HAL
open science

Scheduling for Large Scale Distributed Computing Systems: Approaches and Performance Evaluation Issues

Arnaud Legrand

To cite this version:

Arnaud Legrand. Scheduling for Large Scale Distributed Computing Systems: Approaches and Performance Evaluation Issues. Distributed, Parallel, and Cluster Computing [cs.DC]. Université Grenoble Alpes, 2015. tel-01247932

HAL Id: tel-01247932

https://theses.hal.science/tel-01247932

Submitted on 23 Dec 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE GRENOBLE

Habilitation à Diriger des recherches

Spécialité : **Informatique**

—— 2 novembre 2015 ——

Présentée par

Arnaud LEGRAND

Préparée au sein du **LIG, Laboratoire d'Informatique de Grenoble** et de l'**École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Scheduling for Large Scale Distributed Computing Systems: Approaches and Performance Evaluation Issues

Thèse soutenue publiquement le **2 novembre 2015**,
devant le jury composé de :

M. Yves Robert, Président
Professor at the École Normale Supérieure de Lyon
Senior Member, Institut Universitaire de France

M. David Abramson, Rapporteur
Professor at the University of Queensland, Australia
Director of the UQ Centre for Research Computing

M. Evripidis Bampis, Rapporteur
Professor at the University Pierre et Marie Curie, Paris

M. Marc Snir, Rapporteur
Professor at the University of Illinois at Urbana-Champaign
Director of the Mathematics and Computer Science Division at the Argonne National Laboratory

Mrs. Petra Berenbrink, Examinatrice
Professor at the School of Computing Science at Simon Fraser University

M. Frédéric Desprez, Examinateur
Senior researcher at INRIA Grenoble
Deputy Scientific Director at INRIA

Thanks

God only knows what I'd be without you!
– The Beach Boys, 1966

Fortunately, I did not wait until writing this document to thank all the persons I had the chance to work with. Indeed, the more you wait, the more there are friends and colleagues to thank, which lengthens even more such document. ☺

The Committee

I warmly thank all the members of the thesis committee for accepting, despite the extremely loaded schedule incurred by their respective obligations, to review my work and for giving me their opinion about it. It is a true honor.

My colleagues from the Grenoble teams MESCAL, MOAIS, and NANOSIM/CORSE

I have been now working for over ten years in Grenoble and the atmosphere is so good that I had absolutely no motivation for writing this document. Brainstorming and launching new research directions with kind and brilliant colleagues was much more fun! There has been so many students that it is hard to list them all so I will only list the permanent researchers who accompanied these nice years. I would thus like to thank all my MESCAL and MOAIS colleagues and former colleagues :

- Bruno Gaujal for your bike stops at our home in the week-ends, for being so deeply honest, having brilliant scientific ideas and being always very enthusiastic about the work of others, in short for being such an excellent team leader;

- Jean-Marc Vincent, with whom I shared an office for so long now, for all the excellent cakes of your wife Aline, for sharing and listening together to the complete Bach collection, for properly introducing me to probabilities, statistics, performance evaluation, Markov chains, perfect sampling, ... the list of topics I learnt from you is endless;

- Jean-François Méhaut, for being such a straight colleague, for settling with me the difficult ending of the thesis of a PhD student a long time ago, for providing me with many scientific collaboration opportunities, for accepting to co-advise two thesis with me and making sure everything would go right and in time;

- Corinne Touati for offering me a splendind apple tree, introducing me to game theory and staying up late at night for writing articles and submitting right before the deadline even if we both hate this. Working with you had a definite influence on my research;

- Olivier Richard for sharing so many comics, for always being in such a good mood, for having such a strong position and vision on how infrastructures should be shared and used, and for all the inspiring work you conducted on experimental computer science;

- Denis Trystram, for sharing your deep perspective on scheduling and combinatorial optimization, for animating the scheduling group in such a good way, for always including me in this group and offering me to collaborate. I hope you do not mind too much that I did not seize so many of these proposals;

- Brice Videau, for your boeuf bourguignon in Denver and for sharing tapas in Barcelona, for your obstination in pursuing true scientific research despite the misunderstanding of many, for introducing me to CEA and BSC colleagues and always pushing collaboration opportunities, and for backing me up with advising Augustin and Luka;

- Yves Denneulin, with whom I shared an office, for introducing me to *Strangers in Paradise*, *Le mot musical du jour*, and the subtleties of IO scheduling;

- Brigitte Plateau, who played a definite role in my recruitment in Grenoble and was always in the background to make us benefit from invaluable opportunities;

- Jean-Louis Roch, for your excellent bottles of Roussette, for being always so enthusiastic about research and for making me understand your vision of parallelism and work-stealing;

- Derrick Kondo for being so patient and introducing me to the subtleties of volunteer computing and to the whole BOINC community. I regret I did not take more advantage of your company before you left back to the Bay area. Thanks again for crossing my path and I hope to see you again soon in sunny California;

- Vincent Danjean, whom I know for so long now, for staying always the same quiet, humble, nice and knowledgeable person, ready to share everything you know about threads, operating systems, version control or cooking.

- Pierre-François Dutot, whom I also know for so long now, for bringing excellent beers at the BBQ last time, for having such a nice family, and for managing to convince your scheduling colleagues that using SimGrid was the way to go!;

- Grégory Mounié, for always being in such an incredible good mood, for always being able to always ask relevant questions in seminars, for being an org-mode proselyte and for bringing my HDR file in time at the college doctoral;

- Florence Perronnin for being so enthusiastic about Muramatsu flutes, for your excellent tiramusu, for your values and simply for being such a nice and well organized colleague. I regret I did not take more advantage of your company before you left for Paris and I hope we will still have opportunities to collaborate in the future;

- Vania Marangozova-Martin, for all the tea/coffee/chocolate/candies we shared with Vincent and Florence, for your good mood, and for making sure FrameSoc would be open source;

- Guillaume Huard for advising so well all these PhD students (Lucas, Swann, Damien, David) and with whom I had later the chance to interact. I hope we will both collaborate more in a near future.

- François Broquedis, who recently joined the MOAIS team, for always being in a good mood and whose jokes remind me so much of the ones of Raymond Namyst ☺;

- Nicolas Gast, who recently joined our team, for keeping me up to date with recent developments on lagrangian optimization and multi-path control flow. I hope we will collaborate in the future;

- Panayotis Mertikopoulos, who recently joined our team, for being so humble, patient and good at explaining, for introducing me to potential games, equilibrium learning, the Ising model, ...
- Clément Pernet, for all the wonderful mountain and ski pictures you shared with us. I wish I were able to enjoy nature as you do;
- Thierry Gautier, whom I should have taken the time to get to know more as our offices have always been quite close to each others. I'm sure you'll have a lot of fun with your new Avalon colleagues in Lyon!
- Philippe Waille for the nice discussions on baroque harpsichord and roleplaying;
- Bruno Raffin, for lending me construction tools and more seriously for always being so nice. I truly appreciate the way you manage research and people, which is nice as I guess we will have to interact more and more in the next years;
- Frédéric Wagner, for being Frédéric! It's quite rare to find in the same person an expert in programming, moucharabieh, bagpipe and mandolin!
- Our platform/sysadmin engineers: Pierre Neyron, Christian Seguy, Bruno Bzeznik, and Romain Cavagna. Having such nice, fun and available colleagues makes our life so much easier. I just hope my music was really not too loud;

I also would like to give a special thank to our assistants Annie Simon, Christine Guimet, and Pascale Poulet who always backed me up with a smile whereas I am always late with administrative stuff... Thanks again Annie for harassing me as I kept delaying this HDR.

The SimGrid team

I thank all the colleagues that believed in and accompanied me in the SimGrid project. None of this would have been possible without them. I owe them a lot.

- Henri Casanova, who started everything. You definitely inspired the whole spirit of SimGrid and managed to find crazy dudes to implement it with him. It is also clear from chapter 1 that I would probably never have engaged in the work presented in this document if I hadn't had the chance to meet you[1].
- Martin Quinson, for your incredibly communicative enthusiasm, believing in crazy stuff that nobody thought would be possible and actually making them real, for never finishing anyth...
- Frédéric Suter, for managing to cope with my lack of organization and for assuming the role of the serious guy that forces us to stop acting like teenage hackers and to communicate like actual researchers.

I also thank all those who recently joined this adventure: Arnaud Giersch, Lucas Schnorr, Augustin Degomme, Lionel Eyraud, Samuel Thibault, Anne-Cécile Orgerie, Adrien Lèbre, ...

My former colleagues from Lyon

Before moving to Grenoble, I worked for quite some time in Lyon where everything started. I am so deeply thankful to Yves Robert, Olivier Beaumont, Frédéric Vivien, Loris Marchal, Alan Su, Frédéric Desprez, Eddy Caron, ... Working with you has been a real pleasure and decisive in what happened next in Grenoble.

[1] By the way, thanks again Yves...

PhD students, Post-doc and Engineers I had the chance to work with

During these 12 years in Grenoble, I had the chance to work with many master or PhD students and engineers.

- Pedro Velho was my first PhD student and I learnt a lot from this first experience. We worked together on the validation of the simgrid fluid models, which happened to be a much more difficult topic than I anticipated. I thank you for your ever-lasting good mood, for bringing me to the cachaçaria and other nice places of Porto Alegre, and for standing me for almost 5 years! I can only congratulate your for conducting this work until the end as you truly paved the way to subsequent work;

- Bruno Donassolo worked with me as a master student. I think I should say I helped him improving the core of SimGrid, crafting up a faithful simulation of BOINC and conducting an interesting game theory study in this context. Your autonomy and efficiency made you a precious student and I thank you for working with me;

- Pierre Navarro worked on the software maintenance of SimGrid and on cleaning up all my clumsy prototypes and fuzzy ideas. He was patient enough to help me rewriting the main SimGrid's resource sharing function with cache-oblivious data structures and to implement the scalable hierarchical routing we still use to date. Thanks for all this time and the endless nights at the SimGrid User Days;

- Rémi Bertin worked as a PhD student under Corinne Touati's and my own supervision. Although Rémi was a very smart person with brilliant ideas and had obtained very promising results, I decided to stop the thesis because I felt he did not believe in what we were trying to achieve and would therefore not be able to write his thesis. It was a difficult decision to take and I hope it helped him moving on something else although I guess I'll keep asking myself whether I made the right choice.

- Lucas Schnorr is the only person I know with whom it is possible to discuss of a nice visualization idea in the morning who and can send you a video of a working prototype in the afternoon. I have been extremely lucky to work with such a brilliant researcher and I hope we will be able to continue working together in a near future;

- Sascha Hunold helped me discovering how to apply simple experiment designs in scheduling studies and definitely helped me building my own opinion on reproducible research through long and fruitful discussions. Thanks for your strong opinions, these nice evenings in Berkeley, Grenoble or Vienna, for standing my unreliability and for willing to improve the practice of our community.

- Augustin Degomme made an amazing work on maintaining SimGrid and making SMPI a reality and not simply a research prototype. All the hard work he accomplished would clearly have been sufficient for obtaining a PhD thesis and I think I even already have the slides ready for the defense. But it is not the diploma that makes you a valuable computer scientist... It is probably a good thing the law forbade me to exploit your skills any longer but I hope we will keep working together for a long time;

- Luka Stanisic came out of nowhere and it took me a few months to realize how brilliant and reliable this incredibly discrete student was. It has been a pleasure to improve our practice and to implement a reproducible research workflow in your company. I keep being impressed by the work you accomplished and the quality of your results. You are putting a lot of pressure on my next PhD students! ☺

My other co-workers and co-authors

Being a researcher leads you to travel a lot and I therefore had the chance to work with colleagues from San Diego, namely Fran Berman, Larry Carter and Jeanne Ferrante. Their vision and guidance has been decisive.

I also had the chance to travel a lot in Brazil, at UFRGS where Philippe Navaux, Claudio Geyer, Nicolas Maillard and Alexandre Carissimi have always been incredible hosts. Not only did they let me interact at will with their students but they also brought me to churrascaria and to their home as if I were family. I really value such kind of collaborations.

Corinne Touati introduced me to Hisao Kameda whom I helped fixing some proofs and technical results. I am grateful for the confidence Professor Kameda granted me and I wish I had more time to keep working with him on his very interesting point of views.

Derrick Kondo introduced me to David Anderson, who is a living legend to my eyes as the creator of SETI@home and BOINC. Not only is David a visionary but he is also a excellent piano player and rock climber. I enjoyed every single moment we spent together in Berkeley and I hope I will manage one day to truly contribute to the BOINC community.

My family

Finally, I would like to thank my lovely wife and my sons who clearly are what I am the most proud of in my life. They are wonderful in every aspect and allowed me to stay away from them for three summers trying to write this document and each time realizing I had more urgent things to do. It is good I finally managed to put an end on this as I should definitely spend more time with them.

I obviously also thank my parents and my brother whom I definitely should call, visit and spend time with more often...

Contents

1 Research Context and History **1**
1.1 Context . . . 1
1.1.1 Computational Science and Digital Revolution . . . 1
1.1.2 Distributed Computing infrastructures: Technology, Engineering and Research . . . 2
1.1.3 Context of my Research . . . 4
1.2 History . . . 4
1.2.1 Heuristics for Scheduling Parameter Sweep Applications in Grid Environments . . . 4
1.2.2 Later Work . . . 8
1.3 Organization of the Document . . . 8

I Scheduling and Game Theory **11**

2 Scheduling and Game Theory Concepts **13**
2.1 Throughput Optimization . . . 13
2.1.1 Motivation . . . 13
2.1.2 Steady-State Scheduling of the Master Worker Problem . . . 14
2.1.3 Platform Modeling . . . 17
2.1.4 Formal Definition of Steady-state Scheduling . . . 18
2.1.5 Application Domains . . . 21
2.1.6 Issues . . . 21
2.2 Divisible Workload . . . 22
2.2.1 Problem Definition and Notations . . . 22
2.2.2 Linear Programming and Optimality Principle . . . 24
2.2.3 Complexity results . . . 24
2.2.4 Issues . . . 25
2.3 Game Theory . . . 25
2.3.1 Non-cooperative games . . . 25
2.3.2 Index-based Fairness . . . 28
2.3.3 Axiomatic Fairness . . . 29

3 Non-Cooperative Throughput Optimization **31**
3.1 Platform and Application Setting . . . 31
3.2 Non-Cooperative Scheduling . . . 32
3.3 Nash Equilibrium and Pareto Inefficiency . . . 34
3.4 Resource Augmentation . . . 36
3.5 Conclusion and Open Issues . . . 36

4 Max-min Fair Throughput Optimization **39**
4.1 Platform and Application Setting . . . 39
4.2 Computing the Max-min Fair Solution . . . 40
4.3 Demand-driven and Distributed Heuristics . . . 42
4.3.1 Demand-driven Scheduling . . . 42
4.3.2 Distributed Scheduling . . . 43
4.4 Conclusion and Open Issues . . . 44

5 Proportionally Fair Distributed Throughput Optimization **47**
5.1 Platform and Application Setting . . . 47
5.2 Introduction to Flow Control in Multi-path Networks . . . 48
5.2.1 Fairness and Network Protocol Design . . . 49
5.2.2 Flow Control in Multi-path Networks . . . 51
5.3 Toward a Distributed Algorithm for Throughput Optimization . . . 53
5.3.1 Computing Partial Derivatives . . . 53
5.3.2 Distributed Algorithm Design . . . 54
5.3.3 A Disappointing Behavior . . . 55
5.4 A Non-trivial Adaptation . . . 56
5.5 Conclusion and Open Issues . . . 60

6 Centralized Response Time Optimization **63**
6.1 The GriPPS Infrastructure . . . 63
6.2 Scheduling for the Divisible Load Model . . . 65
6.2.1 Framework and Notations . . . 65
6.2.2 Relationships with the Single Processor Case with Preemption . . . 65
6.2.3 Optimization Criteria . . . 66
6.2.4 Sum Stretch: Shortest Processing Time Rules . . . 68
6.2.5 Max stretch: Deadline Scheduling and Linear Programming . . . 69
6.3 A Pragmatic Scheduling Algorithm . . . 72
6.4 Conclusion and Open Issues . . . 74

7 Non-Cooperative Throughput and Response Time Optimization **77**
7.1 The BOINC project . . . 77
7.1.1 History and Evolution of the BOINC Workload . . . 77
7.1.2 The BOINC Server . . . 78
7.1.3 The BOINC Client . . . 79
7.2 A Game Theoretic Modeling . . . 79
7.3 Simulation Results . . . 82
7.3.1 Sensibility Analysis . . . 83
7.3.2 Utility Set Sampling and Nash Equilibrium . . . 84
7.4 Conclusion and Open Issues . . . 87

8 Ongoing and Future Work **89**

II SimGrid: an Open Simulation Framework **91**

9 Context **93**
9.1 Motivation . . . 93
9.2 The SimGrid Architecture . . . 95
9.3 Related Simulators . . . 98

10 The Validation Quest **101**
10.1 Background and Related Work on Network Modeling . . . 102
10.1.1 Packet-level Models . . . 103
10.1.2 Delay-based Models . . . 103
10.1.3 Flow-level Models of TCP . . . 104
10.2 Accuracy of Flow-level Simulation . . . 106
10.2.1 Toward an Informed Flow-Level Network Model . . . 106
10.2.2 Following the (In)validation Path . . . 109
10.3 Adaptation to the HPC Context . . . 110
10.4 Conclusion . . . 113

11 The Scalability Quest **115**
11.1 Flow-level Models vs. Packet-level Models . . . 115
11.2 Efficient Simulation Kernel . . . 118
11.2.1 Efficient Resource Sharing . . . 118
11.2.2 Lazy Activity Updates . . . 120
11.2.3 Trace Integration . . . 120
11.2.4 Illustrating Scalabity With a Volunteer Computing Simulation . . . 121
11.2.5 Illustrating Scalability With Unstructured Communications Simulation . . 122
11.3 Efficient Platform Representation . . . 123
11.3.1 Illustrating the Efficiency of Platform Representation With Grid Computing Simulations . . . 124
11.4 Efficient Process Representation and Simulator Architecture . . . 125
11.4.1 Illustrating Process Scalability with Peer-to-peer Simulations . . . 126
11.5 Future Work . . . 128

12 Ongoing and Future work **129**
12.1 Accurate and Scalable Simulation of HPC Applications . . . 129
12.1.1 The SMPI Project . . . 129
12.1.2 StarPU over SIMGRID . . . 131
12.1.3 And Beyond . . . 133
12.2 Methodological Aspects . . . 136

III Appendix **139**

A Collaborations and Grants **141**
A.1 Projects and Grants . . . 141
A.1.1 ANR SONGS (Simulation of Next Generation Systems): 2012–2015 . . . 141
A.1.2 European Mont-Blanc projects: 2012–2016 . . . 141
A.1.3 ANR USS-SimGrid (Ultra-Scalable Simulations with SimGrid): 2009–2012 . 142
A.1.4 ADT SimGrid for Human Beings: 2010–2012 . . . 142
A.1.5 ANR DOCCA (Design and Optimization of Collaborative Computing Architectures): 2007–2011 . . . 142
A.1.6 ANR ALPAGE (Algorithms for Large-Scale Platforms): 2005–2008 . . . 143
A.2 Collaborations and Joint Laboratories . . . 143
A.2.1 Joint laboratory on *petascale* and *extreme-scale* computing: 2011–2015 . . . 143
A.2.2 Action d'Envergure Inria HEMERA (developing large scale parallel and distributed experiments): 2010–2014 . . . 144
A.2.3 Grenoble - Berkeley Associated Team: 2009–2013 . . . 144
A.2.4 Grenoble - Porto Alegre Associated Team and Joint Laboratory . . . 144

B Publications **145**

C Bibliography **153**

Chapter 1

Research Context and History

This document presents a summary of the research activities I have conducted after obtaining my PhD degree in December 2003 from the École Normale Supérieure de Lyon, under the supervision of Yves Robert and Olivier Beaumont. This research work has been done at ÉNS Lyon until September 2004, at University of California at San Diego until September 2005, and at the computer science laboratory of Grenoble (LIG) where I hold a tenured full time researcher position from CNRS since September 2004.

In this chapter, I first present the scientific context in which my research was done and try to provide a brief justification of the state of mind in which it was conducted. Then I present the evolution of my work since its early stages in June 1999 when I was a Master intern at UCSD, working under the supervision of Henri Casanova and Francine Berman. This work and the discussions I had with Henri and Fran at this time were decisive for the rest of my work and I think it is thus justified to discuss it in this document as it should help understanding the paths I have followed since almost 16 years now.

1.1 Context

1.1.1 Computational Science and Digital Revolution

Computer science is a very recent science that really emerged only in the late 20th century although one could argue its premises start with the Sumerians back in 2700–2300BC. It has been the basis of a massive worldwide industry and has influenced both society and other fields of science through both **technology** and **science**.

The digital revolution started in the second half of the 20th century with the mass production and widespread usage of computer related technologies. The consequences on society have been tremendous and are due both to computer/telecommunication technologies and computer science (e.g., RSA algorithms that secure our transactions, model checking of complex systems, compression algorithms, recommendation algorithms,...).

Interestingly, computer science has somehow influenced other sciences like biology that now use notions of information and coding as common tools and concepts [Win06]. DNA sequences are thought as strings of a language, cells are seen as self-regulatory systems similar to an electronic circuit, interactions between molecules such as proteins and RNA are seen as a process calculus. There is a clear hope data structures and algorithms can help us understand the structure and interactions of proteins in ways that elucidate their function at a global scale. Computational thinking is changing the way biologists think because it offers new ways to conceive phenomena and it has also started influencing other disciplines like physics, chemistry, geo-sciences, economics, laws...

Interestingly, computer science and technology also offer new ways of discovering and analyzing phenomena. Scientists have quickly realized that pencil and paper alone cannot solve all our problems and that computer could be used be used as a scientific instrument. The ability to

dig the vast amounts of data now accessible to scientists and to solve complex numerical systems has brought us two new scientific paradigms:

Big data The widespread adoption of computer technology has enabled the collection of huge data sets and their exploitation. Although the term "big data" was coined rather recently (e.g., in [Die03]), this was clearly a concern decades ago and motivated the development of major computational infrastructures. Along computational infrastructures comes the need to develop software infrastructures, algorithms, machine learning techniques to support data analytics. Data sources can be sensors, transaction records, genome and protein databanks,... and this obviously changed approaches in both science, industry and society that now rely on such techniques. The most striking about such approach is that it enables to discover phenomena or truths that would otherwise remain unseen.

***In silico* science or computational science** Equations and phenomenon investigated by scientists have such a level of complexity that they have quickly become too complex to be conducted solely by hand, hence the need for computers. Scientists model the real world and devise simulations, i.e., computer programs that solve the corresponding equations and compute the evolution the system under study in a very controlled way. Performing the corresponding experiments in real world would generally be too costly (in term of infrastructure, money, energy, or time) and even sometimes simply impossible. Industry and scientists can thus can explore and investigate designs or phenomena in a few hours which would take months or even years to study using classical experiments involving the construction of miniatures. Hence, all fields of science (physics, genomics, astronomy, ecology,...) and industry (drug design, avionics, structural engineering, oil companies,...) now heavily rely on computers to screen among a set of potential designs or investigate completely new designs.

Computing infrastructures have thus become a fundamental tool for all domains of science and industry.

1.1.2 Distributed Computing infrastructures: Technology, Engineering and Research

The exponential increase of processor speed that has been observed since their creation in the 50s has always been insufficient to answer the ever growing need of scientists for computing power. This unsatisfied appetite is answered by aggregating several (dozens, thousands or millions depending on the context and the decade) processing units with a more or less implicit communication network (see Fig. 1.1). This domain is known under various names such as parallel computing, distributed computing, high performance computing, supercomputing,... and has known several up and downs.

For some time, this research domain was a niche driven by state agencies, military research and major companies and had little direct societal impact. But very quickly new techniques (parallelism, vector processors, pipelining, interconnection technology) were transferred to major industry then to smaller companies and finally in more massive productions, which in turn had a major impact on the way high-end super-computer are built.

Interestingly, research in parallel and distributed computing follows trends and technology developments as well as it influences them. There are many examples of great ideas developed decades ago and that could not be applied because technology was not mature enough or market at this time was not ready and had cheaper alternatives.

As a consequence, although research in this domain has direct applications and is most of the time motivated by current practical considerations, researchers have to anticipate the needs and issues that will arise in the next years or decade. This difficulty is all the more significant as computer science and technology have evolved and keep evolving at an incredible speed. Whenever a machine or a technology goes on the market, it often becomes obsolete within a few

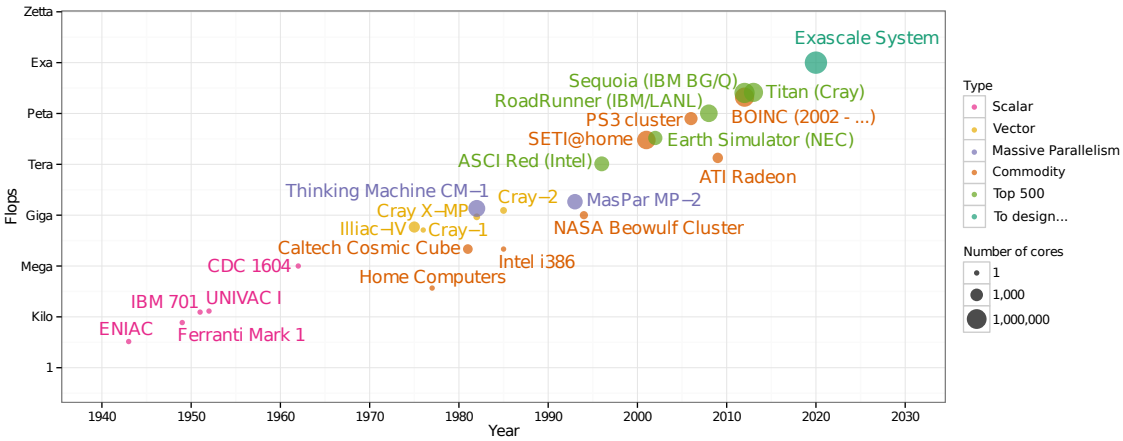


Figure 1.1: Illustrating the evolution of computing infrastructures through a few emblematic platforms

months (not to say weeks). This requires researchers to discriminate real innovations and trends from fad.

Hence, in this domain of computer science, research requires to anticipate technology (r)evolutions, market needs, and societal needs to address what could be the right questions to answer. In this context, the fact that researchers sometimes try to answer questions that are not yet posed by practical situations raises at least two distinct issues that still puzzle me:

- How to evaluate the *relevance of a question* that is supposed to arise from a situation that does not exist yet? It may be the case that this question, even motivated by an hypothetical practical situation, will simply never appear. Fortunately, history (both in classical sciences and in the short history of computer sciences) is full of examples where even though some questions have never aroused, the answer to such questions has been useful to answer other unforeseen questions (note that this may be one of the reason why we have seen such an explosion of knowledge in the last century along with the ability to make connections between various fields of science thanks to the development of communication technologies.) Although society may be satisfied (or not) by such an answer, I consider it is not that obvious to manage for a researcher and this issue is part of my recurrent doubts as I can only answer it by personal faith or support from my peers and a few historical facts.

- How to evaluate the *relevance of the solution* to a practical problem that does not exist yet? This issue is especially tricky as it raises the question of evaluation, which goes to the root of the concerned domain of science. Any science is expected to define its subject and method of inquiry. This may not be sufficient to completely define such an activity but this is clearly an important question to answer. In mathematics, for example, where one aims at building coherent theories and proving hypothesis, the evaluation of a result is rather clear. The proof of a result is either true or false, disregarding of whether it is interesting/elegant or not[1]. In physics, where one aims at investigating natural phenomena, one is concerned by collecting data and building experiments that either support or disprove a theory. In engineering, where one aims at solving problems and constructing systems, the quality of a solution is evaluated by designing, implementing and testing this solution. As explained by Tedre [Ted07], computer science somehow borrows to the essence of these three disciplines (and also from others) and to their method of inquiry, which sometimes leads to methodological confusions. Compared to other disciplines, computer scientists publish relatively few papers with experimentally validated results. While this may not be an issue for those

[1] Although elegance is definitely valuable as it often bring more insights in the result.

of us who are mainly concerned with theoretical considerations, I think this is an issue for those who claim to tackle problems that have a practical application, a direct connection to the real (present or future) world. Again, this issue still puzzles me and motivates some of my research.

Most researchers and engineers implementing and managing these computing infrastructures have to deal with a lot of technical and human details and often have to deal with the most pressing matters first. Some, like Butler Lampson [Lam83], say "*In allocating resources, strive to avoid disaster, rather than to attain the optimum*". This is certainly true when one needs to build a fully functional system taking into account all physical, engineering and budget constraints. Talking about optimality could even seem illusive. Yet, despite all the knowledge and engineering insights collected decades after decades, the complexity of such infrastructure with optimization at every level often turn these systems into bloatware that go beyond understanding. I think that trying to define optimality and how to achieve it is an honorable goal as it may provide interesting insights in the management of such architectures. Indeed, distributed systems appear in a variety of context and have thus been studied with very different tools and approaches. Being able to identify which concepts and techniques can provide the right insight, approach, and structure to manage workload and technology evolution is the key to an efficient management of such systems. Furthermore, I think such investigation has to be done before commercial or societal pressures quietly drive us to designs that are less appropriate.

1.1.3 Context of my Research

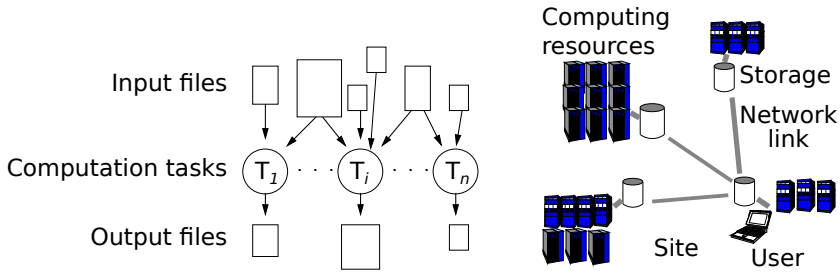
In summary, although our everyday life and society now depends heavily on communication infrastructures and computation infrastructures, scientists and engineers have always been among the main consumers of computing power. My research targets the management and performance evaluation of large scale distributed computing infrastructures such as clusters, grids, desktop grids, volunteer computing platforms, clouds,... when used for scientific computing. More specifically, I have interest in understanding how to make a better use of these platforms and possibly to extend their applicability to other workload than those for which they are already efficiently used. Although my motivations are quite practical, my work is mostly theoretical but done in connection with practitioners and applied whenever possible in order to keep my modeling assumptions as reasonable as possible.

1.2 History

1.2.1 Heuristics for Scheduling Parameter Sweep Applications in Grid Environments

My research really started in 1999 during my Master internship with Henri Casanova at UCSD, in the AppLeS (Application Level Scheduling) group lead by Francine Berman [BWF+96]. This group was involved in what was a very hot topic: grid computing. One can distinguish between several categories of parallel applications described in various formalism (data flow, BSP, moldable tasks, mixed parallelism, ...) and in a complex context such as a grid comprising several computing sites interconnected through an heterogeneous network, it did not seem reasonable to come up a universal efficient solution. That is why the AppLeS group was trying to design an *ad hoc* scheduler, i.e., an entity responsible for the management of computations and data communications, for every such class of application. The motivation for the study of such or such class often came from a collaboration with other scientists that had computation intensive needs and wanted to benefit from the emerging grid technology.

The class of application I have worked on was the one of parameter sweep applications. Such applications are structured as a set of computation tasks that are mostly independent. Each computation task may depend on several files that have to be transferred beforehand and although



(a) Application structure [65] (b) Platform model [CB03]

Figure 1.2: Scheduling parameter sweep applications: APST view.

some computation tasks may share some input files, the output of a task is never used as the entry of an other computation task. In spite of its apparent simplicity, this application model arises in many fields of science and engineering, including computational fluid dynamics, bio-informatics, particle physics, discrete-event simulation, computer graphics, etc.

In the context of AppLeS, we modeled such applications as follows: each application comprises a set of computation tasks T_i that depend on a set of input files I_i and produce an output file O_i (see Fig. 1.2(a)). Hence an input file could be shared or not by multiple tasks. The grid computing platform was modeled as a set of computing sites available to the user (see Fig. 1.2(b)). Each site comprises a storage facility, possibly several computing resources, and is accessible from the user through a network link. We assumed that file transfer time on these links could be estimated from the file characteristics and using tools like the NWS [WSH99]. Likewise, we assumed that task runtime estimates of computation tasks could be obtained for each set of resources. Such estimates were expected to account for potential task/resource affinities. In such a context, scheduling the whole bag of tasks requires to balance the load between the different sites, to exploit data sharing pattern and to organize input file transfers.

From a complexity and algorithmic point of view, there are several difficulties in such a problem:

1. Computation times are heterogeneous and vary from a task to another. This problem is already NP-complete in the case of homogeneous machines and when there is no communication to take into account ($\langle P||C_{\max}\rangle$).

2. When considering negligible computation time, homogeneous communication links, and no data sharing, the problem trivially reduce to $\langle P||C_{\max}\rangle$ if one assumes that communication can be done in parallel from the master. The problem is made even more complex because of data sharing

3. In practice, there would be limits on the amount of data that can be stored on a site, which adds the difficulty of deciding which files need to be deleted and when as the scheduling proceeds.

Since this problem has many sources of combinatorial difficulties, we decided to use simple but robust list scheduling heuristics. Indeed, list scheduling is a 2-approximation for $\langle P||C_{\max}\rangle$ although particular list scheduling can have a better approximation ratio (like LPT, which is a 4/3 approximation for $\langle P||C_{\max}\rangle$). In our context, there was obviously no hope of obtaining a guarantee but list-scheduling was a reasonable starting point. The heuristics we studied were adaptation to this particular context of classical list heuristics like LPT (aka max-min), SPT (aka min-min) and sufferage (that always schedules the task that would "suffer" the most from not being scheduled on its best host). Our heuristics relied on a dynamic estimation of communication and computation time and took into account the presence of files on the different sites. The Xsufferage heuristic that we designed had the better results and was inspired by sufferage but

tried to discriminate between groups of machines that lead to similar characteristics instead of only considering the best two machines.

Now, from a modeling point of view, there are other difficulties that can all be answered in the negative:

1. Is it reasonable to assume that computation time is known beforehand for each combination of task and platform? Is it reasonable to trust user estimations?

2. Although we can easily estimate input data size and how tasks depend on it if user correctly describe their problem, can we really have a good estimation of communication time? In particular, it is likely that there will be contention on the output connection of the user machine. Should we then assume that communications are done sequentially? Or should we assume that they can all occur in parallel with no degradation?

3. Can we assume that communication and computation resources will be dedicated and that their performance will remain stable over time?

For all these reasons, a tightly organized schedule resulting from a long optimization of the previous scheduling problem but using inaccurate information may lead to really bad overall completion time in practice. It would certainly be better to handle this issue by considering directly such uncertainties in the problem modeling but it would certainly involve more elaborate techniques that we did not master and could have other limitations as well. So we decided it would be important to evaluate the sensibility of our algorithms to such uncertainties. Furthermore, since replaying a workload in a non-controlled environment such as a grid was difficult, we decided to rely on simulations to assess the quality of our heuristics and their sensibility to uncertainties.

The conclusion was that although deciding which scheduling algorithm is appropriate for which situation is a difficult question, the XSufferage heuristic seemed effective when large input files are shared by several application tasks and when performance prediction errors are within reasonable bounds. However, in an environment where resource availability varies significantly, thereby making performance unpredictable, or when the amount of application data is small, a greedy non-clairvoyant algorithm seemed more appropriate.

This work led to my first article in the Heterogeneous Computing Workshop, held in conjunction with IPDPS 2000 [65] and is by far the most cited of my articles[2] Interestingly, this work had all the seeds of my current work:

Scheduling independent tasks on a distributed computing platform Many applications are embarrassingly parallel and this simple application context already raises many interesting and non trivial issues. Many people would consider that this is a solved problem but I think this is not true in the general case. We have systems that can manage efficiently a few large compute-bound applications but I do not think we really understand how to deal with more general situations. Although heterogeneity is an inherent key characteristic of such platforms and has to be accounted for from the beginning, it may not necessarily make the problem much more complicated (unlike what I initially thought). However, communication modeling is difficult and needs to be accounted in a reasonable way whenever possible.

Pragmatism and alternate modeling I have always tried to stay away from what I would call artificial complexity. Knowing that a problem is NP-hard does not really help. It merely provides a free ticket for heuristic investigation. Proving that a problem is NP-hard is more interesting though as it may allow to understand where a combinatorial difficulty hides. However, most of the time such NP-completeness reductions are done using widgets and the resulting difficult instance is somehow far away from any instance one may ever encounter in practice. So narrowing the combinatorial difficulty is a good occasion of

[2] 705 citations according to Google scholar in July 2015.

wondering whether simplifying assumptions should not be used to get rid of an "artificial complexity". Using a slightly alternate modeling may lead to simpler problems that can be solved or that allow to introduce further complex aspects that would be to difficult to express in the initial formulation.

Cooperative optimization During my internship at UCSD, Francine Berman mentioned an interesting issue that she liked to call the "Bushel of AppLeS" problem [BW97]. Obtaining an efficient Application Level Scheduler was a good thing from a single user perspective. Unfortunately, it was unclear what would happen if several users sharing machines were to launch separate AppLeS at the same time. They would use the same sources of information and would clearly step on each other's toes. It could also be the case that the high optimization of an AppLeS, although beneficial to its owner, would be somehow harmful to the rest of users.

Performance evaluation In this work, we relied on a custom simulator that tried to be as realistic as possible and to account for important features (heterogeneous computation and transfer time, variability). The main idea was to try to evaluate algorithms in a context that is much more complex and realistic than the one in which they have been designed although it is not exactly the one in which they are supposed to run in the end.

As I already mentioned, this article is the most cited of my articles. Although it was well written and scientifically sound and honest, it does not contain anything revolutionary and this popularity can probably be explained by the fact that it was on particularly hot subject, in the right conference, at the right time. Looking back, I now see several issues with this work. Although it raised very interesting questions, the answers we provided were far from being satisfying. For example, regarding scheduling, using a greedy but sophisticated approach to optimize the completion time of *each* task was naive since all that matters is the completion of the last one.[3] Hence, focusing on completion time of each individual task is somehow irrelevant, especially when it assumes an *a priori* knowledge of their processing time. At that time, we had obviously realized that the file-task structure could be exploited in a less myopic way and thought of modeling it as a graph or hyper-graph partitioning problem. Our time was limited though and we decided to stay with simpler, more classical and potentially more robust approaches like list-scheduling. Aykanat *et al.* [KA06] have worked on this approach a few years after using hyper-graph partitioning techniques that were their specialties. Such an approach was definitely much smarter. Unfortunately it does not handle well heterogeneity or variability, hence its applicability to a real setting remains questionable and I am not aware of recent developments overcoming such issue.

Brasileiro and Cirne *et al.* faced a similar issue when implementing OurGrid [CBA+06]. Their solution is simple and I think has the right intuitions. When scheduling a BoT, one often has to wait for last tasks during an abnormally long time. When the last tasks are allocated to slow or unreliable workers, this may have a dramatic effect on the overall completion time of the BoT [KCC04, KCC07]. Note that using such workers at the beginning of the BoT is generally not an issue and can still be beneficial. A very effective way to deal with such stragglers is the use of replication toward the end. The mechanism proposed by Brasileiro and Cirne *et al.* [SNCBL04] is thus to use a simple work-queue with replication and data affinity, i.e., where replication is done in priority on workers where files do not need to be transferred.

This last solution is probably all that was needed in this context although one may always find situation where large data transfers and specific data sharing patterns would hinder this myopic (but adaptive) approach. This means that characterizing and selecting the right workload is also a critical step that is often overlooked.

[3] Or maybe it allows to actually optimize the average completion time but none of our heuristics has been evaluated this way.

1.2.2 Later Work

This first contact with scheduling for large scale distributed systems has greatly influenced my subsequent research. When I engaged with Yves Robert and Olivier Beaumont into a PhD, the heterogeneity issue and pragmatic modeling were clearly at stake. I have moved to throughput optimization and divisible load scheduling in 2001 as an alternative to classical scheduling formulations. We have shown during my PhD thesis how linear programming and matching decomposition enable to solve such problems in a wide variety of context. These modeling techniques enable to stay away from artificial complexity issues at little cost and to address more complex/general problems than what can usually be done. In some cases, it could even lead to fully distributed solutions, which appeared to me as a good direction toward autonomous scheduling.

In 2003, after the completion of my PhD, I have started to understand some of the limitations of throughput optimization and I have worked on response time/stretch optimization of divisible jobs with Frédéric Vivien. Another limitation of the work I had done during my PhD was the absence of users in the modeling. This made me feel the need to introduce Game Theory notions in my work starting in 2004, as well as the need to design solutions that can be distributed and have resilience/scalability properties. These two aspects (the need to account for users and the need to design distributed solutions) have been the core of my scheduling research work since then.

During my PhD thesis where I had started designing distributed scheduling algorithms, I have faced the same performance evaluation issues as I had encountered in my initial work with Henri Casanova at UCSD. In the meantime, Henri had turned the toy simulator we had developed into SimGrid, an open-source simulation toolkit. I have extended this tool for my own research and taken over its development, first alone, then with Martin Quinson, and later with other colleagues such as Frédéric Suter as the user community was growing and more recently Arnaud Giersch. This tool was initially mainly designed for my own research but its generic design and purpose has proven to be useful to many other contexts, which strengthened our feeling that open-source development of such tools was the only good way to go. The second motivation for open-source development of such a tool was the feeling that I needed to provide others the ability to freely reproduce (and check) my work and build upon it. Amusingly, SimGrid started as a toy tool for scheduling research and has turned to be a very fruitful source of research and "philosophical" questions in itself.

I have always developed these two research activities (scheduling and performance evaluation through simulation) in parallel and they have surprisingly nurtured each other.

In the rest of this document I try to present and summarize my work in a coherent way and to detail without concession its strengths and weaknesses as well as how I think it could be improved. The goal of such presentation is to try to help others to avoid pitfalls I have encountered and list open paths of research, some of which I intend to pursue.

1.3 Organization of the Document

This document is naturally organized in two parts that correspond to my main two research activities. Each chapter builds on (generally several) articles that were published after the completion of my PhD in 2003.

1. **Scheduling and game theory:**

    - In Chapter 2, I briefly introduce some scheduling notions (mainly throughput optimization and divisible load scheduling) that were mainly developed during my doctoral work, on which I have built upon, and which are thus important to understand the rest of the document. I also introduce a few classical game theory notions that I have relied upon.

- As explained in Chapter 2, optimizing throughput of a single application can be easily done through simple strategies on simple platforms. In Chapter 3, I present some results on the analysis of the situation where the throughput of several applications is optimized independently of each other. Such non-cooperative optimization results give us the opportunity to question some classical inefficiency characterization tools.
- Since non-cooperative optimization generally leads to rather inefficient results, I present in Chapter 4 a first approach to a cooperative optimization of throughput trying to achieve max-min fairness. Such approach was mildly successful and allowed us to realize that both the max-min fairness objective, the communication modeling, and the optimization tools we used were not really suited to this context.
- This is why we present a more adapted approach in Chapter 5, which instead leads to a proportionally fair solution and relies on distributed Lagrangian optimization. Interestingly, although such technique is somehow common in the networking context, it revealed more difficult than expected to apply to our context. I think such approach is very promising for a practical usage although it deserves further investigation.
- Considering throughput optimization allows to get rid of several combinatorial difficulties and to incorporate in the modeling important aspects such as communication and complex topologies. There are however situations where the steady-state hypothesis is too strong and where communications are not that critical. I have thus considered the co-scheduling of relatively large bag of tasks belonging to different users. In Chapter 6, I present some work we conducted on response time and stretch optimization in the divisible load framework. This work allowed to rediscover and understand some optimization techniques, to gain a better understanding of how users could be incorporated in the scheduling picture, and of which scheduling ingredients should be used in a practical implementation.
- Some of these ideas are actually partially implemented in production systems such as BOINC. However, they are barely activated in practice and BOINC was designed to be fully distributed. Hence, there is no coordination between applications and non-cooperative optimization may result in inefficiencies. Since some recent related work mentioned experiencing such issues, we decided to study more precisely such non-cooperative optimizations in the BOINC context, which I present in Chapter 7. This study can be seen as the realistic and practical counterpart of the theoretical problem presented in Chapter 3.
- Finally, Chapter 8 concludes this part by summarizing open issues and presenting ongoing and future work I intend to focus on.

2. **SimGrid: an open simulation Framework.** This part presents my investment in the SimGrid project, which aims at providing an open simulation framework for our community. Indeed, simulation is often used in parallel/distributed computing to evaluate the relevance of proposals. However, this if often done without proper training on simulation and performance evaluation and the resulting tools and studies are often quite disappointing. In particular, as I was particularly unsatisfied with my own studies, I decided rather early to invest myself into improving my practices and possibly the ones of my colleagues.

   - In Chapter 10, I start by invalidating popular simulators and then I present the efforts we conducted to improve the quality of the SimGrid simulator in term of prediction capability.
   - Most articles published on simulators in our community emphasize on the software genericity of the simulator but actually trade accuracy and modeling genericity for speed since large scale systems motivate our research. In Chapter 11, I briefly present how we addressed such scalability and expressiveness issues in the SimGrid project.

- Finally, I conclude this part in Chapter 12 by presenting ongoing and challenging work as well as some topics and concerns I discovered during these years and on which I intend to focus in a near future.

Although the two parts can be considered and are presented as independent works, they have nurtured each others. Many simulation developments were motivated by scheduling study needs. Conversely, the scheduling studies provided us the right workload to focus on and also sometimes the right optimization tools to improve our simulations. I think the synergy between these two topics has been particularly fruitful and instructive.

Part I

Scheduling and Game Theory

Chapter 2

Scheduling and Game Theory Concepts

2.1 Throughput Optimization

In this section, I introduce the notion of steady-state scheduling and try to give a slightly different perspective from the one presented in my PhD thesis. In particular I try to emphasize on the importance of dual formulation (even if the primal formulation is always used to solve problems in practice) as it provides an interesting perspective on problem complexity. I also deliberately use slightly more lightweight notations.

2.1.1 Motivation

Let us assume that we are given an application made of a set of n independent computation tasks with identical characteristics. Each task depends on a private input file whose transfer has to be completed before processing can start. If we are given a master/worker platform (see Fig. 2.1), we can note d_i the time required to send an input file from the master to worker P_i and w_i the time required by P_i to process the corresponding task. On the worker side, we assume that a worker can process only one task at a time and that it can receive an input file while processing an other task. On the server side, we assume that only one file can be sent at a time (one-port model).

Such a model is a crude approximation of the kind of scheduling problem that could arise in volunteer computing platforms such as BOINC or Folding@home. Obviously, such projects have to deal with other issues such as dynamicity or volatility of workers, the imperfect uniformity of tasks within a batch, ... Yet, it captures some of the essence of the difficulty arising from platform heterogeneity.

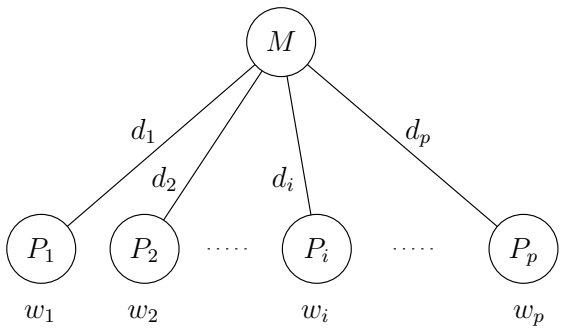


Figure 2.1: Master worker architecture. Worker P_i can process a task in w_i units of time and the master can send a task to worker i in d_i seconds.

Problem 2.1 (**MasterWorker** $(P_1(d_1, w_1), \ldots, P_p(d_p, w_p), n, T)$)**.** Given a master worker platform setting with characteristics $(d_1, w_1), \ldots, (d_p, w_p)$, is it possible to schedule n tasks in less that T units of time?

Unfortunately such formulation raises several issues.

1. In essence, since all the tasks are identical, it would seem reasonable to encode, just as in the previous definition of **MasterWorker**, the input of our problem as a description of the task and the number n of such tasks. Yet, such description raises a complexity issue. A reasonable (and useful) certificate would be a description of when and where each task is executed, hence in size polynomial with n (unless the problem has a particular structure that can be exploited to provide a compact description) and thus exponential in the input whose size is $\Theta(\log n)$. This means that when stating such a problem, a redundant description of all input tasks should be part of the input:

   Problem 2.2 (**MasterWorker-Schedule** $(n, \mathcal{T}, p, \mathcal{W}, \mathcal{D}, T)$)**.** Given

   - a set $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n\}$ of n independent and identical tasks;
   - a set $\mathcal{W} = \{w_1, w_2, \ldots, w_p\}$ of p execution times for each worker;
   - e set $\mathcal{D} = \{d_1, d_2, \ldots, d_p\}$ of p communication delays for each worker;
   - a time bound T;

   is it possible to schedule in less than T units of time the set $\mathcal{T}$ of n tasks on a master worker platform made of p workers P_i whose characteristics are w_i and d_i?

   We have shown in [57] that this problem can be solved in time $O(n^2p^2)$ by a non-trivial greedy algorithm.

2. One may be satisfied with a polynomial complexity of $O(n^2p^2)$ at first but it quickly becomes prohibitive when n and p grow large. What is even more impractical, is that although this problem is solved by a greedy algorithm, the optimal schedule for n tasks may be completely different from the optimal schedule for $n + 1$ tasks. Indeed, although our algorithm is greedy, it builds the schedule backward. This is however a property of optimal solutions and not of a peculiarity of the algorithm.

3. Pierre-François Dutot [Dut03b] extended this result to the case of chain and octopus graph of platforms. Again, the algorithm proceeds backward and has complexity $O(np^2)$. He also proved in [Dut03a] that this problem is NP-complete as soon as the graph as the platform graph is a tree (a chain graph followed by a star graph actually).

Yet, when n grows large, it is expected that some kind of regularity appears with a sort of pipeline where start-up and clean-up phases have little impact on the total completion time if most of the time is spent in steady-state. That is why we proposed in my PhD thesis to optimize the throughput, i.e., the average number of tasks completed per unit of time.

2.1.2 Steady-State Scheduling of the Master Worker Problem

Let us now consider the same platform as earlier but with an unlimited supply of tasks to process.

Problem 2.3 (**MasterWorker-SS** $(p, \mathcal{W}, \mathcal{D})$)**.** Given a master worker setting described as

- a set $\mathcal{W} = \{w_1, w_2, \ldots, w_p\}$ of p execution times for each worker;
- a set $\mathcal{D} = \{d_1, d_2, \ldots, d_p\}$ of p communication delays for each worker;

what is the maximum achievable throughput?

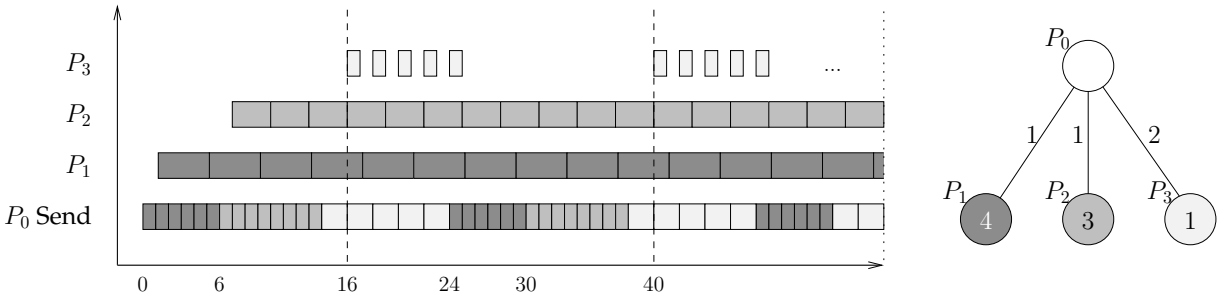


Figure 2.2: Illustrating steady-state scheduling on a simple master-worker platform. Although the third workers is particularly fast, it is kept partially idle as the master cannot send him data sufficiently fast.

Let us consider an infinite schedule of such tasks. Let us denote by $n_i(T)$ the number of tasks processed on P_i in the time interval $[0, T]$. Therefore, the total number of tasks processed in the time interval $[0, T]$ is $n_{total}(T) = \sum_i n_i(T)$ and we have:

$$\begin{cases} \forall P_i, n_i(T) \cdot w_i & \leqslant T \quad \text{(processing constraint)} \\ \sum_i n_i(T) \cdot d_i & \leqslant T \quad \text{(one-port constraint)} \end{cases}$$

Therefore, if we note $\alpha_i = \lim_{T \to \infty} n_i(T)/T$ (assuming such limit exists) the effective task processing rate of P_i, we have

$$\begin{cases} \forall P_i, \alpha_i & \geqslant 0 \\ \forall P_i, \alpha_i \cdot w_i & \leqslant 1 \quad \text{(processing constraint)} \\ \sum_i \alpha_i \cdot d_i & \leqslant 1 \quad \text{(one-port constraint)} \end{cases} \tag{2.1}$$

and the throughput of our schedule can defined as

$$\varrho = \sum_i \alpha_i \tag{2.2}$$

Constraints (2.1) and objective (2.2) define a simple (rational) linear program that can be solved in polynomial time and provides an upper-bound on the throughput of any infinite schedule. From the solution of this linear program, it is possible to easily build a simple periodic schedule that achieves such a throughput. Let us denote by T_p, the lcm of the denominators of the α_i. Let us denote $N_i = \alpha_i \cdot T_p$ the number of tasks that will be sent to P_i every T_p time units. In our schedule, the master simply sends N_i tasks to P_i, one worker after the other and workers process tasks as soon as they have received the input data. Once the master has sent $N_1 + \ldots + N_p$ tasks, it repeats the operation.

Such a technique is illustrated on Fig. 2.1.2. In this toy example, the platform comprises three workers whose communication delays are $d = (1, 1, 2)$ and whose execution times are $w = (4, 3, 1)$. Solving the corresponding linear program leads to $\alpha_1 = \frac{1}{4}$, $\alpha_2 = \frac{1}{3}$ (from the processing constraints), $\alpha_3 = \frac{5}{24}$ (from the one-port constraint), and $\varrho = \frac{19}{24}$. It is thus possible to build a periodic pattern of length 24 where worker 1 processes $\frac{24}{4} = 6$ tasks, worker 2 processes $\frac{24}{3} = 8$ tasks and worker 3 processes $\frac{5 \times 24}{24} = 5$ tasks, hence the desired throughput.

Although, for small values of N, the previous periodic schedule is likely to produce bad performance compared to the solution of **MasterWorker-Schedule**, it is expected that when N grows large, the difference vanishes.

Theorem 2.1. *Let us denote by $\varrho_{opt} = \frac{K}{T_p}$ the optimal throughput, $n_{opt}(T)$ the optimal number of tasks that can be done in T units of time, and $n_{per}(T)$ the number of tasks that are processed in T units of time*

by the periodic schedule.

$$n_{per}(T) \leqslant n_{opt}(T) + \varrho_{opt} T_p, \textit{ hence } \lim_{T \to \infty} \frac{n_{per}(T)}{n_{opt} T} = 1.$$

Proof. We have $n_{opt}(T) \leqslant \varrho_{opt} T$. In time T, there are $\lfloor \frac{T}{T_p} \rfloor$ full periods and since data sent at a given time are completed at most T_p units of time later, at least $\varrho_{opt} T_p (\lfloor \frac{T}{T_p} \rfloor - 1)$ are processed. Therefore $n_{opt}(T) - n_{per}(T) \leqslant \varrho_{opt} T_p$, hence the result. ■

The periodic schedule is thus asymptotically optimal for the **MasterWorker-Schedule** problem. Yet, since **MasterWorker-Schedule** and **MasterWorker-SS** can both be solved in polynomial, one may wonder what steady-state really brought.

1. The periodic algorithm is much simpler and does not depend on n.
2. Although the periodic schedule would probably be of no practical use as such, we can derive a nice intuition. The previous linear program is so simple that there is no need to resort to a classical linear programming solver. It can be interpreted as a rational knapsack, which means that workers should be prioritized according to their delay and saturated. Counter-intuitively, one should not serve the fastest workers first but instead the workers that have the best bandwidth as they are the ones that will keep the master busy as little as possible and allow him to serve other workers. We called such strategy *bandwidth-centric* in [56, 14].
3. Such simple strategy applies not only to star platform but also to tree topologies. Indeed, we proved in [56, 14] that a subtree could be summarized by a single value corresponding to its aggregated computing capacity. By recursively applying this principle (see Fig. 2.3), it is thus possible to compute the overall throughput of a tree topology

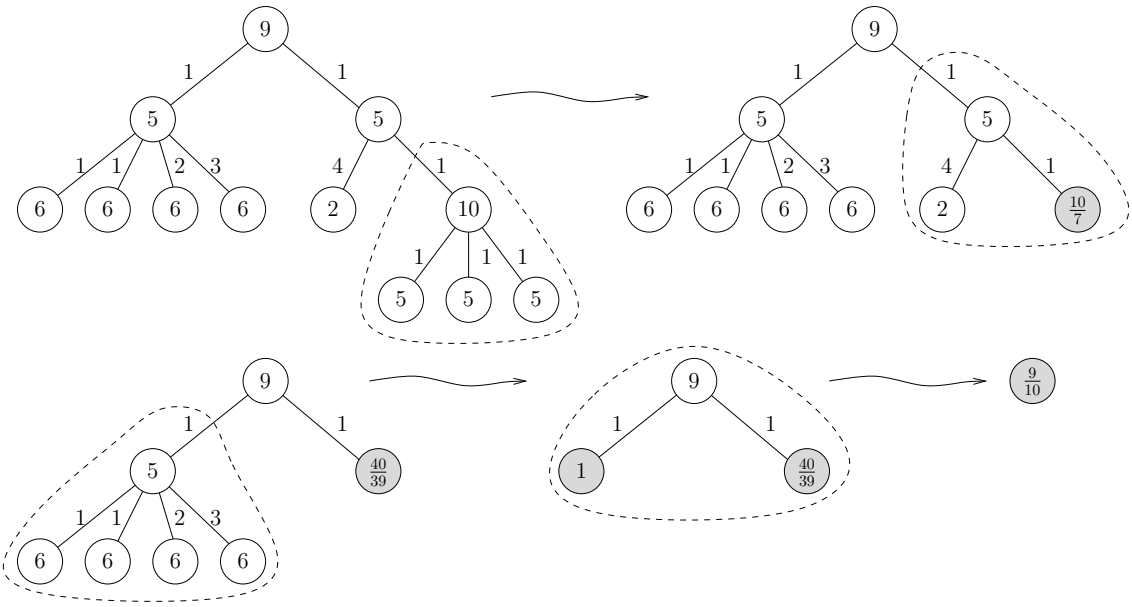


Figure 2.3: Computing the optimal throughput on a tree topology using the bandwidth-centric property and recursively aggregating subtrees. The throughput of a subtree is the inverse of the weight of its root.

4. Last, if on the one hand, **MasterWorker-Schedule** becomes NP-hard as soon as the platforms becomes a little more complex, **MasterWorker-SS** on the other and, remains polynomial, even for arbitrarily complex platforms.

2.1.3 Platform Modeling

The platform model of the previous problem is very simple and limited. For larger platforms, network topology may matter. Therefore, we present in this section how more complex platforms can be modeled and we explain later how this impact the steady-state scheduling problem.

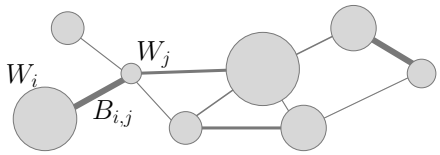


Figure 2.4: A resource graph labeled with node (computation) and edge (communication) weights.

We represent the target computing and communication resources by a *platform graph*, i.e., a node-weighted edge-weighted graph $G = (P, E, W, B)$, as illustrated in Fig. 2.4. Each node $P_i \in P$ represents a computing resource that can deliver W_i floating-point operations per second. Each edge $e_{i,j} : (P_i \to P_j) \in E$ is labeled by a value $B_{i,j}$ which represents the bandwidth between P_i and P_j[1]. We assume a linear-cost communication and computation model. Hence it takes $X/B_{i,j}$ time units to send a message of size X from P_i to P_j.

For the sake of clarity, we ignore processor-task affinities; instead, we assume that only the number of floating-point operations per second (W_i for processor P_i) determines the application execution speed. However, taking such affinities into account does not change any of the results presented in this document. We assume that all W_i are non-negative rational numbers. For any i, $W_i = 0$ means that P_i has no computing power but yet, can forward data to other processors. Similarly, we assume that all $B_{i,j}$ are positive rational numbers (or equal to 0 if there is no link between P_i and P_j).

Generally, one assumes that a processor can only compute one task at a time. However, in the literature, one can distinguish between several communication modes.

1-port models In the **simplex 1-port** model, a node can be engaged in at most one communication at a time. This requires a careful synchronization of all senders and receivers when building the schedule. On the other hand, in the **full-duplex 1-port** model, a node can both send and receive data at the same time although it can be engaged in at most an emission at a time and a reception at a time. In such models, one generally assumes that a data can not be re-emitted until it has been completely received. Fundamentally, 1-port models impose constraints on the time a node spends communicating data.

Multi-port In practice, computers do not have to be engaged in only one communication at a time. In the **multi-port** model, a processor can thus be engaged in communications with all its neighbors at the same time with no degradation [BSP+99, BNGNS00]. This model is found in most DAG scheduling problems involving communications. Note that the ability to send data in parallel with no overhead does not remove the constraint that a link can be used to carry only one file at a time. The multi-port model is thus far less restrictive than the one-port model and easier to handle but it can also lead to unrealistic situations when a node has a large number of neighbors. To address the lack of realism of the previous model, some authors have proposed the **bounded multi-port** [HP07] where each node is associated a bound on the amount of data it can receive and sent at any instant. Fundamentally, such models impose constraints on the amount of data that can be exchanged and are thus less strict than 1-port models which lead to more combinatorial issues.

Communication/computation overlap In the simplest form, one could assume that when a processor is processing a task, it becomes unable to manage communications. This is somehow

[1] Such modeling allows to easily account for asymmetrical connections.

true in single-threaded programs but may not be a reasonable assumption anymore with the advent of multi-core machines and thread programming. On the other hand, one could assume that communications and computations can be perfectly overlapped with no degradation. This is obviously an approximation since in a real machine, both actions would contend for example on the memory bus but attempts to model such level of interferences are not really convincing yet. Hence, in the rest of this work, we always assume that it is possible to both communicate and compute (on other data than the ones that are currently received) at the same time.

2.1.4 Formal Definition of Steady-state Scheduling

We consider an application made of N independent tasks $\mathcal{T}_1, \ldots, \mathcal{T}_N$ to process, with N large and such that all tasks have similar characteristics. Hence, they can be characterized solely by an amount of data b (in bytes) to transfer and an amount w of computation (in Mflop). We further assume that all tasks originate from a particular node (the master) denoted by M. For sake of simplicity, we assume that the only communication required is outwards from the master nodes, i.e., that the amount of data returned by the worker is negligible. Considering inward data would incur only minor modifications to the remaining equations and algorithms. Again, for sake of simplicity, we assume there is no 1-port constraints (we will come back later on the technical difficulties introduced by such kind of constraints). Such modeling allows us to define the following notions, which we try to keep lightweight and intuitive:

Definition 2.1 (Allocation)**.** An *allocation* π is a function that associates to each tasks $\mathcal{T}_i$ a path in G_P originating from M. Such a path indicates along which links the input data should be forwarded and on which machine the task should be processed. We denote by Π, the finite (but exponential in G_P) set of all such allocations π **of a single task**.

Definition 2.2 (Schedule)**.** A schedule associated to an allocation π is a timing function t_π that indicates when each communication starts on each link of the path and when the processing of the corresponding task starts on the final machine. Such a schedule needs to respect the precedence constraints (i.e., the processing or the re-emission of a file cannot start before it is completely received) and the resource exclusivity constraints.

Definition 2.3 (Duration of a schedule)**.** The duration of schedule is defined as the time elapsed between the beginning of the first emission and the end of the last processing.

Definition 2.4 (Cyclic schedule and throughput)**.** We now assume our set of tasks is not finite anymore but instead that our tasks are indexed by $\mathbb{N}$. The definition of allocation and schedule still make sense but are called *cyclic schedules* (t^c_π) and *cyclic allocations* (π^c) [HM95b, HM95a]. Note that the term cyclic does not impose any notion of cycle or periodicity. It simple means infinite.
The notion of duration has to be slightly adapted. Instead, one can define the duration D_N of the N first tasks as the time elapsed between the beginning of the first (among the N first tasks) communication and the last (among the N first tasks) end of the processing. Such notion allows to define the *throughput* of a cyclic schedule as, provided it exists, the limit:

$$\varrho = \lim_{N \to \infty} \frac{N}{D_N}$$

One may then ask the following question:

Problem 2.4 (**CyclicScheduling** (G_P,b,w))**.** What is the maximal throughput ϱ^* of a cyclic schedule?

Unfortunately, such question raises two issues in term of complexity:

1. Although, one may easily define the set of such achievable throughputs, it may not necessarily contain its supremum.

2. Even if the supremum is achieved by a cyclic schedule, is there such a schedule that can be described in a compact (polynomial) way?

The first issue is answered by looking at a seemingly complex formulation of this problem.

Let us assume that we are given a cyclic schedule (π^c, t^c_π). For a given *single task allocation* $\pi \in \Pi$, let us denote by $n_\pi(N)$ the number of time π is used to schedule the N first tasks with π^c. This allows us to define $\alpha_\pi = \lim_{N\to\infty} \frac{n_\pi(N)}{D_N}$ the frequency of apparition of π in π^c. Such a frequency is thus a positive real number:

$$\forall \pi \in \Pi, \alpha_\pi \geqslant 0$$

A given link $(P_i \to P_j)$ may be used several times by different allocations to schedule the N first tasks of π^c. The amount of data going through this link is thus equal to $\sum_{\pi \ni (P_i \to P_j)} n_\pi(N).b$ and is necessarily smaller than $B_{i,j}.D_N$. Therefore the α_π verify:

$$\forall (P_i \to P_j) \in E, \sum_{\pi \ni (P_i \to P_j)} \alpha_\pi.b \leqslant B_{i,j}$$

Likewise, the computing constraints on (π^c, t^c_π) imply that

$$\forall P_i \in P, \sum_{\pi \ni P_i} \alpha_\pi.w \leqslant W_i$$

Finally, the throughput ϱ of (π^c, t^c_π) verifies

$$\varrho = \sum_{\pi \in \Pi} \alpha_\pi$$

Hence, the α_π are solutions of the following linear program:

$$\begin{array}{l} \text{MAXIMIZE } \varrho = \sum_{\pi\in\Pi} \alpha_\pi, \\ \text{UNDER THE CONSTRAINTS} \\ \left\{ \begin{array}{ll} \text{(2.3a)} & \forall \pi \in \Pi, \alpha_\pi \geqslant 0 \\ \text{(2.3b)} & \forall (P_i \to P_j) \in E, \displaystyle\sum_{\pi \ni (P_i \to P_j)} \alpha_\pi.b \leqslant B_{i,j} \\ \text{(2.3c)} & \forall P_i \in P, \displaystyle\sum_{\pi \ni P_i} \alpha_\pi.w \leqslant W_i \end{array} \right. \end{array} \tag{2.3}$$

The previous definition may seem pointless at first since it has an exponential ($|\Pi|$) number of variables and an exponential number of constraints ($|\Pi| + |E| + |P|$). However, this program provides us several important information:

1. Its optimal solution is rational and is an upper-bound of ϱ^*.
2. There is an optimal solution that corresponds to a vertex of the constraint polyhedron. Since there are $|\Pi|$ variables, such a vertex corresponds to the intersection of $|\Pi|$ hyper-planes, i.e., it corresponds to $|\Pi|$ equalities in the previous constraints. Yet, most (Π) of these constraints correspond to the positiveness of our variables. This means that most of our variables are actually 0, hence that $|E| + |P|$ allocations are sufficient to achieve such optimal solution.
3. Although, it would not be reasonable to use such formulation with a linear programming solver, the dual formulation can be solved with the ellipsoid method. Indeed, the dual formulation of linear program (2.3) is:

$$\begin{array}{l} \text{MINIMIZE } \sum_{(P_i \to P_j)\in E} \frac{B_{i,j}}{b} \beta_{i,j} + \sum_{P_k \in P} \frac{W_k}{w} \gamma_k, \\ \text{UNDER THE CONSTRAINTS} \\ \left\{ \begin{array}{ll} \text{(2.4a)} & \forall (P_i \to P_j), \beta_{i,j} \geqslant 0 \\ \text{(2.4b)} & \forall P_i, \gamma_i \geqslant 0 \\ \text{(2.4c)} & \forall \pi = ((P_M \rightsquigarrow P_{P_k})) \in \Pi, \gamma_k + \displaystyle\sum_{(P_i \to P_j)\in\pi} \beta_{i,j} \geqslant 1 \end{array} \right. \end{array} \tag{2.4}$$

For a given vector (β, γ) checking constraints (2.4a) and (2.4b) is obviously done in polynomial time. Although there is an exponential number of constraints (2.4c), they can be checked in polynomial time. Indeed, let us weight the graph G_P by β and γ and compute the shortest path from M to all other P_i. If the sum of the weights of this path is larger than 1, then (2.4c) are valid otherwise, we have found an invalid constraint. Such a cutting-plane oracle can then be used in the ellipsoid algorithm. Program (2.3) can thus be solved in polynomial time although it requires a rather involved algorithm.

Such formulation and consideration may seem purely theoretical and of little practical use. However, it brings the light on the fact that a steady-state schedule is fundamentally nothing else than the weighted combination of a few allocations. The maximal throughput of Problem 2.4 can be achieved by relying only on the (rational) combination of a few (at most $|E| + |P|$, which is a polynomial number) allocations.

Let us prove now that such an upper-bound can be achieved.

Definition 2.5 (***K***-periodic sequence)**.** A sequence u is said to be K-periodic if and only if it is increasing and there is an $n_0 > 0$ and a $T_p > 0$ such that

$$\forall n \geqslant n_0 : u_{n+K} = u_n + T_p$$

K is the periodicity factor and T_p is the period of the sequence. A K-periodic sequence such that $n_0 = 0$ is said to be strictly periodic

Definition 2.6 (***K***-periodic schedule)**.** A K-periodic schedule is a cyclic schedule such that the corresponding timing are K-periodic. In other words, there is an n_0 such that for each $n \geqslant n_0$, task $\mathcal{T}_{n+K}$ has the same allocation as task $\mathcal{T}_n$ and is scheduled exactly T_p units of time after $\mathcal{T}_n$. Likewise a cyclic schedule is said strictly K-periodic when $n_0 = 0$.
The throughput of a K-periodic schedule with period T_p is $\frac{K}{T_p}$.

The main strength of periodic schedules is their compactness In term of throughput, a (strict) K-periodic schedule is completely characterized by the (rational) dates modulo T_p of the K first tasks. Such dates are called a pattern and are sufficient to build a valid K-periodic schedule of period T_p. This means that one does not need to check precedence constraints when checking the validity of a schedule. Checking resource constraints of the pattern is sufficient.

This means that problem 2.4 is actually in NP since the optimal throughput is rational and a valid certificate is a polynomial set of weighted allocations along with how they are organized (possibly to respect 1-port constraints). Obviously, since we have proved that the linear program could be solved in polynomial time, there is no need to prove that it is in NP anymore. Yet, there are more complex situations (e.g., multicasting in a network) where the steady-state formulation is no more polynomial although it can be proved that the problem remains in NP with this exponential formulation.

Theorem 2.2. ***CyclicScheduling*** *can be solved in polynomial time and the optimal throughput is achieved by a compact periodic schedule.*

In practice, one prefers a different formulation with a polynomial number of variables. Let us consider α_i the average number of tasks processed on P_i per time unit and $sent_{i,j}$ the average number of tasks sent on $(P_i \to P_j)$. Resource impose the following constraints:

$$\begin{cases} \forall P_i, \alpha_i \cdot w & \leqslant W_i \quad \text{(processing constraint)} \\ \forall (P_i \to P_j), sent_{i,j} \cdot b & \leqslant B_{i,j} \quad \text{(communication constraint)} \end{cases}$$

Furthermore, we have the conservation law:

$$\forall P_i \neq M : \sum_{(P_j \to P_i)} sent_{j,i} = \alpha_i + \sum_{(P_i \to P_j)} sent_{i,j}$$

These constraints can then be gathered in the following compact linear program:

$$\begin{array}{l}\text{MAXIMIZE } \varrho = \sum_i \alpha_i, \\ \text{UNDER THE CONSTRAINTS} \\ \begin{cases} \text{(2.5a)} & \forall P_i, \alpha_i \geqslant 0 \text{ and } \forall (P_i \to P_j), sent_{i,j} \geqslant 0 \\ \text{(2.5b)} & \forall P_i, \alpha_i \cdot w \leqslant W_i \\ \text{(2.5c)} & \forall (P_i \to P_j), sent_{i,j} \cdot b \leqslant B_{i,j} \\ \text{(2.5d)} & \forall P_i \neq M : \sum\limits_{(P_j \to P_i)} sent_{j,i} = \alpha_i + \sum\limits_{(P_i \to P_j)} sent_{i,j} \end{cases} \end{array} \tag{2.5}$$

This program has the same solution as (2.3) and is obviously polynomial. The corresponding values of $sent$ and α indicate which fraction of tasks should be processed locally or sent to each neighbor. Such a local view is also more suited to a practical implementation.

2.1.5 Application Domains

As we have previously seen, when the number of tasks grows large, steady-state scheduling provides an efficient way of circumventing the difficulty raised by the combinatorial aspect of classical scheduling formulations. Such problem can be solved through a linear program that summarizes the resource constraints imposed by the platforms to any schedule. This provides an upper-bound on the achievable throughput. By "peeling" the solution of the linear program, it is possible to compute a weighted superposition of (spatial) allocations, that can then be arranged into a valid (temporal) schedule. Such a schedule reaches the desired optimal throughput by construction.

Such approach has been applied to more complex situations where applications are made of an infinite amount of identical DAGs, which happens when managing workflows. The formalism seamlessly handles the fact that some tasks can only be processed on particular machines, which is common in this context and generally raises combinatorial issues, or have speed affinities (e.g., because of GPUs). Such approach has also been applied to collective communications (scatter, gather, broadcast) in Loris Marchal's PhD thesis [Mar06]. A few collective communications (e.g., reduce and multicast) remain NP-hard even in this model.

2.1.6 Issues

Despite the aforementioned benefits, this approach has several drawbacks that need to be pointed out:

- The previous periodic schedule has a very large period (the lcm of the denominators of solutions of the linear program), which has several consequences. As such, start-up and clean-up phases are very long and not optimized at all. This is particularly problematic when scheduling workflows and not solely independent tasks. Furthermore, the time elapsed between the emission of a task by the master and its processing can be of the order of several periods. Last but not least, since all data received within a period are sent at the next period, this requires huge buffers at each node. Furthermore, it is unlikely that the machine characteristics will remain perfectly stable during large periods of time. Obviously such a rigid schedule is of no practical interest.

  On the theoretical side, imposing limits on buffer size or trying to minimize latency quickly leads to NP-hard problems. However, on the practical side, it is reasonable to maintain a weighted list of allocations and to dynamically choose which allocation to use while trying to respect the corresponding weights. Such approach can even be used locally. If each node knows which fraction of tasks should be processed locally and which fraction should be sent to each neighbor, such information can be used to dynamically load balance tasks. This approach somehow enables to adapt to slow load variation assuming that such weights are periodically recomputed.

- Except for a few cases such as star platforms presented in Section 2.1.2 or for tree platforms [56, 14], the values of α and $sent$ are obtained by centrally solving a large linear program whose parameters depend on the whole platform. Combined with the previous load-balancing approach, being able to compute such values in a distributed way would provide a fully distributed approach that may be more scalable and likely to adapt to load variations or churn.

- In this brief and general presentation of steady-state scheduling, we have deliberately omitted to go into the difficulties raised by 1-port constraints and non-overlapping of communications and computations. Such constraints incur an additional difficulty, which is the synchronization of senders and receivers when building the pattern of the periodic schedule. In the full-duplex model, this requires a matching decomposition which seems inherently centralized on general platform graphs. In the simplex model, formulation (2.3) and (2.5) do not have the same solution anymore and matching constraints have to be incorporated to formulation 2.4. Although the problem remains polynomial, 1-port constraints make it significantly more difficult to implement, which is why the multi-port model or the bounded multi-port model should be preferred when possible, especially when seeking for a large scale distributed algorithm.

- The steady state scheduling approach has been successfully used to a rather wide variety of domains but always with only one user. Yet, large scale distributed computing platforms are likely to be used by several users and it would be interesting to know how such additional constraint impact this approach.

2.2 Divisible Workload

2.2.1 Problem Definition and Notations

In the problem of divisible load scheduling, we still consider a large amount of computations that need to be performed by a certain number of worker nodes. Again, the master node originally holds all the data, and the goal is to organize the communications to distribute this data among the participating workers. Albeit, the model is called "divisible" because computations can be divided in chunks of arbitrary sizes. It is thus suited for large "bag-of-tasks" applications, in which the granularity is sufficiently low to be neglected. Just like steady-state, such model enables to stay away from some combinatorial issues that arise with more standard modeling.

In this section, we make the following assumptions that are commonly used in the divisible load scheduling literature. We assume that the master cannot send chunks to more than one worker at a time, following the one-port model. We also assume that a worker may compute and receive data simultaneously. However a worker has to wait for a chunk to be completely transferred before starting processing it. We do not consider transfer of output data back to the master.

Consider a DL application that consists of W independent units of load to be processed. The processing of each load unit involves performing some computations on some input data. Without loss of generality, we assume that the master does not perform any computation. Worker P_i can process a chunk of x load units in xw_i seconds, and the master can send a chunk of x load units to worker i in $s_i + xb_i$ seconds (s_i is a startup term for initializing the communication) . We assume that the w_i's, s_i's, b_i's, x and W are rational. A major difference with the previous model is that communications and computations can be divided in chunks of arbitrary sizes.

The problem we consider is: how should the master partition the load into chunks and send those chunks to the workers so that the application makespan, i.e., the time at which the last unit of load is completed, is minimized? A schedule consists of a sequence of workers to which the master sends load chunks in order, which is called the *activation sequence*, and the size of each load chunk. We denote by $\alpha_i^{(j)}$ the size of the jth chunk of load sent to worker i, measured as a

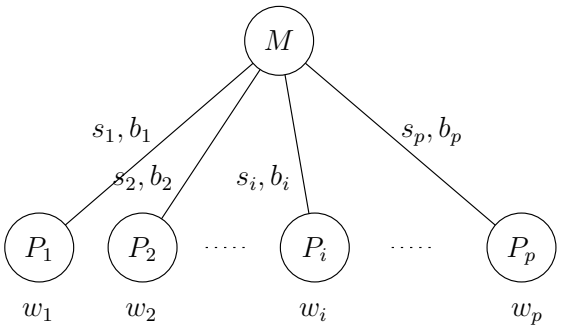


Figure 2.5: Master worker architecture in the DL framework. Worker P_i can process a chunk of x load units in xw_i seconds, and the master can send a chunk of x load units to worker i in $s_i + xb_i$ seconds.

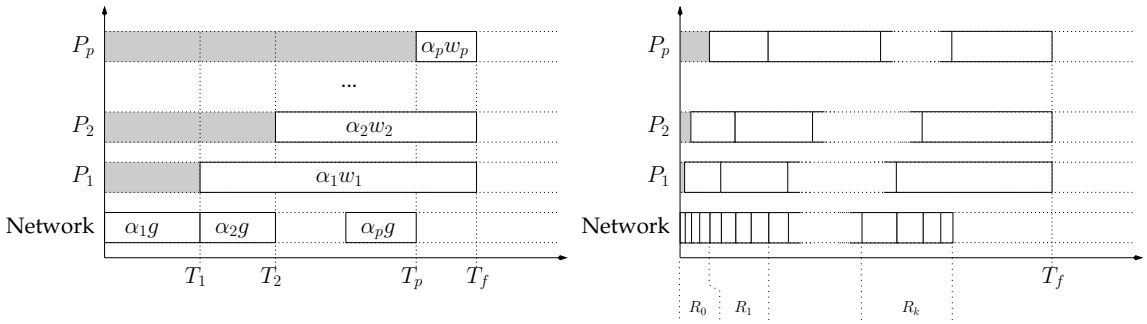


Figure 2.6: Single round versus multi-round for one-port divisible load scheduling. The gray area represents idle-time.

rational number of load units. α_i denotes the size of the chunk of load sent to worker i in case only one chunk is sent to worker i in the schedule.

We define the associated decision problem as:

Problem 2.5 (DLS)**.** Given p workers, $(w_i)_{1 \leqslant i \leqslant p}$, $(s_i)_{1 \leqslant i \leqslant p}$, $(b_i)_{1 \leqslant i \leqslant p}$, and two rational numbers $W \geqslant 0$ and $T \geqslant 0$, is it possible to compute all W load units within T seconds after the master starts sending out the first load unit?

Computing a DL schedule entails three steps: (i) select which workers should participate in the computation; (ii) decide in which order workers should receive load chunks and how many times; and (iii) compute how much work each load chunk should comprise. Most proposed solutions to the DLS problem fall into two categories: *one-round* schedules and *multi-round* schedules. In one-round schedules, each worker receives only one load chunk. In multi-round schedules, each worker may receive multiple load chunks throughout application execution. As illustrated in Fig. 2.6, multi-round schedules are preferable to one-round schedules because they allow for better overlap of computation and communication. Albeit, they are also generally less regular and more difficult to compute.

In the following, we will consider some restrictions of the DLS problem. These restrictions will be denoted as DLS{*restriction*} where restriction may be for example: *1Round* (all processors are used at most one time), $b_i = 0$ (bandwidths from the master to the slaves are infinite), $s_i = 0$ (no startup), and so on.

Regarding unconstrained multi-round schedules, it is important to understand that DLS$\{s_i = 0\}$ is a problem that does not make any sense. Indeed, for any finite schedule, it is possible to find a better schedule: when everything is perfectly linear, idle time can be strictly decreased by adding a new round. Therefore, scheduling problem involving several rounds generally require

startup constraints.

2.2.2 Linear Programming and Optimality Principle

Some workers may not be used in the schedule and hence not appear in the activation sequence. In the following, we will denote as act_{max}, the largest number of activations allowed in an activation sequence. In the one-round case, a worker can only appear once in the activation sequence. The multi-round notion commonly used in the literature assumes that the activation sequence is periodic but this is an artificial restriction.

Consider a given instance $I = (s, b, w)$ of the problem. Let $\sigma : \{1, \ldots, n\} \to \{1, \ldots, p\}$ denote a given activation sequence of size n. Then if we denote by α_j the amount of workload sent to $P_{\sigma(j)}$, DLS{*FixedActivation*} is equivalent to determining whether the following linear constraints define a non-empty set:

$$\begin{cases} \text{(2.6a)} & \displaystyle\sum_{j=1}^{n} \alpha_j = W \\ \text{(2.6b)} & \displaystyle\forall k \leqslant n : \sum_{j=1}^{k} \left(s_{\sigma(j)} + \alpha_j b_{\sigma(j)}\right) + \sum_{j \geqslant k \,:\, \sigma(j) = \sigma(k)} \alpha_j w_{\sigma(k)} \leqslant T \\ \text{(2.6c)} & \forall j \leqslant n : \alpha_j \geqslant 0 \end{cases} \tag{2.6}$$

The leftmost part of Constraint (2.6b) represents the time at which the kth communication ends and the middle one is a lower bound on the computation time of worker $\sigma(k)$ after this communication. The sum of these two times has thus to be smaller than the makespan T. Considering in backward order the activations where a given worker l is used, it is not hard to see from the constraints that one will obtain a feasible schedule [Dro97].

Theorem 2.3. *DLS{FixedActivation} is polynomial and can be solved with linear programming*

Linear programming can be used to prove what is known in the literature as the optimality principle.

Theorem 2.4 (Optimality Principle)**.** *For an optimal activation sequence and the corresponding optimal load distribution, all messages, except maybe the trailing ones, convey some load and there is no idle time.*

It is important to understand that this optimality principle only holds for the optimal activation sequence. This result allows to derive elegant closed-form formula in many simple cases (e.g., for DLS$\{b_i = b, s_i = 0\}$) [BRG96, Rob03].

2.2.3 Complexity results

Since DLS{*FixedActivation*} is polynomial, the difficult part in DLS is to select and order workers. Interestingly, the introduction of startup times creates a sharp complexity gap in the *1Round* problems:

- DLS{*1Round*, $s_i = 0$} is a polynomial problem. Workers should be ordered by decreasing b_i, just like the bandwidth-centric principle of Section 2.1.2.
- DLS{*1Round*, $b_i = 0$} is weakly NP-hard and there is a pseudo-polynomial algorithm to solve it [YCD+07]. To the best of our knowledge, it is unknown whether DLS{*1Round*} is strong NP-hard or not. In essence, the weak NP-completeness results is based on the selection issue. It would thus be interesting to know whether the problem is hard once the selection is done, i.e., if the ordering problem is hard. It could provide a direction toward strong NP-completeness.

As we have already mentioned, multi-round optimization requires to introduce startup. Unfortunately, there are very simple instances for which the optimal number of round grows with $\sqrt{W}$. The definition of DLS{*Multi-Round*}, even with startup is problematic. There are several options to circumvent this issue:

- The first one is to set a bound on the number of activations, in which case the problem can be solved by a mixed integer linear program. It is also possible to resort to branch and bound techniques or genetic algorithms to build the activation sequence [DL05].
- One may also restrict to particular class of schedules like periodic ones. Using the same kind of technique as in steady-state scheduling, it is possible to derive asymptotically optimal periodic schedules such that $T_{per} \leqslant T^* + O(\sqrt{T^*})$. Note that such schedules have $\Theta(\sqrt{W})$ round and that it strengthens the intuition that the optimal number of rounds is always $\Theta(\sqrt{W})$.
- A major drawback of periodic schedules is their rigidity: the exact same amount of workload is sent to each worker during every period. Intuitively, rounds should be smaller in the beginning to allow a better overlap of communications and computations and a better start-up time. The UMR and RUMR heuristics [YdRC05, Yan05] build upon such heuristic and propose schedule where the workload sent at each round is always shared with the same proportions and where chunk size grows geometrically.

2.2.4 Issues

- The efficiency or multi-round schedules over 1-round schedules makes the introduction of startup times a necessary evil. Yet, the selection difficulty they induce seems somehow artificial to me as the difficulty occur only when the workload is relatively small compared to startup delays. Efficient algorithms exist when there is lot of workload to process. And when there is too little work to process, it is probably not worth using a parallel platform anyway... It may be the case that network latency is very large but it is generally for setting up a connection. If the connections can remain open or be open in advance, then latency is not really an issue anymore.
- Lots of variants on divisible load can be found in the literature (with return messages, with buffers, with several workload to distribute, with non linear cost). Such variants generally make the problem more complicated and somehow explore how far the divisible model can be pushed. Although such research path is interesting, I prefer to go the other way around, i.e., to start from practical situations and see how such computationally solvable framework can be used.

2.3 Game Theory

Game theory is a branch of mathematics that has been developed to study conflict and cooperation between intelligent rational decision-makers. Algorithmic game theory has recently emerged at the intersection of game theory and algorithm design. Unlike classical game theory, there is a stronger emphasis on constructiveness, approximation ratio and polynomial running times.

2.3.1 Non-cooperative games

It is generally assumed that a player has a real-valued utility function whose value is to be maximized and depends on a set of possible strategy. The previous scheduling problems can thus be seen as one player situations, the utility being for example the inverse of the time spent processing load (i.e., their throughput), and the strategy being how the load is distributed to the

workers. Such a modeling is sound if the distributed system is dedicated to a single user. Such an assumption is rarely reasonable since large scale distributed systems generally result from the collaboration of several entities (universities, computing centers, volunteers, ...). When the system is shared by several users, the utility of a player does not solely depend on his own strategy but also on the ones of all the other players. Indeed, if two users decide to use the same set of machines, it will create a contention that will generally be detrimental to both of them despite potential transparent sharing mechanisms that may have been set up. This makes scheduling problem much more complicated but also much more realistic and interesting.

Definition 2.7 (Game)**.** Let us consider a set $\mathcal{P} = \{1, \ldots, m\}$ of *players*, each player k having a set of *strategies* S_k at his disposal. Each player k can select freely every strategy $s_k \in S_k$ he wants. A *game* is a function $U : S_1 \times S_2 \times \ldots S_m \to \mathbb{R}^m$ that associate to a *strategy profile* $(s_1, \ldots, s_m)$ a reward or utility to every player. Each player k is thus trying to optimize his own *utility* $U_k(s_1, \ldots, s_m)$ but only controls s_k and has no influence on $s_1, \ldots, s_{k-1}, s_{k+1}, \ldots, s_m$.

Nash introduced in 1950 [Nas50b] a notion of equilibrium that plays a central role in game theory. A Nash equilibrium is a strategy profile such that no player has interest in changing unilaterally his own strategy. Each strategy in a Nash equilibrium is a *best response* to the strategies of the other players in that equilibrium.

Definition 2.8. Nash equilibrium A strategy profile $(s_1, \ldots, s_m)$ is a Nash equilibrium iff for all k and for all $s'_k \in S_k$: $U_k(s_1, \ldots, s_{k-1}, s'_k, s_{k+1}, \ldots, s_m) \leqslant U_k(s_1, \ldots, s_{k-1}, s_k, s_{k+1}, \ldots, s_m)$.

Obviously such model is simple and does not account for temporal aspects, asymmetry of information, intimidation that some players may exert on others, random variability, or the fact that some players may be smarter than others. Several extensions have been proposed since to alleviate these issues but all assume the rationality of players. It is yet a good basis for the study of non-cooperative situations. Although it may be debatable to apply such modeling to human-being, game theory provides an interesting framework to study interactions between computer systems or to design their interactions.

Nash equilibria raise several difficulties though. In general, there is not necessarily a Nash equilibrium (unless we consider mixed games where players have to choose a probability distribution over their strategy space) and even if such an equilibrium exists, there is no known polynomial algorithm to find it (the problem is PPAD-complete [DGP06]). One may consider the dynamic where every player is asked one after the other to update his strategy to maximize his utility. Such a *best response* dynamic has no particular reason to converge but if so, it would be toward a Nash equilibrium. This is one of the reason that motivates the interest for such equilibria. If every player selfishly optimize his own utility, the only possible outcome, if any, is a Nash equilibrium.

Nash equilibria are somehow the result of non-cooperative optimization and are *stable* (hence the name equilibrium) in the sense that no player has interest in unilaterally deviating from its strategy. Yet, Nash equilibria are not resilient to coalitions. A subset of players could have interest in simultaneously changing their strategy to all obtain a better utility. Such coalition notions are more advanced game theory notions [Mye97] and will not be discussed in this document.

As one would expect, such Nash equilibria, when they exist, have no particular reason for being "efficient" or "fair". But let us define this more precisely.

Definition 2.9 (Pareto optimality)**.** Let us denote by $\mathcal{U} = U(S_1 \times \ldots \times S_m) \subset \mathbb{R}^m$ the utility set. An outcome $u \in \mathcal{U}$ is said to be Pareto optimal if

$$\forall v \in \mathcal{U} : (\exists i, v_i > u_i) \Rightarrow (\exists j, v_j < u_j)\,.$$

By extension, a strategy profile $(s_1, \ldots, s_m)$ is said to be *Pareto-optimal* if $U(s_1, \ldots, s_m)$ is Pareto-optimal.

Pareto optimal strategies are thus strategies where it is impossible to increase someone's utility without decreasing the utility of someone else (see Fig. 2.7).

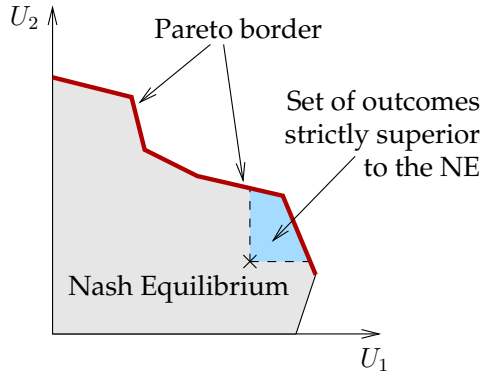


Figure 2.7: Utility set for two users. Any point in the gray area correspond to a possible outcome. Different strategies may lead to the same outcome. The Pareto border (i.e., the set of points that are not dominated by any other strategy) is depicted with a bold line. As illustrated, the Nash equilibrium may be dominated by many other solutions.

Nash equilibria are generally not Pareto-optimal: it could be the case that **every one** obtains a better utility by using a different strategy profile. Such strategy profile would however probably not be an equilibrium: a player could increase his own utility by deviating, which would impact the utility of the others that would then change their strategy, ... The stability of Nash equilibrium (they are resilient to the deviation of individual users) is what may make them desirable. It is thus important to evaluate how inefficient they may be.

The Price of Anarchy has been introduced by Koutsoupias and Papadimitriou [KP99] to measure how the efficiency of a system degrades due to selfish behavior of its agents. This function relies on the definition of an index, also called welfare function, $W : \mathbb{R}^m \to \mathbb{R}$ like the sum of players utilities (also called social welfare, and which is the most commonly used welfare function), the minimum of players utilities (that would rather represent an egalitarian objective) or any increasing function that may make sense for the game under study.

Let us consider s^{NE} a Nash equilibrium. The inefficiency of this Nash equilibrium can be defined as:

$$I_W(s^{NE}) = \frac{\max_{s \in S} W(U(s))}{W(U(s^{NE}))} \tag{2.7}$$

The price of anarchy of a game is the largest inefficiency of a Nash equilibrium, hence

$$POA_W = \max_U \frac{\max_{s \in S} W(U(s))}{\min_{s^{NE} \in NE} W(U(s^{NE}))} \tag{2.8}$$

Let us illustrate a last notion related to the inefficiency of Nash equilibria. When studying properties of Nash equilibria in routing systems, Braess exhibited in 1968 an example in which, by adding resource to the system (in his example, a route), the performance of *all* the users were degraded [Bra68]. This is somehow surprising as one may expect that at least a few benefit from additional resources.

Definition 2.10 (Braess paradox)**.** Let us denote (S, U) the original game and (S', U') a new game with additional resources (i.e., we have $S \subset S'$ and $U(s) = U'(s)$ for each $s \in S$.)
Obviously $\mathcal{U} \subset \mathcal{U}'$, the new utility set is larger and provides potentially better outcomes. Unfortunately, since "the" Nash equilibrium s'_{NE} of $\mathcal{U}'$ is not Pareto optimal, it can be anywhere in $\mathcal{U}'$, even below "the" Nash equilibrium s_{NE} of $\mathcal{U}$. There is a Braess paradox if $U_k(s_{NE}) > U_k(s'_{NE})$ for each player k.

The condition of existence of Braess paradox are not well understood yet.

Nash equilibria are thus potentially interesting because of their stability and because an uncoordinated selfish best-response dynamic may converge only toward them. However, they may

be inefficient and this inefficiency needs to be assessed. In situation where this inefficiency is not harmful, one may want to let the system evolve with no particular intervention. Should it be otherwise (e.g., large price of anarchy or potential Braess paradox), one may want to design particular control mechanisms that promotes cooperation or enforce a rational usage of resource (e.g., through pricing).

2.3.2 Index-based Fairness

In network engineering, fairness index have been commonly used to quantify how fair is the resource sharing between users or applications. The optimization of these index leads to different fairness notions.

Definition 2.11 (Social welfare)**.** An allocation $u \in \mathcal{U}$ is said to be *socially optimal* if it is maximal in $\mathcal{U}$ for the index $\Sigma : \mathbb{R}^m \to \mathbb{R}$ defined by $\Sigma(u) = \sum_k u_k$.

A common drawback of such allocations is that they often comprise players with null utility. To circumvent this issue, one may consider other kind of indexes.

Definition 2.12 (Max-min fairness)**.** One may consider the index $\min : \mathbb{R}^m \to \mathbb{R}$. However there may be several $u \in \mathcal{U}$ that maximize this index. Furthermore, since $\min$ is not strictly increasing, points that achieve the maximum in $\mathcal{U}$ may not be Pareto optimal. Hence, this index should be recursively optimized.
An allocation $u \in \mathcal{U}$ is said to be *max-min fair* if it is maximum for the lexicographic order in $\mathcal{U}$. In other words, if the smallest u_k is maximized and the second smallest u'_k is maximized and etc.

Max-min fair allocations somehow ensure that everything is done to provide as much as possible to the "poorest" user, even if it has to be done at the detriment of the whole system. Other indexes allow to find a trade-off between such extreme choices (i.e., between social welfare optimization and max-min fairness).

Definition 2.13 (Proportional fairness)**.** An allocation $u \in \mathcal{U}$ is said to be *proportionally fair* if it is maximal in $\mathcal{U}$ for the index $\Pi : \mathbb{R}^m \to \mathbb{R}$ defined by $\Pi(u) = \prod_k u_k$.
Alternative and equivalent definitions involve maximizing $\sum_k \log(u_k)$ or being such that for any other utility $v \in \mathcal{U}$, $\sum_k \frac{v_k - u_k}{u_k} < 0$.

An interesting property of proportional fairness is that it is scale-free. If the utility of a player is uniformly scaled up or down by a constant factor, it will not change the resulting solution. In other words, proportional fairness is not sensitive to the units in which utilities are expressed (if these units are linear ...). Proportional fairness also obviously ensures that no player can get a null utility. Another way to avoid such a situation is to consider the inverse of utilities.

Definition 2.14 (Potential delay minimization)**.** An allocation $u \in \mathcal{U}$ is said minimize potential delay if it is minimal in $\mathcal{U}$ for the index $\Sigma^{-1} : \mathbb{R}^m \to \mathbb{R}$ such that $\Sigma^{-1}(u) = \sum_k 1/u_k$. The term of delay minimization may seem awkward but comes from the context of elastic bandwidth sharing where utility is the bandwidth alloted to each flow. If all flows need to transfer the same amount of data M, then the time perceived by user k for transmitting his file will be M/u_k and the total time will thus be $M \sum_k 1/u_k$.

These three fairness criteria respectively correspond to the arithmetic, geometric and harmonic mean of u. This provides a perspective for generalizing fairness.

Definition 2.15 (α-fairness)**.** For a given $\alpha > 0$, an allocation $u \in \mathcal{U}$ is said to be **α-fair** if it is maximum in $\mathcal{U}$ for the index $f_\alpha : \mathbb{R}^m \to \mathbb{R}$ such that

$$f_\alpha(u) = \begin{cases} \frac{1}{1-\alpha} \sum_k u_k^{1-\alpha} & \text{, if } \alpha \neq 1 \\ \sum_k \log(u_k) & \text{, otherwise} \end{cases}$$

The index f_α is continuous in α and has the interesting property to correspond to Social welfare when $\alpha = 0$, to proportional fairness when $\alpha = 1$, to potential delay minimization when $\alpha = 2$, and to max-min fairness when $\alpha = \infty$. It has been thought for some time that such metric enabled to adjust the trade-off between particularly fair solutions ($\alpha = \infty$) and efficient solutions ($\alpha = 1$). This is far from obvious as explained for example in [TWL06] or [LKCS10].

2.3.3 Axiomatic Fairness

Another (maybe more natural) way to conceive fairness is to define it through axioms, i.e., through properties that one would expect from a fair solution. This is the approach that Nash proposed in [Nas50a].

Definition 2.16 (Nash Bargaining Solution)**.** A **bargaining problem** can be defined as a pair $(\mathcal{U}, d)$ where $\mathcal{U}$ is the utility set of our game and $d \in \mathcal{U}$ and is a disagreement which every player will end up with if they cannot agree. We assume that $\mathcal{U}$ is convex and compact and that there exist a $v \in \mathcal{U}$ such that $\forall k, v_k > d_k$.
A **bargaining solution** is a mechanism f that selects a vector $u \in \mathcal{U}$ for a bargaining problem $(\mathcal{U}, d)$. One would expect from a bargaining solution that it respects the following properties:

- **Pareto efficiency**: A Pareto inefficient solution is unlikely as it would leave room for renegotiation.
- **Symmetry**: Let us consider two players i and j. If $\mathcal{U}$ is symmetrical with respect to i and j and they both have the same disagreement values ($d_i = d_j$), then they should obtain the same utility ($f(\mathcal{U}, d)_i = f(\mathcal{U}, d)_j$)).
- **Independence of irrelevant alternatives**: Let $(\mathcal{U}, d)$ and $(\mathcal{U}', d)$ be two bargaining problems such that $\mathcal{U}' \subset \mathcal{U}$. If $f(\mathcal{U}, d) \in \mathcal{U}'$, then $f(\mathcal{U}', d) = f(\mathcal{U}, d)$.
- **Scale invariance**: Let us consider two vectors $\alpha > 0$ and β.

$$f(\alpha \cdot \mathcal{U} + \beta, \alpha \cdot d + \beta) = \alpha f(\mathcal{U}, d) + \beta$$

Interestingly, when $\mathcal{U}$ is convex and compact, the proportionally fair allocation (with respect to the disagreement point) is the only allocation verifying the Nash axioms [Nas50a].

Most previous axioms are very natural. The interpretation of scale invariance is less obvious and may or not make sense depending on the context. Alternative axioms have been proposed by Kalai and Smorodinsky [KS75] and can be related to max-min fairness.

Chapter 3

Non-Cooperative Throughput Optimization

This chapter builds on two articles written with Corinne Touati at INFOCOM'07 [38] and GAMECOM'07 [37]. We consider a very simple topology and illustrate that even under idealistic conditions, when letting everyone optimize its own throughput, bad things can happen. The resulting insights allow us to question classical definitions like the price of Anarchy and to propose a possibly more meaningful definition.

3.1 Platform and Application Setting

We consider a master-worker platform with p workers (see Fig. 3.1). Each worker P_i is characterized by its processing capability W_i (in Mflop/s) and the capacity B_i (in MB/s) of its connection. We define the *communication-to-computation* ratio C_i of worker P_i as B_i/W_i. We assume that the platform performs an ideal fair sharing of resources among the various requests. More precisely, let us denote by $N_i^{(B)}(t)$ (resp. $N_i^{(W)}(t)$) the number of ongoing communication (resp. computation) from M to P_i (resp. on P_i) at time t. The platform ensures that the amount of bandwidth received at time t by a communication from M to P_i is exactly $B_i/N_i^{(B)}(t)$. Likewise, the amount of processor power received at time t by a computation on P_i is exactly $W_i/N_i^{(W)}(t)$. Therefore, the time T needed to transfer a file of size b from M to P_i starting at time t_0 is such that

$$\int_{t=t_0}^{t_0+T} \frac{B_i}{N_i^{(B)}(t)} \cdot dt = b.$$

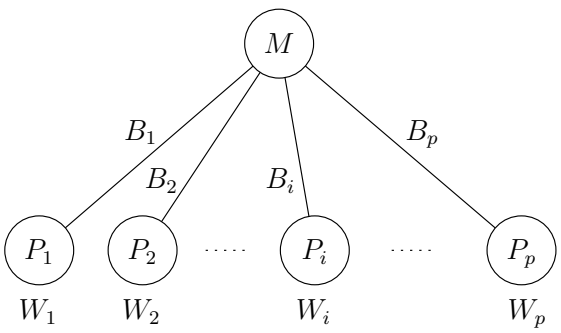


Figure 3.1: Master worker architecture. Worker P_i can deliver W_i floating point operations per second and is connected to the master by a B_i bandwidth connection.

Likewise, the time T needed to perform a computation of size w on P_i starting at time t_0 is such that

$$\int_{t=t_0}^{t_0+T} \frac{W_i}{N_i^{(W)}(t)} \cdot dt = w.$$

Last, we assume that the master is operated under the multi-port model, i.e., that communications to different processors do not interfere.

We consider K BoT applications, A_k, $1 \leqslant k \leqslant K$. Each application is composed of a large set of independent, same-size tasks. Let w_k be the amount of computation (in Mflop) required to process a task of type k. Similarly, b_k is the size (in MB) of (the file associated to) a task of application k. Last, we define the *communication-to-computation ratio* c_k of tasks of type k as b_k/w_k.

Let us define $\alpha_{i,k}$ the average number of tasks of type k performed per time-unit on worker P_i. The throughput for application k of such a schedule is defined as $\alpha_k = \sum_{i=1}^{P} \alpha_{i,k}$. Since the platform is operated under the full-overlap multi-port model, the $\alpha_{i,k}$ verify the following constraints:

Computation $\forall P_i : \sum_{k=1}^{K} \alpha_{i,k} \cdot w_k \leqslant W_i$ (3.1a)

Communication $\forall P_i : \sum_{k=1}^{K} \alpha_{i,k} \cdot b_k \leqslant B_i$ (3.1b)

Remark. Consider a system S with K applications running over P machines. The set of achievable utilities $U(S,P)$, that is to say the set of possible throughput α_k is given by

$$U(S,P) = \left\{ (\alpha_k)_{1 \leqslant k \leqslant K} \;\middle|\; \begin{array}{l} \exists \alpha_{1,1}, \ldots, \alpha_{P,K}, \\ \quad \forall k \in \{1,\ldots,K\} : \sum_{n=1}^{P} \alpha_{i,k} = \alpha_k \\ \quad \forall n \in \{1,\ldots,P\} : \sum_{k=1}^{K} \alpha_{i,k} \cdot w_k \leqslant W_i \\ \quad \forall n \in \{1,\ldots,P\} : \sum_{k=1}^{K} \alpha_{i,k} \cdot b_k \leqslant B_i \\ \quad \forall n \in \{1,\ldots,P\}, \forall k \in \{1,\ldots,K\} : \alpha_{i,k} \geqslant 0 \end{array} \right\}.$$

The utility set is hence convex and compact.

3.2 Non-Cooperative Scheduling

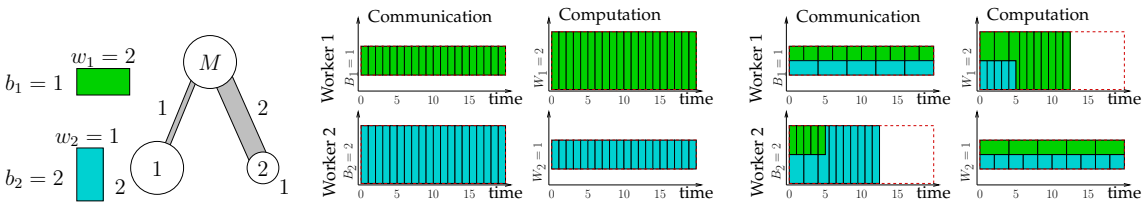
We first study the situation where only one application is scheduled on the platform. This will enable us to simply define the scheduling strategy that will be used by each player (scheduler) in the more general case where many applications are considered. When there is only one application, our problem reduces to the following linear program:

$$\begin{array}{l} \text{MAXIMIZE } \sum_{i=1}^{P} \alpha_{i,1}, \\ \text{UNDER THE CONSTRAINTS} \\ \left\{ \begin{array}{ll} \text{(3.2a)} & \forall P_i : \alpha_{i,1} \cdot w_1 \leqslant W_i \\ \text{(3.2b)} & \forall P_i : \alpha_{i,1} \cdot b_1 \leqslant B_i \\ \text{(3.2c)} & \forall P_i, \quad \alpha_{i,1} \geqslant 0. \end{array} \right. \end{array} \tag{3.2}$$

We can easily show that the optimal solution to this linear program is obtained by setting $\forall P_i$, $\alpha_{i,1} = \min\left(\frac{W_i}{w_1}, \frac{B_i}{b_1}\right)$. In a practical setting, this amounts to say that the master process will saturate each worker by sending it as many tasks as possible. On a stable platform, W_i and B_i can easily be measured and the $\alpha_{i,1}$'s can thus easily be computed. On an unstable one, this may be more tricky. However a simple acknowledgment mechanism enables the master process to ensure that it is not over-flooding the workers, while always converging to the optimal throughput.

In a multiple-applications context, each player (process) strives to optimize its own performance measure (considered here to be its throughput α_k) regardless of the strategies of the other

players. Hence, in this scenario, each process constantly floods the workers while ensuring that all the tasks it sends are performed (e.g., using an acknowledgment mechanism). This adaptive strategy automatically cope with other schedulers usage of resource and *selfishly* maximize the throughput of each application. As the players constantly adapt to each others' actions, they may (or not) reach some Nash equilibrium. One can actually prove [38] that such equilibrium exists and is unique. We denote by $\alpha_{i,k}^{(\mathrm{NE})}$ the rates achieved at such a stable state.



(a) System characteristics

(b) With cooperation. Application 1 uses solely worker 1 while application 2 uses solely worker 2.

(c) Non-cooperative equilibrium. Now both application 1 and 2 use worker 1 (and 2).

Figure 3.2: Non-cooperation can lead to inefficiencies. Fig. 3.2(b) and Fig. 3.2(c) depict steady-state patterns of length 20. The height represents the capacities of the resources.

Consider a system (see Fig. 3.2(a)) with two computers 1 and 2, with parameters $B_1 = 1$, $W_1 = 2$, $B_2 = 2$, $W_2 = 1$ and two applications of parameters $b_1 = 1$, $w_1 = 2$, $b_2 = 2$ and $w_2 = 1$. Fig. 3.2(b)) depicts a steady-state pattern for the situation where the two applications would cooperate and use only their prefered machine (i.e., application 1 uses exclusively worker 1 while application 2 uses exclusively worker 2). On Worker 1 (top left), a green rectangle of height 1 and width 1 represents a communication of the first application. Since b_1, $B_1 = 1$ and only application 1 uses the communication link of worker 1, such communications last for one time unit. Likewise, since $w_1 = 2$ and $W_1 = 2$, computations of application 1 last exactly one time unit. Overall there is a good resource usage and, their respective throughput would be

$$\alpha_1^{(\mathrm{coop})} = \alpha_2^{(\mathrm{coop})} = 1.$$

Yet, with a non-cooperative approach (see Fig. 3.2(c)), both applications will use both workers as much as possible. Communications of application 1 on worker 1 (green rectangles on the top left part) last twice longer (i.e., two time units). Likewise, some of the computations of application 1 are slowed down by those of application 2, which prevents the CPU of worker 1 to be kept fully busy. The same problem occurs with the communication link of Worker 2. One can check that applications only get a throughput of:

$$\alpha_1^{(\mathrm{NE})} = \alpha_2^{(\mathrm{NE})} = \frac{3}{4}$$

In this example, one can easily check that, at the Nash equilibrium, for any worker, there is no resource waste: slave 1 (resp. slave 2) is communication (resp. computation) saturated i.e., Eq. (3.1b) (resp. Eq. (3.1a)) is an equality. However, communication-saturation implies computation idle times and vice-versa. Yet, when communication and computation idle times coexist in a system, the non-cooperative behavior of the applications can lead to important inefficiencies.

The scheduling algorithm used by the players consists in constantly flooding workers. A player k is thus said to be either *communication-saturated* on worker n or *computation-saturated* on worker n. Using different equivalent representations of the schedule and saturation properties, one can prove that there always exists exactly one non-cooperative equilibrium and that it has a closed form expression.

Theorem 3.1 ([38]). *We assume $c_1 \leqslant c_2 \leqslant \cdots \leqslant c_K$. Let us denote by $\mathcal{W}_i$ the set of players that are computation-saturated and by $\mathcal{B}_i$ the set of players that are communication-saturated on a given arbitrary worker i. Depending on the c_k's and the C_i's, we have either:*

1. *If* $\sum_k \frac{C_i}{c_k} \leqslant K$ *then* $\mathcal{W}_i = \emptyset$ *and* $\forall k, \alpha_{i,k}^{(NE)} = \dfrac{B_i}{K.b_k}$.

2. *Else, if* $\sum_k \frac{c_k}{C_i} \leqslant K$ *then* $\mathcal{B}_i = \emptyset$ *and* $\forall k, \alpha_{i,k}^{(NE)} = \dfrac{W_i}{K.w_k}$.

3. *Else,* $\mathcal{B}_i$ *and* $\mathcal{W}_i$ *are non-empty and there exists an integer* $m \in \{1, \ldots, K-1\}$ *such that*

$$\frac{c_m}{C_i} < \frac{m - \sum_{k=1}^{m} \frac{c_k}{C_i}}{K - m - \sum_{k=m+1}^{K} \frac{C_i}{c_k}} < \frac{c_{m+1}}{C_i}.$$

Then, we have $\mathcal{W}_i = \{1, \ldots, m\}$ *and* $\mathcal{B}_i = \{m+1, \ldots, K\}$ *and*

$$\begin{cases} \alpha_{i,k}^{(NE)} = \frac{B_i}{b_k} \frac{|\mathcal{W}_i| - \sum_{p \in \mathcal{W}_i} \frac{c_p}{C_i}}{|\mathcal{W}_i||\mathcal{B}_i| - \sum_{p \in \mathcal{W}_i} c_p \sum_{p \in \mathcal{B}_i} \frac{1}{c_p}} & \text{if } k \in \mathcal{B}_i \\ \alpha_{i,k}^{(NE)} = \frac{W_i}{w_k} \frac{|\mathcal{B}_i| - \sum_{p \in \mathcal{B}_i} \frac{C_i}{c_p}}{|\mathcal{W}_i||\mathcal{B}_i| - \sum_{p \in \mathcal{W}_i} c_p \sum_{p \in \mathcal{B}_i} \frac{1}{c_p}} & \text{if } k \in \mathcal{W}_i \end{cases} \tag{3.3}$$

3.3 Nash Equilibrium and Pareto Inefficiency

As we have seen in the example of Fig. 3.2, the Nash equilibrium may be Pareto inefficient. However, on a single-processor system, the allocation at the Nash equilibrium is always Pareto optimal. On a multi-processor system, Pareto inefficiencies occur as soon as there are both communication-saturated and computation-saturated nodes:

Theorem 3.2 ([38]). *Assume that applications are not all identical, that is to say that there exists* k_1 *and* k_2 *such that* $c_{k_1} < c_{k_2}$. *Then, the allocation at the Nash equilibrium is Pareto inefficient if and only if there exists two workers* n_1 *and* n_2 *such that* $\mathcal{W}_{n_1} = \emptyset$ *and* $\mathcal{B}_{n_2} = \emptyset$.

Since the Nash equilibrium may be Pareto, a natural question is "how much inefficient is it?". Let us thus look at the price of Anarchy of this problem.

Let us consider the following simple system $S_{M,K}$ comprising one machine ($B_1 = 1$ and $W_1 = 1$) and K applications ($b = (\frac{1}{M}, 1, \ldots, 1)$, and $w = (\frac{1}{M}, 1, \ldots, 1)$). It is then easy to compute the following allocations (see Fig. 3.3):

- $\alpha^{(\mathrm{NE})}(S_{M,K}) = \left(\frac{M}{K}, \frac{1}{K}, \ldots, \frac{1}{K}\right)$ corresponds to the non-cooperative allocation;
- $\alpha^{(\Sigma)}(S_{M,K}) = (M, 0, \ldots, 0)$ corresponds to the allocation optimizing the average throughput;
- $\alpha^{(\min)}(S_{M,K}) = \left(\frac{1}{K-1+1/M}, \ldots, \frac{1}{K-1+1/M}\right)$ corresponds to the max-min fair allocation;
- $\alpha^{(\Pi)}(S_{M,K}) = \left(\frac{M}{K}, \frac{1}{K}, \ldots, \frac{1}{K}\right)$ corresponds to the proportionally fair allocation (i.e., the Nash Bargaining Solution when the disagreement point is $(0, \ldots, 0)$). Surprisingly, on this instance, this allocation also corresponds to the non-cooperative one.

Note that, all these allocations are Pareto optimal either by definition or because this is a single-processor system. One can easily compute (using Eq. (2.7) the inefficiencies of the Nash equilibrium:

$$I_\Sigma(S_{M,K}) = \frac{M}{\frac{M}{K} + \frac{K-1}{K}} \xrightarrow[M \to \infty]{} K$$
$$I_{\min}(S_{M,K}) = \frac{K}{MK - M + 1} \xrightarrow[M \to 0]{} K.$$

Theorem 3.3. *The prices of anarchy* POA_Σ *and* $POA_{\min}$ *are unbounded even when restricting to single-processor systems where the Nash equilibrium is always Pareto optimal.*

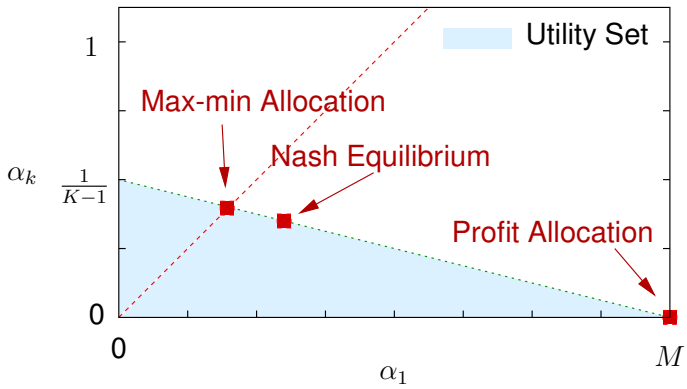


Figure 3.3: Utility set and allocations for $S_{M,K}$ ($K = 3$,$M = 2$).

The fact that the non-cooperative equilibria of such instances are Pareto-optimal and have interesting properties of fairness (it corresponds to the Nash Bargaining Solution in this particular example) questions the relevance of the *price of anarchy* notion as a Pareto efficiency measure. To quantify the degradation of Braess-like Paradoxes (the degree of Paradox), Kameda [Kam06] introduced the Pareto-comparison of two allocations. Working on such notions [37], we realized such a measure can be properly defined only when referring to some kind of distance to the whole Pareto set and no index-based inefficiency measure can reflect such distance. As algorithmicians are used to manipulate approximation factors when evaluating the performance of an algorithm, this distance to the Pareto set should be measured in the log space. Let us denote by $\mathcal{P}(\mathcal{U})$ the Pareto set of $\mathcal{U}$. It may be the case that $\mathcal{P}$ is not closed (even when $\mathcal{U}$ is compact and convex). Hence, we refer to $\overline{\mathcal{P}}(\mathcal{U})$, the closure of $\mathcal{P}(\mathcal{U})$. Let α The distance from β to the closure of the Pareto set $\overline{\mathcal{P}}(\mathcal{U})$ in the log-space is equal to:

$$d_\infty(\log(\beta(\mathcal{U}), \log(\overline{\mathcal{P}}(\mathcal{U}))) = \min_{u \in \overline{\mathcal{P}}(u)} \max_k |\log(\beta(U)_k) - \log(u_k)|$$

Therefore, we proposed in [37] to define the Pareto inefficiency of β as

$$\begin{aligned} \widetilde{I}_\infty(\beta, U) &= \exp(d_\infty(\log(\beta(U), \log(\overline{\mathcal{P}}(U))) \\ &= \min_{u \in \overline{\mathcal{P}}(u)} \max_k \max \left(\frac{\beta(U)_k}{u_k}, \frac{u_k}{\beta(U)_k} \right) \end{aligned} \tag{3.4}$$

This notion is somehow related to the notion of ε-approximation introduced by Papadimitriou and Yannakakis [PY00] in the context of multi-objective approximation. The definition of $\widetilde{I}_\infty$ (Eq. (3.4)) may seem much more complicated than the one of I_W (Eq. (2.7)) used in the price of anarchy, especially as it relies on the Pareto set $\mathcal{P}(U)$ that may be complex and computationally intractable. This seems however required to grasp the notion of Pareto inefficiency. Index-based inefficiency measures can only reflect a particular property of the allocation such as fairness. Note that in mono-criteria situations, it is natural to compare a solution to a computationally intractable optimal solution, generally using approximations or lower bounds. Therefore, it is not surprising that similar approaches should be used in multi-criteria settings to compute $\widetilde{I}_\infty$. This inefficiency measure is thus a natural extension of the classical mono-criteria performance ratio.

Theorem 3.4. *When increasing the unbalance of example of Fig. 3.2 (i.e., when setting $B_1 = 1$, $W_1 = M$, $B_2 = M$, $W_2 = 1$ and two applications of parameters $b_1 = 1$, $w_1 = M$, $b_2 = M$ and $w_2 = 1$), the Pareto inefficiency is equal to $\frac{1}{\frac{M^2+M}{2M^2}} \xrightarrow[M \to +\infty]{} 2$.*

3.4 Resource Augmentation

In this section, we make a few observations on the influence of resource augmentation on the overall performance of the system. We say that $S = ((P, W, B), (K, w, b))$ is a subsystem of $S' = ((P, W', B'), (K, w, b))$ iff for all P_i, $W_i \leqslant W'_i$ and $B_i \leqslant B'_i$.

Let us consider a system (called "initial") and a second one (referred to as the "augmented" system), derived from the first one by adding some quantity of resource. Intuitively, the Nash equilibrium aug in the augmented system should be Pareto-superior to the one in the initial system ini. However, since the Nash equilibrium aug may be Pareto inefficient, it may be the strictly Pareto-inferior to ini, in which case we have a Braess paradox happens. Interestingly, such situation cannot happen in our context. Thanks to the characterization of Theorem 3.1, we have been able to prove:

Theorem 3.5 ([38]). *Even though the Nash equilibria may be Pareto inefficient, in the non-cooperative multi-port scheduling problem considered scenario, Braess paradoxes cannot occur.*

Although there is no Braess paradox, the evolution of resource share with resource augmentation is not very regular. Unexpected behavior of typical performance metrics can occur even for Pareto optimal situations (e.g., on single-processor systems). Let us illustrate this by considering the simple numerical example of Fig. 3.4. Observe that when the bandwidth B is $245/24 \simeq 10.208$ the three metrics (namely the higher throughput, the lower throughput and the average throughput) have lower values than when the bandwidth B is only equal to $560/73 \simeq 7.671$.

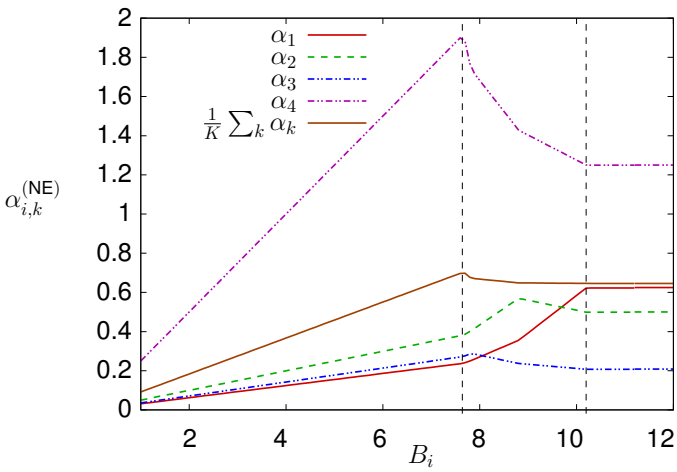


Figure 3.4: The three performance metrics (min, max, average) can simultaneously decrease with resource augmentation: $b = (8, 5, 7, 1)$, $w = (4, 5, 12, 2)$, $K = 4$, $W_i = 10$.

The previous example illustrates the non-monotonicity with resource augmentation of the maximal throughput, of the minimal throughput and of the average throughput. This result can even be strengthened [38].

Theorem 3.6. *For both the maximal throughput, the minimal throughput and the average throughput, there are single-processor systems where degradation when increasing resource can be made arbitrarily large.*

Therefore, although there is no Braess Paradox, it is difficult to tell who will benefit from resource augmentation in the Nash equilibrium.

3.5 Conclusion and Open Issues

The previous study is instructive as it shows that bad situations can happen even with an idealistic model. The bi-dimensional nature of the problem (i.e., applications have both a possibly

unrelated computation and a communication requirement) is sufficient to be the source of inefficiencies. The multi-port non-cooperative steady-state scheduling problem is thus rather minimalist and enables to reconsider classical measures of inefficiency with a different perspective.

Several questions remain open though:

- The ability to analytically compute the Nash equilibrium heavily relies on the multi-port assumption. In this model, all applications constantly flood the workers and the sharing on a worker has no influence on what happens on another worker. One may wonder what happens for 1-port model. In this model, the master should send tasks in priority to workers with the highest bandwidth. Since instantaneous bandwidth depends on the choice of the others, this means that a worker selection that is optimal at time t may become sub-optimal at time $t' > t$. The inability to interrupt communications and reconsider ongoing communications makes the optimal strategy definition much more complex. As will be illustrated in the next chapter, numerical simulations indicate that the quality of the equilibrium can be quite bad under such assumption.

- We have exhibited simple situations where the Pareto inefficiency of the Nash equilibria of the multi-port non-cooperative steady-state scheduling problem could be arbitrarily close to 2. We have never been able to find examples with a larger inefficiency. We suspect that the Pareto inefficiency of this problem is always smaller than 2, which would be an interesting property and would show that the Selfishness Degradation Factor is a useful definition just as natural as the classical approximation ratio notion.

- Even if the Nash equilibrium had bounded Pareto inefficiency, it would be interesting to know if there exists an index for which it has a bounded price of anarchy, i.e., if it somehow optimizes something. We know for sure that it is neither the average throughput nor the minimum throughput. Some simple examples suggest a connection with proportional fairness but this remains to be investigated. Anyway, the previous results advocate for the need of some form of coordination so as to obtain a fair and efficient throughput optimization in a realistic and complex setting.

Chapter 4

Max-min Fair Throughput Optimization

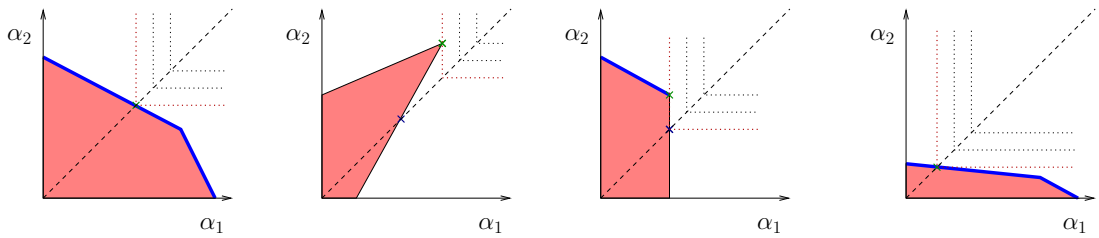
This chapter builds on two articles at IPDPS'06 [39] and in TPDS'08 [8] written with Olivier Beaumont, Larry Carter, Jeanne Ferrante, Loris Marchal, and Yves Robert. This work was initiated during my stay at UC San Diego in 2004 and completed during the ALPAGE ANR project. This was our first attempt to incorporate a notion of users in steady-state scheduling and to investigate whether the linear programming techniques we had developed during my PhD thesis were still effective or not.

4.1 Platform and Application Setting

In this chapter, we consider a platform model similar to the one presented in Section 2.1.3 except that we restrict to rooted tree and single-level tree (or star) topologies. We concentrate on the *full-overlap single-port* model where each processor can simultaneously receive data from one of its neighbors, perform some (independent) computation, and send data to one of its neighbors. At any given time, there are at most two communications involving a given processor, one sent and the other received.

The application setting is similar to the one of the previous chapter: we consider K applications, A_k, $1 \leqslant k \leqslant K$ that originate from the root of the tree P_M, which initially holds all the input data necessary for each application A_k. Each application is composed of a set of independent, equal-sized tasks that can be characterized by w_k, the amount of computation (in Mflop) required to process a task of type k, and by b_k, the amount of data (in MB) to transfer to process a task of application k. Last and as usual, we define the *communication-to-computation ratio* c_k of tasks of type k as b_k/w_k.

Under such model, the set of constraints on $\alpha_{i,k}$ is a set of linear inequalities, and we know that the utility set is thus a convex polyhedron, as illustrated in Fig. 4.1. Fig. 4.1(a) corresponds



(a) Conflict on a worker (b) Synergy (c) Independency (d) Typical utility set for tree

Figure 4.1: A few examples of utility sets. Dotted lines are isolines of $(\alpha_1, \alpha_2) \to \min(\alpha_1, \alpha_2)$ and bold lines represent Pareto optimal points.

to the typical situation where two applications are competing on a single node:

$$\begin{cases} \alpha_1 \cdot b_1 + \alpha_2 \cdot b_2 \leqslant B_1 & \alpha_1 \geqslant 0 \\ \alpha_1 \cdot w_1 + \alpha_2 \cdot w_2 \leqslant W_u & \alpha_2 \geqslant 0 \end{cases}$$

As a consequence (see Fig. 4.1(a)), many points are Pareto-optimal. Any Pareto optimal point is thus *a priori* as worth of interest as any other Pareto optimal point. Defining fairness can be seen as defining a criterion for choosing among Pareto optimal points. Since our previous results heavily relied on linear programming, which we knew pretty well, we proposed to focus on obtaining a max-min fair solution as it is more reasonably fair than for example trying to maximizing the sum of utilities for which some applications may receive nothing at all.

On Fig. 4.1(a), the minimum of the α_k is maximized at a point where all the α_k have the same value. One can easily check that the only Pareto optimal point of Fig. 4.1(b) is also the point such that the minimum of the α_k is maximized. However, one can also check that the points such that the α_k's all have the same value are not efficient, which corresponds to the well-known fact that giving the same thing to each user is not always a good option. In fact the shape of this utility set is rather uncommon and corresponds to a situation where there is a synergy between both users. Such situations may occur with caching mechanisms for example but cannot occur in our framework because all coefficients of the linear constraints are positive.

One may then wonder whether in our context max-min optimal solutions are always such that the throughput of all applications are the same or not. This is true as soon as applications originate from the same location. The utility set of Fig. 4.1(c) is typical of the situation where two applications originate from different locations and where one of them can only use a limited area of the network (e.g., due to a very high communication to computation ratio and a small connectivity to the network). In such case, it may be possible to increase the throughput of the application with the lower ratio (α_2 here) without decreasing the throughput of the other one (α_1 here). However, if both applications start using the same resources, the throughput of one application can only increase at the expense of the throughput of another application. It is important to note that many different points maximize the minimum of the throughputs (all points belonging to U and to the lowest isoline of $\min(\alpha_1, \alpha_2)$). However, only one of them is of interest (i.e., Pareto Optimal). It is well-known that max-min fairness should be *recursively* defined in this case: the first minimum should be maximized, then the second should be maximized, and so on.

Such situations cannot occur on tree-shaped platforms as applications originate from the same location and thus always compete on the same set of resources. That is why in the rest of this chapter, we can restrict to solutions where all application throughputs are equal. But in a more general situation, we should look for Pareto-optimal allocations and the previous stopping condition could not be used anymore.

4.2 Computing the Max-min Fair Solution

Using the same approach as in Eq. (2.5), we can show that an optimal schedule can be derived from the following linear program (where $f(i)$ denotes the father of i in the tree topology):

$$\begin{array}{l} \text{MAXIMIZE } \min_k \sum_i \alpha_{i,k}, \\ \text{UNDER THE CONSTRAINTS} \\ \begin{cases} \text{(4.1a)} \quad \forall P_i, \alpha_{i,k} \geqslant 0 \text{ and } \forall (P_i \to P_j), sent_{i\to j,k} \geqslant 0 & \text{(non-negativity)} \\ \text{(4.1b)} \quad \forall P_i, \displaystyle\sum_k \alpha_{i,k} \cdot w_k \leqslant W_i & \text{(computation limit)} \\ \text{(4.1c)} \quad \forall P_i, \displaystyle\sum_k \sum_{(P_i \to P_j)} \frac{sent_{i\to j,k} \cdot b_k}{B_{i,j}} \leqslant 1 & \text{(single-port)} \\ \text{(4.1d)} \quad \forall P_i \neq M : sent_{f(i)\to i,k} = \alpha_{i,k} + \displaystyle\sum_{(P_i \to P_j)} sent_{i\to j,k} & \text{(data conservation)} \end{cases} \end{array} \tag{4.1}$$

On tree-shaped platforms, the previous solution is the max-min fair solution. On general platform, constraint (4.1d) should simply be rewritten to account for the fact that data may be received from different sources. Yet solving this new linear program would not give the max-min fair solution as the first minimum should be maximized, then the second should be maximized, and so on. This can easily be done in our setting by identifying which applications correspond to the first minimum by looking at saturated constraints (tight inequalities at optimum point). One can then rerun the linear program, with the throughput of these applications fixed, to maximize the smallest throughput of the remaining applications. This process can be repeated until all applications are saturated. Max-min solutions can thus easily be computed in polynomial time, even on complex platforms.

Although the process is slightly simpler for tree platforms than for general graphs, it requires to gather the characteristics of the whole platform before solving the linear program (4.1) to obtain the number of tasks of each type each node should assign to each of its children per time unit. As we explained in Section 2, in the single-application/tree-topology setting, a subtree can be summarized by a single value corresponding to its aggregated computing capacity. Unfortunately, in the multiple-application context, a subtree is characterized by its whole utility set, i.e., by a polyhedron. Therefore, aggregating several subtrees requires to merge the polyhedra and results into a polyhedron whose complexity may grow during the recursion along the tree. We could thus not find any particular gain in such approach when compared to a centralized solution where the characteristics of all resources would be gathered before solving the large linear program 4.1. We also could not find particular properties of the max-min fair solution that could be exploited to limit the growth of the complexity of the polyhedra.

Yet, when the computer platform is a star network, we can prove that the optimal solution has a very particular structure: Fig. 4.2).

Property (Sliced structure)**.** If we order the processors according to their bandwidths (i.e., $B_0 \leqslant B_1 \leqslant \dots B_p$) and the applications by communication-to-computation ratio (i.e., $\frac{b_1}{w_1} \geqslant \frac{b_2}{w_2} \geqslant \dots \frac{b_K}{w_K}$, then there exist indices $a_0 \leqslant a_1 \dots \leqslant a_K$ such that only processors P_i, $i \in [a_{k-1}, a_k]$, execute tasks of type k in the optimal solution.

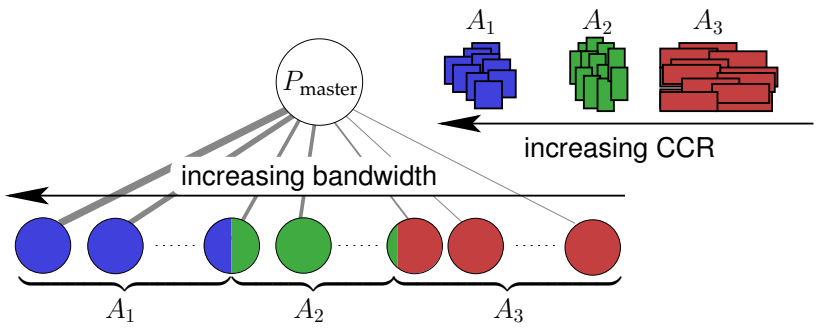


Figure 4.2: Shape of the optimal one-port solution.

In other words, each application is executed by a set of consecutive nodes, which we refer to as a *slice*. The application with the highest communication-to-computation ratio is executed by a first slice of processors, those with largest bandwidths. Then the next most communication-intensive application is executed by the next slice of processors, and so on. There is a possible overlap at the slice boundaries. For instance P_{a_1}, the processor at the boundary of the first two slices, may execute tasks for both applications A_1 and A_2.

Unfortunately, this characterization does not enable to determine the boundaries of the slices nor the $\alpha_{i,k}$ through analytical formulas. Furthermore, we did not succeed in deriving a counterpart of such property for tree-shaped platforms. Intuitively, the problem is that a high-bandwidth child of node P_i can itself have a low-bandwidth, high-compute-rate child, so there is no *a priori* reason to give P_i only communication-intensive tasks.

```
1: loop
2:    If there will be room in your buffer, request work from your parent.
3:    Get incoming requests from your local worker and children, if any.
4:    Move incoming tasks from your parent, if any, into your buffer.
5:    Select which thread (your local worker or a child's scheduler) to assign work to, and the
      type of application that will be assigned.
6:    if you have a request and a task that match your choice then
7:       Send the task to the chosen thread (when the send port is free)
8:    else
9:       Wait for a request or a task
```

Figure 4.3: Demand-driven scheduler thread, run in each node.

4.3 Demand-driven and Distributed Heuristics

Although it is possible from the set of all platform and application parameters to compute an optimal periodic schedule, the previous approach suffers from several serious drawback, which we present in this section.

4.3.1 Demand-driven Scheduling

First, the period of the schedule is the least common multiple of the denominators of the solution of the linear program (4.1). This period may be huge, requiring the nodes to have unreasonably large buffers to ensure uninterrupted steady-state behavior. The problem of buffer size has already been pointed out in [KCCF03, 41], where it is shown that no finite amount of buffer space is sufficient for every tree. It is also known that finding the optimal throughput when buffer sizes are bounded is a strongly NP-hard problem even in very simple situations [41]. Since unlimited buffer space is unrealistic, we consider *demand-driven* algorithms, which try to achieve pre-determined consumption and emission rates and operate as follows. Each node has a local *worker* thread and a *scheduler* thread that both have their own task buffer of limited size. The worker thread is an infinite loop that requests a task from the same node's scheduler thread and then, upon receiving a task, executes it. Fig. 4.3 shows the pseudo-code for the scheduler thread. The "select" choices in line 5 depend on the particular heuristic used and are typically based on the history of requests, task types it has received, and on the communication times it has observed for its children if one wishes to implement a *bandwidth-centric* strategy. When target rates have been predetermined (e.g., by solving linear program (4.1)), each scheduler can try to achieve such rates by selecting the "right" task. To this end, we used a simple 1D load-balancing mechanism [BBP+01] to select a requesting thread and an application type. The 1D load-balancing mechanism works as follows: if task k should be chosen with frequency $f(k)$, and has already been chosen $g(k)$ times, then the next task to be sent will be of type ℓ, where $\frac{g(\ell)+1}{f(\ell)} = \min_k \frac{g(k)+1}{f(k)}$. In such cases, one knows the expected *theoretical* fair throughput, which allows us to explore how implementation issues result in the effective throughput differs from the theoretical throughput. Note that we evaluated effective throughputs by looking at finite schedules (with 200 tasks per application), by considering only the part where all applications are competing with each others, and by getting rid of the initial and final instabilities. Furthermore, since throughputs are expected to be all equal, we compare the smallest effective throughputs of the different applications to the expected theoretical one.

There are many reasons that these quantities might differ, such as the overhead of the request mechanism or a startup period longer than the 10% we neglect. It turned out that the major cause of inefficiency was the limit on the buffer size. Our experiments assumed enough buffer space to hold 10 tasks of any type. For this case, Fig. 4.4 depicts the effective fair throughput deviation from the expected theoretical throughput. The average deviation is equal to 9.4%. However, when we increased the buffer size by a factor ten (and increased the number of tasks

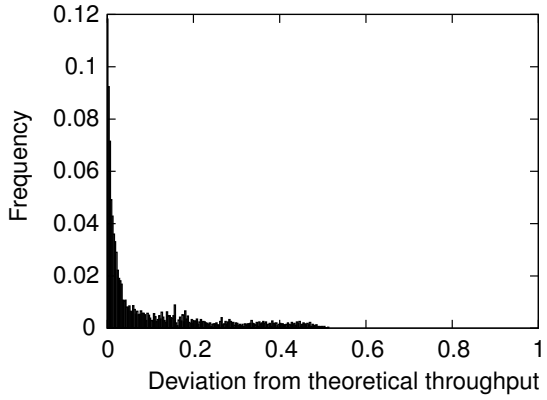


Figure 4.4: Deviation of effective fair throughput from expected theoretical throughput ($1 - \frac{\text{Effective fair throughput}}{\text{Theoretical fair throughput}}$).

per application to 2000), the mean average deviation dropped to 0.3%. Even though the larger buffer size led to much better throughput, we considered it unrealistic, and used size 10 in all other experiments.

4.3.2 Distributed Scheduling

A second problem is that centralized coordination becomes an issue as the size of the platform grows. It may be costly to collect up-to-date information and disseminate it to all nodes in the system. Also when platform characteristics evolves smoothly, it would be interesting to have an algorithm that smoothly adapt to such variations without necessarily recomputing the whole solution. Finally, the centralized solving of the linear program (4.1) induces a single point of failure, which should be avoided if possible. Consequently, a decentralized scheduling algorithm, where all choices are based exclusively on *locally* available information, is desirable.

Building on the previous optimality principles (the *bandwidth-centric* strategy or the *sliced structure* for star networks), we devised several heuristics and evaluated them through simulations:

- Coarse-Grain Bandwidth-Centric (*CGBC*): trying to build on the *bandwidth-centric* strategy, this naive *coarse-grain* heuristic assembles several tasks from each application into a single large one. More precisely, we build a macro-task out of one task of type k, for each k, and the macro-tasks are scheduled using the bandwidth-centric method. Thus, fairness is assured but efficiency is likely to be poor since non communication-intensive tasks will be sent to high-bandwidth nodes.

- Parallel bandwidth-centric (*PBC*): this is the non-cooperative optimization corresponding to a Nash equilibrium. Each application has its own scheduler on each node, running in parallel with the others, and using the bandwidth-centric strategy as if it was the only application running on the platform. In our simulations, the one-port constraint is enforced for each scheduler thread, which gives an unfair advantage to *PBC* over the other heuristics since a node may send as many as K tasks concurrently, one of each type. As we will see, such advantage is anyway largely counter-balanced by the inefficiency of the non-cooperative optimization.

- Data-Centric Scheduling (*DATA-CENTRIC*): we also designed a rather complex heuristic that starts from the optimal distribution for the most communication-intensive application and progressively trades some of these tasks for more computation-intensive ones until

all applications have the same throughput (we refer to [8] for more details). On single-level trees, the slice structure of optimal solutions allows this heuristic to find the optimal solution but its performance on general trees was rather unclear.

We compared these three heuristics to the following two baseline strategies:

- First Come First Served (*FCFS*): the *FCFS* heuristic is a very simple and common decentralized heuristic. Each scheduler thread simply fulfills its requests on a *First Come First Served* basis, using the tasks it receives in order from its parent. The root ensures fairness by selecting task types using a round-robin selection. Although fair between the applications, this simplistic heuristic does not perform any resource selection and may use extremely slow communication links, hence a poor resource usage.

- Centralized LP-based (*LP*): in this heuristic, platform resources are constantly benchmarked so as to solve linear program (4.1) with up-to-date values. The solution of this linear program is then instantaneously broadcast and used by the demand-driven schedulers as target rates. We might hope the *LP* heuristic would always converge to the optimal throughput, but this is not always the case, primarily because of insufficient buffer space.

All strategies were implemented and evaluated with SimGrid (see [8] for detailed information on the simulation setup). For each experimental setting, we compute the (natural) logarithm of the ratio of the effective fair throughput of *LP* with the effective fair throughput of a given heuristic (applying a logarithm enables us to have a symmetrical value). This value is called in the following *logarithmic deviation*. Therefore, a positive value means that *LP* performed better than the other heuristic. Fig. 4.5 depicts the histogram plots of these values.

First of all, we observe that most values are positive, which demonstrates the superiority of *LP*.

As expected *FCFS* is rather inefficient since the geometrical average of the ratio of the performance between *FCFS* and *LP* is 1.564. In the worst case, its performance is more than 8 times worse than *LP*.

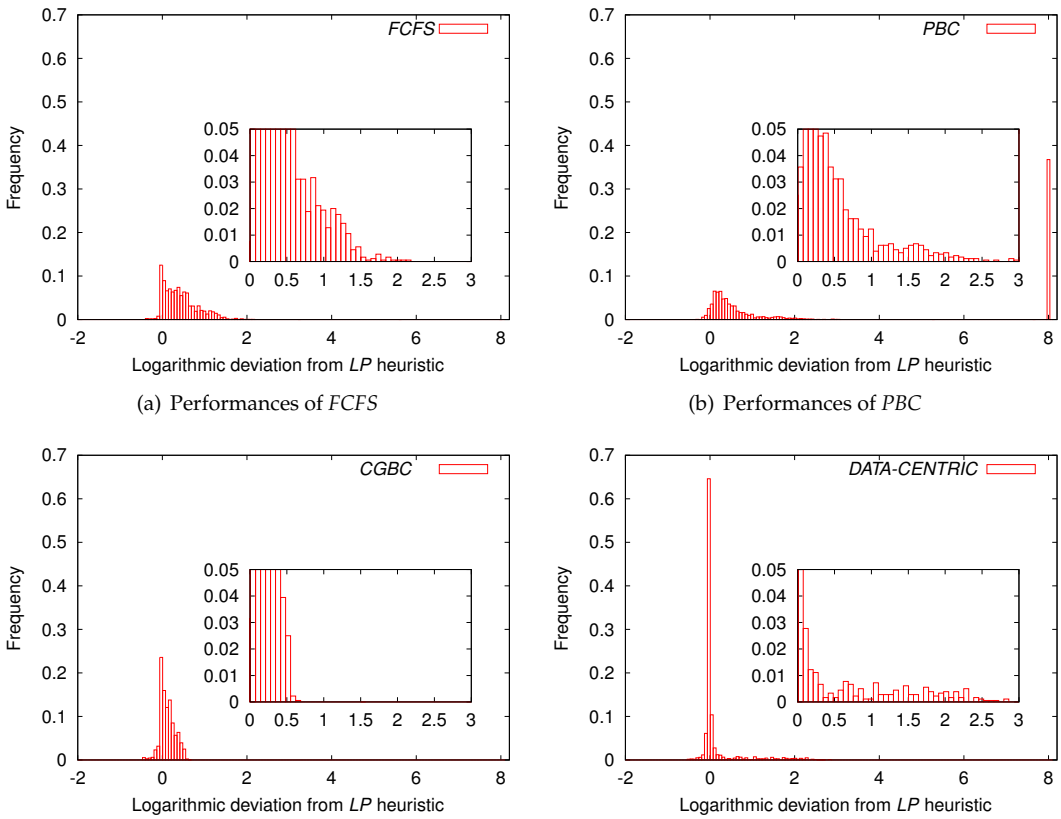
Likewise, *PBC* leads to very bad results. In many situations (more than 35%), an application has been particularly unfavored and the fair effective throughput was close to 0. The logarithm of the deviation for these situations has been normalized to 8. These poor results advocate the need for fairness guarantees in distributed computing environments like the ones we consider.

In contrast, *CGBC* (see Fig. 4.5(c)) is much more stable since its worst performance is only twice that of *LP* and its geometric average is 1.156.

Finally, one can observe on Fig. 4.5(d) that *DATA-CENTRIC* is very close to *LP* most of the time, despite the distributed computation of the expected $\alpha_{i,k}$ and $sent_{i \to j,k}$ values. However, the geometric average of these ratios is equal to 1.164, which is slightly larger than the geometric average for *CGBC* (1.156). The reason is that even though in most settings *DATA-CENTRIC* ends up with a very good solution, in a few instances it performed very badly (up to 16 times worse than *LP*!). This can be seen in the zoom window of Fig. 4.5(d). Our investigation did not allow us to really understand why this heuristic fails on some instances. We suspect that it is due to the use of the (sometimes misleading) intuition of the sliced structure of single-level trees. Indeed, in this heuristic, applications with a high communication-to-computation ratio are performed mainly on the subtrees with the best bandwidths at the root while applications with a low communication-to-computation are performed primarily on the subtrees with the worst bandwidths at the root, which is definitely not optimal on particular instances.

4.4 Conclusion and Open Issues

This was our first attempt to incorporate a notion of users in steady-state scheduling and to reuse the linear programming techniques we had developed during my PhD thesis. This attempt was not very successful but quite instructive.



(a) Performances of *FCFS*

(b) Performances of *PBC*

(c) Performances of *CGBC*

(d) Performances of *DATA-CENTRIC*

Figure 4.5: Logarithmic deviation from *LP* performances ($\ln\left(\frac{\min_k \alpha_k \text{for } LP}{\min_k \alpha_k \text{for other heuristic}}\right)$).

Indeed, none of the heuristics we developed was really satisfying. All our attempts to solve this linear program in a decentralized way failed, even on simple examples like rooted trees. We could not obtain any good algorithmic intuition like the recursion principle we had in the single application context. It seemed quite difficult to use single-user or single-level tree results and intuition to obtain a satisfying solution in a multi-user context, hence a radically different approach was required.

This study also allowed us to realize that the non-cooperative approach was clearly unacceptable in our context. In our simulations, non-cooperative scheduling in the one-port model lead to very unfair situations compared to the expected max-min optimal solution: some applications often obtained a close to 0 throughput. This motivated the work presented in the next chapter.

A positive result though is that if feasible rates are provided, the demand-driven autonomous schedulers presented in Section 4.3 achieve reasonable approximation of these rates even in a bounded buffer setting. The remaining question that puzzled me was thus **How to calculate these rates in a distributed way**?

This study lead us to question two hypothesis we had taken:

Max-min fairness solving max-min fairness seemed more difficult to us than we initially solved despite the linearity of such objective. Furthermore playing with parameters of the simulations allowed us to realize that such an objective was probably meaningless in our context. Indeed, since applications are likely to have very different characteristics, the shape of the utility set is generally very distorted (see Fig. 4.1(d)). Since the max-min fair solution tends to equate throughputs, serving the more demanding application requires to impose hard limits on resource usage of the other applications. For example, in Fig. 4.1(d), slightly de-

creasing the throughput α_2 would free a lot of resources, hence allowing an important improvement of α_1 . This allowed us to realize that the fairness notion to use was not really clear and that the choice of a criteria had to be done very carefully. In particular, in our context, it would be easy for an application to change the granularity of its task, for example, by grouping them by two. This would result in a seemingly twice lower throughput and a max-min fair solution would then increase further the resource share of such an application. Interestingly, proportional fairness is scale free and would not suffer from such attempts from the applications to acquire more resources than they should. Although we had initially avoided to use such criteria because of its non-linearity, we realized it was probably the right criteria for this context.

1-port model As we briefly mentioned in Chapter 2, the 1-port model requires to resort to matching decompositions to organize communications, which seems inherently centralized on general platform graphs. Such constraint makes everything more complicated and is somehow artificial as threads can easily be used to overcome such limitation. That is why we think the **bounded multi-port** [HP07] where each node is associated a bound on the amount of data it can receive and sent at any instant is more adequate for this context.

Therefore, in the next chapter, we present our attempt to design a proportionally fair distributed optimization of throughput under the multi-port model.

Chapter 5

Proportionally Fair Distributed Throughput Optimization

This chapter builds on two articles at Grid'08 [34] and in JPDC'13 [4] written with Rémi Bertin, Sascha Hunold and Corinne Touati. It was conducted in the context of the DOCCA and USS-SimGrid projects. From our previous attempts in trying to obtain a max-min fair sharing of resources, we try to design a distributed and proportionally fair sharing of resources by relying completely different techniques. We inspire from work conducted in the network community, and more particularly on the flow control problem in multi-path networks.

5.1 Platform and Application Setting

In this chapter, we consider the same platform model as the one presented in Section 2.1.3. The target computing and communication resources are represented by a general *platform graph*, i.e., a node-weighted edge-weighted graph $G = (N, E, W, B)$, as illustrated in Fig. 5.1. Each node $P_n \in N$ represents a computing resource that can deliver W_n floating-point operations per second. Each edge $e_{i,j} : (P_i \to P_j) \in E$ is labeled by a value $B_{i,j}$ which represents the bandwidth between P_i and P_j. The operation mode of the processors is the *full overlap, multi-port model* for both incoming and outgoing communications. In this model, a processor node can simultaneously receive data from all its neighbors, perform some (independent) computation, and send data to all its neighbors at arbitrary rate while respecting the resource constraints (i.e., the bandwidth and processing speed bounds). Note that this framework also comprises the bounded multi-port extension [HP07] where each node has an additional specific bandwidth bound. This extension would simply amount to change slightly the graph. However, no specific assumption is made on the interconnection graph, which may well include cycles and multiple paths.

The application setting is similar to the one of the previous chapter: we consider K applications, A_k, $1 \leqslant k \leqslant K$. Each application originates from a master node $P_{m(k)}$ that initially holds all the input data necessary for each application A_k (see Fig. 5.1). Each application is composed of a very large set of independent, equal-sized tasks characterized by a computational cost w_k and a communication cost b_k.

We further assume that each application A_k is deployed on the platform as a tree. This assumption is reasonable as this kind of hierarchical deployment is used by many grid services [CD06]. Therefore, if an application k uses node P_n, all its data will use a single path from $P_{m(k)}$ to P_n denoted by $(P_{m(k)} \rightsquigarrow P_n)$. If there is no such path or if application k cannot access node n (e.g., for administrative reasons), then $(P_{m(k)} \rightsquigarrow P_n)$ is empty. We do not assume that there is only a single way from a location to another (which generally does not hold true in a grid environment). We rather assume that if several ways exist, only one is actually used. Given the proximity between our problem and the multi-path flow control problem, it would not be difficult to consider several possible routes whenever needed as long as this number of alter-

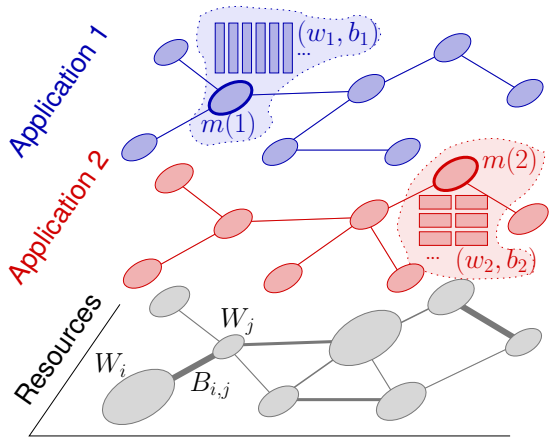


Figure 5.1: A resource graph labeled with node (computation) and edge (communication) weights. Two application deployments with different sources (respectively $m(1)$ and $m(2)$) and task characteristics.

natives remains limited. However, it would uselessly complexify the equations and algorithms presented thereafter.

As we explained earlier, we choose to focus on proportional fairness, which is a *scale-free* measure and ensures that no starvation can occur. Yet, the algorithms we present can be straightforwardly adapted to α-fairness, which accounts for other types of fairness. Therefore, we aim at finding $(\alpha_{n,k})_{1\leqslant k\leqslant K, 1\leqslant n\leqslant N}$ that solve

$$\begin{array}{l}
\text{MAXIMIZE } \sum_k \log(\sum_n \varrho_{n,k}), \\
\text{UNDER THE CONSTRAINTS} \\
\left\{\begin{array}{lll}
\text{(5.1a)} & \forall n, \sum_k \varrho_{n,k} w_k \leqslant W_n & \text{(computation capacity)} \\
\text{(5.1b)} & \forall (P_i \to P_j), \sum_k \sum_{\substack{n \text{ such that} \\ (P_i \to P_j) \in (P_{m(k)} \rightsquigarrow P_n)}} \varrho_{n,k} b_k \leqslant B_{i,j} & \text{(communication capacity)} \\
\text{(5.1c)} & \forall n, \forall k,\ \varrho_{n,k} \geqslant 0 & \text{(non-negativity)}
\end{array}\right.
\end{array} \tag{5.1}$$

Solving such optimization problem can be expressed as a semi-definite program and hence solved with a classical SDP solver such as DSDP [BY05]. This provides a quick and reliable, yet centralized, way to validate of our algorithms.

5.2 Introduction to Flow Control in Multi-path Networks

Lagrangian optimization and dual decomposition is an old idea in optimization, and traces back at least to the early 1960s. Decentralized optimization has been an active topic of research since the 1980s and in the last decade, the network community has heavily used such techniques to both analyze and design network protocols like TCP (see for example [Low03a, KMT98]). This technique has also recently been applied to devise supply and demand algorithms for smart grids [DYS12].

To introduce such technique, we first present how it can be used for point-to-point network protocol design and analysis. Then, we explain how the technique can be extended to study flow control in multi-path networks. Indeed, such problem is very similar to our optimization problem (5.1). Unfortunately, as we will explain, a naive adaptation to our context is completely ineffective.

5.2.1 Fairness and Network Protocol Design

Assume we are given a network made of a set of links$\mathcal{L}$ whose capacity B_l for $l \in \mathcal{L}$ is to be shared among a set of flows$\mathcal{F}$. Let us denote by ϱ_f the bandwidth allotted to flow $f \in \mathcal{F}$. Let us assume that the network operator has decided to share bandwidth according to some fairness criteria expressed through utility functions U_f (e.g., $U_f = \log$ or one of the α-fairness index). The bandwidth sharing can be written as follows:

$$\begin{array}{l} \text{MAXIMIZE } \sum_{f\in\mathcal{F}} U_f(\varrho_f), \\ \text{UNDER THE CONSTRAINTS} \\ \begin{cases} \text{(5.2a)} \quad \forall l \in \mathcal{L}, \displaystyle\sum_{f \text{ going through } l} \varrho_f \leqslant B_l \\ \text{(5.2b)} \quad \forall f \in \mathcal{F},\ \varrho_f \geqslant 0 \end{cases} \end{array} \tag{5.2}$$

Checking that all constraints are satisfied requires some form of global coordination, which is very hard to implement. Fortunately, Lagrangian optimization enables us to put the previous problem in a form more amenable to distribution. This is achieved by introducing a dual variable for each constraint and hence for each resource, which we will denote by λ_l. The variables ϱ_f of the initial problem are called primal variables. The Lagrangian function is then defined as:

$$L(\varrho, \lambda) = \underbrace{\sum_{f\in\mathcal{F}} U_f(\varrho_f)}_{\text{objective function}} + \sum_{l\in\mathcal{L}} \underbrace{\lambda_l \cdot \left(B_l - \sum_{f \text{ through } l} \varrho_f \right)}_{\text{constraints}} \tag{5.3}$$

The original problem (5.2) can be safely rewritten (primal problem):

$$\max_{\varrho\geqslant 0} \min_{\lambda\geqslant 0} L(\varrho, \lambda).$$

Indeed, if an allocation ϱ is unfeasible, then one of the constraints is violated and the inner minimization problem (the minimization over λ) is thus solved by setting the corresponding λ_l to $+\infty$. Conversely, if ϱ is a feasible allocation, then the inner minimization problem is solved by setting the corresponding λ_l to either 0 when the constraints are not tight or to any positive value when the constraint is tight. The objective value is then equal to the original objective function and this formulation of the primal problem is thus strictly equivalent to our original problem. Under very mild assumptions it can be proven that there is no duality gap [BT89], i.e., that

$$\underbrace{\max_{\varrho\geqslant 0} \min_{\lambda\geqslant 0} L(\varrho, \lambda)}_{\text{Primal problem}} = \underbrace{\min_{\lambda\geqslant 0} \max_{\varrho\geqslant 0} L(\varrho, \lambda)}_{\text{Dual problem}} \stackrel{\text{def}}{=} \min_{\lambda\geqslant 0} d(\lambda)$$

In most cases U_f is chosen to be continuous, increasing and strictly concave and the dual function d is thus a convex function. Solving the original problem is then equivalent to find the saddle point of L. The main advantage of such reformulation is that now constraints are very simple ($\varrho \geqslant 0$ and $\lambda \geqslant 0$) and do not require any global coordination. Since both concave maximization and convex minimization problems can be solved through gradient descent (see for example [BT89, Chapter3], on the convergence analysis of descent algorithms), the saddle point is generally obtained by applying a gradient descent simultaneously for both inner and outer optimization problems. A simple constant step-size (γ_ϱ) ascent on the primal variables simultaneous to a constant step-size (γ_λ) descent on the dual variables leads the following update equations

$$\begin{cases} \varrho_f(t+1) = \varrho_f(t) + \gamma_\varrho \cdot \dfrac{\partial L}{\partial \varrho_f}(\varrho(t), \lambda(t)) \\ \lambda_l(t+1) = \lambda_l(t) - \gamma_\lambda \cdot \dfrac{\partial L}{\partial \lambda_l}(\varrho(t), \lambda(t)) \end{cases} \tag{5.4}$$

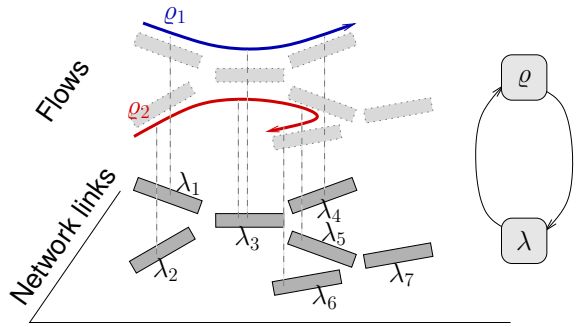


Figure 5.2: Distributed sharing algorithm based on Lagrangian optimization and gradient descent. Flows adapt their rate ϱ based on prices λ advertised by the network links they use and conversely.

Expanding the partial differentiates, the previous update equations are rewritten:

$$\begin{cases} \varrho_f(t) = \varrho_f(t) + \gamma_\varrho \cdot \Big(U'_f(\varrho_f(t)) - \sum\limits_{l \text{ used by } f} \lambda_l(t) \Big) \\ \lambda_l(t) = \lambda_l(t) - \gamma_\lambda \cdot \Big(B_l - \sum\limits_{f \text{ through } l} \varrho_f \Big) \end{cases} \tag{5.5}$$

λ_l is generally called shadow price for link l and the previous equations lead to a surprisingly simple algorithm that can be interpreted as follows (see Fig. 5.2):

- Every flow f evaluates the "total price" of the resources it uses (i.e., the sum of the $\lambda_l(t)$) and adapts its emission rate to account for both its utility and the virtual price it should pay. Whenever the price gets "too expensive" compared to the utility increase $U'_f(\varrho_f)$, the flow decrease its emission rate and conversely.

- Every resource l evaluates whether it is saturated or not and adapts its price accordingly. Whenever a resource is saturated, it will increase its price so that the flows going through it decrease their usage and whenever a resource is underused, it will lower its price so that the flows going through it can increase their rate.

This "supply and demand" inspired algorithm is a simultaneous gradient descent on both primal and dual variables that will converge to the saddle point, which is the optimal solution of the original problem. By adapting the step-size, or the utility functions, one gets a different protocol. Such techniques have for example been used either to design protocols achieving a given fairness criteria or to reverse-engineer existing protocols. For example, by making an analogy between the window adjusting protocols and the primal update equations, Low *et al.* proved [Low03a] that, under some stability assumptions, TCP Vegas achieves some form of proportional fairness, while first versions of TCP Reno behaved as if $\arctan$ based utility functions were optimized.

To summarize, the general approach of distributed Lagrangian optimization is based on the following three steps:

1. **Modeling.** Model the problem as a concave non-linear maximization problem;

2. **Partial derivatives.** Convert this problem into two coupled optimization problems using Lagrangian multipliers and differentiate the Lagrangian function L with respect to each primal and dual variables;

3. **Algorithm design.** From the structure of these partial derivatives, devise a distributed algorithm implementing coupled gradient descent (on dual variables) and ascent (on primal variables). This algorithm can be interpreted as a bargaining of applications over resources.

The key ingredients to turn the partial derivatives into a distributed algorithms (i.e., move from step 2 to step 3) are (1) the separability of objective functions (it is a sum over the flows of quantities that depend only on each flow) and (2) the structure of the constraints (each constraint corresponds to a resource).

5.2.2 Flow Control in Multi-path Networks

A similar approach relying on these three steps has been used in the context of network flows that may choose among a predetermined set of routes [WPL03]. In such a context, each flow f is subdivided into sub-flows $f_1,\ldots,f_k$ and the optimal flow control is written as:

$$\begin{array}{l} \text{Maximize} \sum_{f\in\mathcal{F}} U_f(\sum_k \varrho_{f,k}) \\ \text{s.t.} \begin{cases} \forall l \in \mathcal{L}, \displaystyle\sum_{f_k \text{ going through } l} \varrho_{f,k} \leqslant B_l \\ \forall f_k \in \mathcal{F},\ \varrho_{f,k} \geqslant 0 \end{cases} \end{array} \tag{5.6}$$

Wang *et al.* [WPL03] specifically addressed this problem with the additional constraint that each flow has minimum and maximum requirements: $\forall f, m_f \leqslant \varrho_f \stackrel{\text{def}}{=} \sum_k \varrho_{f,k} \leqslant M_f$. As this kind of constraints is not relevant in our context, for the sake of clarity, we only present in the following, simplified versions of the equations and algorithms proposed in [WPL03].

Now that the first *modeling step* is achieved, we can move on to the *partial derivatives step*. Using the same technique as before, a constant step-size gradient algorithm leads to the following updates:

$$\begin{cases} \varrho_{f,k}(t+1) = \varrho_{f,k}(t) + \gamma_\varrho \cdot \Big(U'_f(\varrho_f(t)) - \displaystyle\sum_{l \text{ used by } f_k} \lambda_l(t)\Big) \\ \lambda_l(t+1) = \lambda_l(t) - \gamma_\lambda \cdot \Big(B_l - \displaystyle\sum_{f_k \text{ through } l} \varrho_{f,k}\Big) \end{cases} \tag{5.7}$$

We can now move on to the *algorithm design step*. The main difference with the previous setting is that each sub-flow f_k has its own rate and requires the aggregate throughput of flow f to be updated. More concretely, each flow f evaluates the price of each sub-flow f_k and updates the sub-flow rates accordingly, slowly moving toward the cheapest alternatives.

Unfortunately, a technical issue prevents the previous equations to be used directly. Since the original objective function is not strictly convex (it is *strictly* convex in any of the ϱ_k, but not with respect to the $\varrho_{f,k}$), the dual function d is not twice differentiable and so, a gradient descent algorithm based on this approach may oscillate and exhibit convergence instabilities. This problem can be circumvented by adding a quadratic term, which makes the primal cost function strictly convex[1]. This technique is called *proximal optimization* (see for example [BT89, Chapter3]) and is used in [WPL03] where two alternative algorithms are proposed to solve the flow control problem in multi-path networks.

Consider the new modified optimization problem:

$$\max_{\widetilde{\varrho}\geqslant 0} \max_{\varrho\geqslant 0} \sum_f U_f\left(\sum_k \varrho_{f,k}\right) - \sum_k \sum_f \frac{c}{2}(\varrho_{f,k} - \widetilde{\varrho}_{f,k})^2, \tag{5.8}$$

where $\widetilde{\varrho}_{f,k}$ is an auxiliary variable and c a constant (set to 1 in [WPL03]).

At the optimum, $\widetilde{\varrho}_{f,k} = \varrho_{f,k}$ and hence the solution of (5.8) is the same as the one of (5.6). This optimization problem is strictly concave in each variable $\widetilde{\varrho}_{f,k}$ and $\varrho_{f,k}$, and is equivalent to

$$\max_{\widetilde{\varrho}\geqslant 0} \max_{\varrho\geqslant 0} \min_{\lambda\geqslant 0} L(\widetilde{\varrho}, \varrho, \lambda), \tag{5.9}$$

[1] Han *et al.* [HSH+06] propose to add a term $\varepsilon \sum_{k,f} \log(\varrho_{f,k}))$ and letting $\varepsilon \to 0$. Although this approach is interesting, it may not be well-suited to a system that needs to operate continuously and where applications and machines regularly join and leave the system.

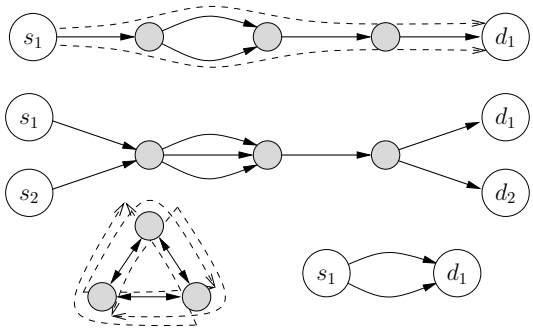


Figure 5.3: Four test topologies used in [WPL03, LS06, HSH+06] to illustrate the behavior of Lagrangian based flow control algorithms in multi-path networks. s_i denotes the source of a flow and d_i the destination. Dotted lines are one particular path that a flow may use.

where any of the minimization and maximization problems is a convex or concave optimization problem of a twice differentiable function. A classical fixed step-size gradient descent algorithm can be used for each level. Such three-level resolution would however be extremely inefficient in practice as it would require to detect several times the convergence (in a distributed way) of the inner problems before further proceeding on the outer problems. This is why Wang *et al.* [WPL03] propose to update all variables ϱ, $\widetilde{\varrho}$ and λ simultaneously, hence breaking the very constraining three-level hierarchical structure of the proximal optimization problem from Eq. (5.9). In essence, this leads to the same equations and algorithm as (5.7), except that the $\widetilde{\varrho}$ acts as a smoothing term for the ϱ variables to dampen oscillations.

Note that Wang *et al.* do not provide a proof of the convergence of this algorithm. This was studied in a more recent work of Lin and Schroff [LS06], in a similar setting, where the structure of the two outermost optimization problems is broken. In particular, they prove sufficient conditions on the step-sizes for the algorithm to converge and also study the effect of measurement noise.

More precisely, let us denote by $E_{f,k}$ the routing vector for sub-flow f_k. This means that $E^l_{f,k}$ is equal to 1 if route f goes through link l and 0 otherwise. One of the main results of [LS06] is that the algorithm converges if the step-size γ_λ satisfies

$$\gamma_\lambda < \frac{c}{2 \cdot S \cdot M}, \text{ where } \begin{cases} M = \max_{f,k} \sum_l E^l_{f,k} \text{ and} \\ S = \max_l \sum_k \sum_f E^l_{f,k} \end{cases} \tag{5.10}$$

More concretely, M is the length of the longest path and S is the largest number of sub-flows going through a link.

It should be mentioned that beside convexity issues, one of the main problems addressed in most of the previous work is the fact that in network protocols such as TCP or MP-TCP (Multi-Path TCP) [IET13], the price is interpreted as a measure of congestion and is thus implicitly measured. Such value is hence neither precise nor up-to-date, which creates instabilities. A large part of the existing related work is devoted to solving this problem (see for example [KGP+12]). However, as it will be explained in the next sections, we are not really concerned with such issue in our context nor in the experiments we performed, as we can develop an application-level protocol where price estimation can follow perfectly the dynamic induced by Lagrangian optimization.

Finally, although the multi-path flow control problem has been extensively studied on a theoretical point of view, it is interesting to note that, to the best of our knowledge, experimental validation of the resulting algorithms is rather limited. The only tested situations reported in [WPL03, LS06, HSH+06] are shown on Fig. 5.3 involve at most 8 nodes and 3 pairs of sources/destinations. In all theses studies, the proposed step-sizes for each setting lead to a satisfactory convergence within a few dozens to a few hundreds of iterations. Yet, it is difficult

to know how sensitive the algorithms are to these step-sizes, as well as how dependent good step-sizes are on the platform shape and size.

5.3 Toward a Distributed Algorithm for Throughput Optimization

It is easy to notice that problem (5.1) is completely similar to problem (5.6) and the same technique can thus be applied.

5.3.1 Computing Partial Derivatives

Applying the Lagrangian methodology to this context leads to the introduction of dual variables for both computation resources (λ_i for P_i) and communication resources ($\mu_{i,j}$ for $(P_i \to P_j)$). Again, the resulting algorithm will be governed by the following dynamic:

$$\begin{cases} \varrho_{n,k}(t+1) = \varrho_{n,k}(t) + \gamma_\varrho \cdot \dfrac{\partial L}{\partial \varrho_{n,k}}(\varrho(t), \widetilde{\varrho}(t), \lambda(t), \mu(t)) \\ \widetilde{\varrho}_{n,k}(t+1) = \widetilde{\varrho}_{n,k}(t) + \gamma_{\widetilde{\varrho}} \cdot \dfrac{\partial L}{\partial \widetilde{\varrho}_{n,k}}(\varrho(t), \widetilde{\varrho}(t), \lambda(t), \mu(t)) \\ \lambda_i(t+1) = \lambda_i(t) - \gamma_\lambda \cdot \dfrac{\partial L}{\partial \lambda_i}(\varrho(t), \widetilde{\varrho}(t), \lambda(t), \mu(t)) \\ \mu_{i,j}(t+1) = \mu_{i,j} - \gamma_\mu \cdot \dfrac{\partial L}{\partial \mu_{i,j}}(\varrho(t), \widetilde{\varrho}(t), \lambda(t), \mu(t)) \end{cases} \tag{5.11}$$

Expanding the partial derivatives for $\varrho_{n,k}$, we get:

$$\frac{\partial L}{\partial \varrho_{n,k}} = U'_k(\varrho_k(t)) - \underbrace{\Big(b_k \cdot \sum_{\substack{(P_i \to P_j) \text{ from} \\ m(k) \text{ to } P_n}} \mu_{i,j}(t) + w_k \cdot \lambda_n(t) \Big)}_{p_k^n(t)\text{: aggregate price to use } P_n}$$

As expected, the aggregate price to use P_n accounts for both communication link usage (the $\mu_{i,j}$) and CPU usage (the λ_n). Furthermore, this usage is weighted by communication (b_k) and computation (w_k) requirements of application k. Again, updating $\varrho_{n,k}$ requires the knowledge of ϱ_k, which is the aggregate throughput of application k. Since we aim at proportional fairness, use $U_k(\varrho_k) = \log(\varrho_k)$ and thus the corresponding update equation would write as follows:

$$\varrho_{n,k}(t+1) \leftarrow \left[(1-\gamma_\varrho)\varrho_{n,k}(t) + \gamma_\varrho \widetilde{\varrho}_{n,k}(t) + \gamma_\varrho \left(\frac{1}{\varrho_k(t)} - p_k^n(t) \right) \right]^+ . \tag{5.12}$$

A small value of ϱ_k leads to huge updates and thus to severe oscillations. As mentioned in [WPL03], it is possible to use independent step sizes for the $\widetilde{\varrho}$ part and for the price p_k^n part and to normalize this update as follows:

$$\varrho_{n,k}(t+1) \leftarrow \left[(1-\gamma_{\widetilde{\varrho}})\varrho_{n,k}(t) + \gamma_{\widetilde{\varrho}} \widetilde{\varrho}_{n,k}(t) + \gamma_\varrho \left(1 - \varrho_k(t).p_k^n(t)\right) \right]^+ . \tag{5.13}$$

Expanding partial derivatives with respect to other variables is straightforward and does not bring particular insight except that it relies on a set of aggregated quantities, which we will detail in the next section. They lead to update equations analogous to Eq. (5.7): as soon as the capacity of a resource is exceeded, its price increases and vice-versa.

5.3.2 Distributed Algorithm Design

In grids, the master-worker pairs are analogous to routes in the flow control problem, and applications are analogous to connections. Compared to the flow control problem, there is thus a huge number of "routes" and a very few "sources," which may have some important impact on the convergence rate.

Last, a subsequent difference lies in the decision points. While in networking context, sources adapt and choose their transmission rate, whereas in grids, we would like the intermediate nodes (between a master and each of its workers) to adjust the rates, so as to prevent overloading the master with information management and decision taking. Hence, we propose to use a "source" algorithm, which is based on a distributed aggregation of various quantities[2]:

$$\sigma_k^n(t) = \sum_{n' \text{ such that } n \in (P_{m(k)} \rightsquigarrow P_{n'})} \varrho_{n',k}(t) \tag{5.14}$$

$$\eta_k^n(t) = \sum_{(P_i \to P_j) \in (P_{m(k)} \rightsquigarrow P_n)} \mu_{i,j}(t) \tag{5.15}$$

σ_k^n is the aggregate throughput of application k at node n. Hence, it reflects how much data will need to flow through node n for application k. η_k^n is the price per byte to pay for sending data from the master $m(k)$ to node n. The price p_k^n can be computed from η_k^n as follows:

$$p_k^n(t) = b_k \cdot \eta_k^n(t) + w_k \cdot \lambda_n(t) \tag{5.16}$$

Expanding the partial derivatives for all variables and substituting them in the update equations, we obtain the following formulas for our algorithm:

$$\begin{aligned}
&\text{(5.17a)} \quad \varrho_{n,k}(t+1) \leftarrow \left[(1-\gamma_{\widetilde{\varrho}})\varrho_{n,k}(t) + \gamma_{\widetilde{\varrho}}\widetilde{\varrho}_{n,k}(t) + \gamma_{\varrho}\left(1 - \varrho_k(t).p_k^n(t)\right)\right]^+ \\
&\text{(5.17b)} \quad \widetilde{\varrho}_{n,k}(t+1) \leftarrow \left[(1-\gamma_{\widetilde{\varrho}})\widetilde{\varrho}_{n,k}(t) + \gamma_{\widetilde{\varrho}}\varrho_{n,k}(t)\right]^+ \\
&\text{(5.17c)} \quad \lambda_i(t+1) \leftarrow \left[\lambda_i(t) + \gamma_\lambda\left(\sum_k w_k \varrho_{i,k}(t) - W_i\right)\right]^+ \\
&\text{(5.17d)} \quad \mu_{i,j}(t+1) \leftarrow \left[\mu_{i,j}(t) + \gamma_\mu\left(\sum_k b_k \sigma_k^j(t) - B_{i,j}\right)\right]^+
\end{aligned} \tag{5.17}$$

Note that in the previous equation, one should use either σ_k^j or σ_k^i depending on whether i or j is met first in the deployment of application k.

Using this particular structure, we propose to implement this dynamic using classical traversal algorithms [Tel01] initiated by the master of each application.

Prerequisites Each node P_n is responsible for computing primal variables $\varrho_{n,k}$ and $\widetilde{\varrho}_{n,k}$, while the master nodes are responsible for the aggregation ϱ_k of the $\varrho_{n,k}$. Each resource (CPU or link) is responsible for its dual variable (λ_n or $\mu_{i,j}$). All these variables are initialized with random non-negative values.

Loop Master, workers and resources interact through the following two steps along the application deployment trees in an infinite loop.

- **Step 1: Propagation phase** Each master propagates its aggregate throughput ϱ_k along the tree to the workers. During the propagation, the aggregate price η_k^n for sending data from the master is computed based on the price μ of the communication resources encountered along the path down to node n. Therefore, upon reception, each node has all required information to compute p_k^n and update its contribution $\varrho_{n,k}$ to application k using Eq. (5.17a).

[2] Aggregate quantities are noted x_k^n instead of $x_{k,n}$

- **Step 2: Aggregation phase** Upon reception of ϱ_k and aggregate communication price, the leaves of the tree send back their new $\varrho_{n,k}$ value up-tree, which are in turn aggregated in σ_k^n up to the master.
  During the aggregation phase, every communication (resp. computation) resource has access to the load $b_k \cdot \sigma_k^j$ (resp. $w_k \cdot \varrho_{n,k}$) incurred by application k and can thus update its price $\mu_{i,j}$ (resp. λ_n). Such interaction ensures a permanent improvement of resource sharing and a seamless adaptation to variations of W_i of $B_{i,j}$ and to the arrival or departure of new nodes and applications.

Similarly to the original algorithm of [WPL03], there is no need for any global information, such as the number or the kind of nodes that are in the grid. Nodes only need to communicate with their neighbors and to update the variables they are responsible for. The wave algorithms seamlessly aggregate all required quantities with no direct interaction among the different applications. Furthermore, the resulting algorithm only requires very simple computations and few message exchanges.

We call the algorithm based on Eq. (5.17), the "/naive/ algorithm" as it is a straightforward application of the distributed Lagrangian optimization proposed in [WPL03] to our context.

5.3.3 A Disappointing Behavior

We implemented this naive algorithm using the SIMGRID simulator (details on experimental setup and results are reported in [34]). Our first attempts were done on rather large platforms and were complete disasters. Therefore, we quickly fell back on using a very simple platform consisting of only 5 nodes (see Fig. 5.4). For ease of interpretation, we even assumed a homogeneous set of nodes and links, although neither the algorithm nor our prototype requires that. We used three kinds of applications of respective (b, w): $(1000, 5000)$, $(2000, 800)$, and $(1500, 1500)$ that roughly correspond to a computation intensive application, a communication intensive application, and an intermediate application. We considered that each application originates from a different master, as represented on Fig. 5.4. Namely, application 1, 2 and 3 are hosted by nodes D, A and C respectively.

For each of the simulations presented below, we compared the obtained solutions to the ones provided by the free open source SDP solver called DSDP [BY05]. Our simulations showed that, even in this simple case study like the one presented in this section, step sizes (for ϱ, λ and μ) cannot be properly set. We illustrate this issue on Fig. 5.5(a) and 5.5(b). For relatively large value of γ_ϱ (e.g., $\gamma_\varrho = 100$), the algorithm is quite unstable, each update on the ϱ values being too large. Large updates can be avoided by using a smaller step size (for instance $\gamma_\varrho = 0.1$, see Fig. 5.5(b)). Yet, although more stable, this scheme does not converge well. Despite an extensive trial of parameters, the algorithm never converges within 5% of the optimal values, even after a thousand iterations. Since we thought it may be an implementation error on our side, we tried different simpler settings. First, we tried to reduce the setup to a single application and found out it was not too difficult to find an adequate set of steps to obtain convergence. Then we tried with several identical applications and found out again that it was possible to find effective step sizes. We were thus somehow able to reproduce the convergence of [WPL03] since

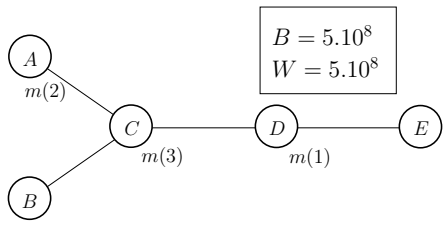
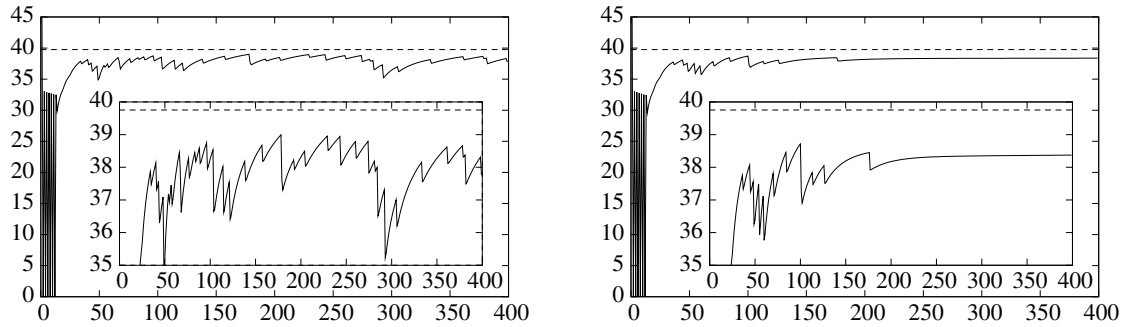


Figure 5.4: Simple platform topology. All applications originate from different nodes.



(a) "Normal" step sizes: ($\gamma_\lambda = 5.10^{-3}$, $\gamma_\lambda = 5.10^{-4}$, $\gamma_\varrho = 100.0$, $\gamma_{\tilde{\varrho}} = 0.02$). The algorithm is numerically unstable, which leads to global inefficiencies.

(b) Smaller step sizes ($\gamma_\lambda = 5.10^{-4}$, $\gamma_\lambda = 5.10^{-5}$, $\gamma_\varrho = 1.0$, $\gamma_{\tilde{\varrho}} = 0.02$) help reducing the oscillations and erratic values but incur a very slow convergence (if any).

Figure 5.5: Evolution of the objective value for the naive adaptation and comparison to the optimal solution. The upper dashed line represents the optimal solution.

the situation where applications are identical can perfectly be reduced to a multi-path routing problem. Further investigations presented in [4] show that, as long as applications are identical, this naive algorithm shows a reasonable convergence even for platform size as large as 500 nodes. However, as soon as applications are different, finding step sizes that lead to convergence seems impossible.

5.4 A Non-trivial Adaptation

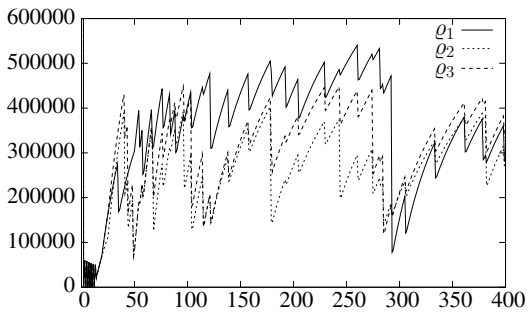
To understand where the problem could come from, we looked in more details to the evolution of each variables. First, if we study more carefully the evolution of Fig. 5.5(a), it can be seen that not only do the objective functions fluctuate, but also the throughput of each application, as represented in Fig. 5.6(a). To understand this behavior, let us recall that the throughput of each application is actually the sum of the rates on each of the 5 paths it follows: $\varrho_k = \sum \varrho_{i,k}$. Fig. 5.6(b) shows how ϱ_1 decomposes. We can note that the different $\varrho_{i,1}$ have exactly the same shape, and only varies by the instants they are reset to zero. These time epochs actually correspond to sudden jumps in the prices.

Indeed, a closer look at the system indicates erratic values of the prices in each node and link. Indeed, for each resource, the prices are null except for a finite number of points, where the values can be arbitrarily large (Fig. 5.6(c)).

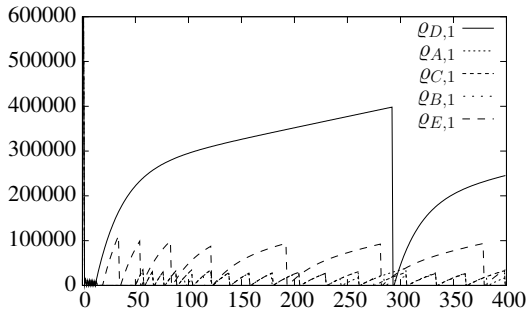
Hence, each application steadily increases it throughput while the prices of the resources it uses are null. Then, at some point a resource becomes saturated and its price suddenly "jumps," leading to a null throughput of all applications that use it. Then the process reiterates, which explains that the system cannot converge. In Eq. (5.17c) it is easy to see that the price update is directly proportional to the saturation of the resource. This saturation has the same order of magnitude as the $w_k \varrho_k$ whereas at the equilibrium, the price has the same order of magnitude as the inverse of the ϱ_k's. This explains why every time the objective function goes nearby the optimal value, it instantaneously bounces away.

Such instability is thus inherent to these update equations (regardless of the values of γ_λ and γ_μ): updating ϱ has an impact on the prices λ and μ, which in turn impact on the ϱ's update. Let us assume that we have reached the equilibrium. Further assume that the price λ_i of P_i is increased by $\Delta\lambda_i$. From Equation (5.17a), we derive that such an increase incurs a variation $\Delta\varrho_{i,k}$ of $\varrho_{i,k}$:

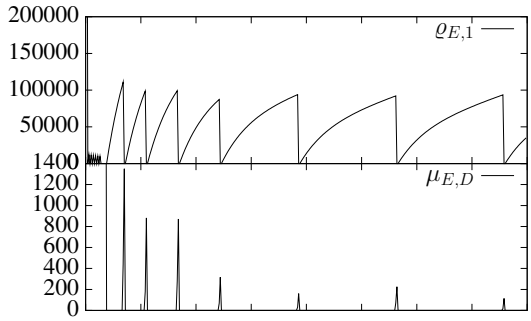
$$\Delta\varrho_{i,k} = -\gamma_\varrho w_k \Delta\lambda_i \varrho_k.$$



(a) Throughput of each of the three applications during 400 iterations: between two iterations, a decrease or increase of magnitude five or more can happen.



(b) The throughput of application seen as the sum of the rates through the different paths.



(c) Correlation between the throughput of an application on a given route and the price it experiences.

Figure 5.6: Investigating convergence issues. Price updates are over-reactive, which prevents the algorithm to converge.

In turn, from Equation (5.17c), we see that such a variation incurs a variation of λ_i:

$$\sum_{k \text{ s.t. } \varrho_{n,k} > 0} \gamma_\lambda . w_k . \Delta\varrho_{i,k} = -\Delta\lambda_i . \left(\sum_k \gamma_\lambda . \gamma_\varrho w_k^2 \varrho_k \right).$$

Thus, the solution of our gradient is stable only if

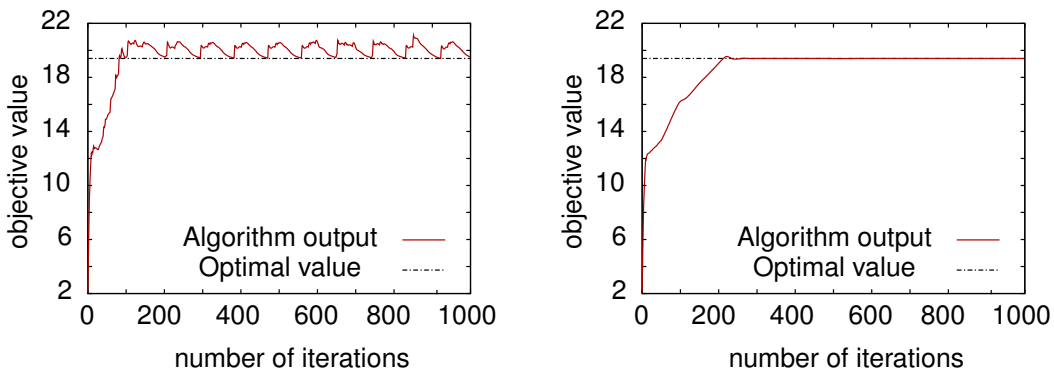
$$\sum_{k \text{ s.t. } \varrho_{n,k} > 0} \gamma_\lambda . \gamma_\varrho w_k^2 \varrho_k < 1.$$

Therefore, Eq. (5.17c) should be normalized as follows:

$$\lambda_i(t+1) \leftarrow \left[\lambda_i(t) + \gamma_\lambda \left(\frac{\sum_k w_k \varrho_{i,k}(t) - W_i}{\sum_{k \text{ s.t. } \varrho_{n,k} > 0} w_k^2 \varrho_k} \right) \right]^+$$

Obviously, the same **stability condition around the equilibrium** should also hold for μ. This first modification was presented in [34] and allows convergence for small platforms although it requires a careful tuning of step sizes. To obtain a more robust algorithm, we used two other techniques:

1. **Fast convergence of the primal** As we have previously seen, constant step-size gradient descent on a convex function F is done by repeating the following updates: $x(t+1) \leftarrow x(t) - \gamma \nabla F(x(t))$. It is well known that Newton's algorithm has much faster convergence than simple gradient projection algorithm. In *Newton's algorithm*, the updates are as follows:



(a) Setting the prices to zero if a division by zero occurs

(b) Damping with geometrical update

Figure 5.7: Stabilization using geometric update.

$x(t+1) \leftarrow x(t) - \gamma \left(\nabla^2 F(x(t))\right)^{-1} \cdot \nabla F(x(t))$. Inverting the Hessian matrix $\nabla^2 F(x(t))$ is however very time consuming, which is why *approximate Newton methods* are often used [BT89]. In such methods, the $\nabla^2 F(x(t))$ matrix is often replaced by a simpler matrix (like its diagonal), whose inversion is straightforward and still has the right order of magnitude. Computing the Hessian matrix for our particular problem leads to a non-invertible matrix because of the non-strict convexity of our initial objective function. Considering only diagonal elements, we get a new scaling that replaces Eq. (5.12) (instead of Eq. (5.13) proposed in [WPL03]):

$$\varrho_{n,k}(t+1) \leftarrow \left[(1-\gamma_{\widetilde{\varrho}})\varrho_{n,k}(t) + \gamma_{\widetilde{\varrho}}\widetilde{\varrho}_{n,k}(t) + \gamma_{\varrho}\big(1 - \varrho_k(t) \cdot p_k^n(t)\big)\varrho_k(t).\right]^+ . \tag{5.18}$$

Obviously, such modification requires to revisit the stability condition around the equilibrium but it only incurs mild modifications.

2. **Avoiding division by very small values and discontinuities** An important point on which we have not insisted yet is that every variable needs to remain non-negative (hence, the need in the previous updates of primal variables to only consider variables $\varrho_{n,k}$ that are positive). Hence, in any such distributed gradient algorithm, if any update step leads to a negative value, the variable is set to 0. This kind of projection is done with the operator $[x(t)+u]^+ = \max(0, x(t)+u)$ and is applied to every variable (both primal and dual). It is typical for such methods but raises several issues in our context. Indeed, it may be the case from an iteration to another that a denominator experiences a very important variation, which may cause a large negative step. Whenever many dual (resp. primal) variables suddenly drop to 0, it generally causes a large increase of the primal (resp. dual) variables. This is why these projections need to be smoothed. We used the following smooth projection operator, which revealed extremely efficient:

$$[x(t)+u]^{\alpha+} = \max(\alpha \cdot x(t), x(t)+u), \text{with } 0 < \alpha < 1$$

With such updates, variables never suddenly drop to 0. Instead, variables geometrically decrease to zero, until the corresponding resource is used again. In our experiments, we set α to $1/2$. As illustrated in Fig. 5.7, this technique proves very efficient for removing oscillations.

During our investigations, we observed that Newton method on the primal, stability condition on the dual and geometric updates all improved convergence without completely solving the issue. Removing any of these ingredients leads to a dynamic that generally exhibits severe convergence issues, which makes it hard to evaluate their respective influence. Our new "*adaptive*

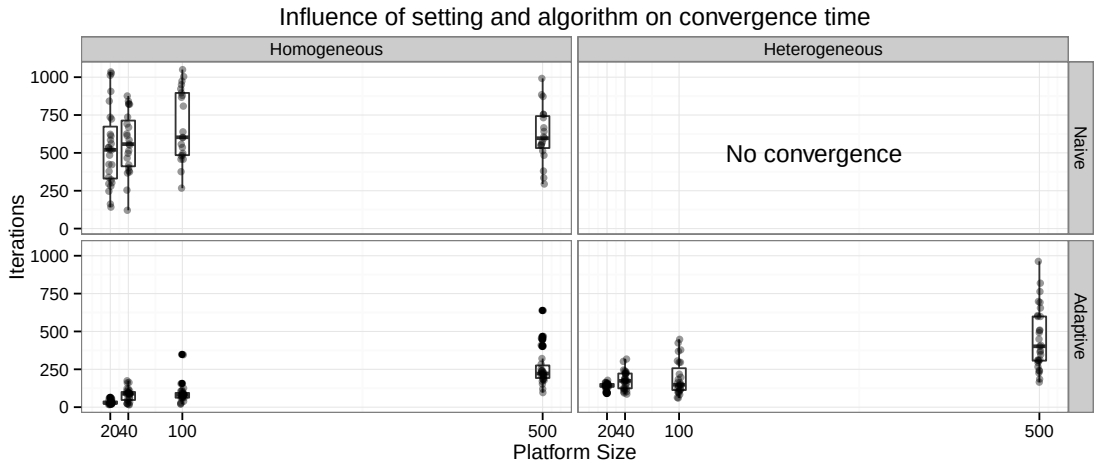


Figure 5.8: Convergence time distribution as a function of platform size.

algorithm" is thus governed by the following equations:

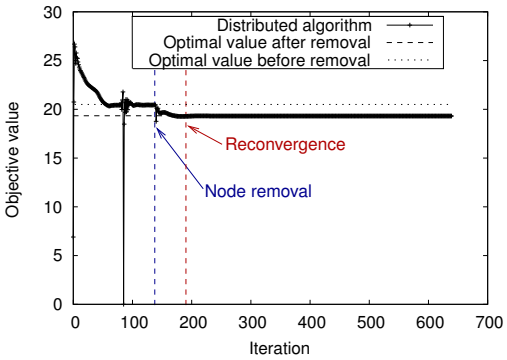
$$
\begin{aligned}
&(5.19a) \quad \varrho_{n,k}(t+1) \leftarrow \left[(1-\gamma_{\widetilde{\varrho}})\varrho_{n,k}(t) + \gamma_{\widetilde{\varrho}}\widetilde{\varrho}_{n,k}(t) + \gamma_{\varrho}\left(1-\varrho_k(t).p_k^n(t)\right)\varrho_k(t)\right]^{\alpha+} \\
&(5.19b) \quad \widetilde{\varrho}_{n,k}(t+1) \leftarrow \left[(1-\gamma_{\widetilde{\varrho}})\widetilde{\varrho}_{n,k}(t) + \gamma_{\widetilde{\varrho}}\varrho_{n,k}(t)\right]^{\alpha+} \\
&(5.19c) \quad \lambda_i(t+1) \leftarrow \left[\lambda_i(t) + \gamma_\lambda \frac{\sum_k w_k \varrho_{i,k}(t) - W_i}{\sum_{k \text{ s.t. } \varrho_{n,k}>0} w_k^2 \varrho_k^2(t)}\right]^{\alpha+} \\
&(5.19d) \quad \mu_{i,j}(t+1) \leftarrow \left[\mu_{i,j}(t) + \gamma_\mu \frac{\sum_k b_k \sigma_k^j(t) - B_{i,j}}{\sum_k \sum_{\substack{n \text{ such that} \\ (P_i \to P_j) \in (P_{m(k)} \rightsquigarrow P_n) \\ \text{and } \varrho_{n,k}>0}} b_k^2 \varrho_k^2(t)}\right]^{\alpha+}
\end{aligned}
\tag{5.19}
$$

Note that this scaling does not require any additional aggregation operation since all processors already receive ϱ_k to perform the update of ϱ.

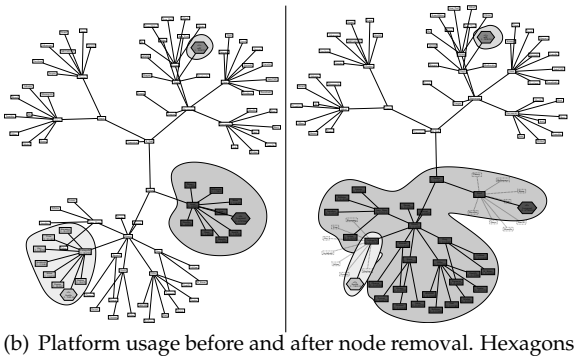
In [4], we show how to use designed factorial experiments to determine robust step-sizes and obtain convergence or the new algorithm. Although such approach is very effective, it turns out that a more careful look at the number of iterations required to converge is highly dependent on platform size. Fig. 5.8 summarizes the evolution of convergence time distribution depending on platform size, application setting (homogeneous vs. heterogeneous) and algorithm (naive vs. adaptive).

At first sight, it may look like convergence time of the naive algorithm in the homogeneous application setting (upper left part of Fig. 5.8), although slower, is not really sensitive to platform size. Yet, one should recall that this naive algorithms never converges in the heterogeneous application setting and that only 20 configurations out of 30 converged in the 500 nodes setting whereas 28 configurations out of 30 converged in the 20 nodes setting. Convergence for the 500 node setting could probably be improved as only a crude step-size tuning was done but not to the extent where convergence time dramatically decreases. As we explained in Section 5.3.3 and as can be seen in the upper right part of Fig. 5.8, the naive algorithm simply never converges when heterogeneous applications are deployed. On the other hand, the adaptive algorithm we proposed converges both in the homogeneous and heterogeneous application setting and even within much better time bounds than the naive algorithm for homogeneous applications.

However, as can be observed on the lower part of Fig. 5.8, platform size has a tremendous impact on the convergence time. For a 500 node platform, starting arbitrarily far from the optimal



(a) Worst node removal: all nodes contributing to the computation are suddenly removed from the network at the 140th iteration. The optimum of the objective function decreases, and the algorithm re-converges in a few dozen of iterations, without oscillations.



(b) Platform usage before and after node removal. Hexagons represent the masters and gray nodes participate to the computations. After having removed all nodes involved in computations, one application was redeployed on another part of the network and now interferes with another application.

Figure 5.9: Behavior of the algorithm when removing all participating nodes.

solution, the 95% confidence interval for the expected number of steps is $[373, 531]$. A 500 node platform with $d_{max} = 15$ has a diameter of roughly 10 and thus the expected convergence time would be less than nine minutes (assuming a $50ms$ RTT between machines).

5.5 Conclusion and Open Issues

We have succeeded in obtaining a distributed algorithm which computes a proportionally fair sharing of resources in steady-state. Although the adaptation from multi-path routing should have been straightforward, several non-trivial adaptations were required by our context.

All previous work relying on this technique had evaluated the effectiveness of their proposal in very limited settings comprising at most a dozen of nodes and a few pairs of sources/destinations. Instead, we evaluated our algorithms in a much more complex setting with up to 500 node platforms. To the best of our knowledge, it is the first time such kind of algorithm is evaluated at such scale and our study reveals issues that had been unnoticed until now. In particular, this enabled to show that the difficulties in the multi-path flow-control problem are different from ours. Indeed, one may have believed that our context is simpler since we can implement the true dynamic at application level, which is not possible when designing network protocols that have to be compatible with the rest of the network stack. However, the heterogeneity of applications in our context raises a new challenge in term of convergence.

I think this study leaves several paths open:

- An interesting feature of this approach is the ability to seamlessly adapt to variations of W_i of $B_{i,j}$ and to the arrival or departure of new nodes and applications (see Fig. 5.9(a)). We almost haven't investigated such feature so far but it is yet of uttermost importance before a practical usage can be expected.

- Regarding the fundamentals of the algorithm, although the similitude between flow control in multi-path networks and steady-state scheduling of BoT has motivated the use of the augmented Lagrangian method, other methods could be used. As we have explained, one of the key ingredients for an effective implementation of such method is the smooth projection operator that shares similarities with a barrier function. Lagrangian algorithms move toward the optimal solution by oscillating around the constraints and constantly trying to overuse resources. Besides the resulting potentially slow convergence, the solution may not always be feasible at a given time step. Interior point methods do not suffer from such

issues and have received a lots of attention lately. A rigorous treatment of the alternating direction method of multipliers was also recently brought to my attention [BPC+11]. Such approach slightly differs from the one we used but its convergence is ensured through a potential function.

- Like most scheduling work, we have assumed that an accurate description of resources, in particular of the topology, was available. We also assume a somehow perfect matching of the application overlay with the physical topology. Our attempts to obtain such information in an automatic way [49, 36] taught us that this assumption is probably too strong. In existing large-scale systems that may deal with communication-bound, one would not have access to the right level of information. Furthermore, we would not be able to deploy scheduling agents on routers and one would thus have to deal with invisible resources and the fact that our application topology is only a crude approximation of the real topology. Therefore, although this seems very difficult to study, I think that checking the influence of the application overlay as being only a crude approximation of the physical topology would be very instructive.

- Another possible usage of such technique would be to exploit affinities in BOINC. Indeed applications have several versions depending on architecture and operating system. As a consequence, some processor may be better fitted to some applications than others (unrelated speed). Since sharing is determined by the volunteer priorities, the same kind of inefficiencies as the ones we explained in Chapter 3 are possible. The same kind of algorithm could be used and would be even simpler. Our first simulations showed that it would work perfectly and would allow for smart optimization of the whole system. However our first investigation of the BOINC workload revealed that there would not be too much to gain and that the already existing pragmatic strategy is probably sufficient for now. If heterogeneity keeps increasing such ideas could provide an interesting solution.

- Finally, the previous study heavily relies on the steady-state throughput formulation and hence achieves a flow-based fairness that accounts for the different application characteristics and needs. However, such modeling by essence ignores temporal aspects such as application campaign termination and creation, which is highly criticizable [Bri07]. In a context where a platform is shared among different users, it would make sense that users can express that they have a urgent need of resources to meet a deadline and that they are willing to release critical resources later. Likewise, whenever a user does not use resources for some time it may make sense that whenever he comes back with new work he obtains a higher priority than those that have used "more than their share". As we have seen, this notion of share is quite difficult to define in a static context (i.e., without task arrival and termination) and including sound temporal components seems even more challenging. Although related notions of repeated games (in the game theory field) or of regret minimization (in the stochastic optimization field) have been introduced, I have not found any really satisfying formulation yet. The next two chapters present some attempts in this direction that have been pursued in parallel with the work presented in the previous three chapters. Although they are clearly no definitive solution to this fair resource sharing problem with temporal constraints, I think they provide some interesting insights in how this problem is difficult.

Chapter 6

Centralized Response Time Optimization

This chapter builds on two articles published at HCW'05 [42], SPAA'06 [40] and in JoS'08 [9] with Fréderic Vivien and Alan Su. This work was initiated right after my PhD defense, while I was still in Lyon and was pursued in the context of the ALPAGE ANR project.

This work was originally motivated by a French ACI GRID project involving the IBCP (Institute of Biology and Chemistry of Proteins), the IN2P3 computing center and the LIP in which I was working. The goal of this project was to set up a grid infrastructure to perform protein pattern/profile scanning: GriPPS [BCGD00, Gri05]. It turned out that this application was well modeled as a divisible load scheduling problem with release dates, which can be related to scheduling with preemption. We we have thus revisited many results from this literature. Doing so, we have gathered all related complexity results and drawn the corresponding complexity landscape. This has allowed to fill a few holes (e.g., by improving some inapproximability bounds and proving a few NP-completeness results) and to clearly identify what was still open.

Probably more important, this theoretical scheduling study has allowed us to extract some important techniques and intuitions. One of the outcome is a pragmatic heuristic that performs in practice much better than already existing algorithms that have proven performance guarantees.

6.1 The GriPPS Infrastructure

The GriPPS framework is based on large databases of information about proteins; each protein is represented by a string of characters denoting the sequence of amino acids of which it is composed. Biologists need to search such sequence databases for specific patterns that indicate biologically significant structures. The GriPPS software enables such queries in grid environments, where the data may be replicated across a distributed heterogeneous computing platform.

As a matter of fact, there seems to be two common usages in protein comparison applications. In the first case, a biologist working on a set of proteins builds a pattern to search for similar sequences on the servers (this is the case for the GriPPS framework). In the second case, canonical patterns are known and should be used for comparison with daily updates of the databanks. This is the only case we are aware of where a very large set of motifs is sent to all databanks. This is however a typical background process whereas the first case is a typical online problem as many biologists concurrently use the servers. Therefore in this first case, the motifs are very small and communication cost they incur can really be neglected. To develop a suitable application model for the GriPPS application scenario, we performed a series of experiments to analyze the fundamental properties of the sequence comparison algorithms used in this code. Here we report on the conclusions of this study whose details can be found in [42, LSV04].

From our modeling perspective, the critical components of the GriPPS application are:

- **Protein databanks**: the reference databases of amino acid sequences, located at fixed locations in a distributed heterogeneous computing platform;

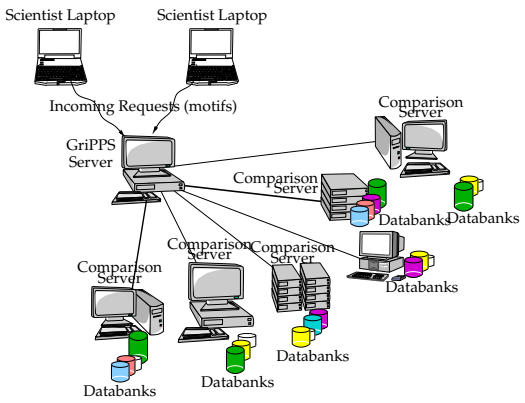


Figure 6.1: GriPPS Setting

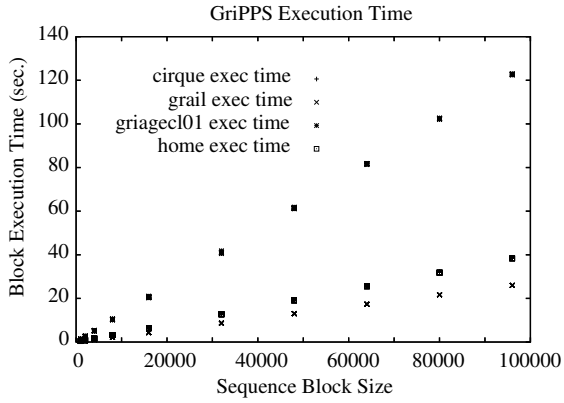


Figure 6.2: Sequence databank divisibility studies.

- **Motifs**: compact representations of amino acid patterns that are biologically important and serve as user input to the application.
- **Sequence comparison servers**: computational processes co-located with some protein databanks that accept as input sets of motifs and return as output all matching entries in any subset of a particular databank.

The main characteristics of the GriPPS application are:

1. **Negligible communication costs**. A motif is a relatively compact representation of an amino acid pattern. Therefore, the communication overhead induced while sending a motif to any processor is negligible compared to the processing time of a comparison. Although the transfer times of results is larger, it also remains negligible compared to the computational requirements of this application.

2. **Divisible loads**. As illustrated on Fig. 6.2, the processing time required for sequence comparisons against a subset of a particular databank is linearly proportional to the size of the subset. This property allows us to distribute the processing of a request among many processors at the same time without additional cost. The GriPPS protein databank search application is therefore an example of a *linear divisible workload without communication costs*.

   In the classical scheduling literature, preemption is defined as the ability to suspend a job at any time and to resume it, possibly on another processor, at no cost. Our application implicitly falls in this category. Indeed, we can easily halt the processing of a request on a given processor and continue the pattern matching for the unprocessed part of the database on a different processor (as it only requires a negligible data transfer operation to move the pattern to the new location). From a theoretical perspective, divisible load without communication costs can be seen as a generalization of the *preemptive execution model* that allows for simultaneous execution of different parts of a same job on different machines.

3. **Uniform machines with restricted availabilities**. A set of jobs is uniform over a set of processors if the relative execution times of jobs over the set of processors does not depend on the nature of the jobs. More formally, for any job J_j, the time $p_{i,j}$ needed to process job J_j on processor i is equal to $w_j \cdot W_i$, where W_i describes the speed of processor i and w_j represents the size of J_j. Our experiments indicated a clear constant relationship between the computation time observed for a particular motif on a given machine, compared to the computation time measured on a reference machine for that same motif. This trend supports the hypothesis of uniformity. However, in practice a given databank may not be available on all sequence comparison servers. Our model essentially represents a *uniform*

machines with restricted availabilities scheduling problem, which is a specific instance of the more general *unrelated machines* scheduling problem.

6.2 Scheduling for the Divisible Load Model

6.2.1 Framework and Notations

An instance of our problem is defined by n jobs, $J_1,\ldots,J_n$ and m machines (or processors), $M_1,\ldots,M_m$. The job J_j arrives in the system at time r_j (expressed in seconds), which is its release date; we suppose that jobs are numbered by increasing release dates.

The value $p_{i,j}$ denotes the amount of time it would take for machine M_i to process job J_j. Note that $p_{i,j}$ can be infinite if the job J_j cannot be executed on the machine M_i, e.g., for our motivating application, if job J_j requires a databank that is not present on the machine M_i. Finally, each job may be assigned a *weight* or *priority* ω_j.

As we have seen, for the particular case of our motivating application, we could replace the unrelated times $p_{i,j}$ by the expression w_j/W_i, where w_j denotes the size (in Mflop) of the job J_j and W_i denotes the computational capacity of machine M_i (in Mflop/s). To maintain correctness for the biological sequence comparison application, we separately maintain a list of databanks present at each machine and enforce the constraint that a job J_j may only be executed on a machine that has a copy of all data upon which job J_j depends. However, since the theoretical results we present do not rely on these restrictions, we retain the more general scheduling problem formulation that is, we address the unrelated machines framework in this article. As a consequence, all the values we consider in this article are nonnegative rational numbers (except the previously mentioned case in which $p_{i,j}$ is infinite if J_j cannot be processed on M_i).

The time at which job J_j is completed is denoted as C_j. Then, the *flow time* of the job J_j, defined as $F_j = C_j - r_j$, is essentially the time the job spends in the system.

Due to the divisible load model, each job may be divided into an arbitrary number of sub-jobs, of any size. Furthermore, each sub-job may be executed on any machine at which the data dependencies of the job are satisfied. Thus, at a given moment, many different machines may be processing the same job (with a master scheduler ensuring that these machines are working on *different* parts of the job). Therefore, if we denote by $\alpha_{i,j}$ the fraction of job J_j processed on M_i, we enforce the following property to ensure each job is fully executed: $\forall j, \sum_i \alpha_{i,j} = 1$.

When a size w_j can be defined for each job J_j (e.g., in the single processor case) we denote by Δ the ratio of the sizes of the largest and shortest jobs submitted to the system: $\Delta = \frac{\max_j w_j}{\min_j w_j}$.

6.2.2 Relationships with the Single Processor Case with Preemption

We first prove that any schedule in the uniform machines model with divisibility has a canonical corresponding schedule in the single processor model with preemption. This is especially important as many interesting results in the scheduling literature only hold for the preemptive computation model (denoted *pmtn*).

Lemma 6.1. *For any platform M_1, ..., M_m composed of uniform processors, i.e., such that for any job J_j, $p_{i,j} = w_j/W_i$, one can define a platform made of a single processor $\widetilde{M}$ with $\widetilde{W} = \sum_i W_i$, such that: - For any divisible schedule of J_1, ..., J_n on $\{M_1, ..., M_m\}$ there exists a preemptive schedule of J_1, ..., J_n on $\widetilde{M}$ with smaller or equal completion times. - Conversely, for any preemptive schedule of J_1, ..., J_n on $\widetilde{M}$ there exists a divisible schedule on $\{M_1, ..., M_m\}$ with equal completion times.*

Fig. 6.3 illustrates the underlying idea (see [LSV06] for details). The reverse transformation simply processes jobs sequentially, distributing each job's work across all processors.

As a consequence, any complexity result for the preemptive single processor model also holds for the uniform divisible model. Thus, throughout this article, in addition to addressing the multi-processor case, we will also closely examine the single processor case.

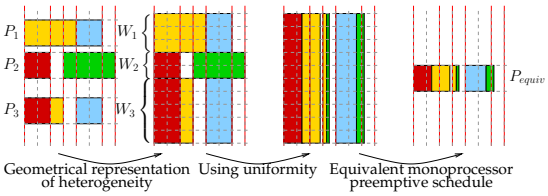


Figure 6.3: Geometrical transformation of a divisible uniform problem into a preemptive single processor problem.

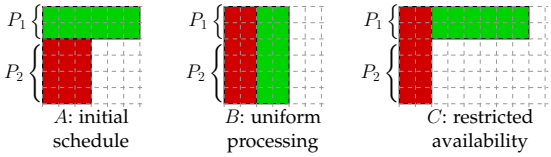


Figure 6.4: Illustrating the difference between the uniform model and the restricted availability model.

Unfortunately, this line of reasoning is no longer valid when the computational platform exhibits restricted availability. In the single processor case, a schedule can be seen as a priority list of the jobs (see the article of Bender, Muthukrishnan, and Rajaraman [BMR04] for example). For this reason, whenever we will present heuristics for the uniprocessor case they will follow the same basic approach: maintain a priority list of the jobs and at any moment, execute the one with the highest priority. In the multi-processor case with restricted availability, an additional scheduling dimension must be resolved: the spatial distribution of each job.

The example in Fig. 6.4 explains the difficulty of this last problem. In the uniform situation, it is always beneficial to fully distribute work across all available resources: each job's completion time in situation B is strictly better than the corresponding job's completion time in situation A. However, introducing restricted availability confounds this process. Consider a case in which tasks may be limited in their ability to utilize some subset of the platform's resources (e.g., their requisite data are not present throughout the platform). In situation C of Fig. 6.4, one task is subject to restricted availability: the P_2 computational resource is not able to service this task.

Deciding between various scheduling options in this scenario is non-trivial in the general case (for example schedule A has a better max flow than schedule C, but schedule C has a better max stretch than schedule A), so we apply the following simple rule to build a schedule for general platforms from single processor heuristics: While some processors are idle, one selects the job with the highest priority and distribute its processing on all available processors that are capable of processing it.

6.2.3 Optimization Criteria

An other important characteristic of our problem is that we target a platform shared by many users. As a consequence, we need to ensure a certain degree of fairness between the different requests. Given a set of requests, how should we share resources among the different requests?

The most common objective function in the parallel scheduling literature is the *makespan*: the maximum of the job termination times, or $\max_j C_j$. Makespan minimization is supposed to be a system-centric approach, seeking to ensure efficient platform utilization. Makespan minimization is meaningful when there is only one user and when all jobs are submitted simultaneously. It is also common to consider average completion time instead ($\sum_j C_j$) but although such metric accounts for every user, it does not take into account their release dates.

Individual users sharing a system are typically more interested in job-centric metrics, such as *job flow time* (also called *response time*): the time an individual job spends in the system (i.e., $C_j - r_j$). It is thus common to consider minimization of the maximum flow time, $\max_j F_j$. On a single processor with preemption, this metric is minimized with the simple *first come first served* (*FCFS*) policy, which prioritizes job by the inverse of their release date. Therefore, we do not even need to preempt jobs. Another interesting property of this policy is that it is non-clairvoyant, i.e., it does not relies on the processing time p_j of jobs. The optimality of this strategy relies on a simple exchange argument. However, whenever a long job is submitted, once it is started, it is never preempted and one would thus hardly consider a system operated with such a strategy to be reactive. Somehow, a max-based criterion focuses on worst-case behavior while one may be

more interested by the average behavior. It would probably more reasonable to sacrifice some large jobs to get something more "reactive" and favor shorter jobs.

It is thus common to consider the average (or total) flow time, $\sum_j F_j$, which accounts for every single user in a single number. On a single processor with preemption, this metric is minimized with the simple *shortest remaining processing time first* (SRPT) policy, whose optimality relies again on a simple exchange argument. Scheduling small jobs first is good for "reactivity" but it requires to know the size of the jobs (i.e., it is a clairvoyant algorithm). Unfortunately, although scheduling small jobs first is good for the average response time, large jobs may be left behind and starvation is thus possible, i.e., some (large) jobs may be delayed to an unbounded extent [BCM98].

To overcome the previous problems, one common approach [CK02] focuses on the *weighted* flow time, using job weights to offset the bias against short or large jobs. *Sum weighted flow* and *maximum weighted flow* metrics can then be analogously defined. Note however that the starvation problem identified for sum-flow minimization is inherent to all sum-based objectives, so the sum weighted flow suffers from the same weakness. The *stretch* is a particular case of weighted flow, in which a job's weight is inversely proportional to its size: $\omega_j = 1/w_j$ [BCM98]. On a single processor, the stretch of a job can be seen as the slowdown it experiences when the system is loaded. In a network context, the stretch can be seen as the inverse of the overall bandwidth allocated to a given transfer (i.e., the amount of data to transfer divided by the overall time needed to complete the transfer). However this kind of definition does not account for the affinity of some tasks with some particular machines (e.g., the scarcity of a particular database). That is why we think a slightly different definition should be used in an unrelated machines context. The stretch is originally defined to represent the slowdown a job experiences when the system is loaded. In the remaining of this article, we will thus define the stretch as a particular case of weighted flow time, in which a job's weight is inversely proportional to its processing time when the system is empty: $\omega_j = \sum_i \frac{1}{p_{i,j}}$ in our divisible load model. This definition matches the previous one in a single processor context and is thus a reasonably fair measure of the level of service provided to an individual job. It is more relevant than the flow in a system with highly variable job sizes. Consequently, in our context, mainly the sum-stretch ($\sum S_j$) and the max-stretch ($\max S_j$) metrics are meaningful.

The two simple heuristics we have presented so far (*FCFS* and *SRPT*) are both optimal (on a single processor with preemption) for a criteria but behave quite differently for the others.

	$\max F_j$	$\sum F_j$	$\max S_j$	$\sum S_j$
FCFS	Optimal	Δ-competitive [9]	Δ-competitive [9]	Δ^2-competitive [9]
SRPT	No guarantee	Optimal [Bak74]	No guarantee	2-competitive [MRSG99]

Note that the previous competitiveness ratios are tight and that, at least for optimal strategies, it is not possible to optimize both sum- and max- criteria at the same time. This is not surprising but one may thus wonder whether a trade-off may be found.

Theorem 6.1 ([9]). *Consider any online algorithm which has a competitive ratio of $\varrho(\Delta)$ for the sum-flow. We assume that this competitive ratio is not trivial, i.e., that $\varrho(\Delta) < \Delta$. Then, there exists for this algorithm a sequence of jobs that leads to starvation, and thus for which the obtained max-stretch is arbitrarily greater than the optimal max-stretch.*

Likewise, for any online algorithm which has a non-trivial competitive ratio of $\varrho(\Delta) < \Delta^2$ for the sum-stretch, there exists a sequence of jobs leading to starvation and where the obtained max-flow is arbitrarily greater than the optimal one.

Intuitively, algorithms targeting max-based metrics ensure that no job is *left behind*. Such an algorithm is thus somehow extremely "fair". Sum-based metrics tend to optimize instead the *utilization* of the platform. The previous theorem establishes that these two objectives can be in opposition on particular instances. As a consequence, it should be noted that any algorithm optimizing a sum-based metric has the particularly undesirable property of potential starvation. This observation, combined with the fact that the stretch is more relevant than the flow in a system

with highly variable job sizes, motivates max-stretch as the metric of choice in designing scheduling algorithms in the GriPPS setting. Yet, sum-stretch should be considered as a secondary objective although simultaneous optimization cannot be guaranteed. Therefore, the following sections present a few results and intuitions on sum stretch and max stretch optimization.

6.2.4 Sum Stretch: Shortest Processing Time Rules

In the general case, without preemption and divisibility, minimizing the sum-stretch is an NP-complete problem, even on a single processor [9]. On the other hand, The complexity of the offline minimization of the sum-stretch with preemption is still an open problem. At the very least, this is a hint at the difficulty of this problem. In the framework with preemption, Bender, Muthukrishnan, and Rajaraman [BMR04] present a Polynomial Time Approximation Scheme (PTAS) for minimizing the sum-stretch with preemption. Chekuri and Khanna [CK02] present an approximation scheme for the more general sum weighted flow minimization problem. As these approximation schemes cannot be extended to work in an online setting, we will not discuss them further. Muthukrishnan, Rajaraman, Shaheen, and Gehrke [MRSG99] prove that there is no optimal online algorithm for the sum-stretch minimization problem when there are three or more distinct job sizes. Furthermore, they give a lower bound of 1.036 on the competitive ratio of any online algorithm. In [40], we improve this bound and show that competitive ratio is actually less than or equal to 1.19484.

We have recalled that **shortest remaining processing time** (*SRPT*) is optimal for minimizing the sum-flow. When *SRPT* takes a scheduling decision, it only considers the *remaining* processing time of a job, and not its *original* processing time. Therefore, from the point of view of the sum-stretch minimization, *SRPT* does not take into account the *weight* of the jobs in the objective function. Nevertheless, Muthukrishnan, Rajaraman, Shaheen, and Gehrke have shown [MRSG99] that *SRPT* is 2-competitive for sum-stretch.

Another well studied algorithm is the Smith's ratio rule [Smi56] also known as **shortest weighted processing time** (*SWPT*). This is a preemptive list scheduling where the available jobs are executed in increasing value of the ratio $\frac{p_j}{\omega_j}$. Whatever the weights, *SWPT* is 2-competitive [SS02] for the minimization of the sum of weighted completion times ($\sum w_j C_j$).

Note that a ϱ-competitive algorithm for the sum weighted flow minimization ($\sum w_j(C_j - r_j)$) is ϱ-competitive for the sum weighted completion time ($\sum w_j C_j$). However, the reverse is not true: a guarantee on the sum weighted completion time ($\sum w_j C_j$) does not induce any guarantee on the sum weighted flow ($\sum w_j(C_j - r_j)$). Therefore, the previous ratio on the minimization of the sum of weighted completion times gives us no result on the efficiency of *SWPT* for the minimization of the sum-stretch. Furthermore, it is not difficult to prove [40] that *SWPT* is not a competitive algorithm for minimizing the sum-stretch.

The weakness of the *SWPT* heuristics is obviously that it does not take into account the remaining processing times: it may preempt a job when it is almost completed. To address the weaknesses of both *SRPT* and *SWPT*, one might consider a heuristic that takes into account both the original and the remaining processing times of the jobs. This is what the **shortest weighted remaining processing time** heuristic (*SWRPT*) does. In the framework of sum-stretch minimization, at any time t, *SWRPT* schedules the job J_j which minimizes $p_j \varrho_t(j)$. Muthukrishnan, Rajaraman, Shaheen, and Gehrke [MRSG99] prove that *SWRPT* is actually optimal when there are only two job sizes.

Unfortunately, neither of the proofs of competitiveness of *SRPT* or *SWPT* can be extended to *SWRPT*. *SWRPT* has apparently been studied by Megow [Meg02], but only in the scope of the sum weighted completion time. So far, there is no guarantee on the efficiency of *SWRPT* for sum-stretch minimization. Intuitively, we would think that *SWRPT* is more efficient than *SRPT* for the sum-stretch minimization. However, one can prove [40] that *SWRPT* is at best 2-competitive, just like *SRPT*. Yet, the experimental results we will present later show yet that, in practice, *SWRPT* is generally better than the simpler *SRPT* algorithm.

The main results are thus that:

1. The complexity of offline sum stretch minimization is still an open problem. It seems rather combinatorial and none of the classical technique and intuition proved useful, hence PTAS were proposed in the literature.

2. The online setting is much simpler since very simple algorithms are 2-competitive and in practice the *SWRPT* heuristic is a good rule of thumb.

6.2.5 Max stretch: Deadline Scheduling and Linear Programming

Max Weighted Flow Minimization and Deadline Scheduling

Let us assume that we are looking for a schedule $\mathcal{S}$ under which the maximum weighted flow is less than or equal to some objective value $\mathcal{F}$. The weighted flow of any job J_j is equal to $\omega_j(C_j - r_j)$. Then, we should have:

$$\max_{1\leqslant j\leqslant n} w_j(C_j - r_j) \leqslant \mathcal{F} \quad \Leftrightarrow \quad \forall j \in [1;n],\ w_j(C_j - r_j) \leqslant \mathcal{F} \quad \Leftrightarrow \quad \forall j \in [1;n],\ C_j \leqslant r_j + \mathcal{F}/\omega_j.$$

Thus, the execution of J_j must be completed before time $d_j(\mathcal{F}) = r_j + \mathcal{F}/\omega_j$ for schedule $\mathcal{S}$ to satisfy the bound $\mathcal{F}$ on the maximum weighted flow. Therefore, looking for a schedule which satisfies a given upper bound on the maximum weighted flow is equivalent to an instance of the deadline scheduling problem. We now show how to solve such a deadline scheduling problem in the divisible load framework.

In *deadline scheduling*, each job J_j has not only a release date r_j but also a deadline d_j. The problem is then to find a schedule such that each job J_j is executed within its executable time interval $[r_j, d_j]$. We consider the set of all job release dates and deadlines: $\{r_1, \ldots, r_n, d_1, \ldots, d_n\}$. We define an *epochal time* as a time value at which one or more points in this set occur; there are between 2 (when all jobs are released at the same date and have the same deadline) and $2n$ (when all job release dates and deadlines are distinct) such values. When ordered in absolute time, adjacent epochal times define a set of *time intervals*. We denote each time interval I_t by $I_t = [\inf I_t, \sup I_t[$. Finally, we denote by $\alpha_{i,j}^{(t)}$ the fraction of job J_j processed by machine M_i during the time interval I_t. In this framework, System (6.1) lists the constraints that should hold true in any valid schedule:

1. *release date*: job J_j cannot be processed before it is released (Equation (6.1a));

2. *deadline*: job J_j cannot be processed after its deadline (Equation (6.1b));

3. *resource usage*: during a time interval, a machine cannot be used longer than the duration of this time interval (Equation (6.1c));

4. *job completion*: each job must be processed to completion (Equation (6.1d)).

$$\begin{cases} \text{(6.1a)} & \forall i, \forall j, \forall t, \quad r_j \geqslant \sup I_t \Rightarrow \alpha_{i,j}^{(t)} = 0 \\ \text{(6.1b)} & \forall i, \forall j, \forall t, \quad d_j \leqslant \inf I_t \Rightarrow \alpha_{i,j}^{(t)} = 0 \\ \text{(6.1c)} & \forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)}.p_{i,j} \leqslant \sup I_t - \inf I_t \\ \text{(6.1d)} & \forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1 \end{cases} \tag{6.1}$$

Lemma 6.2. *System (6.1) has a solution if, and only if, there exists a solution to the deadline scheduling problem.*

System (6.1) can be solved in polynomial time by any linear solver system as all its variables are rational. Building a valid schedule from any solution of System (6.1) is straightforward as for any time interval I_t, and on any machine M_i, the job fractions $\alpha_{i,j}^{(t)}$ can be scheduled in any order.

Solving on a Range

One may think that by applying a binary search on possible values of the objective value $\mathcal{F}$, one would be able to find the optimal maximum weighted flow, and an optimal schedule. However, a binary search on rational values will not terminate. By setting a limit on the precision of the binary search, the number of process iterations is bounded, and the quality of the approximation can be guaranteed. However, as we now show, we can adapt our search to always find the optimal in polynomial time.

So far we have used System (6.1) to check whether our problem has a solution whose maximum weighted flow is no greater than some objective value $\mathcal{F}$. We now show that we can use it to check whether our problem has a solution for some particular *range* of objective values. Later we show how to divide the whole search space into a polynomial number of search ranges.

First, let us suppose there exist two values $\mathcal{F}_1$ and $\mathcal{F}_2$, $\mathcal{F}_1 < \mathcal{F}_2$, such that the relative order of the release dates and deadlines, $r_1, \ldots, r_n, d_1(\mathcal{F}), \ldots, d_n(\mathcal{F})$, when ordered in absolute time, is independent of the value of $\mathcal{F} \in]\mathcal{F}_1; \mathcal{F}_2[$. Then, on the objective interval $]\mathcal{F}_1, \mathcal{F}_2[$, as before, we define an epochal time as a time value at which one or more points in the set $\{r_1, \ldots, r_n, d_1(\mathcal{F}), \ldots, d_n(\mathcal{F})\}$ occurs. Note that an epochal time which corresponds to a deadline is no longer a constant but an affine function in $\mathcal{F}$. As previously, when ordered in absolute time, adjacent epochal times define a set of *time intervals*, that we denote by $I_1, \ldots, I_{n_{\text{int}}(\mathcal{F})}$. The durations of time intervals are now affine functions in $\mathcal{F}$. Using these new definitions and notations, we can solve our problem on the objective interval $[\mathcal{F}_1, \mathcal{F}_2]$ using System (6.1) with the additional constraint that $\mathcal{F}$ belongs to $[\mathcal{F}_1, \mathcal{F}_2]$ ($\mathcal{F}_1 \leqslant \mathcal{F} \leqslant \mathcal{F}_2$), and with the minimization of $\mathcal{F}$ as the objective. This gives us System (6.2).

$$
\begin{array}{l}
\text{MAXIMIZE } \mathcal{F}, \\
\text{UNDER THE CONSTRAINTS} \\
\left\{
\begin{array}{lll}
\text{(6.2a)} & \forall i, \forall j, \forall t, & r_j \geqslant \sup I_t \Rightarrow \alpha_{i,j}^{(t)} = 0 \\
\text{(6.2b)} & \forall i, \forall j, \forall t, & d_j \leqslant \inf I_t \Rightarrow \alpha_{i,j}^{(t)} = 0 \\
\text{(6.2c)} & \forall t, \forall i, & \displaystyle\sum_j \alpha_{i,j}^{(t)} . p_{i,j} \leqslant \sup I_t - \inf I_t \\
\text{(6.2d)} & \forall j, & \displaystyle\sum_t \sum_i \alpha_{i,j}^{(t)} = 1 \\
\text{(6.2e)} & \mathcal{F}_1 \leqslant \mathcal{F} \leqslant \mathcal{F}_2 &
\end{array}
\right.
\end{array}
\tag{6.2}
$$

The relative ordering of the release dates and deadlines only changes for values of $\mathcal{F}$ where one deadline coincides with a release date or with another deadline. We call such a value of $\mathcal{F}$ a *milestone*[1]. In our problem, there are at most n distinct release dates and as many distinct deadlines. Thus, there are at most $\frac{n(n-1)}{2}$ milestones at which a deadline function coincides with a release date. There are also at most $\frac{n(n-1)}{2}$ milestones at which two deadline functions coincides (two affine functions intersect in at most one point). Let n_{q} be the number of distinct milestones. Then, $1 \leqslant n_{\text{q}} \leqslant n^2 - n$. We denote by $\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_{n_{\text{q}}}$ the milestones ordered by increasing values. To solve our problem we just need to perform a binary search on the set of milestones $\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_{n_{\text{q}}}$, each time checking whether System (6.2) has a solution in the objective interval $[\mathcal{F}_i, \mathcal{F}_{i+1}]$ (except for $i = n_{\text{q}}$ in which case we search for a solution in the range $[\mathcal{F}_{n_{\text{q}}}, +\infty[$). There is a polynomial number of milestones and System (6.2) can be solved in polynomial time. Therefore:

Theorem 6.2 ([42]). *The problem of minimizing the maximum weighted flow time in the divisible load model $\langle R|r_j; div|\max w_j F_j\rangle$ can be solved in polynomial time.*

[1] Labetoulle, Lawler, Lenstra, and Rinnooy Kan [LLLR84] call such a value a "critical trial value".

Online Scheduling: Competitiveness Bounds and Competitive Algorithms

Unlike the previous results on sum-stretch, offline max-stretch minimization is thus a relatively easy problem. In the online counterpart, the slightest mistake can be very harmful:

Theorem 6.3 ([40]). *There is no $\frac{1}{2}\Delta^{\sqrt{2}-1}$-competitive preemptive online algorithm minimizing max-stretch if we restrict to instance with at least three different processing times.*

This result is an improvement from the bound of $\frac{1}{2}\Delta^{\frac{1}{3}}$ established by Bender, Chakrabarti, and Muthukrishnan [BCM98]. In fact, we establish this new bound by doing a more precise analysis of the exact same adversary. In their proof, Bender, Chakrabarti, and Muthukrishnan implicitly assumed that the algorithm knew in advance the ratio Δ of the sizes of the largest and shortest jobs. We have roughly bridged half of the gap between the previous lower bound and the best existing algorithms, which are $O(\sqrt{\Delta})$-competitive algorithms and, which we now present.

A first approach that consists in rounding job sizes into two categories was proposed by Bender, Muthukrishnan, and Rajaraman [BMR02]. They define, for any job J_j, a pseudo-stretch $\widehat{\mathcal{S}}_j(t)$:

$$\widehat{\mathcal{S}}_j(t) = \begin{cases} \frac{t-r_j}{\sqrt{\Delta}} & \text{if } 1 \leqslant p_j \leqslant \sqrt{\Delta}, \\ \frac{t-r_j}{\Delta} & \text{if } \sqrt{\Delta} < p_j \leqslant \Delta. \end{cases}$$

Then, they schedule the jobs by decreasing pseudo-stretches, potentially preempting running jobs each time a new job arrives in the system. They demonstrated that this method is a $O(\sqrt{\Delta})$-competitive online algorithm. Although this approach is very simple, it is rather crude and often reaches its bound in practice.

Bender, Chakrabarti, and Muthukrishnan [BCM98] had previously described another $O(\sqrt{\Delta})$-competitive online algorithm for max-stretch. This algorithm works as follows: each time a new job arrives, the currently running job is preempted. Then, they compute the optimal (offline) max-stretch $\mathcal{S}^*$ of all jobs having arrived up to the current time. Next, a deadline is computed for each job J_j: $d_j(\mathcal{F}) = r_j + \alpha \times \mathcal{S}^*/p_j$ Finally, a schedule is realized by executing jobs according to their deadlines, using the *Earliest Deadline First* strategy. To optimize their competitive ratio, Bender *et al.* set their *expansion* factor α to $\sqrt{\Delta}$. For both heuristics, the ratio Δ of the sizes of the largest and shortest jobs submitted to the system is thus assumed to be known in advance.

This is a very nice result as it shows how to build an online algorithm from an offline algorithm and how some of the optimality properties can be transferred from one setting to another. From our point of view, there are however three practical problems with this approach:

1. To obtain the competitiveness bound, the algorithm needs to recompute the optimal offline max-stretch $\mathcal{S}^*$ of **all** jobs having arrived up to the current time. When the system is loaded, recomputing this value becomes more and more expensive, which is all the more harmful as it should be done every time a new job enters the system.

2. Such an algorithm tries only to optimize the stretch of the most constraining jobs. Nothing particular is done for the second max-stretch, the third max-stretch, and so on. This is particularly true with the offline version and although the online version certainly reduces such effect, this means there is room for improvement. This problem is common to all algorithms minimizing a *max* objective. Indeed, such an algorithm could schedule all jobs so that their stretch is equal to the objective, even if most of them could have been scheduled to achieve far lower stretches. Just like that max-min fairness is *recursively* defined, minimization of max stretch should also be defined to produce Pareto-optimal schedules. This problem is far from being merely theoretical and is obviously not specific to stretch minimization. For example, the technique of Shmoys, Wein and Williamson [SWW91] allows to build 2ϱ-competitive polynomial-time online clairvoyant algorithms for $\langle P|\{|s\rangle$ ize$_j$,r_j $\}\{C_{\max}$ $\}$ from a ϱ-approximation for $\langle P|size_j|C_{\max}\rangle$ by scheduling jobs in batches. Yet, in practice and despite the guarantee, backfilling is always used to save all the idle time

wasted by the batch structure. Although we proved [9] that $\langle R|r_j; div|\max w_j F_j Pareto\rangle$ is NP-hard, in practice the combinatorial issue raised by tie-breaking is rare and it is sufficient to recursively optimize the maximum stretch of jobs.

3. Giving some slack is a sound idea in a online setting but it has been quite difficult to estimate its impact compared to a purely greedy strategy. In our simulations, we could not see any difference, which could have two possible explanations: 1) the random instances we tried are not complicated enough to trick a simple online greedy algorithm; 2) as the divisible setting does not incur any preemption cost, a sub-optimal decision is rarely very constraining.

These different observations motivated a pragmatic algorithm, which does not have any theoretical guarantee but largely outperforms other heuristics in simulation.

6.3 A Pragmatic Scheduling Algorithm

The basic online heuristic we proposed is along the same line as the algorithm of Bender, Chakrabarti, and Muthukrishnan [BCM98]: each time a new job arrives we preempt the running job (if any), compute the optimal max-stretch, and schedule the jobs according to the solution of System 6.2. The solution of System 6.2 specifies what fraction of each job should be executed on each processor during each time interval.

Our first modification to this scheme is that, rather than computing the "optimal max-stretch", we compute the "best achievable max-stretch considering the decisions already made". In other words, we take into account our knowledge of which jobs were already (partially) executed, and when. The underlying idea being that we cannot change the past anyway. Also, such an optimization greatly simplifies the linear system. This modification is implemented by making trivial modifications to System 6.2.

Our second modification to the above scheme is more important: we want to optimize more than the max-stretch. The first possibility would be to use in an online framework our offline recursive heuristic for the Pareto minimization of max-stretch of available jobs. However, when the workload becomes important, the recursive solving becomes quickly cumbersome. So instead, we tried to schedule each job in a manner that minimizes its own stretch value, while maintaining the overall maximal stretch value obtained. For example, one could theoretically try to minimize the sum-stretch under the condition that the max-stretch be optimal. However, as we have seen, minimizing the sum-stretch is an open problem. So we consider a heuristic approach based on a relaxation and expressed by System(6.3).

$$\text{MAXIMIZE} \sum_{j=1}^{n} \omega_j \left(\left(\sum_t \left(\sum_{i=1}^{m} \alpha_{i,j}^{(t)} \right) \frac{\sup I_t(\mathcal{S}^*) + \inf I_t(\mathcal{S}^*)}{2} \right) - r_j \right),$$

UNDER THE CONSTRAINTS

$$\begin{cases} \text{(6.3a)} & \forall i, \forall j, \forall t, \quad r_j \geqslant \sup I_t(\mathcal{S}^*) \Rightarrow \alpha_{i,j}^{(t)} = 0 \\ \text{(6.3b)} & \forall i, \forall j, \forall t, \quad d_j(\mathcal{S}^*) \leqslant \inf I_t(\mathcal{S}^*) \Rightarrow \alpha_{i,j}^{(t)} = 0 \\ \text{(6.3c)} & \forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)}.p_{i,j} \leqslant \sup I_t(\mathcal{S}^*) - \inf I_t(\mathcal{S}^*) \\ \text{(6.3d)} & \forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1 \end{cases} \tag{6.3}$$

This system ensures that each job is completed no later than the deadline defined by the optimal (offline) max-stretch $\mathcal{S}^*$. Then, under this constraint, this system attempts to minimize an objective that resembles a rational relaxation of the sum-stretch (or more generally of the sum weighted flow) using as an approximation of the completion time, the weighted sum of the average execution times of a job. As we do not know the precise time within an interval when a part

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
Offline	1.0000	0.0000	1.0000	1.4051	0.2784	2.6685
OfflinePareto	1.0000	0.0000	1.0000	1.2986	0.2605	3.5090
Online-EGDF	**1.0331**	**0.0622**	**1.6613**	**1.0024**	**0.0052**	**1.1095**
Bender98[2]	1.0415	0.0971	2.1521	1.0028	0.0075	1.1393
Bender02	2.9859	2.7071	23.5446	1.2049	0.3087	6.6820
SWRPT	1.0386	0.0729	2.0566	1.0003	0.0014	1.0384
SRPT	1.0596	0.1027	2.1012	1.0048	0.0074	1.1179
SPT	1.0576	0.1032	2.1297	1.0020	0.0048	1.1263
FCFS-Div	5.1353	6.6792	65.9073	1.3767	0.7224	15.4213
MCT	38.4276	24.2626	156.3778	51.9606	36.5202	154.1519
RAND	4.6568	6.9107	87.9141	1.2355	0.4827	10.8549

Table 6.1: Aggregate statistics over all 162 platform/application configurations.

of a job will be scheduled, we approximate it by the mean time of the interval. (This heuristic obviously offers no guarantee on the sum-stretch achieved.)

Finally, the (active) jobs are processed under a list scheduling policy, using the strategy outlined in Section 6.2.2 to deal with restricted availabilities: while some processors are idle, one selects the job with the highest priority and distribute its processing on all available processors that are capable of processing it. Jobs are totally ordered by the interval in which their *total work* is completed, with ties being broken by the *SWRPT* policy. We call such policy *Online-EGDF* and show its effectiveness in the next section.

Evaluation through simulation

To evaluate the efficacy of various scheduling strategies when optimizing stretch-based metrics, we implemented a simulator using the SIMGRID toolkit (details on experimental setup and results are reported in [LSV06]), based on the biological sequence comparison scenario. Our primary goal was to evaluate the proposed heuristics in realistic conditions that include partial replication of target sequence databases across the available computing resources. The simulation campaign involves 162 configurations comprising 3, 10 or 20 clusters, 3, 10 or 20 distinct databases whose availability was 30%, 60%, and 90%, and with different workload density. For each configuration, 200 platforms and application instances are randomly generated and the simulation results for each of the studied heuristics is recorded. Table 6.1 presents the aggregate results from these simulations; finer-grained results based on various partitionings of the data may be found in [LSV06].

Above all, we note that the *MCT* heuristic ("minimum completion time"), which was the policy used in the GriPPS system and simply schedules each job as it arrives on the processor that offers the best job completion time, without exploiting divisibility is unquestionably inappropriate for max-stretch optimization: *MCT* was over 38 times worse on average than the best heuristic. Its deficiency might arguably be tolerable on small platforms but, in fact, *MCT* yielded max-stretch performance over 16 times worse than the best heuristic in all simulation configurations. Even after addressing the primary limitation that the divisibility property is not utilized, the results are still disappointing: *FCFS-Div*, which employs all resources that are able to execute the job, is on average 5.1 times worse in terms of max-stretch than the best approach we found. One of the principal failings of the *MCT* and *FCFS-Div* heuristics is that they are non-preemptive. By forcing a small task that arrives in a heavily loaded system to wait, non-preemptive schedulers cause such a task to be inordinately stretched relative to large tasks that are already running.

We also observe that *SWRPT*, *SRPT*, and *SPT* are all quite effective at sum-stretch optimization. Each is on average within $5‰$ of the best observed sum-stretch for all configurations. In

[2] *Bender98* results are limited to 3-cluster platforms, due to prohibitive overhead costs.

particular, *SWRPT* produces a sum-stretch that is on average $0.3‰$ within the best observed sum-stretch, and attaining a sum-stretch within 4% of the best sum-stretch in all of the roughly 32,000 instances. However, it should be noted that these heuristics *may* lead to starvation. Jobs may be delayed for an arbitrarily long time, particularly when a long series of small jobs is submitted sequentially (the $(n+1)^{th}$ job being released right after the termination of the nth job). Our analysis of the GriPPS application logs has revealed that such situations occur fairly often due to automated processes that submit jobs at regular intervals. By optimizing max-stretch in lieu of sum-stretch, the possibility of starvation is eliminated. Such issue is only mildly reflected by our evaluation as job inter-arrival times was modeled using a Poisson process. We think, this is the reason why *SWRPT*, *SRPT* and *SPT* do not perform that bad regarding max-stretch.

Experimentally, we find that our online heuristic (*Online-EGDF*) is consistently near-optimal (within 4% on average) for max-stretch optimization and actually achieves consistently good sum-stretch (within $3‰$ of the best observed sum-stretch). Furthermore, our online heuristic has far better sum-stretch than the *OfflinePareto* (which is on average almost 30% away of the best observed sum-stretch). This result validates our heuristic optimization of sum-stretch as expressed by Linear Program (6.3). As forecast, *OfflinePareto* has a significantly better average sum-stretch than *Offline*.

Next, we find that the *Bender98* and *Bender02* heuristics are not practically useful in our scheduling context. At first sight, it may seem that the *Bender98* heuristic has a performance similar to the one of our algorithm. Yet, the results shown in Table 6.1 for the *Bender98* heuristic comprise only 3-cluster platforms; simulations on larger platforms were practically unfeasible, due to the algorithm's prohibitive overhead costs. Indeed, for an n-task workload, the *Bender98* heuristic solves n optimal max-stretch problems, many of which are computationally equivalent to the full n-task optimal solution. In several cases the desired workload density required thousands of tasks, rendering the *Bender98* algorithm intractable. To roughly compare the overhead costs of the various heuristics, we ran a small series of simulations using only 3-cluster platforms. The results of these tests indicate that the scheduling time for a 15-minute workload was on average under 0.28 s for our online heuristic, and 0.54 s for the offline optimal algorithm (with 0.35 s spent in the resolution of the linear program and 0.19 s spent in the online phases of the scheduler); by contrast, the average time spent in the *Bender98* scheduler was 19.76s. The scheduling overhead of *Bender02* is far less costly (on average 0.23 s of scheduling time in our overhead experiments), but in realistic scenarios for our application domain, the competitive ratios it guarantees are ineffective compared with our online heuristics for max-stretch optimization.

Finally, we remark that the *RAND* heuristic is slightly better than the *FCFS-Div* for both metrics. Moreover, *RAND* is only 24% away from the best observed sum-stretch on average. This leads us to think that the sum-stretch may not be a discriminating objective for our problem. Indeed, it looks as if, whatever the policy, any list-scheduling heuristic delivers good performance for this metric.

6.4 Conclusion and Open Issues

Motivated by an applicative context that can be modeled as a divisible load scheduling problem with release dates, we have revisited many results from the literature on offline and online scheduling with preemption. Doing so, we have mapped the whole landscape and filled a few holes (e.g., improved a few inapproximability bounds and proved a few NP-completeness results as can be found in Table 6.2). Some, like the optimization of sum stretch remained open despite our efforts in either direction. But more importantly, we have extracted some important techniques and intuitions, which we have sometimes extended. This has enabled us to propose a pragmatic heuristic that performs much better than already existing algorithms in practice.

In term of modeling, this study allowed us to realize that sum stretch and sum flow are not good objectives when several users are involved. First, as we mentioned earlier, these objectives are relatively easy to optimize and do not allow to really discriminate between various heuristics. Second, optimizing such metric does not allow to prevent starvation between users, which I think

	$\beta = \emptyset$	$\beta = pmtn$	$\beta = div$
$\langle 1\|r_j;\beta\|\max w_jF_j\rangle$	NP([BCM98])	$\downarrow$	$\downarrow$
$\langle P\|r_j;\beta\|\max w_jF_j\rangle$	$\uparrow$	$\downarrow$	$\downarrow$
$\langle Q\|r_j;\beta\|\max w_jF_j\rangle$	$\uparrow$	$\downarrow$ (network flow [LLLR84])	$\downarrow$
$\langle R\|r_j;\beta\|\max w_jF_j\rangle$	$\uparrow$	**P(Lin. Prog. [9])**	**P(Lin. Prog. Sec. 6.2.5)**
$\langle 1\|r_j;\beta\|\sum F_j\rangle$	NP([LRB77])	P(SRPT [Bak74])	$\downarrow$
$\langle P\|r_j;\beta\|\sum F_j\rangle$	$\uparrow$	NP(Numerical-3DM [BBC+07])	$\downarrow$
$\langle Q\|r_j;\beta\|\sum F_j\rangle$	$\uparrow$	$\uparrow$	**P(SRPT + Sec. 6.2.2)**
$\langle R\|r_j;\beta\|\sum F_j\rangle$	$\uparrow$	$\uparrow$	**NP(3DM, [9])**
$\langle 1\|r_j;\beta\|\sum S_j\rangle$	**NP([9])**	open	open
$\langle P\|r_j;\beta\|\sum S_j\rangle$	$\uparrow$	open	open
$\langle Q\|r_j;\beta\|\sum S_j\rangle$	$\uparrow$	open	open
$\langle R\|r_j;\beta\|\sum S_j\rangle$	$\uparrow$	open	**NP(3DM, [9])**
$\langle 1\|r_j;\beta\|\sum w_jF_j\rangle$	NP([LRB77])	NP(Numerical-3DM [LLLR84])	$\nwarrow$
$\langle P\|r_j;\beta\|\sum w_jF_j\rangle$	$\uparrow$	$\uparrow$	$\uparrow$
$\langle Q\|r_j;\beta\|\sum w_jF_j\rangle$	$\uparrow$	$\uparrow$	$\uparrow$
$\langle R\|r_j;\beta\|\sum w_jF_j\rangle$	$\uparrow$	$\uparrow$	$\uparrow$

Table 6.2: Summary of complexity results on scheduling with release dates (contributions in bold).

should be somehow guaranteed although this notion of starvation is rather difficult to grasp.

Indeed, in every non-competitiveness proof we built, we build adversaries that force starvation to occur by having a user that saturates the system by sending tasks regularly as if he was alone. In stochastic scheduling and queuing theory, this kind of situation is avoided as the system is then unstable. Yet, our analysis of the GriPPS application logs revealed us that such situations occur often due to automated processes that submit jobs at regular intervals. However, a deeper investigation reveals that rather than regular intervals submission, one should talk about automated interactive submission. Indeed, such behavior corresponds to a robot that resubmits a job as soon as the previous one is terminated. This is why we did not evaluate our strategies by replaying traces from the GriPPS system as we quickly realized it did not make any sense. Instead, we decided to use a simple Poisson process for job submission. A better approach would probably have been to model users by using a think time between job submission but characterizing an existing system was then beyond our skills. In practice, input workload heavily depends on the performance of the system, which complicates even further the comparison of different alternatives. Yet, I think revisiting our study with a better workload would allow to discriminate better between our *Online-EGDF* heuristic and a simpler heuristic such as *SWRPT*.

Finally, it should be noted that although max stretch optimization has the interesting property that it prevents starvation, such modeling is still not satisfying to my eyes. Indeed, such approach focus on minimizing the stretch of *every individual jobs*. Therefore, this *job-centric* approach still lacks a notion of user and is not truthful at all. If a user decides to split one of his job in half instead of submitting it as a whole, it would be considered as two independent smaller (and thus more critical) jobs and would thus gain a higher resource share. Fairness should thus be enforced at the user level and not at the job level, which calls for a two-level aggregation. When defining such aggregation, one should consider the following intuitions:

- Sum-based aggregation favors small jobs and easily lead to starvation of other jobs when a stream of small jobs is constantly submitted. However, if there is only one user, such problem would not really occur as the user would only harm himself and would thus self-regulate. Such criteria is thus rather meaningful from the perspective of a single user.

- The max part of the criteria allows to ensure that starvation between jobs does not occur and should be among users rather than among jobs. A reasonable and strict criteria would

thus be:

$$\max_u \sum_j S_j^{(u)}$$

This would be a "min-max" fair optimization of average stretch of users. If we link such proposal with more classical fairness criteria, we should probably optimize something like[3]:

$$\prod_u \sum_j S_j^{(u)}$$

This would correspond to the Nash Bargaining Solution. It turns out that Agnetis *et al.* [AMPP04, AdPP09, Ale09] have worked on such problem roughly at the same time as we did and that I only relatively recently discovered their work. Although the corresponding problems are NP-hard, they can be efficiently approximated. Using Lagrangian optimization and the properties of Smith's ratio rule, they propose a clever polynomial algorithm that computes a solution which is always within a few percents of the optimal solution in their experiments. Note that their results are for the setting with no release date (and hence no preemption). Yet, I think the technique would apply to the preemptive setting as well (e.g., using *SWRPT* instead of *SPT* rule). The main technical issue I see with such approach in view of a practical implementation is that the complexity of these algorithm tends to grow rather fast with the number of users.

All these observations on the fact that users adapt their workload and strategy to the performance of the system, and that per-user metric should be considered instead, motivated the study we present in the next chapter and where we try to characterize the Nash equilibrium of a system where some users aim at selfishly optimizing their average response time.

[3]We ommit for simplicity the disagreement point but we think a reasonable one would be the outcome of a fair sharing schedule, which is very easy to implement and to compute but is however Pareto inefficient.

Chapter 7

Non-Cooperative Throughput and Response Time Optimization

This chapter builds on an article published in CCGrid'11 [28] with Bruno Donassolo and Claudio Geyer. It was conducted in the context of the DOCCA and USS-SimGrid projects as well as thanks to our collaboration with Bruno Donassolo and Claudio Geyer in the context of the Grenoble-Porto Alegre associated team and with David Anderson in the context of the INRIA Grenoble-Berkeley associated team. It was motivated by the wish to analyze a real system and to see whether our knowledge in game theory and scheduling would help understanding it better and possibly improve it.

This work is a natural continuation of the questions raised in the previous chapters as it addresses a situation where:

1. some users aim at optimizing their throughput while some other users aim at optimizing their average response-time;
2. we assume each user optimizes its own utility while a fair "myopic" resource sharing is globally enforced as in Chapter 3;
3. we investigate the idea that some users may want to regulate their resource usage because they worry not only about their immediate utility but also about their "public image".

7.1 The BOINC project

7.1.1 History and Evolution of the BOINC Workload

Volunteer Computing is a kind of distributed computing platform on which clients donate their idle resources to projects. VC became famous thanks to SETI@home project [ACK+02], which started in 1999, searching for extraterrestrial intelligence. Later, SETI@home evolved and became the open-source BOINC (Berkeley Open Infrastructure for Network Computing) project [BOI]. Nowadays, BOINC harnesses more than 580,000 hosts that deliver over 2,300 TeraFLOP per day. Several projects have been deployed, such as ClimatePrediction.net, Einstein@home or the World Community Grid. Each project has its own server which is responsible for distributing work units to clients, recovering results and validating them. Work units run on clients' machines according to specific rules defined by the clients.

Salient characteristics of such systems are: scalability, heterogeneity, volatility, unpredictability and unreliability. Therefore, typical VC workloads are made of a large (i.e., orders of magnitude larger than the number of available hosts) sets of independent CPU-bound tasks and most projects aim at maximizing their throughput. Supporting new kinds of applications on VC platform is very challenging. A new promising class seems to be the case where applications have a relatively small number of work units in infrequent time intervals. As these projects do not have

large amount of tasks, they are interested in getting tasks back as soon as possible. More precisely, when receiving a batch of tasks, such projects try to minimize the completion time of the last finishing task of the batch. New algorithms and techniques have been proposed to address this problem. Kondo *et al.* [KCC07] claim that rapid application turnaround can be achieved through resource selection, resource prioritization, and replication. More recently, Heien *et al.* [HAH09] proposed to tune the connection interval parameter of BOINC projects to optimize the response time of batches. Last, the GridBot project [SSGS09, Sil11] recently made use of hybrid computing platform composed of grid, cluster, VC and cloud resources to execute workload resulting from mix of throughput and response time oriented applications. Yet, all projects rely on the same BOINC protocol and scheduling algorithms which we expose in the following two subsections.

BOINC relies on a classical client/server architecture. Each project has a specific server from which clients request work units to execute. Clients, during the install process for example, decide the projects that they want to crunch for. The behavior of clients and servers are described below.

7.1.2 The BOINC Server

The main activity of a BOINC server is to distribute jobs to clients. Upon client request it selects from a job list which tasks can run on the client. The server must take care of the system's constraints which could preclude the client from running these tasks. Due to the high resource volatility and unpredictability, the server is also responsible for keeping track of jobs and uses to this end a simple **deadline** mechanism. When work units are distributed to a client, they are associated a deadline before which the client should send the task back. Whenever a work unit is received on time, servers reward the client with credits. If a client takes too much time to execute a task and misses its deadline, no credit is granted to him.

In this work, we consider two kinds of projects:

- **Continuous projects**: Such projects have an extremely large number of tasks and are thus interested in throughput, i.e., the average number of tasks done per day. Most existing BOINC projects actually fall in this category.
- **Burst projects**: Unlike the previous ones, these projects receive batches of tasks (or Bags of Tasks) and are interested in the average response time of these batches.

In order to achieve a better performance, the server may use some strategies, such as replication, deadline or specific scheduling algorithms. Replication can be used to improve average response time, avoiding the last-finishing (a.k.a *straggler*) task issue [KCC07]. Replication has some variants, such as *homogeneous redundancy*, which replicates tasks only to hosts with the same characteristics (OS and CPU) to avoid potential result mismatch due to application numerical instabilities, and *adaptive replication*, which only replicates tasks if the host is not trustful [AR09]. The deadline, in the other hand, can be configured to keep a track of the tasks running on clients. Tighter deadlines implies in more interaction between client and server. Also, it can be used to give urgency to some tasks (last tasks in a batch) and so, get the results earlier. Also, servers may use some special strategy to select which tasks they will send to clients. In short, they are described below.

- **Fixed**: Server does not do any kind of test before sending tasks to clients.
- **Saturation**: Server receives the saturation date of client, i.e., the date when client finishes running all urgent tasks (running in EDF mode). Then, it verifies whether a task, starting at saturation date, will finish before its deadline. This is the most commonly used scheduling algorithm.
- **Earliest Deadline First (EDF)**: The most restrictive test does a detailed simulation of the scheduling of all tasks running on the client (the name comes from the fact that the client

uses an EDF scheduling algorithms when it is getting late). Then, it checks whether, when sending a new task, all already existing tasks (even from other projects) would not miss their deadline by more than they did previously.

BOINC's default configuration utilizes the saturation test. However, each project decides of activating or not these features according to its workload, objectives and clients' characteristics, such that it yields to the best possible behavior (e.g., throughput or response time improvement).

7.1.3 The BOINC Client

According to [AM07], the client scheduling policy has been designed with 3 main goals in mind:

- Maximize the amount of credit given to user;
- Enforce long-term fairness: client must work the same for each project if project shares are equal;
- Maximize variety: client should avoid long periods working to same project.

So, the client tries to fairly share (instantaneously) the resource between projects with respect to their priorities. However, this instantaneously fair share incurs overhead which reduces client throughput and delays task completion. Therefore, BOINC clients implement a complex mix of short-term/long-term fairness scheduling algorithm. The exception is when a task is near to miss its deadline. In order to avoid deadline misses, the scheduling algorithm switches to EDF (Earliest Deadline First) mode and executes such tasks with higher priority. Consequently, continuous projects generally have loose deadlines, whereas burst projects should prefer tighter deadlines so that their tasks are executed in priority. It is important to notice that the long-term debt sharing mechanism prevents projects with tight deadline to always bypass other projects. When a client overworks for a project, it simply momentarily stops downloading new tasks from this project.

7.2 A Game Theoretic Modeling

A Volunteer Computing system such as BOINC is made of volunteers that offer resources to be shared among a set of projects. The underlying client scheduling mechanisms aims at ensuring a fair and efficient sharing of resources but several parameters may affect this sharing. In the following, we explain how this situation can be modeled through the use of game theory notions. We start by defining the main actors of such situations: volunteers and projects.

Definition 7.1 (Volunteer V_j)**.** A BOINC volunteer is characterized by the following parameters:

- a **peak performance** (in $MFLOP.s^{-1}$) indicating the amount of $MFLOP$ it can process per second when it is available.
- an **availability** trace, i.e., an ordered sequence of disjoint time intervals indicating when the volunteer machine is available. This trace is unknown from the volunteer scheduler, which also does not try to use past historical information about this trace to obtain a better schedule.
- the **project shares**, i.e., the list of projects the volunteer is ready to work for and which priorities he/she assigned to each project.

A collection of volunteers is denoted by V.

In our modeling, we consider the volunteers to be passive and to only provide resources. We will see later how their welfare can be accounted for but in a first approximation, they are considered to be completely passive.

We now define the main characteristics of BOINC projects (the modeling of the whole system is depicted in Fig. 7.1.

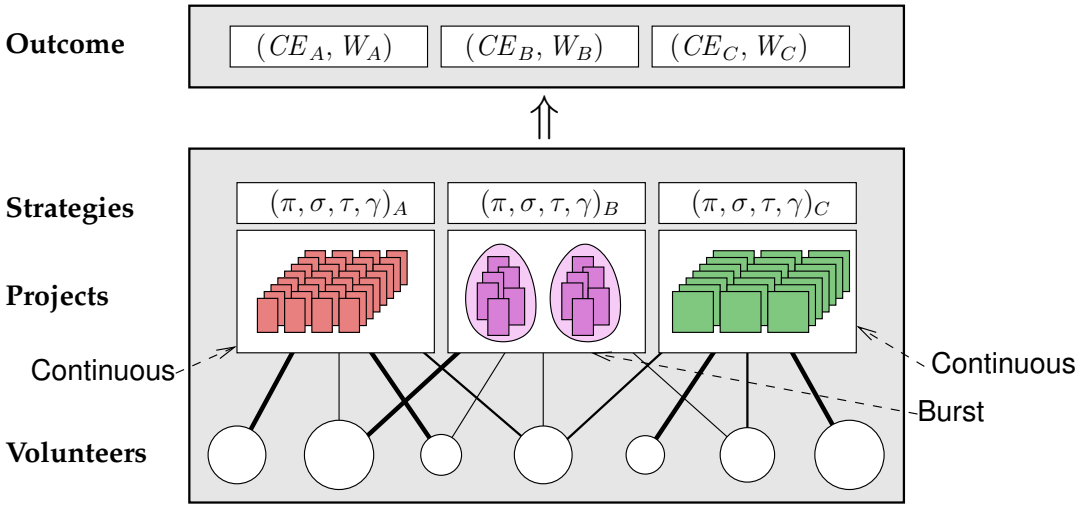


Figure 7.1: Modeling the whole BOINC system. Continuous and burst projects share resources according to the BOINC distributed sharing mechanism and to the project configuration parameters.

Definition 7.2 (Project $\boldsymbol{P_i}$)**.** A BOINC project is characterized by the following parameters:

- w_i $[MFLOP.task^{-1}]$ denotes the **size of a task**, i.e., the number of $MFLOP$ s required to perform a task. We assume that the size of the tasks of P_i is uniform. This is an approximation, but it generally holds true for many projects at the time scale of a few weeks.
- b_i $[task.batch^{-1}]$ denotes the **number of tasks within each batch**. Again, we assume that all batches of a given project have the same size. This is a rather strong assumption for projects like GridBOT [SSGS09] but we consider this as reasonable in a preliminary study such as the one we propose. Furthermore, since we are not interested yet in how a given project should prioritize its bursts depending on their size, this assumption should not really affect our conclusions.
- r_i $[batch.day^{-1}]$ denotes the **input rate**, i.e., the number of batches per day. Again, we assume this is fixed and we neglect the potential bursts and off-peak periods that may arise at the scale of the week or of the month. Also, we also assume that r_i, b_i and w_i are such that they do not fully saturate the system, i.e., such that a batch always ends before the submission of a new one. Indeed, as we previously explained, we are not interested yet in how a given project should prioritize its bursts. We only focus on how the different projects interfere with each others so the previous assumptions should not be harmful to this respect.
- Obj_i is the **objective function** of the project. Depending on the nature of the project, it could be either the **throughput** ϱ_i, i.e., the average number of $tasks$ processed per day, or the **average completion time of a** $batch$ α_i.
- q_i is the **quorum**, i.e., the number of successfully processed results that have to be returned before a task can be considered a valid. In our experiments, we assume that q_i is always equal to 1.

A project P_i is thus a tuple (w_i, b_i, r_i, Obj_i) and a project instance is thus a collection of projects $P = (P_1, \ldots, P_K)$. The set of all such possible project instances is denoted by $\mathcal{P}$.

Using the game theory terminology, projects will be referred to as **players**. Along the same lines, the term **strategy** is used to account for the set of options that players have and that may influence the sharing of the available resources.

Definition 7.3 (Strategy S_i)**.** The server of each project can be configured with specific values that directly influence the performance of the project. In our study, we focused on the following parameters:

- π_i is the task **work send policy** [KAM07] used by the server upon reception of a work request. When a client connects to the server, it asks for enough work to keep him busy for a period of time (e.g., one day). Then, it is the server that determines how many tasks to send. The simplest strategy $\pi_{cste=c}$ sends a fixed amount (c) of tasks, whatever the state of the client and of the server. In this simple strategy, c could be determined from the average task duration and the requested amount of work. More elaborate work send policies like saturation (π_{sat}) and EDF (π_{EDF}) have been introduced in Section 7.1.2.

- σ_i is the **slack** [KAM07]. This value is used to determine the deadlines assigned to tasks upon submission to clients. For example, if the average computation time of a task on a dedicated standard reference machine is one hour, then a fixed slack of 2 would result in a deadline of two hours. The simplest strategy $\sigma_{cste=s}$ means that a fixed slack of s is used. In practice, more elaborate strategies, like adapting the slack to volunteers speed/availability/reliability or to the progress of the batch, could be used but our preliminary study does not explore such possibilities.

- τ_i is the **connection interval** [HAH09] and indicates clients how often they should reconnect to the server. This parameter influences on how fast volunteers realize that new tasks need to be processed for a burst project. In our experiments, this parameter ranges from 12 minutes to 30 hours.

- γ_i is the **replication strategy** [KCC07]. This replication strategy is completely different from the replication used to reach a given quorum. γ_i is used to avoid straggler volunteers to delay the completion of a batch. Therefore, the strategy $\gamma_{cste=r}$ allows to submit at most r replicas of the same task and whose deadlines have not expired. In our experiments, this parameter ranges from 0 to 8. Again, smart strategies adapting to the reliability of volunteers and to the progress of the batch could be used but our preliminary study does not explore such possibilities.

Using game theory terminology, the *strategy* S_i of a project P_i is thus a tuple $(\pi_i, \sigma_i, \tau_i, \gamma_i)$ and the set of all possible strategies for all projects is denoted by $\mathcal{S}$.

Definition 7.4 (Outcome O)**.** Once a set of project P has decided a given strategy S, the resources of a given set of volunteers V are shared through the BOINC scheduling mechanisms.
The outcome $O(V, P, S)$ is the set of all information about completion of tasks and batches from different projects. In the following, when needed, we will denote by O_i the restriction of O to information related to P_i.
From this outcome, we can compute the following values:

- **Throughput** of continuous projects. If P_i is a continuous project, then we can compute the total number of tasks from P_i that have been processed over a given time period.

- **Average batch completion time**. If P_i is a burst project, then we can sum the time needed to complete each batch (from the arrival of the batch in the system to the completion of the last finishing task of the batch) and average it over the total number of batches that have been submitted over a given time period.

- **Waste**. Sometimes, tasks fail to be completed before their deadlines. This can happen because the volunteer's machine was too slow, or because it went unavailable for a long time, or maybe even because the slack of the project was too tight. In such cases, the task is often resubmitted and the client may not get reward for it. The time spent working on missed tasks is thus wasted both from the volunteer and project perspectives.

For a given project P_i and a given volunteer V_j, we denote the waste by $W_{i,j}$, the ratio of tasks from P_i missed by V_j over the total number of tasks he/she received from P_i.

Similarly the waste W_i of P_i denotes the ratio between total number of tasks from P_i missed by volunteers over the total number of tasks.

From a given outcome, we need to define the satisfaction (or **utility** using the game theory terminology) of each player. In our context, depending on the nature of the project, the satisfaction is based either on the effective throughput or on the average batch completion time. Yet, these two metrics do not express in the same units at all. One of them is to be maximized (the throughput) whereas the other one is to be minimized (the average batch completion time). Therefore, we propose to translate these two metrics into a common one: the **cluster equivalence** metric. The cluster equivalence metric was proposed in [KTB+04] in the context of throughput optimization and represents the number of dedicated standard reference machines that would be needed to achieve the same performance. This metric can thus be computed for both types of projects, only with a different formula.

Definition 7.5 (Cluster equivalence $\boldsymbol{CE}$)**.** We denote by W the performance (in $MFLOP.s^{-1}$) of the reference machine chosen to estimate the CE. Depending on the objective of project i, its cluster equivalence can thus be defined either as

$$CE_{continuous} = \frac{TasksDone \cdot w_i}{W \cdot TotalTime} \qquad \text{or as} \qquad CE_{burst} = \frac{b_i \cdot w_i}{W \cdot \alpha_i},$$

where α_i is the average batch response time of project P_i.

We can now define the utility of projects easily:

Definition 7.6 (Utility $\boldsymbol{U_i}$)**.** The utility of P_i is equal to its cluster equivalence:

$$U_i(V, P, S) = CE_{Obj_i}\big(Obj_i(O_i(V, P, S))\big)$$

It is thus very important to understand here that the utility of P_i depends on both its own strategy S_i, but also on the strategy choices made by the other projects.
In the following, $U(V, P, S)$ denotes the utility vector of all projects.

The utility of a project is thus what it should aim at maximizing. Yet, as it has been observed in the context of GridBOT [SSGS09], projects should also try to ensure that the waste they cause to other projects is not too large. Otherwise volunteers may consider this as selfish and decide to sign off from P_i. Therefore, even though we do not model such situations and consider the volunteers to be passive and do not put them in the decision loop, the waste (not only W_i but rather W as a whole) should be considered as a second objective to be minimized.

In the remaining of this work, we investigate the existence, the computation, and the efficiency of Nash equilibria. Even though the BOINC project administrators do not necessarily keep tuning the parameters of their project, they certainly monitor the outcome and we think that such equilibria can be considered as a good approximation of what would happen in reality.

7.3 Simulation Results

The outcome of such complex scheduling algorithms is extremely hard to put into equations and the outcome of the dynamic of thousands of volunteers running it is even more difficult to model. Therefore, we conducted a series of SIMGRID simulations to analyze the performance of many independent players sharing BOINC resources.

We implemented a simulator of the BOINC architecture, including both client and server components [31]. Based on reading of both articles describing BOINC and its freely available source code, we designed our simulator so as to take the main features of BOINC into account and thus perform simulations as realistic as possible.

Since our study is multi-parametric (deadline, connection interval, ...), multi-player (burst and continuous projects) and multi-objective (throughput and response time), we had to bound our parameter space and to perform a careful study of parameter influence on global system performance.

We arbitrarily decided to restrict our study to the following project configurations:

- **Burst projects** receive one batch per day, comprising only short-live jobs (a few hundreds to a thousand depending on the external load) which take about 1 hour running on a standard machine.
- **Continuous projects** have hundreds of thousands of CPU-bound jobs. We used a typical configuration from SETI@home project, with tasks of 30 hours[1]. The deadline of such tasks is set up to 300 hours as it is commonly accepted to provide good results [KAM07].

The duration of each simulation is equivalent to about 140 days. It is important to have a very long period to ensure the effectiveness of the long-term share of BOINC system.

The outcome of the sensibility analysis is summarized in Section 7.3.1 but a detailed version can be found in [28]. In particular, we explain the impact of burst projects on continuous projects Last, we explain in Section 7.3.2 how we restrict our parameter space and how we sample the utility set and search for Nash equilibria.

7.3.1 Sensibility Analysis

In such experiments, continuous projects are considered as passive since their parameters are fixed to sensible values for such projects (i.e., large slack and connection interval). Therefore, we started by checking the influence the parameters of a single burst project on the performance of the whole system:

- The connection interval had almost no influence at all, neither on waste nor on CE (cluster equivalence), so it was finally set to 2 hours in all simulations;
- The slack is a **critical** parameter. A loose slack leads to a very poor response time, which easily result in a CE 6 times lower than the one obtained with a tight slack. Such values should thus be disregarded by burst projects. Our simulations clearly show that the optimal choice for a burst project is to have deadlines close to the actual job completion time on a standard machine (around 1.1 hour for 1 hour jobs). However, too tight deadlines (≤ 1 hour for 1 hour jobs) result in important resource waste and in CE reduction for **all** projects. Carefully setting such parameter is thus critical.
- Fortunately, the optimal configuration of the burst project does not degrade significantly the CE of continuous projects, which augurs well for coexistence of such projects. Furthermore, although the waste for continuous projects remains low when burst projects select their optimal parameter set (less than 3% compared to an absolute minimum of 1.8% in our simulations), the waste of burst projects is extremely high (around 40–60%) when using the simple scheduling strategy π_{cste} where servers satisfy every request sent by clients, regardless of their potential ability to process or not the task in due date.
- Overall, there is little difference in term of CE (both for burst and continuous projects) when switching from π_{cste} to the saturation π_{sat} or EDF π_{EDF} scheduling strategy. Both elaborate scheduling strategies give every project roughly the same CE and a rather low waste (except for burst projects, which is about 12%). We think that π_{EDF} may reveal superior to π_{sat} in more complex situations but this did not appear in our simulations. The fixed π_{cste} strategy allows to obtain a slightly better CE, due to the high number of tasks sent, but incurs a very important waste (around 50% in the optimal configuration). Indeed, in most of our experiments, we observed that this naive strategy often thrashes the system even though

[1] See http://www.boinc-wiki.info/Catalog_of_BOINC_Powered_Projects

it is the most efficient way for burst projects to steal computing resources from continuous projects. Servers should thus disregard this scheduling strategy as it wastes resources and could disappoint clients, since they may not get credits for missed tasks. The important waste of this strategy was already identified by Kondo *et al.* [KAM07] but it had only be compared with the one of the EDF strategy which is considered to be too costly and is thus often not activated. The default π_{sat} used in most deployed BOINC projects seems thus a reasonable choice. Similar issue about unsatisfied users was also reported in [SSGS09].

- When we investigated the impact of speculative replication, in our simulations, allowing up to 2 replicas can improve performance of burst projects about 7% using 2 replicas while further replicas lead to a CE decrease. Unintuitively, such replication also decreases waste from 56% to 48% (with the $\pi_{cste=1}$ strategy).

- Finally, although it was already known [KAM07] that the fixed strategy lead to an unacceptable waste, it was not known (to the best of our knowledge) that it could also increase the waste of other projects. To illustrated this, we replaced continuous projects by burst projects using the fixed scheduling strategy. Although the waste of burst project remains stable (50-60%), using more burst projects also considerably increases the waste of continuous projects (from 3% to over 50%). This shows that the current protocol is unable to enforce fairness and project isolation when using burst projects. Burst projects may thus have an important influence on waste of others projects and hence can cause users dissatisfaction since they will miss more tasks and loose credits.

As a conclusion, the only reasonable configuration for a burst project is to use a replication factor around 2, a connection interval of 2 hours, the saturation π_{sat} scheduling strategy (to minimize waste) and to carefully tune its slack. Although there are definitely interactions between these different parameters, it remains extremely limited compared to the influence of the slack and this preliminary sensibility analysis provides sound justification for focusing on the simpler problem where each burst project only tunes its slack. To further ease the analysis, we assume in the next section that burst projects are identical and thus that, by symmetry, they will use the same strategy.

7.3.2 Utility Set Sampling and Nash Equilibrium

Beside the high waste of the burst project, the previous study seems to indicate that both kind of projects can seamlessly share resources. Yet, this nice behavior is largely due to the fact that the burst project does not exhaust available resources. It is thus always important to check that the input rate of burst projects remains smaller than what the share they receive allows them to process. Otherwise (i.e., when arrival rate, batch size, job duration, or the number of burst projects become too large), the system is saturated and the response time of burst projects artificially grows to infinity, which results in absurdly low CE. In the following experiments, we thus always make sure that the system is not saturated.

Our simulations showed us that the negative effect of burst projects may become more visible when resource become scarce (i.e., when the system is close to saturation). Hence, we first present a configuration with 6 projects among which 4 of them are burst projects receiving a batch of 900 1 hour tasks per day. In this configuration, continuous projects use 24h and 30h tasks with a deadline of 336h and 300h (these values are representative of the Einstein@home and SETI@home projects, respectively).

Although best-response strategy does not necessarily converge, it is known that upon convergence, its limit state is a Nash equilibrium. Furthermore, since in our particular setting, all our burst projects are identical, they should all have the same configuration at equilibrium. Therefore, we apply the following methodology. Assuming burst projects agree on a given value of slack σ_{cons}, we compute the CE_{cons} of burst projects (consensus value in Fig. 7.2). For a given value of slack σ_{cons}, a dissident burst project may increase its CE by unilaterally changing its

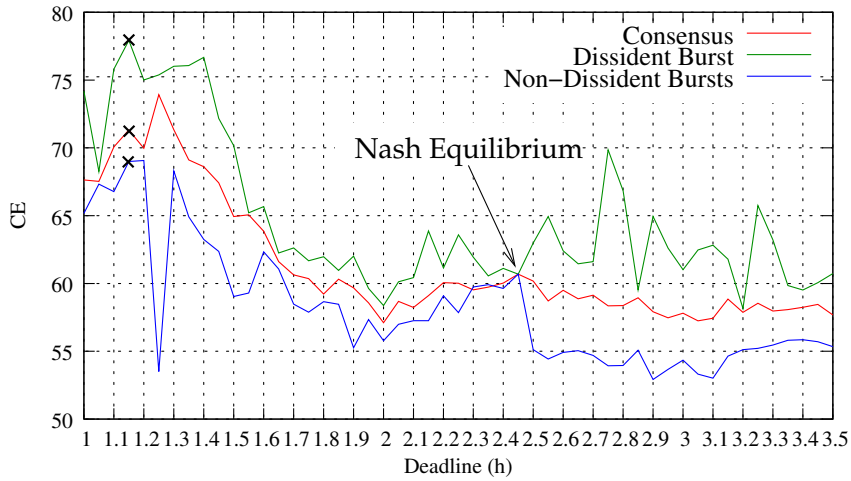


Figure 7.2: Illustrating the willingness to depart from a consensus. The consensus line represents the CE CE_{cons} of each burst project if they agree on a given deadline σ_{cons}. The dissident line represents how much a dissident can improve its CE_{dis} by selecting a different deadline σ_{dis}. The impact of such a strategy modification on the CE of other burst projects is depicted by the line $CE_{non-dis}$.

slack to a different value σ_{dis}. We compute this value so that it maximizes the CE_{dis} of the dissident burst project. Yet, this strategy modification generally results in a decrease of the $CE_{non-dis}$ of the other burst projects. Both CE_{dis} and $CE_{non-dis}$ are represented on Fig. 7.2. For example when $\sigma_{cons} = 1.15$ (i.e., a value close to the optimal one found in the previous section), we see that a burst project can change its CE from 71.3 to 77.5 by changing its slack, thus causing a CE decrease of 2 for every other burst projects. As a consequence, even though this deadline leads to the largest CE_{cons}, the other burst projects should disregard $\sigma_{cons} = 1.15$ and switch to a better position as well. Interestingly, the three curves join for $\sigma_i = 2.45$, which is thus a Nash equilibrium.

Fig. 7.3 depicts for a given σ_{cons}, the corresponding σ_{dis} that leads to the best CE_{dis}. Therefore, if all projects used a simplistic best response strategy, we can follow the dynamic of the system and see that it converges (when initial $\sigma_{init} \in [1.8, 2.45]$) to the Nash equilibrium $\sigma_{NE} = 2.45$, that leads to a CE_{NE} of 60.7 for burst projects. Yet, we can read on Fig. 7.2 that a common choice

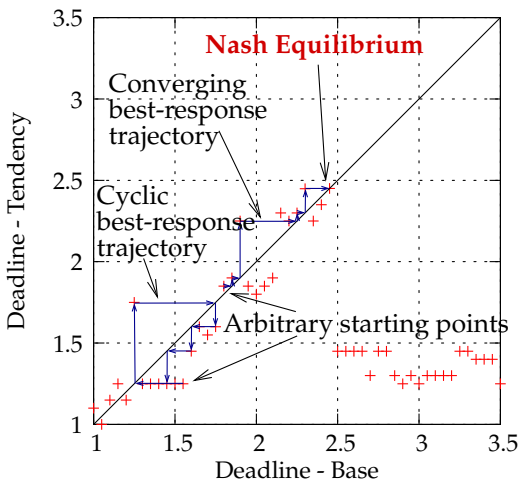


Figure 7.3: Best response dynamic: this plot depicts for a given σ_{cons}, the corresponding σ_{dis} to which burst projects tend to use. With such information, it is possible to depict the outcome of a best-response strategy.

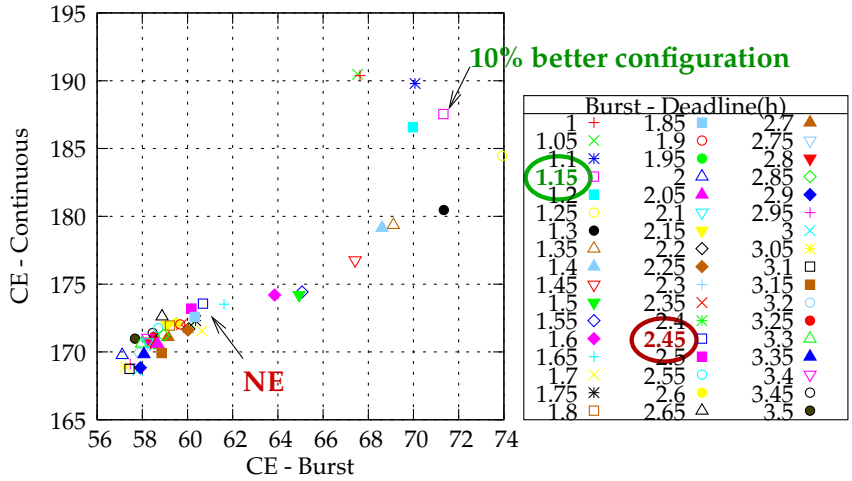


Figure 7.4: Sampling utility set and illustrating the inefficiency of the Nash equilibrium. 2 continuous projects and 4 burst project with 900 1 hour tasks per day each. *CE* could be *improved by more than 10% for all projects* by collaboratively choosing $\sigma_{cons} = 1.15$ instead of the NE ($\sigma_{NE} = 2.45$).

of $\sigma_{cons} = 1.15$ (reference point) would have lead to a $CE_{cons} = 71.3 > CE_{NE}$. Still, it could be the case that this loss of efficiency did benefit to the continuous projects. This is unfortunately not true as can easily be checked by representing the outcome of all such situations in the utility space (Fig. 7.4). With such a representation, we can check that the Nash equilibrium is indeed Pareto inefficient and that the *CE could be improved by more than 10% for all projects* by collaboratively choosing $\sigma_{cons} = 1.15$.

Such inefficiencies can be observed for many different configurations and the utility set samples seem to always have more or less the same structure. For example, when using a configuration with 8 projects among which 7 of them are burst projects receiving a batch of 600 tasks per day, we also obtain an inefficient Nash equilibrium (see Fig. 7.5). Note that in this example, the continuous project uses 30 hours tasks with a deadline of 300 hours, which is a typical configuration from the SETI@home project.

Again, with such a representation, we can check that the Nash equilibrium is indeed Pareto

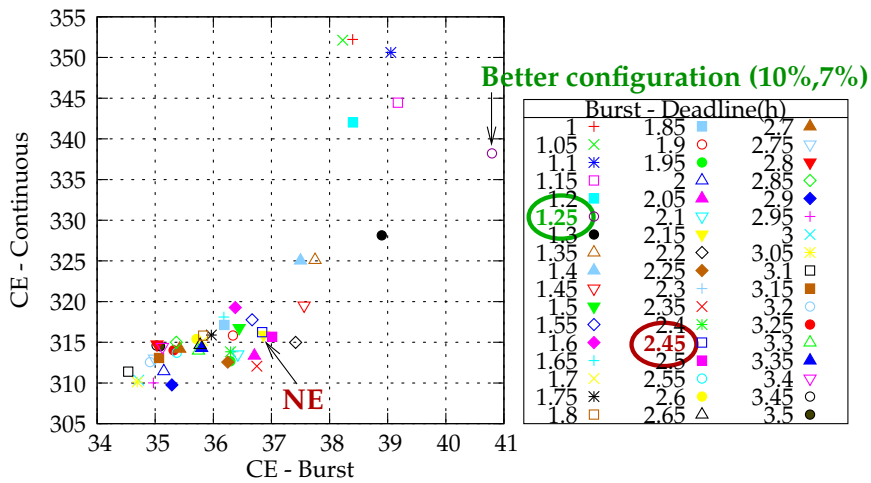


Figure 7.5: Sampling utility set and illustrating the inefficiency of the Nash equilibrium. 1 continuous project and 7 burst project with 600 1 hour tasks per day each. *CE* could be improved by more than 10% for burst projects and by 7% for the continuous project by collaboratively choosing $\sigma_{cons} = 1.25$ instead of the NE ($\sigma_{NE} = 2.45$).

inefficient and that the CE could be improved by 10% for burst projects and by 7% for continuous project by collaboratively choosing $\sigma_{cons} = 1.25$.

The main conclusion to draw from this set of experiments is that burst projects not only step on each others toes but also impact rather negatively the efficiency of continuous projects (not even speaking about their waste).

7.4 Conclusion and Open Issues

In this work, we have studied the situation where burst projects interested in response time share resources with classical throughput-oriented projects. We perform a game theoretic modeling and experimental analysis of such system, in order to verify the potential performance issues raised by such complex configurations. Such game theory concepts are rarely applied in such "complex" systems. Indeed, our modeling considers a very large strategy space, a rather large number of players, and a multi-objective optimization problem since project should not only optimize their throughput or their response time but also their waste (i.e., the fraction of tasks missing their deadline) and the waste they incur to others.

Our experimental analysis relies on thorough and realistic simulations and enables us to study the influence of the main server parameters (slack, replication, work sending strategy, ...) in a multi-player context. This study illustrates that the current scheduling mechanism is unable to enforce fairness and project isolation (burst projects may dramatically impact the performance of other projects).

In particular, we show that when such burst projects share volunteer machines with continuous projects, the non-cooperative optimization of their project configuration may result in rather inefficient sharing of resources. We exhibit situations where every project could use resources 10% more efficiently if burst projects agreed on some of their scheduling parameters. Even though this result should not surprise people acquainted with game theory, it is, to the best of our knowledge, the first time inefficient Nash equilibrium is exhibited in the context of Volunteer Computing systems. It is also interesting to note that these inefficiencies occur even though the whole system relies on a protocol where each volunteer produces a locally fair and efficient sharing of its computing resources. Such results can be linked with those presented in Chapter 3 except that we do not assume an over-simplistic resource model but use a very realistic simulation instead.

From a theoretical point of view, the existence of inefficient Nash equilibria may lead the system to even more undesirable behavior, like Braess's paradox where the addition of resources may degrade the performance of every project. Note that game theory provides many tools (correlated equilibria, pricing mechanisms, Shapley value ...) to cope with and improve such situations and that some of them may be used to implement a form of cooperation (possibly distributed) between servers.

Now, from a practical point of view, the results of this study are subject to the following reservations:

- **Inefficiency of Nash Equilibria**: We exhibit situations where every project could use resources 10% more efficiently if burst projects agreed on some of their scheduling parameters. One may wonder whether it would be possible to find situations that are even more inefficient than this. In our study, we simplified the analysis by assuming that burst projects are identical so that, by symmetry, they use the same strategy. I think a higher degree of heterogeneity would probably lead a higher inefficiencies. Also, note that these inefficient situations correspond to setups which are close to saturation. Practical situations where burst projects compete with continuous projects are not common and our simulation instanciation is thus somehow arbitrary. It is thus quite difficult to evaluate from our study how much will be loss whenever such situation will be more common, and whether it will be worth focusing on such inefficiencies or not. Additional information (in term of workload characterization) are thus required to decide on sound basis whether the BOINC

architecture should be modified to prevent such situations to happen.

- **Relevance of Nash Equilibria**: As David Anderson pointed out, BOINC is typically a production system where project administrator are very conservative. In practice, many projects still run an old and custom version of the server and do not benefit from the latest developments. The EDF server policy is generally considered to be too costly and is thus not often not activated but in most cases it was actually never tried. It is thus quite unlikely that project administrators will spend time playing with server parameters and over-tuning them. However, if the slack choice is critical, coming up with a good default value would be very useful. From a game theory perspective, David Anderson who manages the BOINC source code would serve as an oracle and would provide project administrators with the right value that is globally efficient. One may thus wonder how to compute such good default parameters...

- **Umbrella projects**: setting up and administrating a BOINC server can time- and resource-consuming. Therefore, in the last years, BOINC servers have evolved to accommodate several projects at the same time. Such projects are called umbrella projects and such centralization could provide an effective way to achieve a fair and efficient resource sharing between different projects.

Chapter 8

Ongoing and Future Work

Despite all our efforts, large computing platforms are inherently heterogeneous and, due to their cost, shared by several users. These two aspects are very difficult to model and make optimization of such systems particularly challenging.

As illustrated by the results presented in this part, moving from discrete optimization problems to somehow continuous optimization problems (through steady-state scheduling, divisibility or preemption) is a very effective way to get rid of somehow artificial complexity issues that prevent focusing on key difficulties. In my PhD thesis, we used such approach to address scheduling problems where platform exhibit a high degree of heterogeneity (both in term of computation and communication speed or of topology).

Following this path, the subsequent problems I have worked on included both an important platform heterogeneity as well as a notion of users. To this end, we introduced in our modeling game theory notions and we adapted to our setting optimization tools (e.g., Lagrangian optimization and gradient descent) whose use is more classical in the networking community. Such game theory notions also completely changed our perspective on these problems.

In particular, it raises a question that was not really clear to me twelve years ago. Steady-state optimization does not take time into account while response time optimization requires finite amount of workload and termination of activities. Theses two kind of scheduling problems are generally addressed in quite different ways with different formalism and techniques. Yet, in practice, both make sense and we have mixtures of such workloads. I think understanding how to combine these two aspects in a sound way in a single optimization problem will improve our understanding and help us answering the general question of *fair and efficient resource sharing between steady-state oriented and response time oriented workloads*.

Interestingly, although I was initially very excited by the stretch notion (see Chapter 6), I am not convinced anymore that this is the right approach to handle user and task heterogeneity. Indeed, the rationale for such normalization is to obtain a value that reflects the slowdown perceived by users (or tasks) independently of their need and which also somehow reflects the resource share they obtained. I doubt task-based normalization can help defining a sound fair resource share.

- First, a user should not care about the way resources given to an other user are used. It is every user's duty to determine what is the best way to use resources that are allotted to him. Therefore, using a renormalization like Cluster Equivalence (see Chapter 7) seems quite meaningful to me. Obviously, Cluster Equivalence values are very different for a throughput oriented user and for a response time oriented user but resource-based fairness notion like Nash Bargaining Solution seamlessly handle such heterogeneity of needs between users.

- Second, I doubt a task-based normalization will ever allow to account for temporal variability of resource pressure. I think this notion of resource price that appeared when using Lagrangian optimization (see Chapter 5) and which reflects the instantaneous contention

of a given resource is very elegant and offers many advantages (e.g., in term of distribution and separation of concerns). Although, we could easily use a similar idea related to time and let users bid on resources, this would provide a mechanism and not a clear goal to achieve, which is I think the most difficult part.

In the next years, I intend to revisit the centralized response time optimization problem (see Chapter 6) in particular in the context of BOINC as it is a setting where the multi-user (project) and the mixed throughput/response time workload characteristics are naturally present. In the last three BOINC workshop, I have discussed with many colleagues like David Anderson (BOINC's designer and main developer), Kevin Reed (responsible of the World Community Grid project), Francisco Brasileiro (OurGrid's designer), Wenjing Wu (responsible of the CAS@home project), Derrick Kondo, and Lionel Eyraud about how to reorganize the BOINC scheduler to handle several projects at once (umbrella projects) and response-time oriented projects. Lack of time and the inability to find a satisfying optimization objective somehow stopped me to test the different scheduling ideas we came up with. This is unfortunate as I think it is possible to come up with very pragmatic and effective techniques thanks to the insights gained by theory. Another excuse for not applying these ideas right away is probably the fear to interfere with a production system. Again, looking back, this is an error as I am convinced it would allow to quickly improve the experience of project administrators and widen the usage of BOINC.

Obviously all the questions and solutions related to these scheduling problems are not specific at all to the BOINC setting. Many articles raise similar issues in the context of map-reduce applications or of cloud systems but I am not sure these contexts really cast a new light on these problems nor help answering them. The context of BOINC seems however particularly interesting to me, not only because of its maturity and open-source spirit, but because it offers a diversity of use cases where workload can be well characterized, which is essential to a sound performance evaluation.

Part II

SimGrid: an Open Simulation Framework

Chapter 9

Context

This chapter builds on an article recently published in JPDC'14 [3] with Henri Casanova , Arnaud Giersch, Martin Quinson, and Frédéric Suter.

9.1 Motivation

While large-scale production platforms have been deployed and used successfully in many domains (grid computing, peer-to-peer systems, high performance computing, volunteer computing, ...), many open fundamental questions remain. Relevant challenges include resource management, resource discovery and monitoring, application scheduling, data management, decentralized algorithms, electrical power management, resource economics, fault-tolerance, scalability, performance, ... Regardless of the specific context and of the research question at hand, studying and understanding the behavior of applications on distributed platforms is difficult. The goal is to assess the quality of competing algorithmic and system designs with respect to precise objective metrics. Theoretical analysis like the ones we did in Chapters 2- 6 are typically tractable only when using stringent and ultimately unrealistic assumptions. As a result, relevant research is mostly empirical and proceeds as follows. An *experiment* consists in executing a *software application* on a target *hardware platform*. We use the term "application" in a broad sense here, encompassing a parallel scientific simulation, a peer-to-peer file sharing system, a cloud computing brokering system, etc. The application execution on the platform generates a time-stamped trace of events, from which relevant metrics can be computed (e.g., execution time, throughput, power consumption). Finally, research questions are answered by comparing these metrics across multiple experiments.

One can distinguish three classes of experiments. In *in vivo* experiments an actual implementation of the application is executed on a real-world platform. Unfortunately, real-world production platforms may not be available for the purpose of experiments. Even if a testbed platform is available, experiments can only be conducted for (subsets of) the platform configuration at hand, limiting the range of experimental scenarios. Finally, conducting reproducible in vivo experiments often proves difficult due to changing workload and resource conditions. An alternative that obviates these concerns is *in vitro* experiments, i.e., using emulation (e.g., virtual machines, network emulation). A problem with both in vivo and in vitro experiments is that experiments may be prohibitively time consuming. This problem is exacerbated not only by the need to study long-running applications but also by the fact that large numbers of experiments are typically needed to obtain results with reasonable statistical significance. Furthermore, when studying large-scale applications and platforms, commensurate amounts of hardware resources are required. Even if the necessary resources are available, power consumption considerations must be taken into account: using large-scale platforms merely for performance evaluation experiments may be an unacceptable expense and a waste of natural resources. The third approach consists in running (an abstraction of) the application *in silico*, i.e., using simulation. This ap-

proach is typically less labor intensive, and often less costly in terms of hardware resources, when compared to in vivo or in vitro experiments. Consequently, it should be no surprise that many published results in the field are obtained in silico.

An important observation is that simulators used by parallel and distributed computing researchers are domain-specific (e.g., peer-to-peer simulators, grid simulators, HPC simulators). In some cases, domain-specificity is justified. For instance, wireless networks are markedly different from wired networks and in this work, for instance, we only consider wired networks. But, in general, many simulators are developed by researchers for their own research projects and these researchers are domain experts, not simulation experts. The popular wisdom seems to be that developing a versatile simulator that applies across domains is not a worthwhile endeavor because specialization allows for "better" simulation, i.e., simulations that achieve a desirable trade-off between accuracy and scalability.

Indeed, in our context, two key concerns for simulation are *accuracy* (the ability to run in silico experiments with no or little result bias when compared to their in vivo counterparts) and *scalability* (the ability to run large and/or fast in silico experiments). A simulator relies on one or more simulation models to describe the interaction between the simulated application and the simulated platform. There is a widely acknowledged trade-off between model accuracy and model scalability (e.g., a continuous model based on equations may be less accurate than a complex event-driven procedure but its evaluation would also be less memory- and CPU-intensive). Simulation has been used in some areas of Computer Science for decades, e.g., for microprocessor and network protocol design, but its use in the field of parallel and distributed computing is less developed. While the scalability of a simulator can be easily quantified, evaluating its accuracy is painstaking and time-consuming. As a result, published validation results often focus on a few scenarios, which may be relevant to a particular scope, instead of engaging in a systematic and critical evaluation methodology. Consequently, countless published research results are obtained with simulation methods whose accuracy is more or less unknown.

Worse, most simulators developed in this area are only intended for use by their own developers. Some are sometimes made available to the community but prove short-lived or barely usable for another study than the one they were initially designed for. As a consequence, to date, most simulation results in the parallel and distributed computing literature are obtained with simulators that are ad hoc, unavailable, undocumented, and/or no longer maintained. As an illustration, in 2013, the authors in [BFS+13] point out that out of 125 recent papers they surveyed that study peer-to-peer systems, 52% use simulation and mention a simulator, but 72% of them use a custom simulator. Most published simulation results are thus impossible to reproduce by researchers other than the authors and there is a strong need for recognized simulation frameworks by which simulation results can be reproduced and further analyzed.

As explained in Section 1.2.2, Simgrid was originally developed by Henri Casanova who inspired from code and ideas from the different scheduler simulators developed in the AppLeS group at UCSD into an open-source toolkit. The initial motivation was clearly to avoid others reinventing the ("squared") wheel by providing the "grid" scheduling community a tool that would be suited to its needs. During my PhD, I have extended this tool for my own research and taken over its development, first alone, then with Martin Quinson, and later with other colleagues such as Frédéric Suter as the user community was growing and more recently Arnaud Giersch. In that sense, SimGrid somehow started like most other simulation projects, i.e., by distributed/parallel computing researchers who did not know much about simulation and performance evaluation. However, it fundamentally differed from the plethora of other distributed and parallel computing simulators with respect to at least the three following aspects:

- **Open-source and long-term support**: It was clear to us that releasing code and providing public access to the revision system was the bare minimum but that it was far from sufficient to benefit from free software development. Growing a user and developer community requires both high quality software and adherence to values carried by a long-term project, which is especially difficult in a research context where quick and dirty prototypes are the fastest way to answer questions and to publish articles. The credit for achieving this

clearly goes to my colleagues, Martin Quinson and Frédéric Suter. Martin Quinson always considered as a major priority for SimGrid to be a high quality software and worked hard to make sure it would be the case. SimGrid has more than 500 non-regression tests, nightly builds on a dozen of different architectures/operating systems, and although the favorite target is recent linux OS, it works both on Mac OS X and windows. The only way to ensure a long term support is to grow a sufficiently large user base, which again is quite difficult in a research context and requires to reach a critical mass. This is why when SimGrid started moving from a simple prototype to also a research project in itself, our sole efforts were not sufficient and we needed additional funding and manpower. Frédéric Suter and Martin Quinson always made sure that others would know about our work and that it would be usable despite all my efforts to temper their enthusiasm and pretend it was not ready yet. Thanks to their help, we managed to conduct two very large ANR projects whose outcomes and success was far beyond my expectations. SimGrid is thus clearly a team project that has built on the strengths of several individuals sharing a common goal of improving their practices and the one of their community, which in turn attracted contributors believing in such an objective. This definitely makes SimGrid different from most other simulation projects.

Although I do not feel I have been a key contributor to the open-source aspect of SimGrid, I have definitely contributed to its scalability and its accuracy.

- **Accuracy**: While the scalability of a simulator can be easily quantified, evaluating its accuracy is painstaking and time-consuming. Although many simulation projects do not even consider such issue, Henri Casanova and I had such concerns about 13 years ago and how to "validate" our model was not clear at all by then. Since then, I have had the chance to direct the PhD thesis of Pedro Velho on this very subject and to pursue this effort with Augustin Degomme who worked a research engineer with me and Luka Stanisic who is working as PhD student with me. Mainly, I finally understood that validation (i.e., somehow proving that the simulation would be accurate) was illusive and that only thorough invalidation was possible. The experience gained with Pedro Velho in the context of wide area networks allowed us to extend our approach to the HPC context with Augustin Degomme and Luka Stanisic. I quickly describe some of the results we obtained in Chapter 10.

- **Scalability**: Simulation speed has been a key preoccupation in SimGrid since its creation as one of the main motivation was rapid prototyping of schedulers. Therefore, SimGrid builds on simple and pragmatic models as well as an efficient C implementation. SimGrid is now in its third major version, which could actually as well be its fourth or fifth major version if we consider a major version to be a major rewrite of the whole code. Achieving highly scalable simulation requires advanced skills both from the algorithmic and programming point of view, but also from the workload characterization and modeling perspective and such skills are hardly found within a single person. Along the years, we have gained experience that make SimGrid far more scalable than most other simulators of many fields. I quickly describe some of the techniques we used and results we obtained in Chapter 11.

SimGrid is now a 15 years old project and has become a major player in the field. Interestingly, SimGrid is not restricted at all to grid computing studies anymore. A major outcome of the ANR projects USS-SimGrid and SONGS is that SimGrid also applies to the study of peer-to-peer systems, volunteer computing systems, cloud computing systems and even high performance computing system. This versatility was never a hindrance but rather an asset as it allowed us to reuse the experience gained in other domains and to achieve

9.2 The SimGrid Architecture

Fig. 9.1 shows the main components in the design of SIMGRID and depicts some of the key concepts in this design. The top part of the figure shows the three APIs through which users can

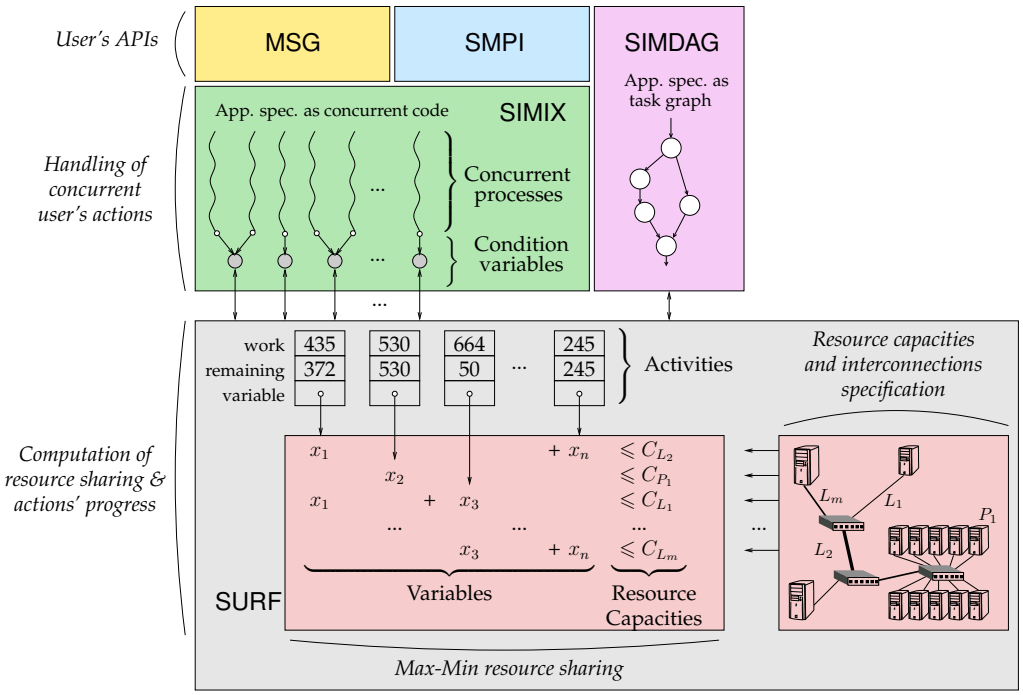


Figure 9.1: Design and internals of SIMGRID.

develop simulators. The MSG API allows users to describe a simulated application as a set of concurrent processes. These processes execute code implemented by the user (in C, C++, Java, Lua, or Ruby), and place MSG calls to simulate computation and communication activities. The SMPI API is also used to simulate applications as sets of concurrent processes, but these processes are created automatically from an existing application written in C or Fortran that uses the MPI standard. SMPI also includes a runtime system, not shown in the figure, that implements necessary MPI-specific functionalities (e.g., process startup, collective communications). MSG thus makes it possible to simulate any arbitrary application, while SMPI makes it possible to simulate existing, unmodified MPI applications. The mechanisms for simulating the concurrent processes for both MSG and SMPI are implemented as part of a layer called SIMIX, which is a kernel (in the Operating Systems sense of the term) that provides process control and synchronization abstractions. The set of concurrent processes is depicted in the SIMIX box in the figure. All processes synchronize on a set of condition variables, also shown in the figure. Each condition variable corresponds to a simulated activity, computation or data transfer, and is used to ensure that concurrent processes wait on activity completions to make progress throughout (simulated) time. The third API, SimDAG, does not use concurrent processes but instead allows users to specify an abstract task graph of communicating computational tasks with non-cyclic dependencies.

Regardless of the API used, the simulation application consists of a set of communication and computation activities which are to be executed on simulated hardware resources. Compute resources are defined in terms of compute capacities (e.g., CPU cycles per time unit). They are interconnected via a network topology that comprises network links and routing elements, defined by bandwidth capacities and latencies. All resources can be optionally associated with time-stamped traces of available capacity values including possible downtime. An example specification of available resources is depicted in the bottom-right of Fig. 9.1, highlighting three network links (L_1, L_2, L_m) and one compute resource (P_1).

The simulation core, i.e., the component that simulates the execution of activities on resources, is called SURF and is shown in the bottom-left of the figure. Each activity is defined by a total amount of work to accomplish (e.g., number of CPU cycles to execute, number of bytes to trans-

fer) and a remaining amount of work. When its remaining amount of work reaches zero the activity completes, signaling the corresponding SIMIX condition variable or resolving a task dependency in SimDAG. Activity i corresponds to a variable, x_i, which represents a resource share used by the activity. A set of constraints over these variables, with one constraint per simulated resource, then describes how the activities compete for using the resources. An example is shown in the figure. The right-hand sides of each constraint is a resource capacity, denoted by C_x where x is a given resource (e.g., C_{L_2} is the capacity of network link L_2). The first activity requires performing 435 units of work, 372 of which remains to be performed at the current simulated date, and is associated to variable x_1. Variable x_1 participates in two constraints, for resources L_2 and L_1, meaning that in this example the first activity is a data transfer that uses network links L_2 and L_1, among others. The third activity is also a data transfer, with 50 units of work remaining out of 664. This transfer uses links L_1 and L_m, meaning that it shares the bandwidth capacity of L_1 with the first activity (as seen in the $x_1 + x_3 \leqslant C_{L_1}$ constraint). The n-th activity uses links L_2 and L_m, and its corresponding variable x_n thus appears in those two constraints. Finally, the second activity in this example corresponds to a computation and its variable x_2 appears in the constraint $x_2 \leqslant C_{P_1}$, showing that this resource is not shared with any other compute activity. Note that the second and n-th activities in this example have yet to begin as their remaining works are equal to their total works. Based on these constraints the simulation core computes resource allocations.

To this end, SIMGRID mostly uses a unified *fluid* model for simulating the execution of activities on simulated resources. This model is purely analytical so as to afford scalability by avoiding costly cycle-, block-, and packet-level simulation of compute, storage, and network resource usage. Such "flow-level" models have been proposed in the networking literature, mostly to study the theoretical behavior of TCP protocols. Inspired by these developments, in SIMGRID we use a flow-level model for the purpose of cpu, disk and network simulation. In a flow-level network model the individual packets of an end-to-end communication are abstracted into a single entity, a *flow*, which is characterized by a data transfer rate, or bandwidth. This bandwidth depends on the network topology and on the interactions with other ongoing network flows. It is assumed that the flows have reached *steady-state*, and the goal is to define analytical bandwidth sharing models that capture the bandwidth sharing behaviors of actual network protocols.

Formally, in the context of SIMGRID, given a resource r, and a set of simulated activities, $\mathcal{A}$, the model is specified by a set of positive linear constraints and a possibly non-linear optimization problem:

$$\begin{array}{l} \text{MAXIMIZE } f(a) \\ \text{UNDER CONSTRAINTS} \\ \left\{ \forall r, \sum_{a \in \mathcal{A} \text{ using resource } r} \varrho_a \leqslant C_r, \right. \end{array} \tag{9.1}$$

where C_r denotes the capacity of resource r, and ϱ_a denotes the resource share allocated to activity a. No particular assumption is done on the way activities are linked to resources, which provides a lot of flexibility when modeling real systems. In practice (but not necessarily), the objective function is often $\min_{a \in \mathcal{A}} \varrho_a$, and corresponds to a popular bandwidth sharing model, Max-Min fairness [BG96], by which the bandwidth allocation is such that increasing the allocation of any flow would necessarily require decreasing the allocation of a less favored flow.

Solving this optimization problem, which boils down to solving a (generally) linear system, yields instantaneous resource shares to activities. Given these computed resource shares at simulated time t_0, for all simulated resources, the SURF component of SIMGRID computes the first activity that will complete, advances the simulated clock to that time, say t_1, removes the completed activity from consideration, accounts for the progress of each activity given its resource shares and the simulated elapsed time $t_1 - t_0$, and possibly adds newly created activities (by yielding to the process whose activity just completed).

The optimization problem in Eq. (9.1) is at the core of the SURF component of SIMGRID (see Fig. 9.1), which implements efficient algorithms and data structures to solve the corresponding linear system quickly. The key aspect of this model is that it is does not only allow to easily simulate coarse-grain characteristics of networks but also of CPU and storage provided a little flexibility in optimization function and constraint definition.

For example, fair sharing of cpu resource or of disk bandwidth is trivially modeled and even compute priorities. The seek time is added as a fixed initial delay when advancing the simulation clock in the same way are latency is accounted for when simulating network flows. Obviously, the resource sharing pattern is much simpler for cpu and disk and does not require such a complex formulation. Yet having a flexible unified formulation is precious and allowed us to reuse optimization techniques and validity results when moving from a domain to an other unlike what most other simulators could afford (see for example Fig. 12.5).

9.3 Related Simulators

Many other simulators have also been developed in the area of *grid computing*, most of which only intended for use by their own developers (e.g., Bricks [TMN+99] or HyperSim [PUK03] for which I have never been able to get access to the code). Some were made available to the community but proved short-lived, such as OptorSim [BCM+03] or ChicSim [RF02]. Besides SIMGRID, the other simulator widely used in grid computing research is GridSim [BM02]. Several simulators like GSSim [KNOW07] have built on gridsim to provide higher-level abstractions.

More recently, simulators have been proposed for simulating *cloud computing* platforms and applications. GroudSim [OPF10] is a framework that enables the simulation of both grid and cloud systems. CloudSim [CRB+11] builds on the same simulation internals as GridSim but exposes specific interfaces for simulating systems that support cloud services. To the best of our knowledge many people who based their work on CloudSim or on GridSim ended up modifyning the internals of the simulator although the proposed API suited their needs. iCanCloud [NVPC+11] has been specifically developed to simulate cloud platforms and applications at a possibly fine-grain level.

Some simulators have also been developed for simulating *volunteer computing* systems. BOINC [BOI] is the most popular volunteer computing infrastructure today, and these simulators attempt to simulate (parts of) BOINC's functionalities. In fact, BOINC itself embeds in its source code a simple time-driven simulator for running the actual client scheduler code in simulation mode. The SimBA simulator [TKE+07] models BOINC clients as finite-state automata based on probabilistic models of availability, and makes it possible to study server-side scheduling policies in simulation. The same authors later developed EmBOINC [ETA09]. Unlike SimBA, EmBOINC executes actual BOINC production code to emulate the BOINC server. SimBOINC [Kon07] goes further and simulates the full BOINC system by linking the BOINC code with SIMGRID, thus allowing for multiple servers and for the simulation of client-side scheduling. Due to BOINC code restructuring and to the need to maintain SimBOINC in sync with BOINC, SimBOINC has however been discontinued, which is why we did not build on it but rewrote our own modeling of BOINC on top of SIMGRID in Chapter 7.

Another area in which simulators have been developed is *peer-to-peer computing* [BFS+13]. Most of these simulators trade off accuracy for scalability, so as to make it possible to simulate up to millions of peers. For instance, it is common to simulate network transfers as fixed delays since message count is a useful metric to evaluate peer-to-peer systems. PeerSim [MJ09] is likely the most widely used simulators for theoretical peer-to-peer studies, and relies on simplistic (i.e., simple delay-based models) but scalable simulation models. OverSim [BHK07] builds on the OMNeT++ [Var01] discrete-event simulation kernel and may thus rely on more realistic packet-level network simulation. Several other simulators have emerged, such as P2PSim [GKL+05] or PlanetSim [GPM+04], but they have been short-lived and are no longer maintained. It is thus difficult to say whether more recent proposals, e.g., D-P2P-Sim [SPS+09], will perdure.

Finally, the simulation of parallel applications on parallel computing platforms has a long history in HPC, in particular in the context of *applications based on MPI* [GLS99]. Two main approaches are used: *off-line* and *on-line* simulation.

- In off-line simulation, a time-stamped log of computation and communication events is first obtained by running the application on a real platform. The simulator then replays

this sequence of events as if they were occurring on another platform with different hardware characteristics. The three most representative such simulators are PSinS [TLCS09], LogGOPSim [HSL10b], Dimemas [BLGE03] and BigSim [ZKK04] but several others are available (e.g., [ZCZ10a, TBB+11, NnFG+10, ZCZ10b, HGWW09]). One issue with off-line simulation is that event logs are tied to a particular application execution (e.g., number of processors, block size, data distribution schemes) so that a new log must be obtained for each simulation scenario. However, extrapolation is feasible as proposed for instance in [WM11, CLT13, HSL10b].

- The alternative to off-line simulation is on-line simulation in which actual application code is executed on a host platform that attempts to mimic the behavior of the target platform. Part of the instruction stream is intercepted and passed to a simulator. Many on-line simulators have been developed with various features and capabilities (e.g., [PWTR09, BDP01, DHN96, Rie06, LRMB09]). In these simulators, the amounts of hardware resources required to run the simulated application on the host platform are commensurate to (or in fact larger than) those needed to run the actual application on the target platform. Simulation scalability is thus achieved by "throwing more hardware" at the problem. In the context of SIMGRID, we always tried to limit our scope to simulations that can be executed on a single computer, so that simulation scalability must be achieved in software.

As we already explained, the code of all these simulators is not always available and when it is, it may not be still actively supported, which makes it quite difficult to compare with. In the next two chapters, we only compare with what we believed to be the most significant ones. Still, we can already list a few differences between SIMGRID and these simulators: (i) SIMGRID is the only one that is not domain specific and is generic enough to "compete" with all these other simulators, (ii) SIMGRID is almost the only one who correctly implements fluid models and can thus account for network topology and contention in a scalable way, and (iii) SIMGRID is written in plain C and uses its own custom data structures and thread abstraction. This last point explains part of the performances we managed to obtain by having a full control on performances but it also hindered the adoption of SIMGRID compared to most tools relying on java for example and which are easier to approach.

The reader may notice that we purposely did not include in our list neither classical network simulators like NS [IH08], GTNetS [Ril03], DaSSF [LN01] nor micro-architecture simulators like GEMS [MSB+05], SIMICS [sim] or GPGPU-Sim [BYF+09]. The reason for this is such simulators are typically in details of protocols or of architecture that none of the previous simulators is actually interested in. SIMGRID is clearly not intended to answer questions like "is such version of TCP more fair or more TCP-friendly or more reactive than such other version of TCP?" or "would our GPU benefit from a different prefetching policy or from a larger texture cache or from a higher number of cores?". SIMGRID is initially rather intended to questions scheduling and resource management questions on large-scale platforms and at a rather large time-scale.

Chapter 10

The Validation Quest

This chapter builds on Pedro Velho's PhD thesis [Vel11] with whom I have studied the validity of fluid network models for wide area networks and, which has been published at SimuTools09 [33] and in TOMACS13 [5] with Pedro Velho, Henri Casanova, and Lucas Schnorr. This chapter also builds on the more recent work on simulation of MPI applications, which has been published at PMBS'13 [24] with Augustin Degomme, Martin Quinson, Frédéric Suter and several others. This work has been conducted in the context of the USS-SimGrid and SONGS ANR project.

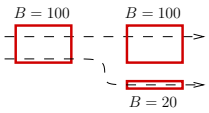
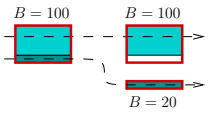
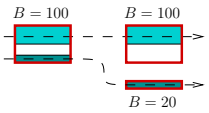
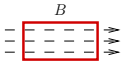
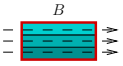
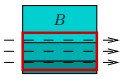
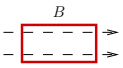
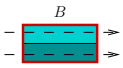
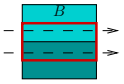
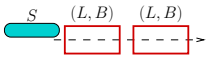
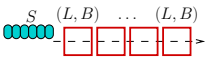
It is common (especially in Physics) to judge a scientific theory by its ability to make correct predictions about the world. Obviously, the simplicity of the theory and how it improves our understanding of the studied phenomenon is also at stake but I have to say that, from my experience, most simulation studies of large distributed computing systems have little care for correctness of predictions. Although using valid simulation models seems a prerequisite for developing a useful simulator, it turns out that many popular grid/cloud simulators use network models that have not been (sufficiently) validated. Table 10.1 illustrates a few simple experiments that invalidate four well-known simulators: OptorSim (2.1, 02/2010) [BCM+03], GridSim (5.2 beta, 11/2010) [BM02], GroudSim (0.11, 06/2010) [OPF10], and CloudSim (3.0.2, 11/2012) [CRB+11]. The version we tested were the latest at the time of writing [5]. These simulators have been used to obtain results published in hundreds of research articles, and also used as building blocks to develop other simulators [CD12, TYM11, SJY11, JK12]. Each invalidating experiment in Table 10.1 can yet be devised through inspection of the simulator's source code, and some are even sometimes documented by the developers themselves.

The flow-level model, which is used in SIMGRID and which we presented in the previous chapter, is a supposedly effective way to account for network topology and contention at low computational cost. As a result it is implemented in several simulators [BCM+03, OPF10, GB02]. Unfortunately, SIMGRID is the only simulator we are aware of which correctly implements such flow-level models.

Some model validation results have been published for other simulators, but they typically consider only a few cases in which simulation models are expected to work well [TLCS09, ZKK04, ZWL+04] (essentially merely verifying that model implementations are correct). However, it is accepted in most of the sciences that model *invalidation* is an important component of the research activity. Invalidation studies must be conducted that explore a wide range of scenarios for which a model's behavior is either unknown or expected to be invalid. Through these invalidation studies, the model is improved or refuted, leading to increasingly precise knowledge of the range of situations in which the model's results are meaningful.

We have thus focused on flow-level network models of the Transmission Control Protocol (TCP) to be used when simulating large-scale grid/cloud distributed systems. For these systems, accounting for network proximity and network topology is paramount for ensuring that simulations are valid. The goal of flow-level models is to capture complex network behavior using tractable mathematical derivations, i.e., that can be computed quickly. In this chapter, we present a short overview of network models used in discrete-event simulation and of flow-level models.

Table 10.1: Invalidating experiments for four popular grid/cloud simulators. Network links are shown as boxes labeled with latencies (L) and/or bandwidth capacities (B). Network flows are dashed arrows that traverse one or more flows. Bandwidth allocations are shown as gray fills when applicable, with a different shade for each flow. The table shows both expected results and flawed results produced by each simulator.

Simulator	Setting	Expected output	Actual output	Cause
OptorSim or GroudSim (flow model)				Each link is shared fairly among all traversing flows, leading to underestimated link usage.
GridSim (flow model)				Bandwidth share is updated incorrectly. Flow #1 receives B, flow #2 receives $B/2$, flow #3 receives $B/3$.
CloudSim (flow model)				Fixes the problem in the row above, but problems still occur when flows start at different times.
GridSim (large message model)		Delay $\approx 2L + \frac{S}{B}$	Delay $= 2L + \mathbf{2}\frac{S}{B}$	A large message of size S is sent using store and forward, without any pipelining.
GridSim (small packet model)	 with B and $L \gg 1$	Delay$\approx$ $4L + \frac{S}{W_{max}/RTT}$, where W_{max} is the maximum TCP window size	Delay$\approx 4L + \frac{S}{B}$	Simple packet-level model that does not account for TCP features, such as congestion window and RTT-unfairness.

Then, we present the SIMGRID flow-level network model and explain how we came up with it as well as how, doing so, we invalidated other models proposed in the literature. Finally, we briefly explain how this model has been extended to the high performance computing setting.

10.1 Background and Related Work on Network Modeling

Network modeling and simulation can be undertaken in many contexts, thus mandating context-specific discussions of known methodologies and results. For instance, simulations for studying the fine-grain properties of the TCP protocol may have little in common with simulations for studying the scalability of some large-scale parallel computing application. In what follows we

discuss works in the area of network modeling categorized by modeling approaches, highlighting the specific contexts in which each approach has been used.

10.1.1 Packet-level Models

Packet-level network simulations are discrete-event simulations with events for packet emission or reception as well as network protocol events. These simulations can reproduce the movements of all network packets and the behavior of the whole TCP stack down to the IP level. Packet-level simulations have been widely used for studying fine-grained properties of network protocols.

The TCP stack models found in simulators such as GTNetsGTNetS [Ril03], or NS2 [IH08] are simplified versions of the TCP stack but recent developments [JM07] allow the use of real TCP stack implementations, which is slower but more realistic (i.e., real TCP stack implementations might have features/bugs that are absent from simplified versions used exclusively for simulation purposes). Except maybe for wireless networks, where the modeling of the physical layer is challenging, packet-level simulation is generally recognized by the network community as trustworthy, and it serves as a reference.

Unfortunately, as will be illustrated in Section 11.1, in the context of simulation of grids or clouds, it generally leads to unacceptably long simulation times since the life cycle of each packet is simulated through all protocol layers all the way to a simulated physical layer. Its use is thus often restricted to studying network protocols or applications that exchange small messages. To mitigate such overhead, simulators in the area of grid computing (e.g., GridSim [BM02], early versions of SIMGRID [Cas01]), have attempted widely simplified packet-level simulation (e.g., wormhole routing, no implementation of any network protocol). Unfortunately, these simulators are easily shown to produce simulated network delays that can be far from those achieved by TCP communications in real networks (see the last row of Table 10.1). Furthermore, these simplified packet-level models face the same scalability issues, if not as severe, as more realistic packet-level simulators.

Finally, for the purpose of simulation, fine-grain network simulation models may be difficult to instantiate with realistic parameter values when targeting large-scale networks as some of the parameter values may be unknown or difficult to obtain (e.g., a full description of a large subset of the Internet). Fortunately, the level of detail provided by packet-level simulation is not necessary for studying large-scale applications that exchange large amounts of data.

10.1.2 Delay-based Models

In some contexts it is sufficient to simulate simple network delays between pairs of communicating hosts. For instance, a peer-to-peer simulator for studying overlay networks may model each communication delay as a constant delay or as a sample from a given statistical distribution [MJ09]. These models do not account for network proximity. As a consequence, some peer-to-peer simulators [GKL+05, MJ09, BHK07] model network delays using coordinate systems. Each peer is provided with coordinates in a Euclidean metric space and the simulator simply computes the corresponding distance in this space to evaluate communication delay. Note that non-Euclidean spaces can lead to more accurate point-to-point communication delays in Wide Area networks [DCKM04]. Coordinate-based models provide a good trade-off between compute time and space as they account for network proximity with a $\Theta(N)$ memory footprint and a $O(1)$ computation time for a network delay. Since coordinates may change over time and suffer from measurements inaccuracies [LGS07], some simulators generally add noise to these coordinates [BHK07]. In the context of simulating High Performance Computing systems (e.g., clusters of servers), similar models have also been proposed [CKP+93, AISS95, KBV00, IFH01, HSL10a]. These models account for partial asynchrony for small messages. They also include specific parameters depending on message size making it possible to account for protocol switching, which can have a large impact on overall performance. These models are extremely scalable since both description size and delay computation time are in $O(1)$.

All models above, whether for peer-to-peer application or for HPC systems, ignore network contention. In the case of peer-to-peer simulations, the rationale is that the amount of exchanged data is small and that network contention is thus not an important factor in the performance of the application. In fact, often the metric of interest is the number of exchanged messages, rather than the data transfer rates achieved. In the case of HPC simulations, the rationale is that the target platform is provisioned with a bisection bandwidth so large that network contention is unlikely to occur. While this assumption may be reasonable for high-end systems (e.g., systems built from high-radix optical switches), it does not hold for all commodity clusters.

10.1.3 Flow-level Models of TCP

When modeling network contention is needed, one cannot use the scalable delay-based models discussed in the previous section. To achieve scalability higher than that afforded by full-fledged packet-level models, one needs an abstraction of the network topology that focuses on the part of the topology in which contention may happen. For instance, when considering the simulation of a peer-to-peer data streaming application or of a volunteer computing infrastructure in which participants donate compute cycles, a detailed model of the core of the network may not be needed as most of the contention occurs at the edges of the network (e.g., when streaming content, when downloading input data for computation). Conversely, when studying collective communications in a cluster with limited bisection bandwidth, it is necessary to model the core of the network since it is where the contention will occur. Similarly, in platforms that span wide-area networks, applications may experience network contention on some wide-area network paths.

An alternative to expensive packet-level modeling, which still makes it possible to account for network contention, is thus flow-level modeling as explained in Section 9.2. In flow-level models, which are used by simulators in various domains [BCM+03, 35, OPF10, GB02, ZKK04], each communication, or *flow*, is simulated as a single entity rather than as a set of individual packets. The time needed to transfer a message of size S between hosts i and j is then given by:

$$T_{i,j}(S) = L_{i,j} + S/B_{i,j}, \tag{10.1}$$

where $L_{i,j}$ (resp. $B_{i,j}$) is the end-to-end network latency (resp. bandwidth) on the route connecting i and j. Although determining $L_{i,j}$ may be straightforward, estimating the bandwidth $B_{i,j}$ is more difficult as it depends on interactions with every other flow. This is generally done by assuming that the flow has reached *steady-state*, in which case the simulation amounts to solving a bandwidth sharing problem, i.e., determining how much bandwidth is allocated to each flow. More formally:

> *Consider a connected network that consists of a set of links $\mathcal{L}$, in which each link l has capacity B_l. Consider a set of flows $\mathcal{F}$, where each flow is a communication between two network vertices along a given path. Determine a "realistic" bandwidth allocation ϱ_f for flow f, so that:*
>
> $$\forall l \in \mathcal{L}, \sum_{f \text{ going through } l} \varrho_f \leqslant B_l \ . \tag{10.2}$$

Given the computed bandwidth allocation (which defines all data transfer rates), and the size of the data to be transmitted by each flow, one can determine which flow will complete first. Upon completion of a flow, or upon arrival of a new flow, the bandwidth allocation can be reevaluated. Usually, this reevaluation is *memoryless*, meaning that it does not depend on past bandwidth allocations. This approach makes it possible to quickly step forward through (simulated) time, and thus is attractive for implementing scalable simulations of large-scale distributed systems with potentially large amounts of communicated data. However, it ignores phenomena such as protocol oscillations, slow start, and more generally all transient phases between two steady-state operation points. Perhaps more crucially, the whole approach is preconditioned on computing a bandwidth sharing solution that corresponds to the bandwidth sharing behavior of real-world network protocols, and in particular of TCP.

Two approaches are used to build models of a computer system, and thus of the network: in the *bottom-up* approach the model is built based on analyzing low-level phenomena; in the *top-down* approach the model is built from observations of the full system. The bottom-up approach should intuitively lead to more accurate models but does not guarantee perfect validity because analyzing low-level phenomena typically requires that approximations be made. One advantage of the top-down approach is that deriving the model is easier since understanding the low-level details of the system is not needed, but the validity of the model is more questionable.

The TCP network protocol is known for its additive increase multiplicative decrease window size policy, which causes the data transfer rate of flows to oscillate. While such oscillation complicates the modeling of the behavior of TCP, in the last decade, following the seminal work of Kelly *et al.* [KMT98], several bottom-up flow-level models and tools for network protocol analysis have been proposed [MLAW99, MW00, YMR10, LPW02, Low03b]. The goal is to provide a quantitative understanding of the macroscopic behavior of TCP given its microscopic behavior, and in particular its window size policy. In these works, the steady-state behavior of the protocol is often characterized as the solution to an optimization problem. By making an analogy between the equations governing expected window size that follow from the specification of TCP and a distributed gradient algorithm, Low *et al.* [Low03b] have proved that the steady-state throughputs of network flows are similar to those obtained by solving a global optimization problem under the constraints in Eq. (10.2).

For example, it is possible to prove that when using the Reno protocol with RED [FJ93] as a queue policy for routers, the steady-state bandwidth converge to a point that maximizes

$$\sum_{f \in \mathcal{F}} \frac{\sqrt{2}}{d_f} \arctan\left(\frac{d_f \varrho_f}{\sqrt{2}}\right) \text{ under constraints in Eq. (10.2)}, \tag{10.3}$$

where d_f be the equilibrium round trip time (propagation plus equilibrium queuing delay) of f, and is assumed to be constant. Similarly, it can be proven that TCP Vegas with DropTail achieves some form of weighted proportional fairness [Low03b], i.e., it maximizes the weighted sum of the logarithm of the throughput of each flow:

$$\sum_{f \in \mathcal{F}} d_f \log(\varrho_f) \quad \text{under constraints in Eq. (10.2)}. \tag{10.4}$$

These bottom-up models are attractive because they capture the specifics of the underlying network protocol. One drawback for using them as the basis for grid/cloud simulation, admittedly not their intended use, is that they involve parameters that may not be straightforward to instantiate by users, e.g., the equilibrium round trip time. The model may be of little use in this context if its instantiation requires in-depth knowledge of networking, or a whole set of complex experimental measurements (either of which would compel users to use unsound "guesses" as parameter values).

Top-down flow-level models have also been proposed. These models are not derived from an analysis of the specification of the TCP protocol, but from observations of the protocol's macroscopic behavior over a range of relevant network topologies. An oft mentioned bandwidth sharing objective, which leads directly to a top-down model, is max-min fairness. This objective is reached by recursively maximizing

$$\min_{f \in \mathcal{F}} w_f \varrho_f \quad \text{under constraints in Eq. (10.2)}, \tag{10.5}$$

where w_f is a weight parameter generally chosen as the round-trip time of flow f. There are two rationales for this objective. First, it corresponds to what one would na\"ively expect from a network, i.e., be "as fair as possible" so that the least favored flows receive as much bandwidth as possible while accounting through weights w_f for the well-known RTT-unfairness of TCP [MPP+07]. Second, there is a simple algorithm for solving the optimization problem [BG96], whereas solving non linear problems (with objectives such as the ones in Eq. (10.3) or Eq. (10.4))

requires more elaborate techniques (such as lagrangian optimization and gradient descent as we used in Chapter 5). Previous studies have shown that max-min fairness does not exactly correspond to bandwidth sharing under TCP but that it is a reasonable approximation in many relevant cases [Chi99, CM02] but our goal was to compare these different kind of fluid models and see whether bottom-up models were significantly better than top-down models for the purpose of simulation.

10.2 Accuracy of Flow-level Simulation

10.2.1 Toward an Informed Flow-Level Network Model

The overarching question that we study in this work is whether flow-level simulation can provide accurate results, in particular when compared to packet-level simulation. Some simulators do implement flawed max-min flow-level models that lead to reasonable results in some situations but also lead to unrealistic bandwidth sharing in other situations (see Table 10.1). Besides such *plainly* invalid models, there is a striking dearth of validation studies in the literature. Those works that do propose flow-level models [LPW02, Low03b] are driven by protocol design goals [LS04], and thus merely present a few test cases to illustrate the correctness of the models.

Evaluating the validity of flow-level models in complex and diverse scenarios raises practical difficulties, such as requiring that real-world networks be configured for each scenario of interest. In the context of grid or cloud computing systems, setting up many network configurations for the purpose of model validation is simply not possible. One convenient solution is to compare results obtained with flow-level models to results obtained using packet-level simulation. This approach raises the question of whether packet-level simulation is representative of real-world networks. Answering this question is out of the scope of this work. However, based on the confidence placed by the network community in its packet-level simulators, it is reasonable in this work to declare packet-level simulation the ground truth. As a result, we talk of the *error* of a flow-level model to denote the discrepancy between its results and the results obtained with packet-level simulators for the same simulated scenario.

Conveniently, SIMGRID provides an interface to the GTNetS packet-level simulator, which greatly eases the comparison of flow-level and packet-level results. GTNetS is no longer officially supported and the latest version of SIMGRID also provides an interface to NS3. (Both GTNetS and NS3 implement the same TCP stack.) The current version of SIMGRID implements flow-level models based either on max-min fairness or on the model by Low *et al.* [Low03b]. These capabilities of SIMGRID make it a convenient framework for studying the validity of flow-level models. In our work, we took advantage of these capabilities to evaluate and improve upon a classical top-down max-min flow-level model. In that work the following improvements were made:

- **Linearity:** The common approach is to model the execution time of a flow that transfers S bytes of data as the latency plus S divided by the bandwidth:

$$T_f(S) = \ell_f + S/\varrho_f, \tag{10.6}$$

  where S is the message size, ϱ_f is the bandwidth share computed by the bandwidth sharing model and ℓ_f is the end-to-end latency, i.e., the sum of the latencies of the links traversed by f. However, ℓ_f and B_l (used in the computation of ϱ_f) are physical characteristics that are not directly representative of what may be achieved by flows in practice. The protocol overhead should be accounted for, which can be done by multiplying all latencies by a factor $\alpha > 1$ and all bandwidths by a factor $\beta < 1$. α accounts for TCP slow-start and stabilization, which prevent flows from instantaneously reaching steady-state. β accounts for packetization and control overheads. A more accurate empirical model is thus

$$T^{\text{improved}} = \alpha\ell_f + \frac{S}{\beta\varrho_f}, \tag{10.7}$$

where α and β (typically $\alpha \in [10, 15]$ and $\beta \in [0.8, 1]$, depending on the version of TCP [33, 5]) are two additional positive real parameters. Packet-level simulations can be used to calibrate parameter values, i.e., to determine the parameter values that minimize modeling error for a set of synthetic simulation scenarios. This simple change leads to an excellent approximation for a single flow on a single link when message size is larger than 100KB [33].

- **Flow Control Limitation:** TCP's flow control mechanism is known to prevent full bandwidth usage as flows may be limited by large latencies [MSMO97, FF99, JPD03]. This well-known phenomenon can be captured in a flow-level model by adding, for each flow, the constraint that:

$$\varrho_f \leqslant W_{\max}/(2L_f) \ , \tag{10.8}$$

where $W_{\max}$ is the configured maximum window size and L_f is the sum of latencies along the path. The validity of this enhancement is demonstrated for a single flow going through a single link [33].

- **Bottleneck Sharing and RTT-unfairness:** TCP protocols are known to be RTT-unfair, hence when two flows contend for bandwidth on a bottleneck link they are assigned bandwidth inversely proportional to their round trip times [MPP+07]. This round-trip-time is thus used as the weights of variables in the max-min or in the other optimization problem formulation (the d_f in Eq. (10.4) and (10.3)). On a simple dumbbell topology, but for a wide range of bandwidth and latency parameters (including highly contended situations), this round trip time is well approximated by the following formula:

$$RTT_f = \sum_{l \text{ traversed by } f} L_l + \frac{\gamma}{B_l} \ , \tag{10.9}$$

where γ is a fixed value for all flows. Tuning this value from packet-level simulations, it turns out that $\gamma = 8{,}775$ provides a good approximation [33].

- **Reverse Traffic Influence:** The data transfer rate of a TCP flow is strongly dependent upon the rate at which acknowledgment packets arrive, which can lead to surprising situations. Although Our expectation is that on a single full-duplex link the bandwidth allocated to flows going in one direction is independent from that allocated to flows going in the reverse direction, it is easily demonstrated to be influenced by flows going in the reverse direction both in packet-level simulation and in real-world experiments. Let us denote by $\{B, (n_1, n_2)\}$ a scenario with a single full-duplex bottleneck link of capacity B, n flows going through the link in one direction, and p flows going through the link in the reverse direction. We denote the outcome of the scenario by (B_1, B_2), where B_1 is the bandwidth allocated to the n flows and B_2 is the bandwidth allocated to the p reverse flows (in all cases all flows in the same direction are allocated the same bandwidth). Fig. 10.1 depicts five example scenarios, showing approximate results achieved in simulation. The bandwidth allocation for $\{B, (n, 0)\}$ (resp. $\{B, (0, p)\}$) is always approximately $(B/n, 0)$ (resp. $(0, B/p)$). Since the network link is full-duplex, expectedly simulations also show that $\{B, (1, 1)\}$ leads to the allocation (B, B). Likewise, $\{B, (2, 2)\}$ leads to $(B/2, B/2)$. Surprisingly though, $\{B, (2, 1)\}$ does not lead to $(B/2, B)$ as one would expect but instead to $(B/2, B/2)$. More generally, our experiments show that $\{B, (n_1, n_2)\}$ leads to allocation $(B/\max(n_1, n_2), B/\max(n_1, n_2))$. Note that, as surprising as this result may seem, it is not a simulation artifact and can easily be observed on real-world networks, with host pairs connected via a direct full-duplex link and host pairs connected via a sequence of full-duplex links. Using network monitoring tools, namely `tcpdump` and `wireshark`, we were able to understand that link under utilization is due to the *interference* between acknowledgments for the traffic in one direction and the traffic in the other direction (the reverse traffic). Acknowledgment packets are queued with data packets from the traffic, thus slowing down the reverse traffic. It turns out that this phenomenon is known in the network community as *ACK compression* [ZSC91], and is very common for ADSL connections. In perfectly

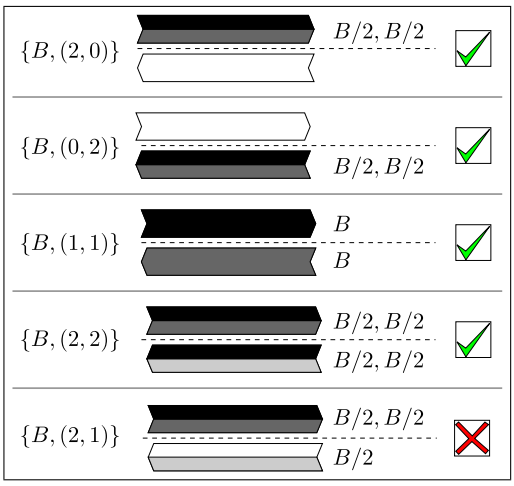


Figure 10.1: Five reverse traffic interference scenarios: $\{B,(n_1,n_2)\}$ denotes a scenario with one full-duplex link of capacity B, with n flows in one direction (shown above the dashed lines) and p flows in the reverse direction (shown below the dashed lines). The bandwidths allocated to the flows are shown on the right-hand side of each scenario. The last scenario shows an asymmetric situation.

symmetrical cases, although *ACK compression* may still occur, delayed ACKs should make it disappear. As recently noted in [HMBD11], the poor link utilization we observe is more likely to be explained by a *data pendulum* effect (also known as *ACK-clocking*) where data and ACK segments alternatively fill only one of the link buffers. At any rate, such a phenomenon, which is not modeled by any of the previously mentioned flow-level models, results in seemingly over-utilized bottleneck links in flow-level simulations.

It turns out that, under the constraints in Eq. (10.2), our flow-level model is inherently incapable of capturing such reverse-traffic effect. However, it is surprisingly straightforward to account for such effect by replacing Eq. (10.2) by the following constraint:

$$\forall l \in \mathcal{L}, \sum_{f \text{ going through } l} \varrho_f + \varepsilon \cdot \left(\sum_{f\text{'s ack through } l} \varrho_f \right) \leqslant B_l \ . \tag{10.10}$$

Although it can hardly be justified by a bottom-up analysis, such simple modification allied with max-min sharing reveals particularly effective in our context and greatly improves the quality of simulations.

As a summary, given a connected network that consists of a set of links $\mathcal{L}$, in which each link l has capacity B_l and a set of flows $\mathcal{F}$, where each flow is a communication between two network vertices along a given path, the current model of SIMGRID shares bandwidth according to the following optimization problem:

$$\begin{array}{l} \text{MAXIMIZE RECURSIVELY } \min_f \left(RTT_f \varrho_f\right) \\ \text{UNDER CONSTRAINTS} \\ \begin{cases} \forall l \in \mathcal{L}, \displaystyle\sum_{f \text{ going through } l} \varrho_f + \varepsilon \cdot \left(\sum_{f\text{'s ack through } l} \varrho_f \right) \leqslant B_l \\ \forall f \in \mathcal{F}, \varrho_f \leqslant W_{\max}/(2L_f) \end{cases} \text{, where } \begin{cases} RTT_f = \displaystyle\sum_{l \text{ traversed by } f} \left(\ell_l + \frac{\gamma}{B_l}\right) \\ L_f = \displaystyle\sum_{l \text{ traversed by } f} \ell_l \end{cases} \end{array} \tag{10.11}$$

Once such shares are determined, completion time is evaluated with Eq. (10.7) (and possibly reevaluated whenever other flow complete or are created). Note that the notions of RTT_f and L_f could certainly be unified and account more specifically for route asymmetry but no situation ever motivated such effort yet.

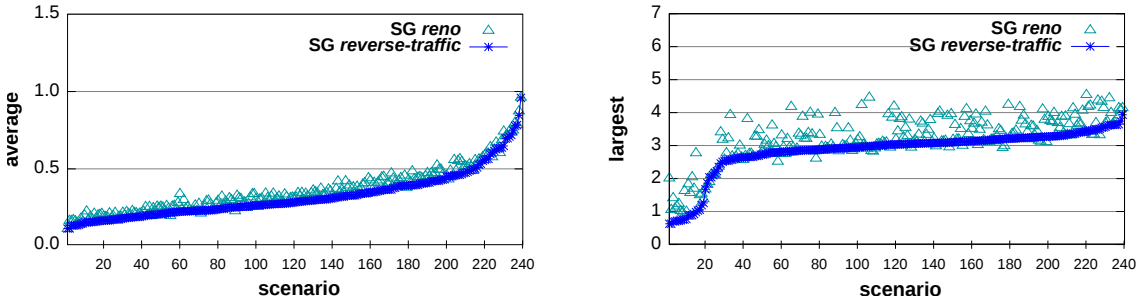


Figure 10.2: Mean logarithmic error (left) and max logarithmic error (right) for all experimental scenarios for the reverse-traffic aware model based on max-min, $SG_{reverse\text{-}traffic}$, and the arctan-based model in [Low03b], SG_{reno}.

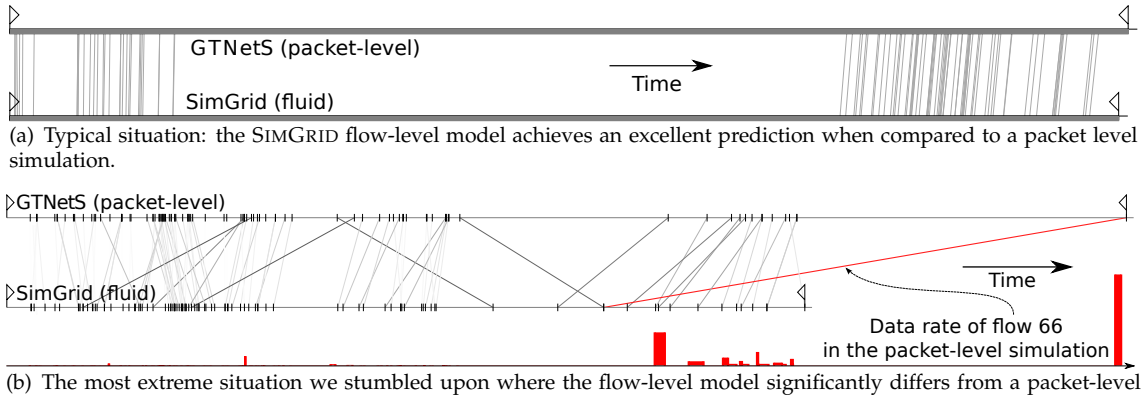
10.2.2 Following the (In)validation Path

The earliest flow-level model of SIMGRID [52] used $\alpha = 1$, $\beta = 1$, $\gamma = 0$ and no modeling of reverse traffic. It was proved to be a good approximation for large messages (above 10 MiB) and relatively large bandwidth values such as the scenarios used in the "validation" of this model in [CM02]. However, whenever we tried more complex scenarios, we quickly ran into troubles that were difficult to explain. Therefore, we engaged into a systematic exploration of configurations and investigation of failing scenarios, which led us to identify all the previous possible improvements.

Indeed, beyond an improved network model for simulation purposes, one of our contributions is the method we use for accomplishing our goal. Rather than showing that a model is valid by exhibiting a possibly large set of scenarios in which the model leads to good results, we instead strive to identify cases that "break" the model. This approach follows the *critical method* [Pop72], which places model refutation at the center of the scientific activity. The main implication is that inductive reasoning, by which one accumulates observations in which the model is valid, should be banned. Instead, the quality of a model should be established by searching for situations that invalidate the model. Invalidation is done via *crucial experiments* that invalidate the current model, thus motivating the need for a new model. This new model should stand the test of these crucial experiments and should also explain why the refuted model is invalid in these experiments. As a consequence, model parameters that do not have clear significance but make it possible to "fit" a model to a set of observations are to be avoided.

What we present in [33, 5] is thus rather atypical in that we repeatedly show "poor" results, while the vast majority of published works strive to show "good" results instead. As an illustration, Fig. 10.3 compares typical outcomes of invalidation studies presented in [5]. Nevertheless, following our method, we are able to measure and understand the inherent limitations of an entire class of network modeling techniques.

Therefore, we evaluated our model on a few hundreds of different scenario. Typically, a hundred of 100 MiB transfers are launched between random pairs of endpoints for several dozens of randomly generated network topologies such as the ones depicted in Fig. 11.1 (see [5] for more details). Yet, in spite of good average results, occasionally a flow has a throughput that is more than a factor 20 away from that obtained using packet-level simulation. Consequently, except for a few ones, many of these scenarios fall into the "accumulating cases in which the model is effective" category instead of being "crucial experiments." Inspecting the good scenarios in detail, we noticed for example that most flows are throughput-limited by their RTTs. Such flows are good cases for a flow-level model because the constraints in Eq. (10.8) essentially bypass the core of the bandwidth sharing formulation (i.e. max-min fairness vs. proportional fairness). Therefore, we modified our set of scenarios by aggressively decreasing network link bandwidth. While such low bandwidth values are likely uncommon and impractical in today's (wired) networks, they increased the number of flows that are not RTT-bound and hence lead to scenarios that stress the



(a) Typical situation: the SIMGRID flow-level model achieves an excellent prediction when compared to a packet level simulation.

(b) The most extreme situation we stumbled upon where the flow-level model significantly differs from a packet-level model for several flows. The lower part depicts the evolution of the throughput of the last finishing flow in the packet-level simulation and for which the flow-level model has a large error. Because of high contention, this flow stalls 65% of the time. When there is no other remaining connection in the network, it does not transmit a single byte for 380 seconds, and finally completes the transfer in a few seconds.

Figure 10.3: Comparison of flow completion times with flow-level simulation (top timeline) and packet-level simulation (bottom timeline); lines connecting timeline markers correspond to the same flow in both simulations and are darker for larger completion time mismatches.

core of the model, i.e., the bandwidth sharing algorithm. Such new topologies where contention is very common indeed lead to situations with larger errors, making "bad cases" easier to identify. A careful analysis of such bad cases lead us to identify the reverse traffic influence described in the previous section.

Such bad cases suggested adjustments and improvements to our model. With the improved model currently implemented in SIMGRID, most scenarios lead to good accuracy, as seen for instance in Fig. 10.3(a). Only a few situations remain, as that shown in Fig. 10.3(b), where we seem to reach the limits of the flow-level approximation. These situations occur for highly contended scenarios (i.e., with extremely small latencies and low bandwidth capacities). In these scenarios the high error is due to the discrete nature of the TCP protocol, which, by design, is not captured by the flow-level approximation.

To the best of our knowledge, our max-min based model augmented with reverse-traffic support is the best flow-level model of TCP available to date to drive simulations in the context of grid/cloud computing. By contrast, some of these modifications (in particular the one accounting for reverse traffic) seem out of reach for the flow-level models proposed by others in the literature (see Fig. 10.2 and [5] for more details). In fact, these models were built in a bottom-up fashion, completely ignoring reverse-traffic effects. Furthermore, their "validation" had been done by illustrating their accuracy on a few simple cases. It is likely that another model (like our max-min-based model) would have produced the same results in these simple settings, showing the inherent problem with a methodology that only explores "good cases."

Our flow-level network model was originally designed for the grid computing domain, which involves very large data transfers. However, with the above improvements it is applicable to other scenarios (e.g., under-provisioned networks, applications that exchange as few as a few hundreds of KiB of data), thereby making SIMGRID more versatile.

10.3 Adaptation to the HPC Context

The improved model described in the previous section expands the versatility of the simulator toward a broader range of wide-area networks, provided the simulated applications exchange messages on the order of a few hundred KiB. At the other end of the spectrum, many users wish to simulate cluster platforms that consist of compute nodes connected via (a hierarchy of) switches. The goal is to simulate a single cluster, or to simulate intra-cluster phenomena in a grid

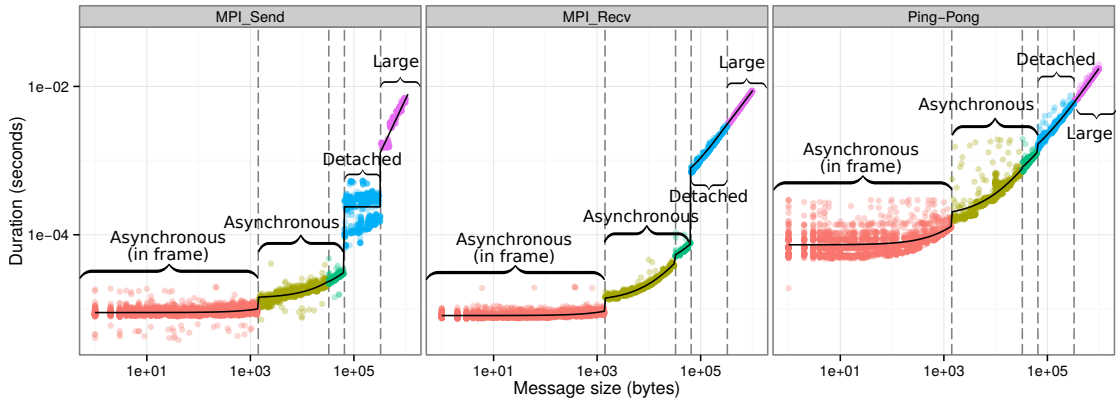


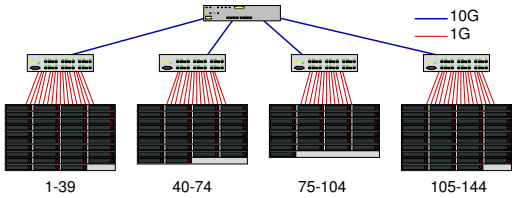
Figure 10.4: Duration of `MPI_Send`, `MPI_Recv`, and a ping-pong (a send immediately followed by a receive of the same size) vs. message size. Different modes can be seen depending on message size. Solid lines represent piece-wise linear regressions.

or cloud platform. In such settings the communication workload often comprises many small messages that consist of a few KiB or even only a few bytes.

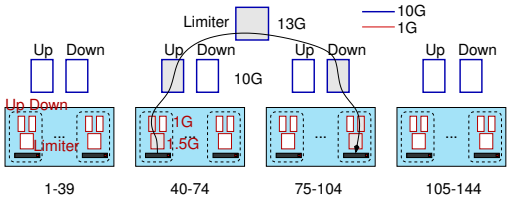
The previous flow-level model fails to capture some fundamental aspects of cluster interconnects with TCP and popular MPI implementations, e.g., OpenMPI [GFB+04] or MPICH2 [Gro02], over Gigabit Ethernet switches. For instance, a message under 1 KiB fits within an IP frame, in which case the achieved data transfer rate is higher than for larger messages although latency is generally also larger. More importantly, implementations for `MPI_Send` typically switch from buffered to synchronous mode above a certain message size. The former involves an extra data copy, while the latter avoids it because copying large amounts of data has high overhead. This "protocol switching" feature is seen in both OpenMPI and MPICH2. Due to such effects, instead of being a linear function of message size as in Eq. (10.7), communication time is rather *piece-wise linear*. Furthermore, depending on the mechanism, communications may be overlapped or not by computations or other communications, which ideally the simulation model should capture. Such synchronization and overlapping aspects can be partially accounted for by the classical LogP family of models [CKP+93, AISS95, KBV00, IFH01], and in particular LogGPS [IFH01].

Fig. 10.4 shows elapsed time vs. message size, using logarithmic scales for both axes, obtained from an experiment conducted with OpenMPI 1.6 on the *graphene* cluster of the Grid'5000 experimental testbed. The measurements were obtained as follows. To avoid measurement bias the message size is exponentially and randomly sampled between 1 byte and 100 MiB, for two types of experiments: "ping" and "ping-pong". The ping experiments aim at measuring the time spent in `MPI_Send` (resp. `MPI_Recv`) by ensuring that the receiver (resp. sender) is always ready to communicate. The ping-pong experiment consists in sending a message and immediately receiving a message of the same size, which allows us to measure the transmission delay. The goal is to study the behavior of MPI from the application's point of view, without any a priori assumptions about the way in which the MPI implementation switches between communication protocols depending on message size, which is in general difficult to determine based solely on the MPI configuration parameters.

Protocol switching effects are clearly seen in Fig. 10.4. For messages below 32 KiB fully asynchronous communication is used, for messages above 256 KiB fully synchronous communication is used, and in between partially synchronous or "detached" communication is used. Each protocol leads to elapsed times that can be accurately modeled through linear regression. Up to three linear regressions, however, should be used for asynchronous communication depending on message size. For instance, a separate mode is required to capture accurately the case when the message is small enough to fit inside a single TCP frame. Overall, we have five distinct modes (modes 2 and 3 are almost identical for the `MPI_Send` and the Ping-Pong results), for an overall



(a) This cluster comprises 144 2.53 GHz Intel Xeon Quad-core X3440 nodes spread across four cabinets, and interconnected by a hierarchy of 10 Gigabit Ethernet switches.



(b) Modeling the graphene cluster: rectangles represent capacity constraints. Grayed rectangles represent constraints involved in a communication from node to node 40 to node 104.

Figure 10.5: The graphene cluster: a hierarchical Ethernet-based cluster.

behavior that is discontinuous and piece-wise linear. The simple linear model in the previous section would be reasonably accurate for small and large messages, but largely inaccurate in between (for more than 30% average error overall, with a worst case at 127%). Likewise, the classical LogP models [CKP+93, AISS95, KBV00, IFH01] do not model the piece-wise linear behavior accurately. The closest contender would be LogGPS [IFH01], but it distinguishes between only two kinds of message sizes (small and large) and is continuous.

In [CSG+11], Clauss *et al.* have proposed a (non-necessarily continuous) piece-wise linear simulation model that can be instantiated for an arbitrary number of linear segments. We extended this model later in [24] to include the overlapping and synchronization aspects, as modeled by LogGPS. This new piece-wise linear model makes it possible to simulate accurately applications that use a wide range of message sizes [24]. The accuracy improvements due to the piece-wise linear model rather than the linear model are large enough to justify the increased number of parameters (2 parameters per mode). In addition, the value of these parameters are easily determined via straightforward experiments and analysis.

An important implication of such accurate piece-wise linear model of point-to-point communication is that, when combined with the bandwidth sharing model described in the previous section, it leads to an immediate simulation model for collective communication operations. Based on topological information, we conduct saturation experiments that enable to identify potential bottlenecks and reverse-traffic interferences to account for (see Fig. 10.5). Such information are critical to simulate the collective operations, which are accessible to the SIMGRID user via the SMPI API (Fig. 9.1). Just like any MPI implementation, the SMPI runtime implements collective communications as sets of point-to-point communications that may contend with each other on the network. This is to be contrasted with monolithic modeling of collective communications, as done in [TLCS09] or [BLGE03] for instance, which rely on coarse approximations to model contention and/or on extensive calibration experiments that must be performed for each type of collective operation.

Indeed, several algorithms exist for each collective operation, each of them exhibiting very different performance depending on various parameters such as the network topology, the message size, and the number of communicating processes [FYL06]. A given algorithm can commonly be almost an order of magnitude faster than another in a given setting and yet slower than this same algorithm in another setting. Every widely-used MPI implementation thus provides several algorithms for each collective operation and carefully selects the best one at runtime. For instance, OpenMPI provides a dozen distinct algorithms for the `MPI_Alltoall` function, and the code to select the right algorithm for a given setting is several thousand lines long. Note that the selection logic of the various MPI implementations is highly dependent on the implementation and generally embedded deep within the source code. To this end, SIMGRID now implements **all** the collective algorithms and selection logics of both OpenMPI and MPICH and even a few other collective algorithms from Star MPI [FYL06]. These different modeling efforts combined with highly technical developments to enable unmodified applications to run on top of SIMGRID have allowed us to obtain excellent prediction results, which we briefly present in Section 12.1.1.

10.4 Conclusion

The network model of all other grid simulators we could test are over-simplistic and can easily be invalidated. To the best of our knowledge, the max-min based model currently implemented in SIMGRID and which has been augmented with reverse-traffic support is the most accurate one to date to drive simulations in the context of grid/cloud computing.

This model has been improved over years thanks to the use of critical method, i.e., thanks to thorough invalidation campaigns and a careful investigation of pathological scenarios. The conclusion of this study is twofold:

1. Flow-level models account for network topology and contention and are quite accurate in a wide range of complex heterogeneous settings. From our experience, they start becoming blatantly inaccurate when the steady-state assumptions break, which occurs for example in extremely contended cases.

2. Bottom-up models have a sound justification and were thus much more promising than the top-down max-min model. Unfortunately, the absence of interference between acknowledgments and data traffic is one of the main assumptions used to build such bottom-up flow-level models, which makes them difficult to use a basis for simulation. The fact that the max-min model can actually be modified to (at least in restricted settings) account for reverse traffic is only a fortunate event (see [5] for more details) and not at all a will to stick with our initial choice.

Finally, although we were not network protocol experts, it is interesting to mention that the use or our critical method lead us to rediscover along the way several non trivial phenomenon like *phase effects* [FJ92] and *reverse traffic influence*, as well as the inherent flaw of previously proposed bottom-up models, which was actually known in the community in which these papers had been published. Indeed, another interesting conclusion of this work is that it illustrates how much we should not blindly trust the results of others. Reading the numerous articles published on bottom-up flow-level models really made it believe that they were exactly answering our needs. These models had even been validated and compared with packet-level simulations even sometimes in not completely trivial scenarios. However, it is only after we clearly identified the reverse traffic issue that we realized that none of the scenario tested to validate the bottom-up models had flows going in reverse directions. Indeed, such scenarios were rather illustrations supporting the interest of the model and they should hence not be considered as a model validation. Trying to reproduce the work of others although time-consuming and not easily rewarding turned out to be very instructive.

Articles attempting to fix the inability of bottom-up models to handle reverse traffic were published in 2008 [TAJ+08, JAT+08] and 2010 [TAJ+10], i.e., in the middle of our invalidation campaign but we only noticed this work a couple years after. Such modeling effort is to the best of our knowledge still ongoing research and although these improved models are very promising, it is unclear whether they can be used effectively in the context of simulation (this is admittedly not their intended use, since the aforementioned works focus on understanding TCP behavior rather than producing instantiable simulation models). First, all these models require solving complex differential equations, which would likely prove unscalable if used to drive simulations with hundreds of flows. Second, although such work is motivated by burstiness caused by reverse-traffic, the reverse-traffic considered in the validation experiments of [TAJ+08, JAT+08, TAJ+10] is always UDP-based and does not saturate links in the opposite directions, hence leaving room for acknowledgment packets. This situation does not correspond to all relevant simulation scenarios. Third, the experimental scenarios used to evaluate the models involve at most three nodes and three flows, and one may wonder how the models would behave if used for simulating hundreds of flows. Consequently, although in the future effective bottom-up simulation models could arise, for the time being our top-down model above appears to be the best available flow-level network simulation model to date, at least in the context of large-scale grid/cloud computing simulations.

Again, not being network protocol experts made this study quite difficult, not only because of understanding protocol peculiarities, but because some of the knowledge and experience can hardly be found by simply reading articles. Our investigation of flow-level models has somehow reached its end because the remaining erroneous behaviors we have can be imputed to violations of the steady-state assumption, which is the central tenet of flow-level modeling. A future direction would consist of providing sufficient conditions for the steady-state assumption to be invalid. This would allow us to better identify the validity domain of flow-level models and allow further investigation of invalidation scenarios. Ultimately, building on such results, a simulator relying on flow-level models should warn its users when it wanders outside the validity domain of its simulation models. Defining such domain remains however quite challenging if not illusive.

Chapter 11

The Scalability Quest

This chapter builds on Pedro Velho's PhD thesis [Vel11], on an article published at LSAP'10 [31] with Pedro Velho and Bruno Donassolo, on an article published at CCGrid'12 [27] with Martin Quinson, Frédéric Suter and other colleagues, and on an article recently published in JPDC'14 [3] with Henri Casanova , Arnaud Giersch, Martin Quinson, and Frédéric Suter. This work has been conducted in the context of the USS-SimGrid and SONGS ANR project.

Striving to make SIMGRID more versatile (so that it can be used for, e.g., volunteer computing or exascale HPC simulations as well as peer-to-peer simulations) has led us to tackling the scalability challenge along several directions. We start by showing that scalability is preconditioned on the use of an analytical simulation model, such as that described in the previous chapter, instead of more commonly used packet-level models. Yet, three major scalability concerns, both in terms of memory footprint and CPU time, remain: (i) the efficiency of the implementation of the simulation model; (ii) the description of large platforms; and (iii) the simulation of large numbers of concurrent processes. In the next three sections we describe how SIMGRID addresses these three concerns. We also provide quantitative comparisons to state-of-the-art domain-specific simulators for relevant case studies in the areas of volunteer computing, grid computing, peer-to-peer computing, and HPC. Together, these case studies illustrate how our scalability solutions developed for specific domains can in fact be combined and applied to different domains.

11.1 Flow-level Models vs. Packet-level Models

The simulators that are used in the grid/cloud computing domain rely on various simulation models for compute components (e.g., servers) and network components (e.g., routers, links, network cards). Models that capture the operation of these components in detail (e.g., cycle-level simulation of a processor, packet-level simulation of a router) generally prove intractable when simulating applications with large computation and communication workloads on large-scale systems with many components. For example, packet-level network simulations are discrete-event simulations with events for packet emission or reception as well as network protocol events. These simulations can reproduce the movements of all network packets and the behavior of the whole TCP stack down to the IP level. Packet-level simulations have been widely used for studying fine-grained properties of network protocols. The most popular packet-level simulator is NS2 [IH08], while more recent simulators include NS3 [NS3], GTNetS [Ril03] and OMNet++ [VH08]. The TCP stack models found in simulators such as GTNets or NS2 are simplified versions of the TCP stack. More recent developments [JM07] even allow the use of real TCP stack implementations, which is slower but more realistic (i.e., real TCP stack implementations might have features/bugs that are absent from simplified versions used exclusively for simulation purposes).

Unfortunately, in the context of simulation of grids or clouds, it can lead to long simulation times since the life cycle of each packet is simulated through all protocol layers all the way to

a simulated physical layer. Its use is thus often restricted to studying network protocols or applications that exchange small messages. An example of such application is a peer-to-peer Distributed Hash Table implementation, and peer-to-peer simulators have been developed that use standard packet-level simulators [BHK07]. Nevertheless, some simulators provide the option of using realistic packet-level simulation in spite of its high overhead (e.g., the iCanCloud cloud simulator [NVPC+11]).

An alternative to expensive packet-level modeling, which still makes it possible to account for network contention, is flow-level modeling as we described in Section 9.2. In flow-level models, which are used by simulators in various domains [BCM+03, 35, OPF10, GB02], each communication, or *flow*, is simulated as a single entity rather than as a set of individual packets and assuming that the flow has reached *steady-state* determining communication time amounts to share bandwidth. The superiority in term of speed of one approach over the other obviously depends on the input workload [YFG+01, YGK+99] and on the quality of the implementation although one expects fluid simulation to be significantly faster for "stable" situations. The goal of this section is to provide the reader a feeling of by how much the performance of such models can differ and why fine grain packet-level models are definitely inadequate for studying large-scale distributed systems. A similar preliminary study was also done by Casanova *et al.* [FC07].

To this end, we compare the default SIMGRID model, whose accuracy was discussed in the previous chapter, with GTNetS [Ril03]. Since GTNetS uses fine grain simulation the GTNetS model is expected to be much slower than SIMGRID's analytic models although. The goal of the following is to evaluate in simple settings whether such speed difference is acceptable or not.

We use two small-scale 50-node platforms. On the left side of Fig. 11.1, we present the results obtained with a 50-node platform generated with the Waxman model [Wax88] and the BRITE generator [MLMB01]. In this platform there are several alternate network paths but the unstructured communication patterns of the simulated application still leads to interference among communications in the network. On the right side of Fig. 11.1 we present results with a 50-node platform generated with the Tiers algorithm [CDZ97], which uses a three-step space-based hierarchical approach. The resulting topology is hierarchical with low bisection bandwidth, and has thus both global (in the core of the network) and local (on the edges of the network) bottleneck links.

We simulate a synthetic application that creates several independent transfers of a fixed size randomly distributed over the platform and waits for all of them to complete. The timings presented here use an AMD Opteron 248 Dual Core (2.2 GHz) with 1MB of L2 cache and 2 GB of RAM. Since execution time is slightly variable from one execution to another we performed each experiment at least 5 times each and the 95% confidence interval based on Student's distribution. All graphs are plotted with the confidence interval enclosing each point although most of the time confidence intervals are too small to be perceived with naked eye.

In our first comparison, we fix the message to 20MB and vary the number of concurrent flows (see Fig. 11.1). Such message size may seem arbitrarily high but in a grid or HPC context, it is very common to transfer large amount of data at once. Varying the number of simultaneous communications directly increases simulation time by increasing the number of variables inside the Max-Min system. As we can observe, SIMGRID is more than one order of magnitude faster than GTNetS. As an illustration, simulating 320 concurrent communications on the Tiers platform requires more than 7 minutes with GTNetS while less than 1 second is required with SIMGRID.

Another major advantage of the macroscopic flow-level model is its independence on message size. To illustrate this, we vary the message size of 80 concurrent transfers (see Fig. 11.1). As expected, SIMGRID simulations are independent of message size while the duration of GTNetS simulations grows linearly with message size, quickly leading to prohibitive simulation times in the context of large scale distributed computing systems.

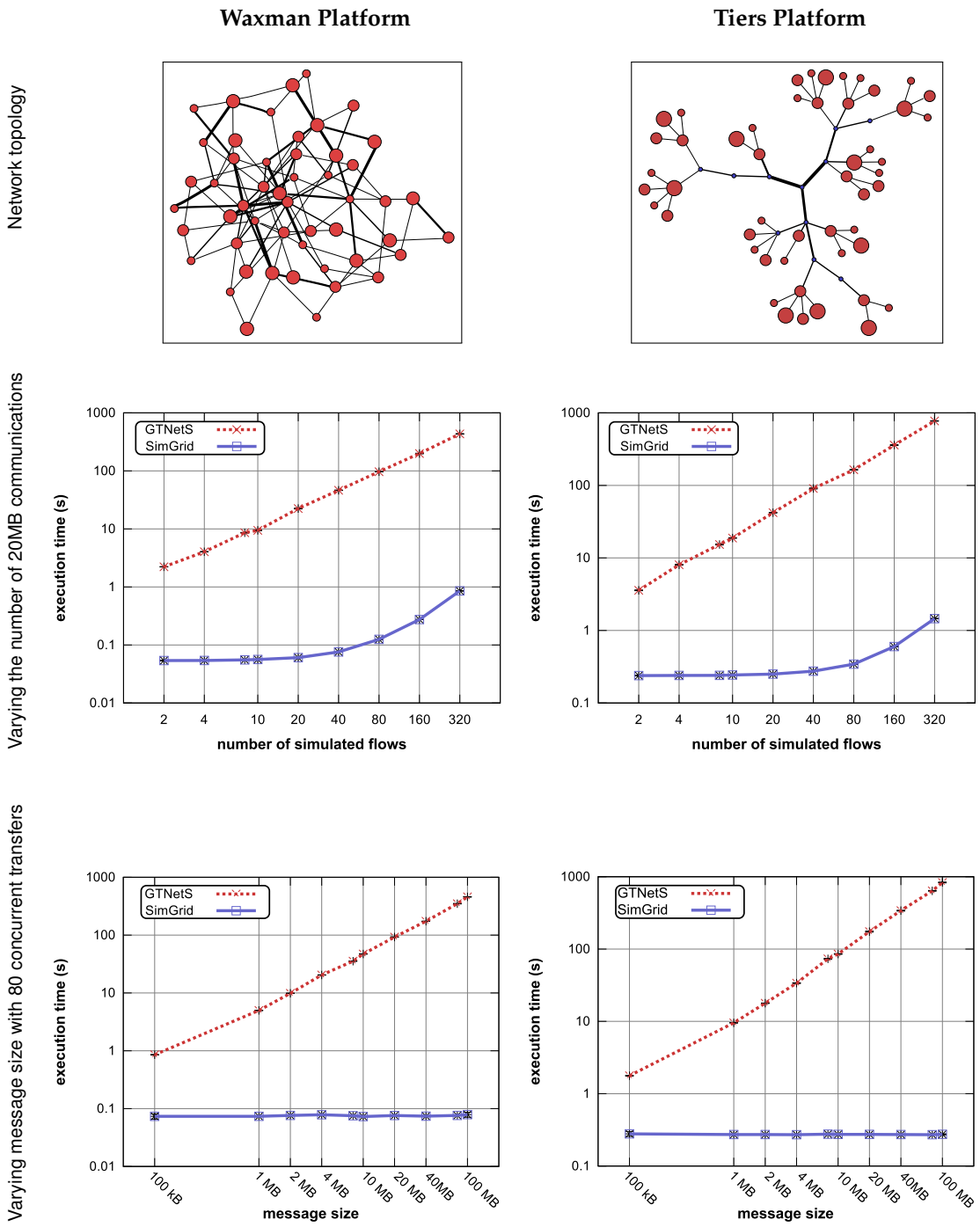


Figure 11.1: SIMGRID versus GTNetS response time when varying the number of simulated flows or the size of messages for two 50-node platforms Waxman (left) and Tiers (Right). GTNetS simulations are much several orders of magnitude slower than SIMGRID in spite of results being within a few percents except for highly contended situations (i.e., the situations with 320 flows for Waxman and 80+ flows for Tiers). As expected, GTNetS performance depends on message size while SIMGRID does not, which is critical when studying communication workloads like those encountered in Grid or HPC context.

11.2 Efficient Simulation Kernel

As explained in the previous chapter, the base simulation model in SIMGRID relies on a steady-state assumption to compute resource shares allocated to pending simulated activities. As a result, each time the set of these activities changes (a new activity is started, a current activity completes), the resource shares must be reevaluated, which amounts to solving a linear system of equations (Eq. (9.1)). The computed resource shares are then used to determine (i) by how much the simulated clock should be advanced and (ii) the progress of each pending activity. Such resource sharing approach is somehow involved compared to what can be found in classical discrete-event simulators. We describe in this section most of the tricks and techniques we resorted on to obtain a scalable implementation of such model. We conclude by two illustrations of their effectiveness.

11.2.1 Efficient Resource Sharing

As we explained in Chapter 10, fair-share of network resources at the flow-level is implemented in several simulators [BCM+03, OPF10, GB02] but incorrectly. The reason for this is that despite a rather simple formulation a correct and efficient implementation can be quite involved. In particular, since simulated activities dynamically appear and disappear during the simulation, an efficient implementation required ad hoc dynamic and sparse data structure to represent Eq. (9.1). In this section, we describe the main algorithmic and implementation principles underlying the SIMGRID implementation of resource sharing.

In its simplest form, the max-min version of Eq. (9.1) is written:

$$\begin{array}{l}\text{MAXIMIZE } \min_i \varrho_i \\ \text{UNDER CONSTRAINTS} \\ \left\{\forall r, \sum_{i \in \mathcal{A} \text{ using resource } r} \varrho_i \leqslant C_r,\right.\end{array}$$

Let us denote as ε the largest possible value for $\min_i \varrho_i$. Since it cannot be enlarged, this mean that at least one constraint of a resource r is tight, i.e., that $\sum_{i \in \mathcal{A} \text{ using resource } r} \varepsilon = C_r$. Such resource is then said to be *saturated*. If we denote by n_r the *load of constraint* r, i.e., the number of activities a using resource r, we have thus $\varepsilon = C_r/n_r$ and $\varepsilon \leqslant C_{r'}/n_{r'}$ for all other r'. Therefore, ε can simply be defined like $\min_r C_r/n_r$ and the (possibly several) r corresponding to this minimum are thus *saturated constraints*. All variables linked to such constraints are *saturated variables* and should receive a resource share ϱ_i to ε. Other variables could however receive a larger resource share which has still to be defined. Therefore, once *saturated variables* have been allotted resource shares, the *remaining capacity* of all the other constraints should be updated accordingly. Such *capacity update* requires to iterate over all constraints linked to saturated variables. Once this is done, the same process can be repeated again with free variables until they all have been saturated.

This leads us to the algorithm given in Fig. 11.2. In the case of SIMGRID, the algorithm is slightly more complicated as variables i have weights w_i and resource usage may not be uniform, i.e., the previous optimization problem is actually written:

$$\begin{array}{l}\text{MAXIMIZE } \min_i w_i \varrho_i \\ \text{UNDER CONSTRAINTS} \\ \left\{\forall r, \sum_{i \in \mathcal{A} \text{ using resource } r} a_{r,i} \varrho_i \leqslant C_r,\right.\end{array}$$

The solving principle remains however the same. If we denote by ε the largest possible value for $\min_i w_i \varrho_i$, then we derive that

$$\varepsilon \leq \frac{C_r}{\sum_{i \in \mathcal{A} \text{ using resource } r} \frac{a_{r,i}}{w_i}}$$

The load on constraint r should thus be defined as $\sum_i \in r \frac{a_{r,i}}{w_i}$. The saturated constraints are thus those reaching this minimum and saturated variables need to be set to ε/w_i. Constraint capacities

```
1: Let R_a be the set of active constraints, i.e., those linked to variables with a non-negative
   weight.
2: repeat
3:   for all active constraint r ∈ R_a do
4:     Compute the constraint load n_r and the constraint demand C_r/n_r.
5:   Let ε be the smallest demand and R_s be the set of saturated resources, i.e., the ones
     achieving such minimum.
6:   Let S be the set of saturated variables, i.e., those linked to a saturated resource r ∈ R_s.
7:   for all variable i in S do
8:     ϱ_i ← ε
9:     for all constraint r linked to i do
10:      Update the capacity of r, i.e., C_r ← C_r − ϱ_i
11:      Update the load of r, i.e., n_r ← n_r − 1
12:      if C_r = 0 or n_r = 0 then
13:        Remove r from R_a
14: until there is no more active constraint
```

Figure 11.2: Sketch of the max-min share algorithm implemented in SIMGRID.

have to be updated with $a_{r,i}\varrho_i$ and constraint demands have to be updated with $a_{r,i}/w_i$. Furthermore, a special treatment should be given to variable specific bounds (i.e., those corresponding to the RTT bounds of Eq. (10.8)) but it does not change the principle of the algorithm either.

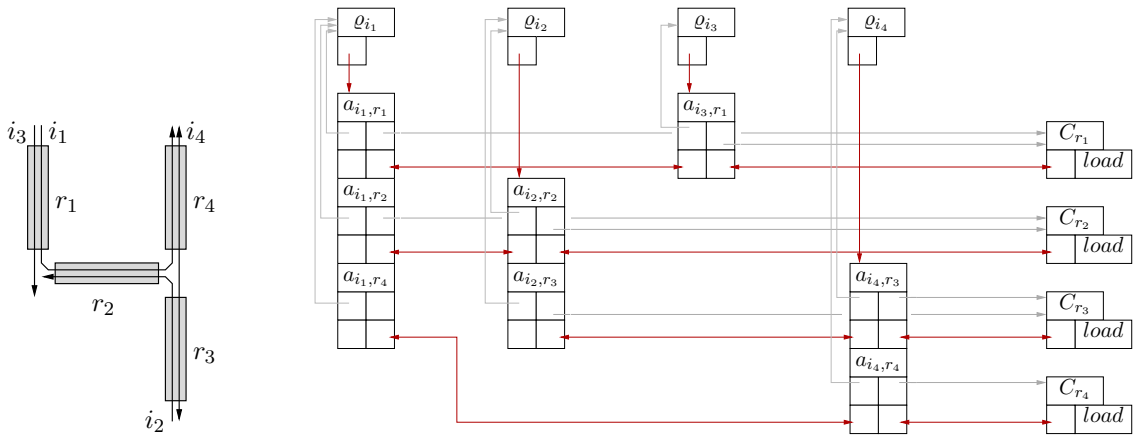
The principle of the max-min fair sharing algorithm is thus quite simple although the recursion, the weights and the non-uniform resource usage make it slightly more complicated. Coming up with an efficient implementation that scales gracefully adds however a level of complexity since we need to maintain and solve such system of equations at minimal cost. From inspecting the algorithm of Fig. 11.2, we can make the following observations:

- The list of constraints r is defined at the beginning of the simulation although the set of active constraints evolves upon activity creation;
- Variables ϱ_i are regularly created and destroyed upon activity life-cycle;
- The connection between ϱ_is and the constraints r is fully known upon the creation of variable ϱ_i;
- We need to iterate over the set of saturated constraints and on the set of active constraints, which both evolve during the execution of the algorithm;
- To efficiently update constraint demand and capacity, it is required to navigate back and forth between the variables ϱ_i and the constraints r.

These observations motivated the *ad hoc* sparse data structure depicted on Fig. 11.3.

Such data structure has redundancy that enable to access any needed value in time $O(1)$ and, allied with a few simple programming tricks to keep useless objects out of sight and to break loops as soon as possible, it allows to compute the resource shares very efficiently. The only operation whose complexity may be improved and amortized over the execution of the algorithm is the computation of the minimum, for example using a heap. However, since such values need to be updated regularly, they would have to be extracted and reinserted in the heap, hence an unclear benefit.

The previous implementation is thus optimal in terms of number of operations, but on particularly large workloads, we realized it suffered from a very poor L1 and L2 cache reuse. To obtain an efficient cache-oblivious implementation we split the data structure in half so as to group together the few fields that are heavily used by the bandwidth sharing algorithm in contiguous arrays. The use of arrays instead of linked lists improves locality and hence the prefetch efficiency. This trimming of data structures is partially done at the beginning of the algorithm while



(a) Typical resource usage.

(b) Corresponding data structure.

Figure 11.3: Illustrating the sparse data structures underlying the max-min resource sharing.

another part is maintained over the calls for a better amortized cost. As one would expect, these optimization lead to improved cache utilization, and thus significantly lowers simulation time, but at the cost of increased implementation complexity.

Beside these data structure optimizations, we have also implemented two algorithmic optimizations motivated by actual large-scale simulation scenarios [31], as detailed in the next two paragraphs.

11.2.2 Lazy Activity Updates

Originally, SIMGRID was intended for the simulation of applications that comprise many communicating tasks running on computers connected by hierarchical networks. In this setting any event related to a simulated activity or resource can impact a large fraction of the other simulated activities and resources. However, when simulating large-scale platforms, such as those used for peer-to-peer or volunteer computing applications, most activities are independent of each other. In this case, reevaluating the full model becomes a performance bottleneck because all activities are examined even though many can simply be ignored most of the time. Our approach is thus to avoid solving the whole linear system in Eq. (9.1) by only recomputing the parts of it that are likely to be impacted by newly arrived or newly terminated activities. Furthermore, if between two resolution of the linear system only a few variables have changed, then only the state of the corresponding activities needs to be updated. Using a heap as a future event set, and efficiently detecting the set of variables that are impacted by activity removal and addition, we are able to lower the computational complexity of the simulation significantly. We term this technique "lazy updates," since the state of a simulated activity is modified only when needed.

11.2.3 Trace Integration

Our second efficiency improvement targets the management of resources whose capacities change frequently. In SIMGRID, the user can specify the capacity of a resource as a time-stamped trace to simulate fluctuating availability due to some out-of-band load (a capacity of zero means a downtime). The linear system in Eq. (9.1) must be reevaluated each time the capacity of a resource changes. In extreme scenarios, many such reevaluations may occur before a single activity completes, which would slow the simulation down unnecessarily. For instance, let us consider a situation in which the capacity of a resource is specified to change 100 times according to a user-specified time-stamped trace. Furthermore, let us assume that all pending activities still have

large amounts of remaining work so that the earliest activity completion occurs after the 100th resource capacity change. In this case, 100 reevaluations of the linear system would take place even though would be possible to perform a single reevaluation. More formally, given current remaining work amounts, one can compute the next activity completion date given all future resource capacity values before this date. This computation can be performed efficiently using "trace integration" (see Fig. 11.4). Essentially, instead of storing a trace as capacity values, one

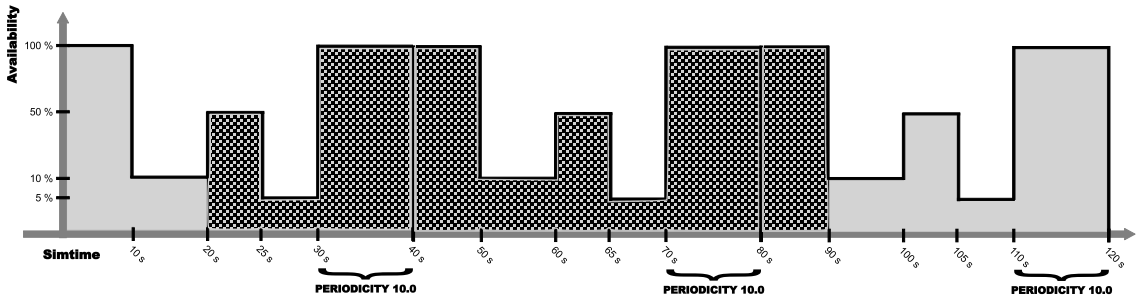


Figure 11.4: Integrating availability trace histograms one is able to advance the simulation to useful actions.

stores its integral. Finding the last resource capacity change before the next activity completion can then be performed using a binary search, i.e., with logarithmic time complexity.

11.2.4 Illustrating Scalabity With a Volunteer Computing Simulation

The scalability enhancements described in this section were initially motivated by the need to simulate large volunteer computing systems efficiently [31]. Let us consider a volunteer computing scenario with N hosts. Each host computes sequentially P tasks, and the compute rate of each host changes T times before the completion of the simulation. With our original implementation the time complexity of this simulation is $O(N^2(P+T))$. With lazy activity updates it becomes $O(N(P+T)\log N)$, and $O(NP(\log(N)+\log(T)))$ when adding trace integration. We have implemented such a simulation, using traces of MFlop/sec rates for SETI@home hosts available from [KJIE10]. Compute tasks have uniformly distributed random compute costs in MFlop between 0 and 8.10^{12} (i.e., up to roughly one day for a median host). Note that such simulation scenarios are commonplace when studying volunteer computing, and in fact this particular scenario was suggested to us by the authors of [HFH08] to highlight scalability issues in previous versions of SIMGRID. Fig. 11.5 shows simulation time measured on a 2.2 GHz AMD Opteron processor vs. N for the initial design, the addition of lazy activity updates, and the addition of trace integration, using a logarithmic scale on the vertical axis. Results make it plain that both proposed improvements decrease simulation time dramatically. For instance, for a simulation with 2,560 hosts, the simulation time is almost at 3h without the enhancements, around 1min with lazy updates, and under 10 s with lazy updates and trace integration. A comparison with the state-of-the-art SimBA simulator [TKE+07], based on timing results published therein and the use of a similar benchmark machine, shows that with our improvements SIMGRID achieves simulation times more than 25 times faster. We refer the interested reader to [31] for more details on the experimental setup. This is an important result given that SIMGRID is more versatile than SimBA. In fact, the behaviors of the network and the software are simulated in much more details in SIMGRID (i.e., flow-level model of TCP, programmatic specification) than in SimBA (i.e., fixed latency, finite automata). Such kind of detail is mandatory to conduct studies like the one of Chapter 7.

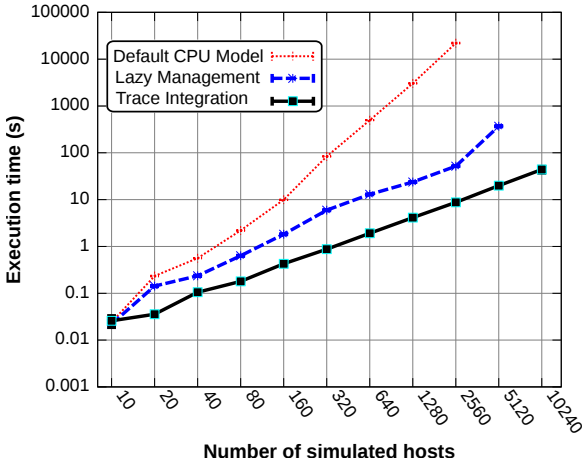


Figure 11.5: Simulation time vs. number of simulated hosts for a volunteer computing simulation using the initial design, with lazy updates, and with lazy updates and trace integration.

11.2.5 Illustrating Scalability With Unstructured Communications Simulation

The trace integration mechanism is specific to CPU simulation, but the lazy update mechanism applies across all kind of resources and activities. To quantify the impact of lazy updates on the

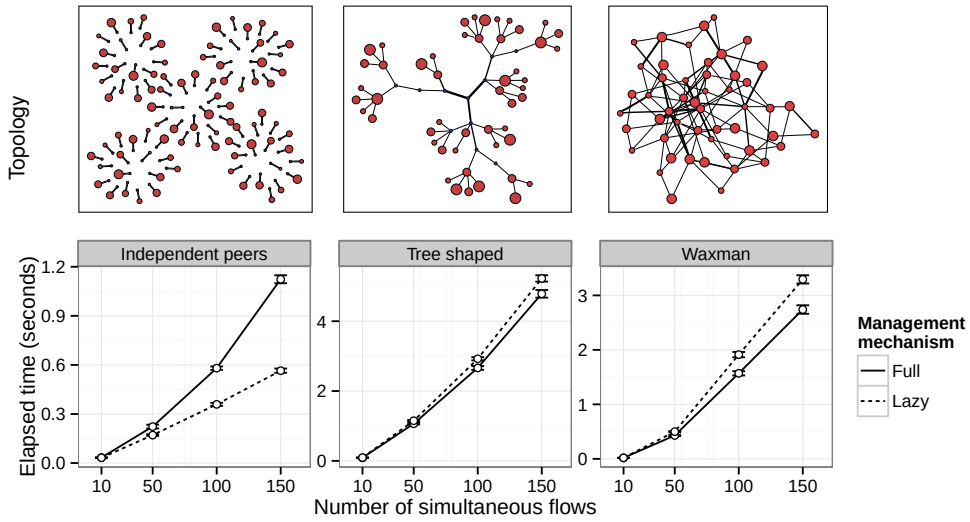


Figure 11.6: Simulation time vs. number of network flows for three different network topologies without and with lazy updates. On loosely connected platforms (e.g., independent peers), the lazy updates mechanism reduces simulation time significantly. On more tightly coupled platforms it can increase simulation time.

speed of network simulation, Fig. 11.6 shows simulation times when simulating various numbers of flows (10, 50, 100, or 150) that are opened and closed at random dates between random pairs of nodes for 1,000 seconds of simulated time, for three representative platforms. A randomized factorial set of experiments with 50 measurements for each combination is run on a 3.3 GHz Core i7 processor and we report the 95% confidence intervals of the average time needed to perform the simulation (platform description parsing time is not included). The first platform consists of 1,740 independent hosts each with its own upstream link and downstream link. When peer A

communicates with peer B, a network flow using the upstream link of A and the downstream link of B is created and the latency of this flow is computed from the link latencies and the Vivaldi [DCKM04] network coordinates of the peers. This platform is thus very loosely coupled, and as such we see in the leftmost graph in Fig. 11.6 that the use of lazy updates reduces the simulation time significantly. The second platform comprises 90 hosts and 20 routers and was created with the Tiers algorithm [CDZ97], which uses a three-step space-based hierarchical approach. The resulting topology is hierarchical with low bisection bandwidth, and has thus both global (in the core of the network) and local (on the edges of the network) bottleneck links. The third platform comprises 200 nodes and is generated with the Waxman model [Wax88] and the BRITE generator [MLMB01]. In this platform there are more alternate network paths but the unstructured communication patterns of the simulated application leads here also to interference among flows in the network. The results in Fig. 11.6 for the second and third platforms show that lazy updates actually increase simulation time. This is because in these platforms the probability that a random flow interferes with another is high. As a result, our lazy updates implementation suffers from some slight overhead when determining that the resource shares of most flows needs to be recomputed. For this reason, lazy updates can be deactivated by the user. However, since lazy updates only incur marginal slowdowns but bring significant speedup when there is locality in the communication patterns, SIMGRID enables them by default.

11.3 Efficient Platform Representation

As can be seen on Fig. 9.1, a specification of resources, of their capacities and of their interconnections is required to create and update the linear system (9.1). For example, when creating a communication from A to B, we need to know the list of network resources used (possibly in both directions) so as to link the newly created variable to all the constraints corresponding to such resources. The simplest and most expressive way to describe a network interconnect is to describe the routing table of each simulated network element explicitly. Unfortunately, this method is quite memory consuming as it grows quadratically with the number of hosts. There are however many situations where network topology exhibit some regularity that could be exploited to reduce memory footprint. For example, when simulating a peer-to-peer system one may not want to precisely simulate the core of the network (, which is often quite difficult to model and instantiate anyway) but one may want to model contention occurring on the edge of the network on ADSL lines. In such case, an effective solution is to assign each host its own upstream link and downstream link. Then, when peer A communicates with peer B, a network flow using the upstream link of A and the downstream link of B is created. Such routing rule can easily be implementing with $\Theta(N)$ memory footprint and a $O(1)$ routing cost. More generally, SIMGRID uses a scalable, efficient, and modular network representation technique, which also drastically reduces the platform description burden placed on users.

SIMGRID's platform representation exploits the *hierarchical* structure of real-world (large-scale) network infrastructures [27], relying on the concept of autonomous systems (AS), including local networks as well as the classical Internet definition. In addition, the representation is recursive within each AS so that regular patterns can be exploited whenever possible. SIMGRID provides stock implementations of well-known routing schemes, including Dijkstra, Dijkstra with cache, Floyd, Flat (i.e., full routing table), Empty routing with Vivaldi network coordinates [DCKM04], cluster (i.e., a regular topology where each node has its own private links and communicates with the others through an additional shared link), and classical HPC topologies such as n-dimensional torus and fat-trees. For the time being, and to favor scalability, SIMGRID assumes that the routing is static over time. This assumption is reasonable (see for instance the study in [BMA06], which shows that less than 20% of Internet paths change in a 24-hour period). Besides, routing changes in real-world networks are known to affect traffic on backbone links. Usually, these links are not communication bottlenecks. Therefore, routing changes can likely be ignored without a large impact on simulation accuracy in a Wide area context. In an HPC context where parallel computers may use dynamic adaptive routing, special adjustments have

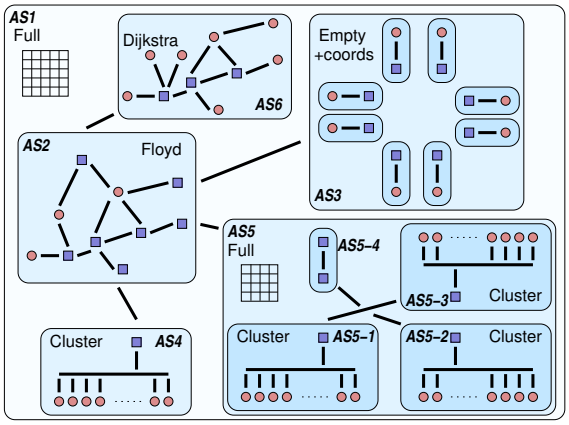


Figure 11.7: Example hierarchical network representation of an AS, AS1. Circles represent compute resources and squares represent network routers. Bold lines represent communication links. Routing schemes are indicated for each AS. AS2 models the core of a national network interconnecting a small cluster (AS4), a larger hierarchical cluster (AS5), a subset of a LAN (AS6), and a set of peers scattered across the Internet (AS3).

to be implemented in the core of SIMGRID. Fig. 11.7 shows an example hierarchical network representation in SIMGRID.

Each AS declares a number of gateways, which are used to compute routes between ASes comprised within a higher-level AS. This mechanism is used to determine routes between hosts that belong to different ASes: simply search for the first common ancestor in the hierarchy and resolve the path recursively. The network representation and this route computation method provide a compact and effective representation for hierarchical networks. Since real-world networks are not purely hierarchical, SIMGRID provides "bypassing" rules that can be used to declare alternate routes between ASes manually.

The above semantic principles of network representation are implemented by the user via an XML file. For convenience, SIMGRID provides a set of XML tags that simplify the definition of two standard and ubiquitous ASes: homogeneous clusters and sets of Internet peers. The cluster tag creates a set of homogeneous hosts interconnected through private links and a backbone, which all share a common gateway. The peer tag allows for the easy creation of peer-to-peer platforms by defining at the same time a host and a connection to the rest of the network (with different upload and download characteristics and network coordinates). SIMGRID also provides an API for generating in-memory network descriptions directly without requiring an XML file.

11.3.1 Illustrating the Efficiency of Platform Representation With Grid Computing Simulations

SIMGRID's network description approach makes it possible to describe large platforms with low memory footprint and without significant computational overhead. It competes in terms of speed and memory usage with state-of-the-art simulators while relying on much more accurate models that are generally considered as prohibitive.

For instance, it allows us to represent the full Grid'5000 platform [BCC+06] (10 sites, 40 clusters, 1,500 nodes) with only 61 KiB. By comparison, the flat representation with SIMGRID v3.2 required 1,065 MiB [FQS08]. It takes more than 4 minutes to parse the flat representation on a 1.6 GHz Intel Core2 Duo with 5 GiB of memory and an SSD drive while the current representation is parsed in less than 150 milliseconds.

We now compare the scalability of SIMGRID to the widely used GridSim toolkit [BM02] (version 5.2 released on Nov. 25, 2010). The experimental scenario is a simple master-worker execution where the master distributes T fixed size tasks to W workers in a round-robin fashion. In

	Simulation Time	Peak Memory Footprint
GridSim	$56\,\text{ms} \times W + 14\,\text{ns} \times T^2$	$2.5\,\text{GiB} + 226\,\text{KiB} \times W + 3\,\text{KiB} \times T$
SIMGRID	$0.1\,\text{ms} \times W + 26\,\mu\text{s} \times T$	$5.2\,\text{MiB} + \quad 80\,\text{KiB} \times W$

Table 11.1: Polynomial fits of simulation times and peak memory footprints of GridSim and SIMGRID for a master-worker simulation with W workers and N tasks. The simulation time is quadratic with T in GridSim while it is linear in SIMGRID. The peek memory footprint of GridSim is several orders of magnitude larger than that of SIMGRID.

GridSim we did not define any network topology, hence only the output and input baud rates are used to determine data transfer rates. By contrast, with SIMGRID we used a full-fledged flow-level network model and the aforementioned Grid'5000 network representation, which models clusters and their cabinets as well as the wide area network interconnecting the different sites. Experiments were conducted using a 2.4 GHz Intel Xeon Quad-core with 8 GiB of RAM. We refer the interested reader to [27] for more details on the experimental setup.

The number of tasks, T, is uniformly sampled in the $[1; 500{,}000]$ interval and the number of workers, W, is uniformly sampled in the $[100; 2{,}000]$ interval We perform 139 experiments for GridSim and 1,000 for SIMGRID (as it was significantly faster), and measure the wall-clock time (in seconds) and the memory consumption (using the Maximum Resident Set Size in KiB as a measurement). As expected, the size (input and output data, and amount of computation) of the tasks have no influence. Experimental results are shown as polynomial fits in Table 11.1. The goodness of fit is high (all coefficients of determinations, or R^2, for all fits are above 0.9972).

The simulation time for GridSim is quadratic in T and linear in W. Surprisingly, GridSim's memory footprint is not polynomial in T and W. Rather, it appears to be piece-wise linear in both (with a very steep slope at first, and less steep as values increase). Furthermore there are a few outlier points that exhibit particularly low or high memory usages (leading to $R^2 = 0.9871$). This is likely explained by the Java garbage collection. For this reason, in Table 11.1 we only report results for scenarios where T is larger than 200,000, which removes most outliers.

Analyzing the results for SIMGRID shows that its simulation time and memory footprint are stable and several orders of magnitude lower. The results reported in Table 11.1 mean that 5.2 MiB are required to represent the Grid'5000 platform and the internals of SIMGRID, with a payload of 80 KiB per worker. By comparison GridSim uses 2.5 GiB with an additional 226 KiB payload per worker. Furthermore, in SIMGRID T has no impact on the memory footprint, which is not the case in GridSim. We conclude that SIMGRID, with its flow network model and a fine-detailed network topology, is several orders of magnitude faster and more memory efficient than GridSim, with its simple delay-based model and no network topology. For instance, while GridSim requires more than one hour and 4.4 GiB of memory to simulate the execution of 500,000 tasks with 2,000 workers, SIMGRID performs this same simulation in less than 14 seconds and with only 165 MiB.

11.4 Efficient Process Representation and Simulator Architecture

SIMGRID allows users to describe the simulated application programmatically as a set of independent but communicating concurrent processes. The goal is to allow users to implement the simulated application in a way that is similar to but simpler than the way in which a real application would be implemented. Such notion of process was not in the earliest versions of SIMGRID designed by Henri Casanova and I introduced it in 2002, providing two different backends. The first one was based on `pthread` and was quite portable although limited in term of scalability. The second one was based on System V `ucontext` and had portability issues on several operating systems (e.g., Solaris and AIX) but had not other scalability limit than the memory available

on the system. This implementation of concurrent sequential process was however quite cumbersome and difficult to maintain. I tried several times, in particular with Bruno Donassolo, to clean this part but with limited success. The good idea to re-architecture this part of the simulator by introducing a mechanism similar to the one of system calls and which is presented in this section came from Martin Quinson, Cristian Rosa and Christophe Thiéry [QRT12].

Due to the optimizations described in the previous sections, for many large-scale simulations the most computationally intensive portion of the simulation is not the evaluation of the simulation model, but instead the execution and the synchronization of the simulated processes! As a result, increasing scalability requires going beyond vanilla implementations based on threads and standard synchronization primitives.

Since the simulation models in SIMGRID can be computed quickly, it is possible and in fact efficient to have a unique execution context (such as a thread) handle all the simulation model computations. We call this context the *core context*, and it interacts with the execution contexts of the simulated concurrent processes. This has led to the layered design shown in Fig. 9.1. At the bottom is the SURF component that runs in the core context and deals with the simulation of the resources and of their usage by the activities issued by the simulated concurrent processes. At the top are the concurrent processes themselves, implemented as user code that places calls to a SIMGRID API (MSG or SMPI) to define activities. In between is a *synchronization kernel*, SIMIX, that mediates every interaction between the simulated processes and the core context.

The synchronization kernel is conceptually close to the kernel of a classical operating system and it emulates a system call interface called *simcalls*. Simcalls are used by simulated processes to interact with the core context. When a simulated process issues a simcall the request and its arguments are stored in a private memory location. The process is then blocked until the completion of the request (e.g., completion of the corresponding simulated activity). When all user processes are blocked in this manner control is passed to the core context. The core context handles the requests sequentially in an arbitrary but deterministic order based on process IDs, and it is the only context that accesses the simulation state. A sequential core context makes for simplified simulation logic due to vastly reduced numbers of context switches between the core context and the simulated processes. To the best of our knowledge it is the first time that this classical OS design is applied to distributed system simulation. An alternate design in which simulated entities actively interact with each other, such as that used for instance in GridSim [BM02], may seem more intuitive but leads to more complex simulation logic due to multi-step interactions between processes/threads.

Our design is scalable only if mechanisms are available to execute thousands or even millions of processes on a single host (standard virtual machine techniques cannot be used to execute our simulated processes as at most dozens of virtual machines instances can run efficiently on a host). The use of regular threads seems like a natural approach, with the code of each simulated concurrent process running in its own thread. But with standard threads, one can scale up to "only" a few thousands simulated processes, thus severely limiting the scale of the simulation. For instance, GridSim, which uses threads, cannot simulate more than 10,000 processes/hosts [DDMVB08]). Instead, we employ cooperative, light-weight, non-preemptive threads (known as *continuations*). They are ideally suited to our needs since our synchronization kernel has to finely control the scheduling of the simulated processes anyway. Additionally, they are much simpler to implement than regular threads. The Windows operating system provides such light-weight execution contexts as *fibers*, while they are called *ucontexts* (for user-contexts) on Unix operating systems, including Mac OSX. In SIMGRID we have aggressively re-implemented a similar mechanism directly in assembly so as to remove a costly and unnecessary system call found in standard implementations.

11.4.1 Illustrating Process Scalability with Peer-to-peer Simulations

In [QRT12], Martin Quinson *et al.* compare the scalability of SIMGRID for a peer-to-peer simulated scenario to that of two popular and reported-to-be-scalable simulators in that domain:

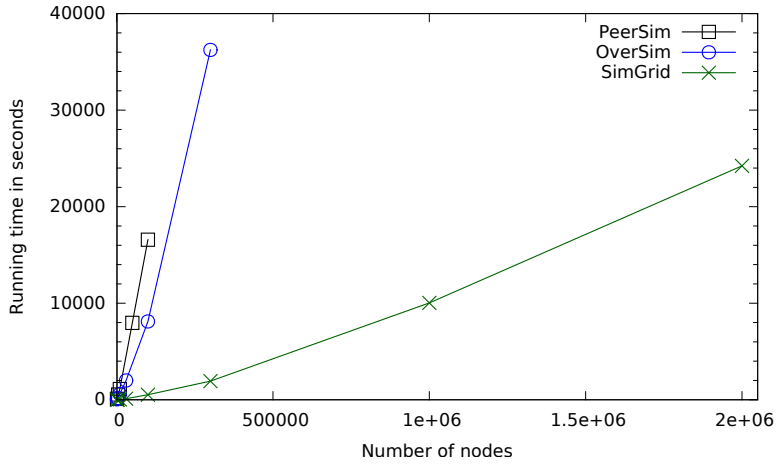


Figure 11.8: Simulation time vs. number of peers for a Chord simulation with SIMGRID (with constant and precise network models), OverSim (with a simple underlay and using the OMNeT++ bindings), and PeerSim.

OverSim [BHK07] and PeerSim [MJ09]. Fig. 11.8 shows the simulation time of the Chord protocol [SMLN+03] vs. the number of simulated peers. For SIMGRID and OverSim they used the experimental scenario initially proposed in [BHK07]: each peer joins the Chord ring at time $t = 0$, then performs a stabilize operation every 20 seconds, a fix fingers operation every 120 seconds, and an arbitrary lookup request every 10 seconds. The simulation ends at $t = 1000$ seconds. For PeerSim, the implementation that is publicly available does not make this experimental scenario possible since there are not stabilize or fix fingers operations. So in the PeerSim experiments a single lookup was generated every 120 seconds.

They used the Chord implementations that are publicly available for OverSim and PeerSim, while they have implemented the Chord protocol themselves for SIMGRID. Therefore, there may be differences (parameters, features, optimizations, or even bugs) among the three implementations of the protocol. To ensure that experiments are comparable in spite of such differences, they tuned the simulated scenario parameters to make sure that the numbers of application messages exchanged during the simulation, and thus the load on the simulator, were comparable across experiments (10,000 peers, 25 million messages). More specifically, they conservatively ensured that more messages are exchanged in the SIMGRID simulation than in the OverSim and PeerSim simulations. Note that the three simulators recorded different information (i.e., simulation event traces), leading to different tracing overheads. However, as seen hereafter, the results show orders of magnitude improvements for SIMGRID over its competitors.

Experiments were conducted on one core of a two-CPU 1.7 GHz AMD Opteron 6164 HE (12 cores per CPU) with 48 GiB of RAM running Linux. OverSim (v20101103) is implemented in C++ (gcc v4.4.5), SIMGRID (SimGrid v3.7-beta, git revision 918d6192) in C (gcc v4.4.5), and PeerSim (v1.0.5) in Java (HotSpot JVM v1.6.0-26). In the experiments, they configured PeerSim so that its simulation model assumes that every communication takes a uniform random amount of time. They configured OverSim to use a simple model in which communication times are based on the Euclidean distance between processes (instead of the less scalable OMNeT++ bindings). By contrast, for this experiment SIMGRID used its full-fledged flow-level model that accounts for more complex network behavior (i.e., contention and TCP congestion avoidance). Therefore, the simulation model of SIMGRID subsumes and is strictly more realistic than the simulation models in OverSim and PeerSim. We refer the interested reader to [QRT12] for full details.

Results (see Fig. 11.8) show that the largest scenario that they managed to run in less than 12 hours using PeerSim was for 100,000 peers (4h36min). This poor result is likely due to the Java implementation. With OverSim, they managed to simulate 300,000 peers in 10 hours. With

SIMGRID they were able to simulate 2,000,000 peers in 6h43min. Simulating 300,000 peers took 32 minutes. The memory footprint for simulating 2 million peers with SIMGRID was about 36 GiB, which amounts to 18 KiB per peer, including 16 KiB for the stack devoted to the user code.

We conclude that SIMGRID leads to drastic scalability improvements when compared to state-of-the-art peer-to-peer simulators, even though these simulators were designed specifically for scalable simulations. For instance, SIMGRID is 15 times faster than OverSim and can simulate scenarios that are 10 times larger even though it uses much more sophisticated (network) simulation models. The reasons for these large performance improvements over domain-specific simulators are the various optimizations/designs of the simulation engine described in this work, which were driven by the need for simulation versatility.

11.5 Future Work

The previous sections illustrate the efforts we conducted to make SIMGRID faster and more scalable than domain-specific simulators in the context of grid computing, volunteer computing and peer-to-peer networks. Although we only present the solutions we came up with, many other developments would have been possible. In our case, the key element in designing effective strategies has been to start from workloads revealing actual limitations. Whenever we tried to guess what may go wrong and what could need to be improved, our guess turned out to be not really judicious. That is why we took care in USS-SimGrid and more particularly in SONGS to incorporate both simulation developments and actual case studies depending on simulation.

The current state of SIMGRID is thus now sufficient for most volunteer computing studies even for complex ones like those presented in Chapter 7. Regarding peer-to-peer systems, although I am not aware of real peer-to-peer studies based on SIMGRID, we somehow achieved our goal since SIMGRID not only scales much better than classical peer-to-peer simulators but it is also now cited as a viable alternative in recent surveys on peer-to-peer simulators [BFS+13]. In the context of grid computing, SIMGRID was selected over other simulators by colleagues from the CERN to study data movement policies at LHC due to its scalability and its ability to account for network contention. Several researchers [KAdBM+13, PIB14] have also used it for studying Map-Reduce applications.

I think little developments will be needed for the cloud context in term of scalability and that further work in this direction can only be motivated by specific use cases where SIMGRID scalability is demonstrated to be not satisfactory. The HPC and exascale domain is likely to raise a few challenges of its own but again, I think only actual case studies and workloads should motivate new developments as the wrong optimization is likely to be developed otherwise.

Chapter 12

Ongoing and Future work

12.1 Accurate and Scalable Simulation of HPC Applications

12.1.1 The SMPI Project

In Section 10.3, we explained how we adapted the SIMGRID fluid network model to MPI workloads and in Section 11.3, we explained how we exploited topology regularity and hierarchy to achieve scalable network representation. Developing a simulator that makes it possible to simulate a few standard HPC benchmarks with reasonable accuracy requires a fair amount of effort, and has merit to demonstrate the potential of the simulator. However, the ultimate goal is for the simulator to be usable (i.e., accurate and scalable) for simulating real applications. For this reason, we also ensured SMPI could also be used to simulate both benchmarks and complex applications (both in FORTRAN and C), including the full LinPACK suite [DLP03], the Sweep3D [BK98] benchmark, the BigDFT Density Functional Theory application [GNG+08], and the SpecFEM3D geodynamics application [PKL+11] that is part of the PRACE benchmark. SMPI is tested on a daily basis for 80% of the MPICH2 test suite and against a large subset of the MPICH3 test suite. We can thus claim that SMPI is not limited to toy applications but can effectively be used for the analysis of real scientific applications.

In [24] we have demonstrated the ability of SMPI to simulate a real, large, and complex MPI application and we report here a part of these results to illustrate the effectiveness of the models presented in Section 10.3. To this end, we use BigDFT, which is the sole electronic structure code based on systematic basis sets that can use hybrid supercomputers and has good scaling (95% efficiency with 4,096 nodes on the Curie supercomputer). For this reason, BigDFT was selected as one of the eleven scientific applications in the Mont-Blanc project [Mon] (see Section A.1.2). The goal of this project is to assess the potential of low-power embedded components, such as commercially available ARM processing and network components, for building exascale clusters. The first Mont-Blanc prototype is expected to become available during 2014. It will include Samsung Exynos 5 Dual Cortex A15 processors with an embedded Mali T604 GPU and will be using Ethernet for communication. To evaluate the applications before the prototype is available, a small cluster of ARM systems-on-chip was built at the Barcelona Supercomputing Center, Tibidabo [RVV+13], which uses NVIDIA Tegra2 chips, each with a dual-core ARM Cortex A-9 processor. The PCI Express support of Tegra2 is used to connect a 1 GbE NIC, and the boards are interconnected hierarchically using 48-port 1 GbE switches. The application execution results that are presented in this section have been obtained on Tibidabo. The OpenMP and GPU extensions of BigDFT were disabled so as to focus on the behavior of the MPI operations. We used MPICH 3.0.4 [TRG05] and we refer the interested reader to [24] for more details on the experimental setup.

BigDFT alternates between computation bursts and intensive collective communications. The collective operations that are used are diverse and can change depending on the instance, hence requiring accurate modeling of a broad range of collective communications for the purpose of

simulating BigDFT executions. BigDFT can be simulated with SMPI with minimal source code modification. BigDFT has a large memory footprint, which precludes running it on a single machine. However, thanks to the memory folding and partial execution techniques implemented as part of SMPI (see [CSG+11]), we were able to simulate the execution of BigDFT with 128 processes, with a peak memory footprint estimated at 71 GiB, on a 1.6 GHz Intel Core2 Duo processor with less than 2.5 GiB of RAM.

Fig. 12.1 shows parallel speedup vs. number of compute nodes, as measured on the Tibidabo cluster for an instance of the BigDFT application. This instance has a relatively low communication to computation ratio in spite of Tibidabo's relatively slow compute nodes (around 20% of time is spent communicating when using 128 nodes), and uses the following collective operations: `MPI_Alltoall`, `MPI_Alltoallv`, `MPI_Allgather`, `MPI_Allgatherv` and `MPI_Allreduce`. This particular instance is a difficult case for simulation. This is because the large number of collective communication operations severely limits the scalability of the application, thus requiring precise simulation of these operations. Yet, accurately assessing such scalability limits in simulation is crucial for deciding how to provision a platform before it is actually purchased and deployed.

In addition to the real speedup measurements, Fig. 12.1 shows the speedup computed based on simulation results obtained with SIMGRID, as well as the speedup computed according to the more standard LogGPS model [IFH01]. As expected, both are more optimistic than the real execution. However, while SIMGRID tracks the trend of the real measurements well (within 8%), LogGPS is overly optimistic (up to 40% error). As explained in Section 10.3, unlike models from the LogP family, SIMGRID relies on a model that combines flow-level models (to account for contention on arbitrary network topologies), a piece-wise linear model (to model the protocol switching feature of MPI implementations), and a LogP model (to model the computation/communication overlap and the communication synchronization semantic). The results in Fig. 12.1 show that this model is significantly more accurate than the LogGPS model. In particular, unlike LogGPS, it successfully accounts for the slowdown of BigDFT incurred by the hierarchical and irregular network topology of the Tibidabo platform.

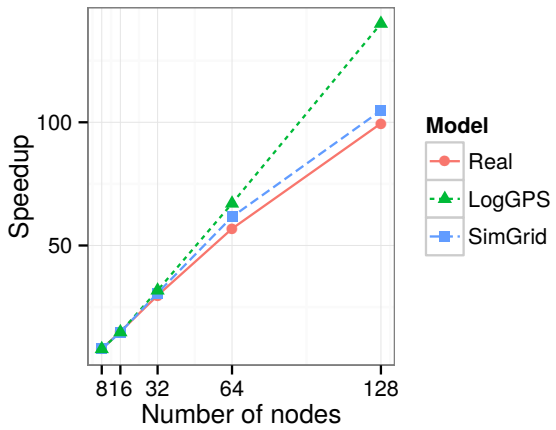


Figure 12.1: Parallel speedup vs. number of compute nodes for BigDFT on Tibidabo, for real executions, SIMGRID simulations, and LogGPS models.

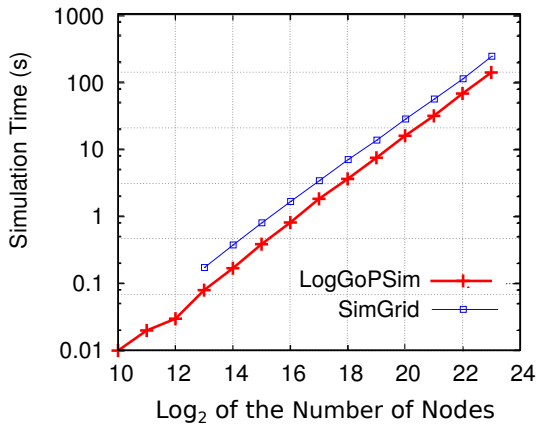


Figure 12.2: Simulation time vs. number of simulated nodes for SIMGRID and LogGOPSim when simulating a binomial broadcast.

To further demonstrate the usability of SMPI, we want to mention that simulating 64 nodes of tibidabo, which is made of relatively slow ARM processors, on a Xeon 7460 with partial on-line simulation takes twice as less time (10 minutes) than running the code for real (20 minutes). This can obviously imputed to the slow speed of ARM processors but still shows that such approach can somehow compete with real experiments when debugging code or conducting scalability studies.

One interesting question is whether the higher accuracy of SIMGRID when compared to the use of the simpler delay-based LogGPS model comes at an acceptable loss in simulation scalability. To answer this question we compare the scalability of SIMGRID to that of LogGOPSim 1.1 [HSL10b], a recent simulator designed specifically to simulate the execution of MPI applications on large-scale HPC systems. LogGOPSim relies on the LogGPS model. We use the same experimental setting described in Section 4.1.2 of [HSL10b], i.e., the execution of a binomial broadcast on various numbers of nodes. Unfortunately, as the input traces used therein were not available, we compare our results to the published results instead of reproducing the experiments with LogGOPSim. We use SIMGRID v3.7-beta (git revision 918d6192 and gcc v4.4.5) to simulate a platform interconnected with a hierarchy of 64-port switches. The binomial broadcast is implemented with the SimDAG API. The original evaluation of LogGOPSim was done on a 1.15 GHz Opteron workstation with 13 GiB of memory. Instead, we use one core of a node with two AMD Opteron 6164 HE 12-core CPUs at 1.7 GHz with 48 GiB of memory, which we scale down to 1 GHz to allow for a fair comparison. We refer the interested reader to [27] for more details on the experimental setup.

Fig. 12.2 shows simulation time vs. the number of simulated nodes for both SIMGRID and LogGOPSim. While using significantly more elaborate platform and communication models, and thus leading in general to much improved accuracy (see Fig. 12.1), SIMGRID is only about 75% slower than LogGOPSim. This percentage slowdown is almost constant up to large scales with millions of simulated nodes. SIMGRID's memory usage for 2^{23} nodes in this experiment is 15 GiB, which is larger than what is achieved in [HSL10b] (whose experiments were conducted on a machine with 13 GiB of RAM). The incurred scalability penalties in terms of simulation time and memory footprint are likely worthwhile for most users given the large improvement in simulation accuracy.

12.1.2 StarPU over SIMGRID

Hybrid multi-core architectures comprising several GPUs have become mainstream in the field of High-Performance Computing. However, obtaining the maximum performance of such heterogeneous machines is challenging as it requires to carefully offload computations and manage data movements between the different processing units. In the past few years, it has become very common to deal with that through the use of an additional software layer, a runtime system, based on the task programming paradigm [ATNW11, ABI+09, BBD+11]. Applications are expressed as a task graph with data dependencies, i.e., a Directed Acyclic Graph (DAG), and provide both CPU and GPU implementations for the tasks. The runtime can then schedule the tasks over all available computation units, and automatically initiate the entailed data transfers. Scheduling heuristics such as HEFT or work stealing are used to automatically optimize that execution [ATNW11]. Application programmers are thus relieved from scheduling concerns and technical details.

As a result, the concern becomes choosing the right task granularity, task graph structure, and scheduling strategies optimizations. Task granularity is of a particular concern on hybrid platforms, since a tradeoff must be found between large tasks which are efficient on GPUs but expose little task parallelism, and a lot of small tasks for CPUs but are less efficient on GPUs. The task graph structure itself can have an influence on execution time, by requiring more or less communication compared to computation, which can be an issue depending on the available bandwidth on the target system.

Getting accurate measurement results for all combinations is not trivial and it requires reserving the target system for a long period, which can become prohibitive. Moreover, experimenting over a wide range of different platforms is also necessary to make sure that the resulting strategy choices are generic, and not only suited to the few target systems which were available to developers. Finally, since execution time on real machine exhibit variability, dynamic schedulers tend to make varying scheduling decisions, and the obtained performance is thus far from deterministic. This makes performance comparisons more questionable and debugging of non-deterministic deadlocks inside such runtimes even harder.

Table 12.1: Machines used for the experiments

Name	Processor	Number of Cores	Frequency	Memory	GPUs
hannibal	Intel Xeon X5550	2×4	2.67GHz	2×24GB	3×QuadroFX5800
attila	Intel Xeon X5650	2×6	2.67GHz	2×24GB	3×TeslaC2050
conan	Intel Xeon E5-2650	2×8	2.0GHz	2×32GB	3×TeslaM2075
frogkepler	Intel Xeon E5-2670	2×8	2.6GHz	32GB	2×K20

Simulation can be an effective technique to address such challenges although the workload induced by such runtimes is very different from what we had been dealing with SIMGRID so far. It may be worth mentioning that studying such application through classical and simple trace-based simulations (as it is commonly done when simulating MPI applications) is not possible as the behavior of the application is completely dynamic. The whole logic of the application scheduler needs thus to be embedded in the simulation. Recently, Luka Stanisic, Samuel Thibault, Brice Videau, Jean-François Méhaut, and myself explained at Europar'14 [23] how we crafted a coarse-grain hybrid simulation/emulation of StarPU [ATNW11], a dynamic runtime system for heterogeneous multi-core architectures, on top of SIMGRID.

We followed a systematic (in)validation approach similar to the one we used in Chapter 10. All conclusions were drawn from analyzing and comparing GFlop/s rate, makespans and traces of StarPU on one hand (called *Native* in the following), and StarPU on top of SIMGRID on the other.

To study the validity of our models, we used the systems described in Table 12.1. These NVIDIA GPUs have distinct characteristics and belong to different generations, which intends to demonstrate the validity of our approach on a range of diverse machines. Regarding applications, we decided to focus on two common dense linear algebra kernels: *cholesky* and *LU* factorization.

As shown in [23], a careless modeling of any aspect of runtime, communications or computations, can lead to gross inaccuracies for particular combinations of machines and applications. Our systematic investigation of discrepancies allowed to cover the most important issues, which enables us to obtain excellent prediction of performances. Fig. 12.3 depicts the performance as a function of the size of the matrix for the two applications LU and Cholesky and for the four different hybrid systems we described in Table 12.1. For most combinations, the prediction obtained with SimGrid is very accurate. The only two scenarios where the error is larger than a few percents is for the LU kernel on conan and frogkepler when our prediction slightly overestimates the (bad) performances for large matrices. The trend is however perfectly predicted as well as the size beyond which performance drops.

A closer look at traces (see Fig. 12.4) allows to see that this approach does not only provide a good estimation of the total runtime but also offers an accurate simulation of the scheduling

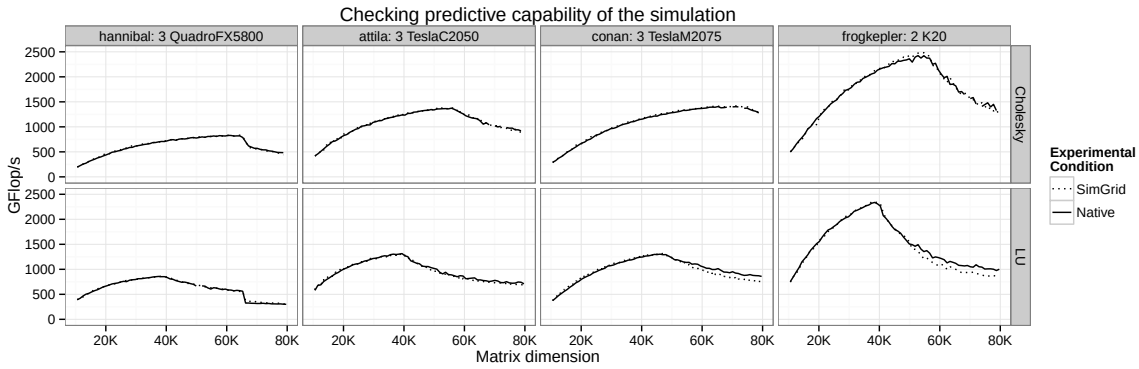


Figure 12.3: Checking predictive capability of our simulator in a wide range of settings

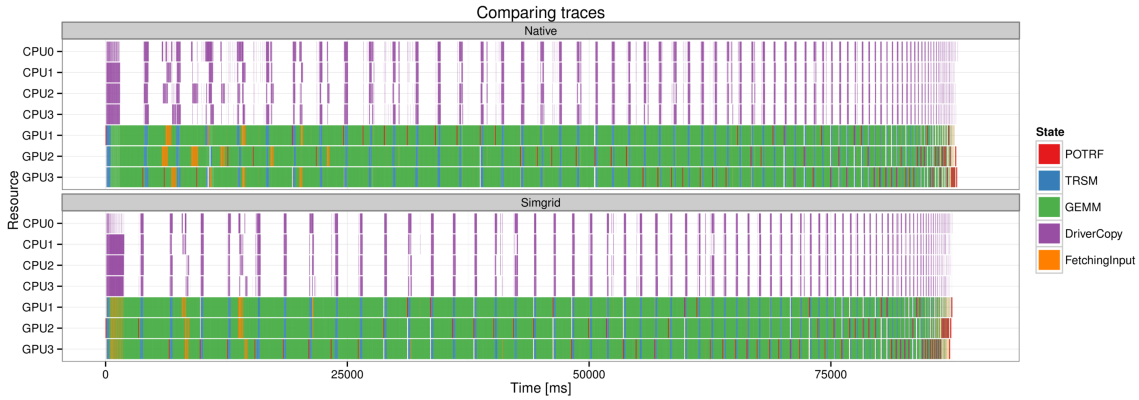


Figure 12.4: Comparing execution traces (native execution on top vs. simulated execution at the bottom) of the Cholesky application with a $72,000 \times 72,000$ matrix on the Conan machine. Traces are not perfectly identical since the execution is not deterministic but the behavior of the simulation is representative of the real execution

details. Since even with the same parameters, native traces differ from an execution to another, a point-to-point comparison with a simulation trace would not make sense. However, we can check that both traces are indeed extremely close, which allows to study and understand the potential weaknesses of a scheduler.

It is important to mention that the time to run each simulation typically takes few seconds compared to sometimes several minutes for a real experiment. Compared to architecture-level simulators whose average slowdown of simulations versus native execution is of the order of magnitude of several dozens of thousands, our coarse-grain simulation allows to obtain a speedup of ten to a hundred depending on the workload and on the speed of the machine. Furthermore, since the target system is not required anymore, it is easy to run series of simulations in parallel.

Such a tool is extremely interesting for both StarPU developers and users as it allows (i) to easily and accurately evaluate the impact of various parameters or scheduling alternatives (ii) to tune and debug applications on a commodity laptop (instead of requiring a dedicated access to a high-end machine) *in a reproducible way* (iii) to obtain reliable comparison of performance estimations that may allow to detect problems with some real experiments (perturbation, configuration issue, etc.).

12.1.3 And Beyond

Our broad conclusion, from this and the other case studies presented in this document, is that a simulator can have both high accuracy and high scalability. It is interesting to note that SIMGRID initially targeted grid computing applications, which led to the development of flow-level network models that account for network contention (Chapter 10) in heterogeneous settings. SIMGRID then began being used for peer-to-peer and volunteer computing simulations (Chapter 7), which required an optimization of the simulation model evaluation algorithm (Section 11.2), the design of efficient platform models and representations (Section 11.3), and the optimization of the simulation of concurrent processes (Section 11.4). Targeting the simulation of HPC systems required improving the network model (Section 10.3). In the end, while these advances were motivated by various domains, their benefits are felt across all these domains. The results achieved for the HPC case study presented in this section would not have been possible had not all these advances been accomplished within a single versatile simulation framework. Fig. 12.5 illustrates the different variations we developed to obtain faithful simulations of HPC scenarios.

Our achievements in the HPC domain can thus be summarized as follows:

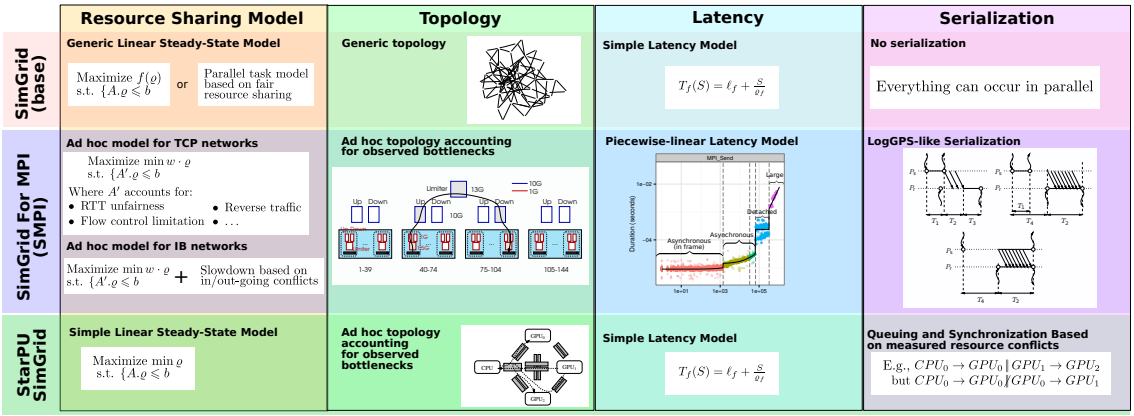


Figure 12.5: Illustration of how the different simgrid components/models have been modified and extended to target specific HPC use cases

- The current version of the SMPI runtime (v3.11) makes it possible to simulate the execution of unmodified MPI applications while accounting for network topology and contention in high-speed TCP networks. Results in [24] show that SIMGRID can simulate collective communications effectively and has consistently a better predictive power than classical LogP-based models for a wide range of scenarios including both established HPC benchmarks and real applications.

- The StarPU/SIMGRID simulator enables to accurately predict the actual running time of dense linear algebra kernels on hybrid nodes [23]. in a large variety of settings. We intend to continue such effort in two main directions. First, StarPU was recently extended to exploit clusters of hybrid machines by relying on MPI [AAF+12]. Since we demonstrated SimGrid's ability to accurately simulate MPI applications, combining both works should allow to obtain good performances predictions of complex applications on large-scale high-end HPC infrastructures. Second, many numerical applications have been recently ported on top of StarPU, including dense (MAGMA/MORSE) and sparse linear algebra (QR-MUMPS), and FMM methods. Such applications are less regular and are thus likely to be more challenging to model but our initial results are very promising.

Our [in]validation experiments allowed us to improve our models and the quality of our prediction to the point where whenever we face a discrepancy between real experiments and simulation, we also have to question the relevance of real experiments. We can list at least a few examples of situations where this happened:

- When conducting the Tibidao experiments of Section 12.1.1, our initial prediction with the fluid model and with the standard LogGPS model were identical and we were thus unable to identify the scalability issue. It turned out that this discrepancy was not due to the model but to the tibidabo machine that was not conforming to its specification. The two cabinets were connected by 7 1 Gb links but due to some machine misconfiguration, only one of them was used. When we modified our platform model accordingly, everything went back to normal and we obtained the prediction presented in Section 12.1.1.

- Additionally, during the Tibidabo experiments, the `MPI_Alltoallv` collective operation happened to be significantly slower than its simulation counterpart. A careful investigation of the corresponding MPI code was mentioning a buffering optimization, which turned out to be very counter-productive in this particular setting. Removing it allowed to achieve the performance predicted by SMPI and which were actually much better.

- Finally, when comparing the execution of the NAS PB with our estimations [24], our scalability predictions are overoptimistic when reaching 128 nodes. However, such bad scalability can be imputed to network collapse that freeze the whole application for about 0.2 s several dozens of time. Although we do not know for sure yet, we think the timeout issues we encountered could be somehow similar to what is known as the *TCP incast problem* and which has been observed in cloud environments [CGL+09]. Such protocol collapse would clearly need to be fixed in a production environment and if it were, the new execution time would perfectly match our predictions. In the case where such phenomenon is too difficult to fix, modifying SimGrid to account for it should be quite easy and may allow to study how much an application is sensitive to such effect, which can help the application developer to improve his code.

- When conducting the StarPU/SIMGRID experiment campaign, some discrepancies could sometime be imputed to GPU not behaving as expected. Such systematic comparison is thus very instructive even for advanced developers and users of such platforms as it can allow to detect hardware or software misconfigurations and unexpected performances that should be fixed.

Most of the work on simulation for exascale that can be found in the literature focuses primarily on how to simulate such very large infrastructures in a reasonable time. As a consequence, the models underlying such simulators are often over simplistic and ignore many important phenomenon such as network contention, network topology or performance peculiarities (e.g., non linearity). Most authors are perfectly aware of such limitations and the study of the validity of the model and of the predictive power of the simulator is generally a secondary objective, which is often left for future work.

The work accomplished on SMPI allows us to seriously envision in the near future the use of simulation as a faithful tool to answer questions that could hardly be addressed with simulators relying on simple delay-based models like LogGPS and simple modeling of collective operations. In particular, we believe, it could be effectively used by application **developers** as it allows non-intrusive tracing and repeatable execution. It also allows to save resources by allowing to test their code on simple laptops under a wide variety of situations before full scale deployment. It could also be useful to application **users** by providing them with sound run time estimates, which can help them to study configuration and deployment trade-offs in simulation rather than by wasting precious resources. Finally, such tool could be used to conduct **capacity planning** studies and to decide how hardware should be upgraded while truly taking into account the characteristics of the codes running the most often.

Simulation can obviously not completely replace real experiments but we think we can improve its quality and usability to the point where it becomes part of the standard set of tools used in the HPC community. Yet, despite all that has been accomplished, a lot remains to be done to address HPC challenges, regardless of the targeted scale. Here are a few examples of topics we intend to address in a near future:

1. Ethernet-based cluster are used in commodity clusters and are one of the envisioned possibility for exascale platforms but Infiniband is also a widespread interconnect technology, whose behavior is not only different in term of performance but also in term of contention management. As shown on Fig. 12.5, a model for Infiniband based on the work of Vienne *et al.* [VMVM08] has been implemented and validated against micro-benchmarks but a broader (in)validation study remains to be conducted. Likewise, many communications take place through shared memory and we are thus currently investigating the design of new models that will widen the application domain of SMPI.

2. The performance interference between computations and computations is currently ignored in SMPI or in StarPU/SIMGRID but the generalization of multi-core processors and the increase of NUMA effects makes it a significant factor to account for. The complexity

of such architectures and of cache mechanisms makes me think that predicting such slowdowns is illusive. However, measuring it and accounting for it in simulations is reasonable although it mandates a specific methodology .

3. When addressing exascale simulation challenges, specific characteristics of applications will have to be exploited. For example, although StarPU applications are currently simulated at a fine-grain level, simulating the execution of every single kernel and ignoring the structure of applications, it should be possible to come up with good models of "macro"-tasks and thus to have multi-scale simulation.

12.2 Methodological Aspects

Finally, the main lesson learned from the SIMGRID project is probably the importance and the interest of methodological questions. The use of critical method for (in)validating models and which has been explained in detail in this part is one of these methodological issues but it is not the only one. In particular, during these years, we also worked on the following themes:

Visualization of Large Distributed Systems Initially, unlike many specialized simulators, SIMGRID did not have any tracing and visualization capability (it merely offered a colored text output). It is indeed difficult to provide such mechanism for a generic-purpose simulation toolkit and we had little experience in this domain. The collaboration with Lucas Schnorr and Jean-Marc Vincent on this subject was very fruitful and allowed me to realize both the importance and the difficulty of such feature. Indeed, a simulation is generally a complex trajectory which is summarized by simple figures (e.g., the average throughput of the system, the convergence time of the algorithm, ...), which are then aggregated over a series of simulations. However, such figures generally only make sense when some hypothesis are verified. Although formally checking them can be quite cumbersome, failing to do so can endangers the conclusions drawn from the simulation campaign. For example, when studying the fairness of a resource sharing protocol, we should at least ensure that the different application/users which may originate from different locations actually interfere with each others and compete for resources. We should also ensure that steady-state was reached. Custom visualization of the simulation trajectory is the most effective way to check all these different hypothesis and I have to confess that when we applied such techniques to some of my previously published work, we discovered that some of the simulations used to draw our conclusions were actually meaningless. It turned out that modifying the simulation setup to obtain more meaningful scenarios did not modify our initial conclusion but this was still a rather unpleasant surprise.

Likewise, when we developed the BOINC simulator, early visualization of the simulations allowed us to detect non-trivial bugs and behaviors that would probably never have been noticed otherwise [7]. Such behaviors would have been impossible to detect by computing simple statistics and with classical visualization. Only interactive and highly configurable visualization tools like the ones developed by Lucas Schnorr could allow to grasp such behaviors. Yet, there is a surprising dearth of tools allowing such exploration.

Looking into such visualization issues revealed me a whole new field of investigation with its own unanticipated challenges. In particular, I learned that just like many simulators should not be trusted when the underlying model has not been validated, many visualizations should be questioned too as they generally aggregate data in a implicit way that may strongly bias visualizations [68]. Designing meaningful visualization of distributed computing systems or comparing two traces of parallel applications can be particularly challenging even at small scale. At large scale, everything remains to be invented. I believe that the understanding obtained from designing faithful simulations and thus faithful models of such complex systems can be leveraged to design meaningful analysis and visualizations. Although I do not know what will be my contribution to this field, I intend to

keep working on this domain at least to improve my use of such non-trivial technique as well as the one of my colleagues.

Reproducibility of Experiments As explained in Chapter 5 or in Chapter 10, in the past years, I have tried several times to build on the work of others and thus to reproduce their results. Every time, even when trying to reproduce the work of very close colleagues that were sensible to such reproducibility issues (e.g., when Pedro Velho and myself tried to reproduce the results of Casanova and Fujiwara [FC07] on the comparison of SIMGRID with GTNetS, despite our long-term collaboration within the same project and the fact that they gave us access to all the code and data they had), it turned out to be extremely difficult not to say impossible. Several times, we came to (sometimes slightly but also sometimes radically) different conclusions. Although it was often instructive and we were not always able to explain these differences. Without even mentioning how time consuming this has been, the results of such work is generally very hard to publish and can easily be considered as offensive to the original authors. I quickly realized that my own work was no better and that even myself was sometimes unable to repeat some previous studies. Making experiment code publicly accessible and writing a documentation afterwards is insufficient and generally proves barely of any help. Therefore I spent a lot of time trying to improve my own methodology both in term of *design of experiments*, *conduct of experiments*, and *provenance tracking*. The last three articles we have published [4, 24, 23] have gradually improved in term of quality thanks to this. We now use *laboratory notebooks* and *literate programming* on a daily basis, which considerably smooth the writing of articles. For example, in the StarPU/SIMGRID study [23], not only the whole software (StarPU and Simgrid) are free software available online but all experiment results are publicly available on *figshare* [SPU14] as well as supplementary data, which is not presented due to space limitation, along with all the scripts, raw data files and traces needed to regenerate this article. The raw data files we provide not only contain experiment outputs but also experimental meta-data such as all the code revision and configuration information, architectural information about the machine (cache hierarchy, CPU type, ...) and operating system information (linux version, frequency governor, compiler version and options, ...). Such methodology greatly increased the confidence we have in our experiments and makes it possible for others to find all the details needed to reproduce our work. We described our approach in various events and articles [22, 1]. The learning curve of such approach is quite steep but the time gained by having access to all relevant information when in doubt or simply when presenting final results largely compensates for the time spent learning and using the right tools. Again, I do not know what could be my contribution, beyond the proselytism I have undertaken, in term of experimental methodology since in the end the techniques we use are generally "relatively simple" and should be the basis of common scientific practice. But just like for simulation and visualization, there is still room for improvement in my own methodology as well as the one of my colleagues.

Part III

Appendix

Appendix A

Collaborations and Grants

During these years I have been heavily involved in the following national or international projects, which I briefly describe here as they may give an idea of the kind of collaborations I had.

A.1 Projects and Grants

A.1.1 ANR SONGS (Simulation of Next Generation Systems): 2012–2015

The SONGS project was funded for four years (2012–2015) by the The French National Research Agency (ANR) under contract no. ANR-11-INFRA-13. It was the largest project (1.8M€) funded for the INFRA ("Infrastructures matérielles et logicielles pour la société numérique") call and was a platform project, which means that we aimed at building a shared, generic and open infrastructure that tackles a technological lock. This project was lead by Martin Quinson helped by Frédéric Suter, Lionel Eyraud, Stéphane Genaud, Olivier Dalle and myself.

The goal of the SONGS project was to extend the applicability of the SimGrid simulation framework from Grids and Peer-to-Peer systems to Clouds and High Performance Computation systems. Building on the experience of the USS-SimGrid project, each type of large-scale computing system was be addressed through a set of use cases and lead by researchers recognized as experts in this area. I had mainly the responsibility of the work packages on the simulation of HPC systems, on the analysis/visualization of simulations, and on support to experimental methodology.

This project involved colleagues from many laboratories (Algorille/LORIA, MESCAL/LIG, CC/IN2P3, AVALON/LIP, CEPAGE-HIEPACS-RUNTIME/LABRI, ICPS/LSIIT, ASCOLA/LINA-Inria Rennes, MASCOTTE-MODALIS/I3S) among which Martin Quinson, Arnaud Legrand Derrick Kondo, Jean-François Méhaut, Jean-Marc Vincent, Frédéric Suter, Frédéric Desprez, Lionel Eyraud-Dubois, Denis Barthou, Olivier Beaumont, Nicolas Bonichon, Brice Goglin, Abdou Guermouche, Samuel Thibault, Emmanuel Agullo, Stéphane Genaud, Julien Gossa, Adrien Lèbre, Olivier Dalle, and Hélène Renard as well as colleagues from CERN, Bull, ... Collaborating with so many people was a incredibly fruitful experience.

During this period I had the chance to advise and collaborate with Luka Stanisic and Augustin Degomme.

A.1.2 European Mont-Blanc projects: 2012–2016

Energy efficiency is a primary concern for the design of any computer system and it is clear that designing the envisioned Exascale systems within a reasonable power envelope will require to fully redesign software and architecture. Since October 2011, the aim of the European project called Mont-Blanc has been to design a new type of computer architecture capable of setting future global HPC standards, built from energy efficient solutions used in embedded and mobile

devices such as ARMv8 64-bit processors. This project is coordinated by the Barcelona Supercomputing Center (BSC) and is funded by the European Commission. Two years later, the European Commission granted additional 8 million Euro funds to extend the Mont-Blanc project activities until September 2016.

My role in these projects is to improve performance evaluation and simulation techniques to conduct network and resource provisioning studies raised by such context as well as possibly improve the parallel software development process.

A.1.3 ANR USS-SimGrid (Ultra-Scalable Simulations with SimGrid): 2009–2012

USS-SimGrid was an ANR Project from the "Systèmes embarqués et grandes infrastructures (ARPEGE)" theme, which was lead by Martin Quinson, Frédéric Suter and myself.

Computer Science differs from other experimental sciences, such as biology of physics, in the way experimental results are presented in articles. In those other disciplines articles always begin with a detailed presentation of the methods employed to produce the results that often rely on previously described and acknowledged procedures. In computer science, and more particularly in the field of application simulation, only a short description of a (sometime unavailable) ad-hoc simulation framework is provided. This prevents reproducibility of published results and thus objective comparisons between new research results and the state of the art. To reduce this gap between computer science and other experimental sciences, there is need for powerful, validated, available and well advertised tools and methods.

The general goal of this project was to provide such an application simulation framework that meets the needs of both the High Performance Computing and the Large Scale Distributed Computing communities. In this project, we worked on extending SimGrid to target the Large Scale Distributed Computing community, increasing simulation realism, and improving the analysis methodology of such discrete event simulations.

This project involved colleagues from many laboratories (Algorille/LORIA, ASAP/Inria-Saclay, Cepage/LABRI, GRAAL/LIP,MESCAL/LIG, Syscom/CReSTIC) among which Martin Quinson, Frédéric Suter, Stéphane Genaud, Fabrice Lefessant, Lionel Eyraud-Dubois, Olivier Beaumont, Nicolas Bonichon, Frédéric Vivien, Frédéric Desprez, Jean-Marc Vincent, Alain Bui, Olivier Flauzac.

During this period I had the chance to advise and collaborate with Pedro Velho on simulation accuracy, Lucas Schnorr on visualization, and Sascha Hunold on design of experiments and experimental methodology.

A.1.4 ADT SimGrid for Human Beings: 2010–2012

From 2010 to 2012, the SimGrid project received support from the Inria to improve the quality of our codebase. I coordinated this project in collaboration with Martin Quinson (Algorille/LORIA) and we two young engineers, whose role has been to set up a quality continuous integration infrastructure, to answer the user questions and improve the documentation, to improve code portability (java, windows) and rewrite/stabilize some prototype parts that had been developed during the ANR USS-SimGrid project.

During this period I supervised and coordinated the work of Pierre Navarro.

A.1.5 ANR DOCCA (Design and Optimization of Collaborative Computing Architectures): 2007–2011

The DOCCA project was a young researcher ANR project lead by Florence Perronnin (Associate Professor, Université Joseph Fourier) and which involved Corinne Touati (Research scientist, INRIA),Fanny Pascual (Associate Professor, Université Pierre et Marie Curie), Olivier Richard (Associate Professor, Polytech'Grenoble), and Lucas Nussbaum (Associate Professor, Lorraine University).

The goal of the project was to design a peer-to-peer collaborative computing protocol with a strong emphasis on theoretical aspects of fairness issues and collaboration incentives. Our target system was a fully decentralized architecture, where desktops are both volunteers and clients (i.e., they can submit jobs). Our goals were to:

1. Leverage the pluridisciplinarity of the team to combine theoretical tools and metrics from the parallel computing community and from the network community, and to explore algorithmic and analytical solutions to the specific resource management problems of such systems.

2. Design a P2P architecture based on the algorithms designed in the second step and to create a novel P2P collaborative computing system.

During this period, I advised Bruno Donassolo, Rémi Bertin, and Rémi Vannier.

A.1.6 ANR ALPAGE (Algorithms for Large-Scale Platforms): 2005–2008

ALPAGE was an ANR project from the "Masse de données: Modélisation, Simulation, Applications" theme, which was lead by Olivier Beaumont.

The new algorithmic challenges associated with large-scale platforms have been approached from two different directions. On the one hand, the parallel algorithms community has largely concentrated on the problems associated with heterogeneity and large amounts of data. Algorithms have been based on a centralized single-node, responsible for calculating the optimal solution; this approach induces significant computing times on the organizing node, and requires centralizing all the information about the platform. Therefore, these solutions clearly suffer from scalability and fault tolerance problems.

On the other hand, the distributed systems community has focused on scalability and fault-tolerance issues. The success of file sharing applications demonstrates the capacity of the resulting algorithms to manage huge volumes of data and users on large unstable platforms. Algorithms developed within this context are completely distributed and based on peer-to-peer communications. They are well adapted to very irregular applications, for which the communication pattern is unpredictable. But in the case of more regular applications, they lead to a significant waste of resources.

The goal of the ALPAGE project was to establish a link between these directions, by gathering researchers (MESCAL, LIP, LORIA, LaBRI, LIX, LRI) from the distributed systems and parallel algorithms communities: Philippe Baptiste, Olivier Beaumont, Philippe Duchon, Christoph Dürr, Pierre Fraigniaud, Cyril Gavoille, Nicolas Hanusse, Anne-Marie Kermarrec, Martin Quinson, Yves Robert, Sébastien Tixeuil, and Frédéric Vivien.

During this period, I advised Lionel Eyraud-Dubois.

A.2 Collaborations and Joint Laboratories

A.2.1 Joint laboratory on *petascale* and *extreme-scale* computing: 2011–2015

The joint laboratory between University of Illinois at Urbana-Champaign, Inria, the CNRS, Argonne National Laboratory, Barcelona Supercomputing Center and Jülich Supercomputing Center targets software and hardware issuers raised by the design and exploitation of supercomputers. I participate since 2011 to this joint laboratory in particular on topics related to modeling and performance evaluation.

In this context, I organized the summer school on *Performance Metrics, Modeling and Simulation of Large HPC Systems* funded by thePartner University Fund and the joint laboratory in June 2014 in Sophia Antipolis[1].

[1] http://mescal.imag.fr/membres/arnaud.legrand/research/events/puf_jlpc_workshop_14.php

A.2.2 Action d'Envergure Inria HEMERA (developing large scale parallel and distributed experiments): 2010–2014

HEMERA is an Inria research action, which started in 2010 and whose goal is to federate the research efforts linked to large-scale experimentation, particularly in the context of the Grid'5000 infrastructure. One of the goals of this action was to animate the high performance/distributed computing French research community. This project was lead by Christian Pérez and I was responsible with Martin Quinson of the {Modeling Large Scale Systems and Validating their Simulators} theme.

A.2.3 Grenoble - Berkeley Associated Team: 2009–2013

The MESCAL team has been associated to several researchers from the Bay area and in particular David Anderson, the leader of the BOINC project but also Walfredo Cirne from Google Inc. This collaboration was initially lead by Derrick Kondo and then by myself and focused on many performance evaluation aspects of very large scale computing systems such as volunteer computing systems and cloud computing systems. Derrick Kondo and Jean-Marc Vincent worked on statistically characterizing the availability and unavailability of resources in such systems based on traces collected by our American colleagues. Such models can then be used to predict load or improve replication strategies and collective availability. On my side I worked rather on how scheduling and game theory could be applied to possibly better understand such systems as well as on how to simulate them efficiently. This collaboration was initially lead by Derrick Kondo. I became the coordinator in 2012 and I organized the BOINC workshop in 2013.

A.2.4 Grenoble - Porto Alegre Associated Team and Joint Laboratory

Grenoble and Porto Alegre Universities (in particular the Universidade Federal do Rio Grande do Sul) have a long standing collaboration that dates back from the end of the 1970s. There has been several associated teams and research/student exchanges (through Inria, CNRS, CAPES, CNPq, FAPERGS, ...) and these collaborations have recently evolved into the LICIA (*Laboratoire International en Calcul Intensif et Informatique Ambiante*), a joint laboratory between the computer science department of UFRGS and the LIG. I have thus visited regularly Porto Alegre in the last decade to give lectures and collaborate with Philippe Navaux, Nicolas Maillard, Claudio Geyer, Alexandre Carrissimi. This gave me the opportunity to advise many Brazilian students during the Msc, PhD or postdoc among which Pedro Velho, Lucas Schnorr, Bruno Donassolo, Wagner Kolberg, Rafael Tesser, ...

Appendix B

Publications

International peer reviewed journal [ACL]

2015

[1] L. Stanisic, A. Legrand, and V. Danjean. An Effective Git And Org-Mode Based Workflow For Reproducible Research. *ACM SIGOPS Operating Systems Review*, 49:61 – 70, 2015.

[2] L. Stanisic, S. Thibault, A. Legrand, B. Videau, and J.-F. Mehaut. Faithful Performance Prediction of a Dynamic Task-Based Runtime System for Heterogeneous Multi-Core Architectures. *Concurrency and Computation: Practice and Experience*, page 16, May 2015.

2014

[3] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter. Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms. *Journal of Parallel and Distributed Computing*, June 2014.

2013

[4] R. Bertin, S. Hunold, A. Legrand, and C. Touati. Fair scheduling of bag-of-tasks applications using distributed Lagrangian optimization. *Journal of Parallel and Distributed Computing*, Aug. 2013.

[5] P. Velho, L. Schnorr, H. Casanova, and A. Legrand. On the Validity of Flow-level TCP Network Models for Grid and Cloud Simulations. *ACM Transactions on Modeling and Computer Simulation*, 23(3), Oct. 2013.

2012

[6] H. Kameda, E. Altman, C. Touati, and A. Legrand. Nash Equilibrium Based Fairness. *Mathematical Methods of Operations Research*, 76(1), 2012.

2011

[7] L. M. Schnorr, A. Legrand, and J.-M. Vincent. Detection and analysis of resource usage anomalies in large distributed systems through multi-scale visualization. *Concurrency and Computation: Practice and Experience*, 2011.

2008

[8] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, L. Marchal, and Y. Robert. Centralized Versus Distributed Schedulers for Multiple Bag-of-Tasks Applications. *IEEE Trans. Parallel Distributed Systems*, 19(5):698–709, May 2008.

[9] A. Legrand, A. Su, and F. Vivien. Minimizing the Stretch When Scheduling Flows of Divisible Requests. *Journal of Scheduling*, 2008.

2005

[10] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, and Y. Yang. Scheduling Divisible Loads on Star and Tree Networks: Results and Open Problems. *IEEE Trans. Parallel Distributed Systems*, 16(3):207–218, 2005.

[11] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Pipelining Broadcasts on Heterogeneous Platforms. *IEEE Trans. Parallel Distributed Systems*, 16(4):300–313, Apr. 2005.

[12] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Steady-State Scheduling on Heterogeneous Clusters. *Int. J. of Foundations of Computer Science*, 16(2):163–194, 2005.

[13] A. Legrand, L. Marchal, and Y. Robert. Optimizing the Steady-State Throughput of Scatter and Reduce Operations on Heterogeneous Platforms. *Journal Parallel and Distributed Computing*, 65(12):1497–1514, 2005.

2004

[14] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Scheduling Strategies for Master-Slave Tasking on Heterogeneous Processor Platforms. *IEEE Trans. Parallel Distributed Systems*, 15(4):319–330, 2004.

[15] A. Legrand, H. Renard, Y. Robert, and F. Vivien. Mapping and Load-Balancing Iterative Computations on Heterogeneous Clusters with Shared Links. *IEEE Trans. Parallel Distributed Systems*, 15(6):546–558, 2004.

2003

[16] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Scheduling Strategies for Mixed Data and Task Parallelism on Heterogeneous Clusters. *Parallel Processing Letters*, 13(2):225–244, 2003.

[17] O. Beaumont, A. Legrand, and Y. Robert. Scheduling Divisible Workloads on Heterogeneous Platforms. *Parallel Computing*, 29:1121–1152, 2003.

[18] O. Beaumont, A. Legrand, and Y. Robert. The Master-Slave Paradigm with Heterogeneous Processors. *IEEE Trans. Parallel Distributed Systems*, 14(9):897–908, 2003.

2002

[19] O. Beaumont, A. Legrand, F. Rastello, and Y. Robert. Dense Linear Algebra Kernels on Heterogeneous Platforms: Redistribution Issues. *Parallel Computing*, 28:155–185, 2002.

[20] O. Beaumont, A. Legrand, and Y. Robert. Static Scheduling Strategies for Heterogeneous Systems. *Computing and Informatics*, 21:413–430, 2002.

2001

[21] O. Beaumont, A. Legrand, F. Rastello, and Y. Robert. Static LU Decomposition on Heterogeneous Platforms. *Int. Journal of High Performance Computing Applications*, 15(3):310–323, 2001.

International peer-reviewed conference proceedings [ACT]

2014

[22] L. Stanisic and A. Legrand. Effective Reproducible Research with Org-Mode and Git. In *1st International Workshop on Reproducibility in Parallel Computing*, Porto, Portugal, Aug. 2014.

[23] L. Stanisic, S. Thibault, A. Legrand, B. Videau, and J.-F. Mehaut. Modeling and Simulation of a Dynamic Task-Based Runtime System for Heterogeneous Multi-Core Architectures. In *Euro-par - 20th International Conference on Parallel Processing*, pages 50–62, Porto, Portugal, Aug. 2014. Springer International Publishing Switzerland.

2013

[24] P. Bédaride, A. Degomme, S. Genaud, A. Legrand, G. S. Markomanolis, M. Quinson, M. Stillwell, F. Suter, and B. Videau. Toward Better Simulation of MPI Applications on Ethernet/TCP Networks. In *4th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, Denver, CO, 2013.

[25] L. M. Schnorr, A. Legrand, and J.-M. Vincent. Interactive Analysis of Large Distributed Systems with Scalable Topology-based Visualization. In *International Symposium on Performance Analysis of Systems and Software (ISPASS'13)*. IEEE Computer Society Press, Apr. 2013.

[26] L. Stanisic, B. Videau, J. Cronsioe, A. Degomme, V. Marangozova-Martin, A. Legrand, and J.-F. Mehaut. Performance Analysis of HPC Applications on Low-Power Embedded Platforms. In *Proceedings of the Conference on Design, Automation, Test in Europe (DATE'13)*, Grenoble, Mar. 2013. Special Day on High-Performance Low-Power Computing.

2012

[27] L. Bobelin, A. Legrand, D. A. Gonzàlez Màrquez, P. Navarro, M. Quinson, F. Suter, and C. Thiery. Scalable Multi-Purpose Network Representation for Large Scale Distributed System Simulation. In *Proceedings of the 12th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'12)*. IEEE Computer Society Press, May 2012.

2011

[28] B. De Moura Donassolo, A. Legrand, and C. Geyer. Non-Cooperative Scheduling Considered Harmful in Collaborative Volunteer Computing Environments. In *Proceedings of the 11th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'11)*. IEEE Computer Society Press, May 2011.

[29] L. M. Schnorr, A. Legrand, and J.-M. Vincent. Multi-scale analysis of large distributed computing systems. In *Proceedings of the third international workshop on Large-scale system and application performance*, LSAP '11, pages 27–34. ACM, 2011.

2010

[30] R. Bertin, P. Coucheney, A. Legrand, and C. Touati. Practical Implementation Issues of Lagrangian Based Distributed Optimization Algorithms. In *12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (Synasc)*, 2010.

[31] B. De Moura Donassolo, H. Casanova, A. Legrand, and P. Velho. Fast and Scalable Simulation of Volunteer Computing Systems Using SimGrid. In *Workshop on Large-Scale System and Application Performance (LSAP)*, 2010.

2009

[32] H. Kameda, E. Altman, C. Touati, and A. Legrand. Nash Equilibrium Based Fairness. In *Proc. of the International Conference on Game Theory for Networks (GameNets)*, 2009.

[33] P. Velho and A. Legrand. Accuracy Study and Improvement of Network Simulation in the SimGrid Framework. In *SIMUTools'09, 2nd International Conference on Simulation Tools and Techniques*, 2009.

2008

[34] R. Bertin, A. Legrand, and C. Touati. Toward a Fully Decentralized Algorithm for Multiple Bag-of-tasks Application Scheduling on Grids. In *IEEE/ACM International Conference on Grid Computing (Grid)*, Tsukuba, Japan, 2008.

[35] H. Casanova, A. Legrand, and M. Quinson. SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In *Proceedings of the 10th Conference on Computer Modeling and Simulation (EuroSim'08)*, 2008.

2007

[36] L. Eyraud, A. Legrand, M. Quinson, and F. Vivien. A First Step Towards Automatically Building Network Representations. In *Proceedings of EUROPAR'07*, Rennes, France, May 2007.

[37] A. Legrand and C. Touati. How to measure efficiency? In *Proceedings of the 1st International Workshop on Game theory for Communication networks (Game-Comm'07)*, 2007.

[38] A. Legrand and C. Touati. Non-Cooperative Scheduling of Multiple Bag-of-Task Appplications. In *Proceedings of the 25th Conference on Computer Communications (INFOCOM'07)*, Alaska, USA, May 2007.

2006

[39] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, L. Marchal, and Y. Robert. Centralized Versus Distributed Schedulers Multiple Bag-of-Task Applications. In *International Parallel and Distributed Processing Symposium IPDPS'2006*. IEEE Computer Society Press, 2006.

[40] A. Legrand, A. Su, and F. Vivien. Minimizing the Stretch When Scheduling Flows of Biological Requests. In *Symposium on Parallelism in Algorithms and Architectures SPAA'2006*. ACM Press, 2006.

2005

[41] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Independent and Divisible Tasks Scheduling on Heterogeneous Star-Schaped Platforms with Limited Memory. In *PDP'2005, 13th Euromicro Workshop on Parallel, Distributed and Network-Based Processing*, pages 179–186. IEEE Computer Society Press, 2005.

[42] A. Legrand, A. Su, and F. Vivien. Off-Line Scheduling of Divisible Requests on an Heterogeneous Collection of Databanks. In *Proceedings of the 14th Heterogeneous Computing Workshop*, Denver, Colorado, USA, Apr. 2005. IEEE Computer Society Press.

2004

[43] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Assessing the Impact and Limits of Steady-State Scheduling for Mixed Task and Data Parallelism on Heterogeneous Platforms. In *HeteroPar'2004: International Conference on Heterogeneous Computing, Jointly Published with ISPDC'2004: International Symposium on Parallel and Distributed Computing*, pages 296–302. IEEE Computer Society Press, 2004.

[44] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Complexity Results and Heuristics for Pipelined Multicast Operations on Heterogeneous Platforms. In *2004 International Conference on Parallel Processing (ICPP'2004)*, pages 267–274. IEEE Computer Society Press, 2004.

[45] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Pipelining Broadcasts on Heterogeneous Platforms. In *International Parallel and Distributed Processing Symposium IPDPS'2004*, page 19b (10 pages). IEEE Computer Society Press, 2004.

[46] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Steady-State Scheduling on Heterogeneous Clusters: Why and How? In *6th Workshop on Advances in Parallel and Distributed Computational Models APDCM*, page 171a (8 pages). IEEE Computer Society Press, 2004.

[47] E. Caron, P. K. Chouhan, and A. Legrand. Automatic Deployment for Hierarchical Network Enabled Server. In *Heterogeneous Computing Workshop*. IEEE Computer Society Press, 2004.

[48] A. Legrand, L. Marchal, and Y. Robert. Optimizing the Steady-State Throughput of Scatter and Reduce Operations on Heterogeneous Platforms. In *6th Workshop on Advances in Parallel and Distributed Computational Models APDCM 2004*, page 176a (8 pages). IEEE Computer Society Press, 2004.

[49] A. Legrand and M. Quinson. Automatic Deployment of the Network Weather Service Using the Effective Network View. In *High Performance Grid Computing Workshop*. IEEE Computer Society Press, 2004.

2003

[50] O. Beaumont, A. Legrand, and Y. Robert. Optimal Algorithms for Scheduling Divisible Workloads on Heterogeneous Systems. In *HCW'2003, the 12th Heterogeneous Computing Workshop*. IEEE Computer Society Press, 2003.

[51] O. Beaumont, A. Legrand, and Y. Robert. Scheduling Strategies for Mixed Data and Task Parallelism on Heterogeneous Clusters and Grids. In *PDP'2003, 11th Euromicro Workshop on Parallel, Distributed and Network-Based Processing*, pages 209–216. IEEE Computer Society Press, 2003.

[52] H. Casanova, A. Legrand, and L. Marchal. Scheduling Distributed Applications: the SimGrid Simulation Framework. In *Proceedings of the Third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)*. IEEE Computer Society Press, May 2003.

[53] A. Legrand, H. Renard, Y. Robert, and F. Vivien. Load-Balancing Iterative Computations on Heterogeneous Clusters with Shared Communication Links. In *PPAM-2003: Fifth International Conference on Parallel Processing and Applied Mathematics*, LNCS 3019, pages 930–937. Springer Verlag, 2003.

[54] A. Legrand, H. Renard, Y. Robert, and F. Vivien. Mapping and Load-Balancing Iterative Computations on Heterogeneous Clusters. In *Euro-PVM-MPI-2003: Recent Advances in Parallel Virtual Machine and Message Passing Interface*, LNCS 2840, pages 586–594. Springer Verlag, 2003.

2002

[55] C. Banino, O. Beaumont, A. Legrand, and Y. Robert. Scheduling Strategies for Master-Slave Tasking on Heterogeneous Processor Grids. In *PARA'02: International Conference on Applied Parallel Computing*, LNCS 2367, pages 423–432. Springer Verlag, 2002.

[56] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Bandwidth-Centric Allocation of Independent Tasks on Heterogeneous Platforms. In *International Parallel and Distributed Processing Symposium IPDPS'2002*. IEEE Computer Society Press, 2002.

[57] O. Beaumont, A. Legrand, and Y. Robert. A Polynomial-Time Algorithm for Allocating Independent Tasks on Heterogeneous Fork-Graphs. In *ISCIS XVII, Seventeenth International Symposium On Computer and Information Sciences*, pages 115–119. CRC Press, 2002.

[58] O. Beaumont, A. Legrand, and Y. Robert. Mixed Task and Data Parallelism. In *Parallel Matrix Algorithms and Applications*. Université de Neuchâtel, 2002.

[59] O. Beaumont, A. Legrand, and Y. Robert. Static Scheduling Strategies for Dense Linear Algebra Kernels on Heterogeneous Clusters. In *Parallel Matrix Algorithms and Applications*. Université de Neuchâtel, 2002.

[60] O. Beaumont, A. Legrand, and Y. Robert. Static Scheduling Strategies for Heterogeneous Systems. In *ISCIS XVII, Seventeenth International Symposium On Computer and Information Sciences*, pages 18–22. CRC Press, 2002.

2001

[61] O. Beaumont, V. Boudet, A. Legrand, F. Rastello, and Y. Robert. Heterogeneous Matrix-Matrix Multiplication, or Partitioning a Square into Rectangles: NP-Completeness and Approximation Algorithms. In *EuroMicro Workshop on Parallel and Distributed Computing (EuroMicro'2001)*, pages 298–305. IEEE Computer Society Press, 2001.

[62] O. Beaumont, A. Legrand, and Y. Robert. Master-Slave Tasking with Heterogeneous Processors. In *2001 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001)*, pages 857–863. CSREA Press, 2001.

[63] O. Beaumont, A. Legrand, and Y. Robert. The Master-Slave Paradigm with Heterogeneous Processors. In D. S. Katz, T. Sterling, M. Baker, L. Bergman, M. Paprzycki, and R. Buyya, editors, *Cluster'2001*, pages 419–426. IEEE Computer Society Press, 2001.

2000

[64] O. Beaumont, V. Boudet, A. Legrand, F. Rastello, and Y. Robert. Dense Linear Algebra Kernels on Heterogeneous Platforms. In *Parallel Matrix Algorithms and Applications*. Université de Neuchâtel, 2000.

[65] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. In *Heterogeneous Computing Workshop*, pages 349–363, 2000.

Short communications [COM] and posters [AFF] in conferences and workshops

2006

[66] A. Legrand, M. Quinson, K. Fujiwara, and H. Casanova. The SimGrid Project - Simulation and Deployment of Distributed Applications. In *Proceedings of the IEEE International Symposium on High Performance Distributed Computing (HPDC-15)*, pages 385–386. IEEE Computer Society Press, 2006.

2000

[67] O. Beaumont, V. Boudet, A. Legrand, F. Rastello, and Y. Robert. Heterogeneity Considered Harmful to Algorithm Designers. In *Cluster'2000*, pages 403–404. IEEE Computer Society Press, 2000.

Scientific books and chapter [OS]

2013

[68] L. M. Schnorr and A. Legrand. Visualizing More Performance Data Than What Fits on Your Screen. In A. Cheptsov, S. Brinkmann, J. Gracia, M. M. Resch, and W. E. Nagel, editors, *Tools for High Performance Computing 2012*, pages 149–162. Springer Berlin Heidelberg, 2013.

2009

[69] A. Legrand and L. Eyraud. *Introduction to Scheduling*, chapter Influence of Platform Models on Scheduling Techniques. Taylor and Francis publisher, 2009.

2008

[70] H. Casanova, A. Legrand, and Y. Robert. *Parallel Algorithms*. Chapman & Hall, 2008.

2003

[71] A. Legrand and Y. Robert. *Algorithmique Parallèle – Cours Et Exercices Corrigés*. Dunod, 2003.

2002

[72] O. Beaumont, V. Boudet, A. Legrand, F. Rastello, and Y. Robert. *Annual Review of Scalable Computing*, volume 4, chapter Static Data Allocation and Load Balancing Techniques for Heterogeneous Systems, pages 1–37. World Scientific, 2002.

National peer reviewed journal [ACLN]

2002

[73] A. Legrand. Équilibrage de Charge Statique Pour Noyaux D'algèbre Linéaire Sur Plate-Forme Hétérogène. *Technique Et Science Informatique, Numéro Spécial RenPar'13*, pages 711–734, 2002.

National peer-reviewed conference proceedings [ACTN]

2002

[74] O. Beaumont, A. Legrand, and Y. Robert. Ordonnancement En Régime Permanent Pour Plateformes Hétérogènes. In *GRID'2002, Actes de L'école Thématique Sur la Globalisation Des Ressources Informatiques Et Des Données*, pages 325–334. INRIA Lorraine, 2002.

[75] A. Legrand. Simulation Pour L'ordonnancement Distribué. In *GRID'2002, Actes de L'école Thématique Sur la Globalisation Des Ressources Informatiques Et Des Données*, pages 155–164. INRIA Lorraine, 2002.

2001

[76] A. Legrand. Équilibrage de Charge Statique Pour la Décomposition LU Sur Une Plate-Forme Hétérogène. In *13ième Rencontres Francophones Du Parallélisme Des Architectures Et Des Systèmes*, Paris, La Villette, Apr. 2001.

Doctoral Dissertations and Habilitations Theses [TH]

2003

[77] A. Legrand. *Algorithmique Parallèle Hétérogène Et Techniques D'ordonnancement : Approches Statiques Et Dynamiques*. PhD thesis, École Normale Supérieure de Lyon, Dec. 2003.

Appendix C

Bibliography

[AAF+12] Cédric Augonnet, Olivier Aumage, Nathalie Furmento, Raymond Namyst, and Samuel Thibault. StarPU-MPI: Task Programming over Clusters of Machines Enhanced with Accelerators. In *Proceedings of the 19th European Conference on Recent Advances in the Message Passing Interface (EuroMPI)*, pages 298–299. Springer-Verlag, 2012.

[ABI+09] Eduard Ayguadé, Rosa M. Badia, Francisco D. Igual, Jesús Labarta, Rafael Mayo, and Enrique S. Quintana-Ortí. An Extension of the StarSs Programming Model for Platforms with Multiple GPUs. In *Proceedings of the 15th Euro-Par Conference*, August 2009.

[ACK+02] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. SETI@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[AdPP09] Alessandro Agnetis, Gianluca de Pascale, and Dario Pacciarelli. A lagrangian approach to single-machine scheduling problems with two competing agents. *J. Scheduling*, 12(4):401–415, 2009.

[AISS95] Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman. LogGP: Incorporating Long Messages Into the LogP Model – One Step Closer Towards a Realistic Model for Parallel Computation. In *Proceedings of the 7th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 95–105, Santa Barbara, CA, 1995.

[Ale09] Marco Pranzo Alessandro Agnetis, Gianluca De Pascale. Computing the nash solution for scheduling bargaining problems. *International Journal of Operational Research*, 2009.

[AM07] David P. Anderson and John McLeod VII. Local scheduling for volunteer computing. In *Parallel and Distributed Processing Symposium (IPDPS 2007)*. IEEE International, 2007.

[AMPP04] Alessandro Agnetis, Pitu B. Mirchandani, Dario Pacciarelli, and Andrea Pacifici. Scheduling problems with two competing agents. *Operations Research*, 52(2):229–242, 2004.

[AR09] David P. Anderson and Kevin Reed. Celebrating diversity in volunteer computing. In *HICSS '09: Proceedings of the 42nd Hawaii International Conference on System Sciences*, Washington, DC, USA, January 2009. IEEE Computer Society.

[ATNW11] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. *Concurrency and Computation: Practice and Experience*, 23:187–198, February 2011.

[Bak74] Kenneth R. Baker. *Introduction to Sequencing and Scheduling*. Wiley, New York, 1974.

[BBC+07] Philippe Baptiste, Peter Brucker, Marek Chrobak, Christoph Dürr, Svetlana A. Kravchenko, and Francis Sourd. The complexity of mean flow time scheduling problems with release times. *Journal of Scheduling*, 10(2):139–146, April 2007.

[BBD+11] George Bosilca, Aurelien Bouteiller, Anthony Danalis, Thomas Herault, Pierre Lemarinier, and Jack Dongarra. DAGuE: A Generic Distributed DAG Engine for High Performance Computing. In *IEEE International Symposium on Parallel and Distributed Processing*, pages 1151–1158. IEEE Computer Society, 2011.

[BBP+01] Olivier Beaumont, Vincent Boudet, Antoine Petitet, Fabrice Rastello, and Yves Robert. A proposal for a heterogeneous cluster ScaLAPACK (dense linear solvers). *IEEE Trans. Computers*, 50(10):1052–1070, 2001.

[BCC+06] Raphaël Bolze, Franck Cappello, Eddy Caron, Michel Daydé, Frédéric Desprez, Emmanuel Jeannot, Yvon Jégou, Stephane Lanteri, Julien Leduc, Noredine Melab, Guillaume Mornet, Raymond Namyst, Pascale Primet, Benjamin Quetier, Olivier Richard, El-Ghazali Talbi, and Iréa Touche. Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed. *IJHPCA*, 20(4):481–494, 2006.

[BCGD00] Christophe Blanchet, Christophe Combet, Christophe Geourjon, and Gilbert Deléage. MPSA: Integrated System for Multiple Protein Sequence Analysis with client/server capabilities. *Bioinformatics*, 16(3):286–287, 2000.

[BCM98] Michael A. Bender, Soumen Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *Proceedings of the 9th Annual ACM-SIAM Symposium On Discrete Algorithms (SODA'98)*, pages 270–279. Society for Industrial and Applied Mathematics, 1998.

[BCM+03] William H. Bell, David G. Cameron, A. Paul Millar, Luigi Capozza, Kurt Stockinger, and Floriano Zini. OptorSim: A grid simulator for studying dynamic data replication strategies. *International Journal of High Performance Computing and Applications*, 17(4):403–416, 2003.

[BDP01] Rajive Bagrodia, Ewa Deelman, and Thomas Phan. Parallel Simulation of Large-Scale Parallel Applications. *IJHPCA*, 15(1):3–12, 2001.

[BFS+13] Anirban Basu, Simon Fleming, James Stanier, Stephen Naicken, Ian Wakeman, and Vijay K. Gurbani. The State of Peer-to-peer Network Simulators. *ACM Computing Survey*, 45(4):46–70, August 2013.

[BG96] Dimitri P. Bertsekas and Robert G. Gallager. *Data Networks*. Prenctice Hall, second edition, 1996.

[BHK07] Ingmar Baumgart, Bernhard Heep, and Stephan Krause. OverSim: A Flexible Overlay Network Simulation Framework. In *Proceedings of the 10th IEEE Global Internet Symposium*, pages 79–84. IEEE, May 2007.

[BK98] Randal S. Baker and Kenneth R. Koch. An s_n algorithm for the massively parallel CM-200 computer. *Nuclear Science and Engineering*, 128(3):312–320, March 1998.

[BLGE03] Rosa M. Badia, Jesús Labarta, Judit Giménez, and Francesc Escalé. Dimemas: Predicting MPI Applications Behaviour in Grid Environments. In *Proceedings of the Workshop on Grid Applications and Programming Tools*, June 2003.

[BM02] Rajkumar Buyya and Manzur Murshed. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, December 2002.

[BMA06] Kevin. Butler, Patrick. McDaniel, and William Aiello. Optimizing BGP security by exploiting path stability. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 298–310, 2006.

[BMR02] Michael A. Bender, S. Muthukrishnan, and Rajmohan Rajaraman. Improved algorithms for stretch scheduling. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 762–771, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[BMR04] Michael A. Bender, S. Muthukrishnan, and Rajmohan Rajaraman. Approximation algorithms for average stretch scheduling. *J. of Scheduling*, 7(3):195–222, 2004.

[BNGNS00] Amotz Bar-Noy, Sudipto Guha, Joseph (Seffi) Naor, and Baruch Schieber. Message multicasting in heterogeneous networks. *SIAM Journal on Computing*, 30(2):347–358, 2000.

[BOI] Berkeley Open Infrastructure for Network Computing. http://boinc.berkeley.edu.

[BPC+11] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 3(1):1–122, January 2011.

[Bra68] Dietrich Braess. Über ein paradoxien aus der verkehrsplanung. *Unternehmensforschung*, 12:258–268, 1968.

[BRG96] Veeravalli Bharadwaj, Thomas G. Robertazzi, and Debasish Ghose. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press, 1996.

[Bri07] Bob Briscoe. Flow rate fairness: Dismantling a religion. *ACM SIGCOMM Computer Communication Review*, 37(2):63–74, April 2007.

[BSP+99] Mohammad Banikazemi, Jayanthi Sampathkumar, Sandeep Prabhu, Dhabaleswar K. Panda, and P. Sadayappan. Communication modeling of heterogeneous networks of workstations for performance characterization of collective operations. In *HCW'99, the 8th Heterogeneous Computing Workshop*, pages 125–133. IEEE Computer Society Press, 1999.

[BT89] David P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, 1989.

[BW97] Fran Berman and Rich Wolski. TheAppLeS project: A status report. In *Proceedings of the 8th NEC Research Symposium*, 1997.

[BWF+96] Francine D. Berman, Rich Wolski, Silvia Figueira, Jennifer Schopf, and Gary Shao. Application-level scheduling on distributed heterogeneous networks. In *Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '96, Washington, DC, USA, 1996. IEEE Computer Society.

[BY05] Steven J. Benson and Yinyu Ye. DSDP5: Software For Semidefinite Programming. Technical Report ANL/MCS-P1289-0905, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, September 2005.

[BYF+09] Ali Bakhoda, George L. Yuan, Wilson W. L. Fung, Henry Wong, and Tor M. Aamodt. Analyzing CUDA workloads using a detailed GPU simulator. In *ISPASS*, pages 163–174, 2009.

[Cas01] Henri Casanova. SimGrid: A toolkit for the simulation of application scheduling. In *First IEEE International Symposium on Cluster Computing and the Grid (CCGrid'01)*, Brisbane, Australia, May 2001.

[CB03] Henri Casanova and Fran Berman. Parameter Sweeps on the Grid with APST. In Geoffrey Fox Fran Berman and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, chapter 33. John Wiley & Sons, 2003.

[CBA+06] Walfredo Cirne, Francisco Brasileiro, Nazareno Andrade, Lauro Costa, Alisson Andrade, Reynaldo Novaes, and Miranda Mowbray. Labs of the world, unite!!! *Journal of Grid Computing*, 4(3):225–246, 2006.

[CD06] Eddy Caron and Frédéric Desprez. DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid. *International Journal of High Performance Computing Applications*, 20(3):335–352, 2006.

[CD12] Weiwei Chen and Ewa Deelman. Workflowsim: A toolkit for simulating scientific workflows in distributed environments. In *8th IEEE International Conference on eScience*. IEEE, oct 2012.

[CDZ97] Kenneth Calvert, Matthew Doar, and Ellen Zegura. Modeling Internet Topology. *IEEE Communications Magazine*, 35(6):160–163, June 1997.

[CGL+09] Yanpei Chen, Rean Griffith, Junda Liu, Randy H. Katz, and Anthony D. Joseph. Understanding tcp incast throughput collapse in datacenter networks. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, WREN '09, pages 73–82. ACM, 2009.

[Chi99] Dah Ning Chiu. Some observations on fairness of bandwidth sharing. Technical report, Sun Microsystems, 1999.

[CK02] Chandra Chekuri and Sanjeev Khanna. Approximation schemes for preemptive weighted flow time. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 297–305. ACM Press, 2002.

[CKP+93] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. LoGP: Towards a Realistic Model of Parallel Computation. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 1993.

[CLT13] Laura Carrington, Michael Laurenzano, and Ananta Tiwari. Inferring large-scale computation behavior via trace extrapolation. In *Large-Scale Parallel Processing workshop (IPDPS'13)*, 2013.

[CM02] Henri Casanova and Loris Marchal. A network model for simulation of grid application. Technical Report 2002-40, LIP, October 2002.

[CRB+11] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, Cesar A. F. De Rose, and Rajkumar Buyya. CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. *Software: Practice and Experience*, 41(1):23–50, January 2011.

[CSG+11] Pierre-Nicolas Clauss, Mark Stillwell, Stéphane Genaud, Frédéric Suter, Henri Casanova, and Martin Quinson. Single Node On-Line Simulation of MPI Applications with SMPI. In *Proceedings of the 25th IEEE International Parallel and Distributed Processing Symp (IPDPS)*, pages 661–672. IEEE, May 2011.

[DBL08] *Proceedings of the 2008 IEEE International Conference on Cluster Computing, 29 September - 1 October 2008, Tsukuba, Japan*. IEEE Computer Society, 2008.

[DCKM04] Frank Dabek, Russ Cox, Frans M. Kaashoek, and Robert Morris. Vivaldi: A Decentralized Network Coordinate System. In *Proceedings of ACM SIGCOMM*, pages 15–26, 2004.

[DDMVB08] Wim Depoorter, Nils De Moor, Kurt Vanmechelen, and Jan Broeckhove. Scalability of Grid Simulators : An Evaluation. In *Proceedings of the 14th EuroPar Conference*, volume 5168 of *LNCS*, pages 544–553. Springer, 2008.

[DGP06] Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a nash equilibrium. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, STOC '06, pages 71–78, New York, NY, USA, 2006. ACM.

[DHN96] Phillip Dickens, Philip Heidelberger, and David Nicol. Parallelized Direct Execution Simulation of Message-Passing Parallel Programs. *IEEE Transactions on Parallel and Distributed Systems*, 7(10):1090–1105, 1996.

[Die03] Francis X. Diebold. 'Big data' dynamic factor models for macroeconomic measurement and forecasting. In M. Dewatripont, L. P. Hansen, and S. Turnovsky, editors, *Advances in Economics and Econometrics, Eighth World Congress of the Econometric Society*, pages 115–122. Cambridge University Press, 2003.

[DL05] Maciej Drozdowski and Marcin Lawenda. On optimum multi-installment divisible load processing in heterogeneous distributed systems. In J. C. Cunha and P. D. Medeiros, editors, *Proceedings of Euro-Par 2005, LNCS*, volume 3648, pages 231–240. Springer-Verlag, 2005.

[DLP03] Jack J. Dongarra, Piotr Luszczek, and Antoine Petitet. The LINPACK benchmark: Past, Present, and Future. *Concurrency and Computation: Practice and Experience*, 15(9):803–820, 2003.

[Dro97] Maciej Drozdowski. *Selected problems of scheduling tasks in multiprocessor computing systems*. Poznań University of Technology, Poznań, Poland, 1997.

[Dut03a] Pierre-François Dutot. Complexity of master-slave tasking on heterogeneous trees. *European Journal of Operational Research*, 2003.

[Dut03b] Pierre-François Dutot. Master-slave tasking on heterogeneous processors. In *International Parallel and Distributed Processing Symposium IPDPS'2003*. IEEE Computer Society Press, 2003.

[DYS12] Qifen Dong, Li Yu, and Wen-Zhan Song. Distributed Demand and Response Algorithm for Optimizing Social-Welfare in Smart Grid. In *26th IEEE International Parallel & Distributed Processing Symposium (IPDPS'12)*, May 2012.

[ETA09] Trilce Estrada, Michela Taufer, and David Anderson. Performance prediction and analysis of BOINC projects: An empirical study with emBOINC. *Journal of Grid Computing*, 7(4):537–554, December 2009.

[FC07] Kayo Fujiwara and Henri Casanova. Speed and accuracy of network simulation in the simgrid framework. In *Proceedings of the First International Workshop on Network Simulation Tools (NSTools)*, 2007.

[FF99] Sally Floyd and Kevin Fall. Promoting the Use of End-to-end Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4):458–472, 1999.

[FJ92] Sally Floyd and Van Jacobson. On traffic phase effects in packet-switched gateways. *Internetworking: Research and Experience*, 3:115–156, 1992.

[FJ93] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4), 1993.

[FQS08] Marc-Eduar Frincu, Martin Quinson, and Frédéric Suter. Handling Very Large Platforms with the New SimGrid Platform Description Formalism. Technical Report 348, INRIA, 2008.

[FYL06] Ahmad Faraj, Xin Yuan, and David Lowenthal. STAR-MPI: self tuned adaptive routines for MPI collective operations. In *Proceedings of the 20th annual international conference on Supercomputing*, ICS '06, pages 199–208. ACM, 2006.

[GB02] Thomas J. Giuli and Mary Baker. Narses: A Scalable Flow-Based Network Simulator. Technical Report cs.PF/0211024, Stanford University, 2002.

[GFB+04] E. Gabriel, G. Fagg, G. Bosilca, T. Angskun, J. Dongarra, J. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. Castain, D. Daniel, R. Graham, and T. Woodall. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In *Proceedings of the 11th European PVM/MPI Users' Group Meeting*, volume 3241 of *Lecture Notes in Computer Science*, pages 97–104. Springer, September 2004.

[GKL+05] Thomer M. Gil, M. Frans Kaashoek, Jinyang Li, Robert Morris, and Jeremy Stribling. P2PSim, a Simulator for Peer-to-Peer Protocols. http://pdos.csail.mit.edu/p2psim/, 2005.

[GLS99] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. Scientific And Engineering Computation Series. MIT Press, 2nd edition, 1999.

[GNG+08] Luigi Genovese, Alexey Neelov, Stefan Goedecker, Thierry Deutsch, Seyed Alireza Ghasemi, Alexander Willand, Damien Caliste, Oded Zilberberg, Mark Rayson, Anders Bergman, and Reinhold Schneider. Daubechies Wavelets as a Basis Set for Density Functional Pseudopotential Calculations. *Journal of Chemical Physics*, 129(014109), 2008.

[GPM+04] Pedro García, Carles Pairot, Rubén Mondéjar, Jordi Pujol Ahulló, Helio Tejedor, and Robert Rallo. PlanetSim: A New Overlay Network Simulation Framework. In *Proceedings of the 4th International Workshop on Software Engineering and Middleware (SEM)*, volume 3437 of *LNCS*, pages 123–136. Springer, September 2004.

[Gri05] http://gripps.ibcp.fr/, 2005.

[Gro02] William Gropp. MPICH2: A new start for MPI implementations. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 9th European PVM/MPI Users' Group Meeting*, volume 2474 of *Lecture Notes in Computer Science*. Springer, October 2002.

[HAH09] Eric Heien, David Anderson, and Kenichi Hagihara. Computing low latency batches with unreliable workers in volunteer computing environments. *Journal of Grid Computing*, 7:501–518, 2009.

[HFH08] Eric M. Heien, Noriyuki Fujimoto, and Kenichi Hagihara. Computing low latency batches with unreliable workers in volunteer computing environments. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–8, April 2008.

[HGWW09] Marc-André Hermanns, Markus Geimer, Felix Wolf, and Brian Wylie. Verifying Causality between Distant Performance Phenomena in Large-Scale MPI Applications. In *Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pages 78–84, Weimar, Germany, February 2009.

[HM95a] Claire Hanen and Alix Munier. An approximation algorithm for scheduling dependent tasks on m processors with small communication delays. In *EFTA 95: INRIA / IEEE Symposium on Emerging Technology and Factory Animation*, pages 167–189. IEEE Computer Science Press, 1995.

[HM95b] Claire Hanen and Alix Munier. Cyclic scheduling on parallel processors: an overview. In P. Chrétienne, E. G. Coffman, J. K. Lenstra, and Z. Liu, editors, *Scheduling Theory and its Applications*, pages 193–226. John Wiley & Sons, 1995.

[HMBD11] Martin Heusse, Sears A Merritt, Timothy X Brown, and Andrzej Duda. Two-way TCP Connections: Old Problem, New Insight. *ACM Computer Communication Review*, 41(2):5–15, April 2011.

[HP07] Bo Hong and Viktor K. Prasanna. Adaptive Allocation of Independent Tasks to Maximize Throughput. *IEEE Transactions on Parallel and Distributed Systems*, 18(10):1420–1435, October 2007.

[HSH+06] Huaizhong Han, Srinivas Shakkottai, C. V. Hollot, R. Srikant, and Don Towsley. Multi-path TCP: a joint congestion control and routing scheme to exploit path diversity in the internet. *IEEE/ACM Trans. Netw.*, 14, December 2006.

[HSL10a] T. Hoefler, T. Schneider, and A. Lumsdaine. LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model. In *Proceedings of the 2nd Workshop on Large-Scale System and Application Performance*, 2010.

[HSL10b] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model. In *Proceedings of the ACM Workshop on Large-Scale System and Application Performance*, pages 597–604, June 2010.

[IET13] IETF. Multipath TCP working group. http://datatracker.ietf.org/wg/mptcp/charter/, 05 2013.

[IFH01] Fumihiko Ino, Noriyuki Fujimoto, and Kenichi Hagihara. LogGPS: a Parallel Computational Model for Synchronization Analysis. In *Proceedings of the eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming (PPoPP)*, pages 133–142, Snowbird, UT, 2001.

[IH08] Teerawat Issariyakul and Ekram Hossain. *Introduction to Network Simulator NS2*. Springer Link, July 2008.

[JAT+08] Krister Jacobsson, Lachlan Andrew, Ao Tang, Karl Johansson, H$r_a$kan Hjalmarsson, and Steven Low. ACK-Clocking Dynamics: Modelling the Interaction between Windows and the Network. In *Proceedings of the 27th IEEE International Conference on Computer Communications (INFOCOM)*, pages 2146–2152, 2008.

[JK12] Jongtack Jung and Hwangnam Kim. Mr-cloudsim: Designing and implementing mapreduce computing model on cloudsim. In *International Conference on ICT Convergence (ICTC)*, pages 504 –509, October 2012.

[JM07] Sam Jansen and Anthony McGregor. Validation of simulated real world TCP stacks. In *Winter Simulation Conference*, 2007.

[JPD03] Manish Jain, Ravi S. Prasad, and Constantinos Dovrolis. The TCP Bandwidth-Delay Product Revisited: Network Buffering, Cross Traffic, and Socket Buffer Auto-Sizing. Technical Report GIT-CERCS-03-02, Georgia Institute of Technology, 2003.

[KA06] Kamer Kaya and Cevdet Aykanat. Iterative-improvement-based heuristics for adaptive scheduling of tasks sharing files on heterogeneous master-slave environments. *IEEE Transactions on Parallel and Distributed Systems*, 17(8):883–896, 2006.

[KAdBM+13] Wagner Kolberg, Julio C.S. Anjos, Pedro de B. Marcos, Alexandre K.S. Miyazaki, Claudio R. Geyer, and Luciana B. Arantes. MRSG - a MapReduce simulator over SimGrid. *Parallel Computing*, 2013.

[Kam06] Hisao Kameda. Bounds on benefits and harms of adding connections to noncooperative networks. In Nikolas Mitrou, Kimon Kontovasilis, George N. Rouskas, Ilias Iliadis, and Lazaros Merakos, editors, *NETWORKING 2004*, volume 3042 of *LNCS*, pages 405–417. Springer Verlag, 2006.

[KAM07] Derrrick Kondo, David Anderson, and John VII McLeod. Performance evaluation of scheduling policies for volunteer computing. In *Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing e-Science'07*, Bangalore, India, December 2007.

[KBV00] Thilo Kielmann, Henri E. Bal, and Kees Verstoep. Fast Measurement of LogP Parameters for Message Passing Platforms. In *Proceedings of the 4th Work. on Run-Time Systems for Parallel Programming*, IPDPS, pages 1176–1183, London, UK, UK, 2000. Springer-Verlag.

[KCC04] Derrrick Kondo , Andrew A. Chien , and Henri Casanova . Rapid application turnaround on enterprise desktop grids. In *ACM Conference on High Performance Computing and Networking, SC2004*, 11 2004.

[KCC07] Derrrick Kondo, Andrew A. Chien, and Henri Casanova. Scheduling task parallel applications for rapid application turnaround on enterprise desktop grids. *Journal of Grid Computing*, 5(4):379–405, December 2007.

[KCCF03] Barbra Kreaseck, Larry Carter, Henri Casanova, and Jeanne Ferrante. Autonomous Protocols for Bandwidth-Centric Scheduling of Independent-task Applications. In *Proceedings of IPDPS 2003, Nice, France*, April 2003.

[KGP+12] Ramin Khalili, Nicolas Gast, Miroslav Popovic, Utkarsh Upadhyay, and Jean-Yves Le Boudec. MPTCP is not pareto-optimal: performance issues and a possible solution. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '12)*, pages 1–12. ACM, 2012.

[KJIE10] Derrick Kondo, Bahman Javadi, Alexandru Iosup, and Dick Epema. The Failure Trace Archive: Enabling Comparative Analysis of Failures in Diverse Distributed Systems. In *Proceeding of the 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 398–407. IEEE, May 2010.

[KMT98] Frank Kelly, Aman Maulloo, and David Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49:237–252, 1998.

[KNOW07] Krzysztof Kurowski, Jarek Nabrzyski, Ariel Oleksiak, and Jan Weglarz. Grid scheduling simulations with gssim. In *Proceedings of the 13th International Conference on Parallel and Distributed Systems*, ICPADS '07, Washington, DC, USA, 2007. IEEE Computer Society.

[Kon07] Derrick Kondo. SimBOINC: A Simulator for Desktop Grids and Volunteer Computing Systems. http://simboinc.gforge.inria.fr/, 2007.

[KP99] Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. In *Proceedings of the 16th annual conference on Theoretical aspects of computer science*, STACS'99, pages 404–413, Berlin, Heidelberg, 1999. Springer-Verlag.

[KS75] Ehud Kalai and Meir Smorodinsky. Other solutions to nash's bargaining problem. *Econometrica*, 43(3):513–518, 1975.

[KTB+04] Derrrick Kondo, M. Taufer, C. Brooks, Henri Casanova, and Andrew A. Chien. Characterizing and evaluating desktop grids: An empirical study. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'04)*, 4 2004.

[Lam83] Butler W. Lampson. Hints for computer system design. *ACM Operating Systems Review*, 15(5):33–48, October 1983.

[LGS07] Jonathan Ledlie, Paul Gardner, and Margo Seltzer. Network Coordinates in the Wild. In *Proceedings of the 4th Symposium on Networked Systems Design and Implementation (NSDI)*, 2007.

[LKCS10] Tian Lan, David Kao, Mung Chiang, and Ashutosh Sabharwal. An axiomatic theory of fairness in network resource allocation. In *Proceedings of the 29th Conference on Information Communications*, INFOCOM'10, pages 1343–1351. IEEE Press, 2010.

[LLLR84] Jacques Labetoulle, Eugene L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Preemptive scheduling of uniform machines subject to release dates. In W. R. Pulleyblank, editor, *Progress in Combinatorial Optimization*, pages 245–261. Academic Press, 1984.

[LN01] Jason Liu and David M. Nicol. *DaSSF 3.1 User's Manual*, April 2001.

[Low03a] Steven Low. A Duality Model of TCP and Queue Management Algorithms. *IEEE/ACM Transactions on Networking*, 11(4):525–536, 2003.

[Low03b] Steven H. Low. A duality model of TCP and queue management algorithms. *IEEE/ACM Transactions on Networking*, 11(4):525–536, 2003.

[LPW02] Steven H. Low, Larry L. Peterson, and Limin Wang. Understanding vegas: a duality model. *Journal of the ACM*, 49(2), March 2002.

[LRB77] J. K. Lenstra, A. H. G. Rinnooy Kan, and Peter Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[LRMB09] Edgar León, Rolf Riesen, Arthur Maccabe, and Patrick Bridges. Instruction-Level Simulation of a Cluster at Scale. In *Proceedings of the International Conference for High Performance Computing and Communications (SC)*, November 2009.

[LS04] Steven H. Low and R. Srikant. A mathematical framework for designing a low-loss, low-delay internet. *Network and Spatial Economics*, 4:75–102, 2004.

[LS06] Xiaojun Lin and Ness B. Shroff. Utility Maximization for Communication Networks With Multipath Routing. *IEEE Transactions on Automatic Control*, 51(5):766–781, 2006.

[LSV04] Arnaud Legrand, Alan Su, and Frédéric Vivien. Off-line scheduling of divisible requests on an heterogeneous collection of databanks. Research report 5386, INRIA, November 2004. Also available as LIP, ENS Lyon, research report 2004-51.

[LSV06] Arnaud Legrand, Alan Su, and Frédéric Vivien. Minimizing the Stretch When Scheduling Flows of Divisible Requests. Technical Report 2006-19, LIP, October 2006. Also available as INRIA research report 6002 http://hal.inria.fr/inria-00108524.

[Mar06] Loris Marchal. *Communications collectives et ordonnancement en régime permanent sur plates-formes hétérogènes*. PhD thesis, École Normale Supérieure de Lyon, France, October 2006.

[Meg02] Nicole Megow. Performance analysis of on-line algorithms in machine scheduling. Diplomarbeit, Technische Universität Berlin, April 2002.

[MJ09] Alberto Montresor and Márk Jelasity. PeerSim: A Scalable P2P Simulator. In *Proceedings of the 9th International Conference on Peer-to-Peer*, pages 99–100, September 2009.

[MLAW99] Jeonghoon Mo, Richard La, Venkat Anantharam, and Jean Walrand. Analysis and comparison of tcp reno and tcp vegas. In *INFOCOM*, 1999.

[MLMB01] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: An Approach to Universal Topology Generation. In *Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS)*, pages 346–356, Cincinnati, Ohio, August 2001.

[Mon] Mont-Blanc: European Approach Towards Energy Efficient High Performance. Montblanc. http://www.montblanc-project.eu/.

[MPP+07] Gustavo Marfia, Claudio Palazzi, Giovanni Pau, Mario Gerla, M.Y. Sanadidi, and Marco Roccetti. TCP Libra: Exploring RTT-fairness for TCP. In Ian F. Akyildiz, Raghupathy Sivakumar, Eylem Ekici, Jaudelice Cavalcante de Oliveira, and Janise McNair, editors, *Proceedings of the 6th International IFIP-TC6 Networking Conference on Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*, volume 4479 of *Lecture Notes in Computer Science*, pages 1005–1013. Springer Berlin Heidelberg, 2007.

[MRSG99] S. Muthukrishnan, Rajmohan Rajaraman, Anthony Shaheen, and Johannes Gehrke. Online scheduling to minimize average stretch. In *IEEE Symposium on Foundations of Computer Science*, pages 433–442, 1999.

[MSB+05] Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *SIGARCH Comput. Archit. News*, 33(4):92–99, November 2005.

[MSMO97] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *Computer Communications Review*, 27(3):67–82, 1997.

[MW00] Jeonghoon Mo and Jean Walrand. Fair end-to-end window-based congestion control. *IEEE/ACM Transactions on Networking*, 8(5), 2000.

[Mye97] Roger B. Myerson. *Game Theory: Analysis of Conflict*. Harvard University Press, 1997.

[Nas50a] John F. Nash. The bargaining problem. *Econometrica*, 18(2):155–162, 1950.

[Nas50b] John F. Nash. Equilibrium points in N-person games. *Proceedings of the National Academy of Sciences USA*, 36:48–49, 1950.

[NnFG+10] Alberto Núñez, Javier Fernández, José Daniel García, Félix García, and Jesús Carretero. New Techniques for Simulating High Performance MPI Applications on Large Storage Networks. *Journal of Supercomputing*, 51(1):40–57, 2010.

[NS3] The ns-3 Network Simulator. http://www.nsnam.org.

[NVPC+11] A. Núñez, J. Vázquez-Poletti, A. Caminero, J. Carretero, and I. M. Llorente. Design of a New Cloud Computing Simulation Platform. In *Proceedings of the 11th International Conference on Computational Science and its Applications*, pages 582–593, June 2011.

[OPF10] Simon Ostermann, Radu Prodan, and Thomas Fahringer. Dynamic Cloud Provisioning for Scientific Grid Workflows. In *Proceedings of the 11th ACM/IEEE International Conference on Grid Computing (Grid)*, pages 97–104, 2010.

[PIB14] Tien-Dat Phan, Shadi Ibrahim, and Gabriel Antoniuand Luc Bougé. A simulation approach to evaluate mapreduce performance under failure. Master's thesis, Master Recherche en informatique (MRI), Université de Rennes 1, 2014.

[PKL+11] Daniel Peter, Dimitri Komatitsch, Yang Luo, Roland Martin, Nicolas Le Goff, Emanuele Casarotti, Pieyre Le Loher, Federica Magnoni, Qinya Liu, Céline Blitz, Tarje Nissen-Meyer, Piero Basini, and Jeroen Tromp. Forward and Adjoint Simulations of Seismic Wave Propagation on Fully Unstructured Hexahedral Meshes. *Geophysical Journal International*, 186(2):721–739, 2011.

[Pop72] Karl Popper. *Objective Knowledge: An Evolutionary Approach*. Oxford University Press, 1972.

[PUK03] Sugree Phatanapherom, Putchong Uthayopas, and Voratas Kachitvichyanukul. Dynamic scheduling ii: fast simulation model for grid scheduling using hypersim. In *Winter Simulation Conference*, pages 1494–1500, 2003.

[PWTR09] Brad Penoff, Alan Wagner, Michael Tüxen, and Irene Rüngeler. MPI-NeTSim: A network simulation module for MPI. In *Proceedings of the 15th IEEE Intl. Conference on Parallel and Distributed Systems*, Shenzen, China, December 2009.

[PY00] Christos H. Papadimitriou and Mihalis Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, page 86, Washington, DC, USA, 2000. IEEE Computer Society.

[QRT12] Martin Quinson, Cristian Rosa, and Christophe Thiéry. Parallel simulation of peer-to-peer systems. In *Proceedings of the 12th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'12)*, pages 668–675. IEEE Computer Society Press, May 2012.

[RF02] Kavitha Ranganathan and Ian Foster. Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications. In *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 352–358, 2002.

[Rie06] Rolf Riesen. A Hybrid MPI Simulator. In *Proceedings of the IEEE International Conference on Cluster Computing*, September 2006.

[Ril03] George F. Riley. The georgia tech network simulator. In *ACM SIGCOMM workshop on Models, Methods and Tools for Reproducible Network Research*, pages 5–12, Karlsruhe, Germany, 2003.

[Rob03] Thomas G. Robertazzi. Ten reasons to use divisible load theory. *IEEE Computer*, 36(5):63–68, 2003.

[RVV+13] Nikola Rajovic, Lluis Vilanova, Carlos Villavieja, Nikola Puzovic, and Alex Ramirez. The Low-Power Architecture Approach Towards Exascale Computing. *Journal of Computational Science*, 4(6):439–443, 2013.

[Sil11] Mark Silberstein. Building an online domain-specific computing service over non-dedicated grid and cloud resources: The superlink-online experience. In *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2011, Newport Beach, CA, USA, May 23-26, 2011*, pages 174–183. IEEE, 2011.

[sim] Simics: a full-system simulator to run unchanged production binarie. http://www.windriver.com/simics/.

[SJY11] Yuxiang Shi, Xiaohong Jiang, and Kejiang Ye. An energy-efficient scheme for cloud resource provisioning based on cloudsim. In *IEEE International Conference on Cluster Computing (CLUSTER)*, pages 595 –599, September 2011.

[Smi56] Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[SMLN+03] Ion Stoica, Robert Morris, David Liben-Nowell, David Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, 2003.

[SNCBL04] Elizeu Santos-Neto, Walfredo Cirne, Francisco Brasileiro, and Aliandro Lima. Exploiting replication and data reuse to efficiently schedule data-intensive applications on grids. In *JSSPP*, pages 210–232, 2004.

[SPS+09] Spyros Sioutas, George Papaloukopoulos, Evangelos Sakkopoulos, Kostas Tsichlas, and Yannis Manolopoulos. A Novel Distributed P2P Simulator Architecture: D-P2P-Sim. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM)*, pages 2069–2070, November 2009.

[SPU14] Companion of the Europar'14 StarPU+SimGrid article. Hosted on Figshare at http://dx.doi.org/10.6084/m9.figshare.928338, 2014. Online version of this article with access to the experimental data and scripts (in the org source).

[SS02] Andreas S. Schulz and Martin Skutella. The power of α-points in preemptive single machine scheduling. *Journal of Scheduling*, 5(2):121–133, 2002.

[SSGS09] Mark Silberstein, Artyom Sharov, Dan Geiger, and Assaf Schuster. Gridbot: execution of bags of tasks in multiple grids. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, New York, NY, USA, 2009. ACM.

[SWW91] David B. Shmoys, Joel Wein, and David P. Williamson. Scheduling parallel machines on-line. *Symposium on Foundations of Computer Science*, 0:131–140, 1991.

[TAJ+08] Ao Tang, Lachlan Andrew, Krister Jacobsson, Karl Johansson, Steven Low, and H$r_a$kan Hjalmarsson. Window Flow Control: Macroscopic Properties from Microscopic Factors. In *Proceedings of the 27th IEEE International Conference on Computer Communications (INFOCOM)*, pages 91–95, 2008.

[TAJ+10] Ao Tang, Lachlan Andrew, Krister Jacobsson, Karl Johansson, H$r_a$kan Hjalmarsson, and Steven Low. Queue Dynamics With Window Flow Control. *IEEE/ACM Transactions on Networking*, 18(5):1422–1435, 2010.

[TBB+11] Ehsan Totoni, Abhinav Bhatele, Eric J. Bohm, Nikhil Jain, Celso L. Mendes, Ryan M. Mokos, Gengbin Zheng, and Laxmikant V. Kale. Simulation-based Performance Analysis and Tuning for a Two-level Directly Connected System. In *Proceedings of the 17th International Conference on Parallel and Distributed Systems*, pages 340–347, 2011.

[Ted07] Matti Tedre. Know your discipline: Teaching the philosophy of computer science. *Journal of Information Technology Education*, 6:105–122, 2007.

[Tel01] Gerard Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, New York, NY, USA, 2nd edition, 2001.

[TKE+07] Michela Taufer, Andre Kerstens, Trilce Estrada, David Flores, and Patricia J. Teller. SimBA: A discrete event simulator for performance prediction of volunteer computing projects. In *PADS '07: Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation*, pages 189–197, Washington, DC, USA, 2007. IEEE Computer Society.

[TLCS09] Mustafa Tikir, Michael Laurenzano, Laura Carrington, and Allan Snavely. PSINS: An Open Source Event Tracer and Execution Simulator for MPI Applications. In *Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, number 5704 in LNCS, pages 135–148. Springer, August 2009.

[TMN+99] Atsuko Takefusa, Satoshi Matsuoka, Hidemoto Nakada, Kento Aida, and Umpei Nagashima. Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms. In *In Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC-8)*, pages 97–104, August 1999.

[TRG05] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of Collective Communication Operations in MPICH. *International Journal of High Performance Computer Applications*, 19(1):49–66, 2005.

[TWL06] Ao Tang, Jiantao Wang, and Steven H. Low. Counter-intuitive throughput behaviors in networks under end-to-end control. *IEEE/ACM Trans. Netw.*, 14(2):355–368, April 2006.

[TYM11] Fei Teng, Lei Yu, and Frédéric Magoulès. Simmapreduce: A simulator for modeling mapreduce framework. In *5th FTRA International Conference on Multimedia and Ubiquitous Engineering (MUE)*, pages 277 –282, june 2011.

[Var01] András Varga. The OMNeT++ Discrete Event Simulation System. In *Proceedings of the 15th European Simulation Multiconference (ESM)*, June 2001.

[Vel11] Pedro Velho. *Accurate and Fast Simulations of Large- Scale Distributed Computing Systems*. PhD thesis, University Joseph Fourier, Grenoble, June 2011.

[VH08] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *Simutools*, 2008.

[VMVM08] Jérôme Vienne, Maxime Martinasso, Jean-Marc Vincent, and Jean-François Méhaut. Predictive models for bandwidth sharing in high performance clusters. In *Proceedings of the 2008 IEEE International Conference on Cluster Computing, 29 September - 1 October 2008, Tsukuba, Japan* [DBL08], pages 286–291.

[Wax88] Bernard Waxman. Routing of Multipoint Connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, December 1988.

[Win06] Jeanette Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, March 2006.

[WM11] Xing Wu and Frank Mueller. ScalaExtrap: trace-based communication extrapolation for SPMD programs. In *Proc. of the 16th ACM symposium on Principles and Practice of Parallel Programming (PPoPP'11)*, pages 113–122, 2011.

[WPL03] Wei-Hua Wang, Marimuthu Palaniswami, and Steven Low. Optimal Flow Control and Routing in Multi-path Networks. *Performance Evaluation*, 52:119–132, 2003.

[WSH99] Rich Wolski, Neil Spring, and Jim Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computer Systems*, 15(5-6):757–768, 1999.

[Yan05] Yang Yang. *Scheduling Divisible Loads in Multiple Rounds*. PhD thesis, University of California at San Diego, Department of Computer Science and Engineering, July 2005.

[YCD+07] Yang Yang, Henri Casanova, Maciej Drozdowski, Marcin Lawenda, and Arnaud Legrand. On the Complexity of Multi-Round Divisible Load Scheduling. Research Report 6096, INRIA, January 2007.

[YdRC05] Yang Yang, Krijn Van de Raadt, and Henri Casanova. Multi-round algorithms for scheduling divisible loads. *TPDS*, 16(11):1092–1102, 2005.

[YFG+01] Benyuan Liu Yang, Daniel R. Figueiredo, Yang Guo, Jim Kurose, and Don Towsley. A Study of Networks Simulation Efficiency: Fluid Simulation vs. Packet-level Simulation. In *IEEE INFOCOM*, April 2001.

[YGK+99] Benyuan Liu Yang, Yang Guo, Jim Kurose, Don Towsley, and Weibo Gong. Fluid simulation of large scale networks: Issues and tradeoffs. In *In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 2136–2142, 1999.

[YMR10] Haïkel Yaïche, Ravi R. Mazumdar, and Catherine Rosenberg. A Game Theoretic Framework for Bandwidth Allocation and Pricing in Broadband Networks. *IEEE/ACM Transactions on Networking*, 8(5), 2010.

[ZCZ10a] Jidong Zhai, Wenguang Chen, and Weimin Zheng. PHANTOM: Predicting Performance of Parallel Applications on Large-Scale Parallel Machines Using a Single Node. In *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 305–314, January 2010.

[ZCZ10b] Jidong Zhai, Wenguang Chen, and Weimin Zheng. PHANTOM: Predicting Performance of Parallel Applications on Large-Scale Parallel Machines Using a Single Node. In *Proceedings of the 15th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, pages 305–314, January 2010.

[ZKK04] Gengbin Zheng, Gunavardhan Kakulapati, and Laxmikant V. Kalé. BigSim: A parallel simulator for performance prediction of extremely large parallel machines. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS)*, April 2004.

[ZSC91] Lixia Zhang, Scott Shenker, and David D. Clark. Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic. In *ACM Computer Communication Review*, pages 133–147, 1991.

[ZWL+04] Gengbin Zheng, Terry Wilmarth, Orion Sky Lawlor, Laxmikant V. Kalé, Sarita Adve, and David Padua. Performance modeling and programming environments for petaflops computers and the blue gene machine. In *Computer Science*. IEEE Press, 2004.