HAL
open science

Evaluation de précision et vitesse de simulation pour des systèmes de calcul distribué à large échelle

Pedro Antonio Madeira de Campos Velho

To cite this version:

Pedro Antonio Madeira de Campos Velho. Evaluation de précision et vitesse de simulation pour des systèmes de calcul distribué à large échelle. Autre [cs.OH]. Université de Grenoble, 2011. Français. NNT : 2011GRENM027 . tel-00625497

HAL Id: tel-00625497

https://theses.hal.science/tel-00625497v1

Submitted on 21 Sep 2011

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

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministérial : 123123

Présentée par

Pedro Velho

Thèse dirigée par **M. Jean-François Méhaut**
et codirigée par **M. 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**

Accurate and Fast Simulations of Large-Scale Distributed Computing Systems

Thèse soutenue publiquement le **date à définir**,
devant le jury composé de :

Mme. Isabelle Guérin Lassous
Professeur, Université Lyon I, Rapporteur
M. Olivier Dalle
Maître de Conférence, Université de Nice, Rapporteur
Mme. Françoise Baude
Professeur, Université de Nice, Examinatrice
M. Frédéric Desprez
Directeur de Recherche, INRIA Rhône-Alpes, Examinateur
M. Hermes Senger
Professor Adjunto, *Universidade de São Carlos*, Examinateur
M. Jean-François Méhaut
Professeur, Université de Grenoble, Directeur de thèse
M. Arnaud Legrand
Chargé de Recherche, CNRS, Co-Directeur de thèse

“Essentially, all models are wrong, but some are useful.”
George E. P. Box

“The Devil is in the details.”

Acknowledgement

Résumé

De nos jours, la grande puissance de calcul et l'importante capacité de stockage fournie par les systèmes de calcul distribué à large échelle sont exploitées par des applications dont les besoins grandissent continuellement. Les plates-formes de ces systèmes sont composées d'un ensemble de ressources reliées entre elles par une infrastructure de communication. Dans ce type de système, comme dans n'importe quel environnement de calcul, il est courant que des solutions innovantes soient étudiées. Leur adoption nécessite une phase d'expérimentation pour que l'on puisse les valider et les comparer aux solutions existantes ou en développement.

Néanmoins, de par leur nature distribuée, l'exécution d'expériences dans ces environnements est difficile et coûteuse. Dans ces systèmes, l'ordre d'exécution dépend de l'ordre des événements, lequel peut changer d'une exécution à l'autre. L'absence de reproductibilité des expériences rend complexe la conception, le développement et la validation de nouvelles solutions. De plus, les ressources peuvent changer d'état ou intégrer le système dynamiquement ; les architectures sont partagées et les interférences entre applications, ou même entre processus d'une même application, peuvent affecter le comportement général du système. Enfin, le temps d'exécution d'application à large échelle sur ces systèmes est souvent long, ce qui empêche en général l'exploration exhaustive des valeurs des éventuels paramètres de cette application. Pour toutes ces raisons, les expérimentations dans ce domaine sont souvent basées sur la simulation. Diverses approches existent actuellement pour simuler le calcul distribué à large-échelle. Parmi celles-ci, une grande partie est dédiée à des architectures particulières, comme les grappes de calcul, les grilles de calcul ou encore les plates-formes de calcul bénévole. Néanmoins, ces simulateurs adressent les mêmes problèmes : modéliser le réseau et gérer les ressources de calcul. De plus, leurs besoins sont les même quelle que soit l'architecture cible : la simulation doit être rapide et passer à l'échelle. Pour respecter ces exigences, la simulation de systèmes distribués à large échelle repose sur des techniques de modélisation pour approximer le comportement du système. Cependant, les estimations obtenues par ces modèles peuvent être fausses. Quand c'est le cas, faire confiance à des résultats obtenus par simulation peut amener à des conclusions aléatoires. En d'autres mots, il est nécessaire de connaître la précision des modèles que l'on utilise pour que les conclusions basées sur des résultats de simulation soient crédibles. Mais malgré l'importance de ce dernier point, il existe très rarement des études sur celui-ci.

Durant cette thèse, nous nous sommes intéressés à la problématique de la précision des modèles pour les architectures de calcul distribué à large-échelle. Pour atteindre cet objectif, nous avons mené une évaluation de la précision des modèles existants ainsi que des nouveaux modèles conçus pendant cette thèse. Grâce à cette évaluation, nous avons proposé des améliorations pour atténuer les erreurs dues aux modèles en utilisant SimGrid comme cas d'étude. Nous avons aussi évalué les effets des ces améliorations en terme de passage à l'échelle et de vitesse d'exécution. Une contribution majeure de nos travaux est le développement de modèles plus intuitifs et meilleurs que l'existant, que ce soit en termes de précision, vitesse ou passage à l'échelle. Enfin, nous avons mis en lumière les principaux enjeux de la modélisation des systèmes distribuées à large-échelle en montrant que le principal problème provient de la négligence de certains phénomènes importants.

Abstract

Large-Scale Distributed Computing (LSDC) systems are in production today to solve problems that require huge amounts of computational power or storage. Such systems are composed by a set of computational resources sharing a communication infrastructure. In such systems, as in any computing environment, specialists need to conduct experiments to validate alternatives and compare solutions. However, due to the distributed nature of resources, performing experiments in LSDC environments is hard and costly. In such systems, the execution flow depends on the order of events which is likely to change from one execution to another. Consequently, it is hard to reproduce experiments hindering the development process. Moreover, resources are very likely to fail or go off-line. Yet, LSDC architectures are shared and interference among different applications, or even among processes of the same application, affects the overall application behavior. Last, LSDC applications are time consuming, thus conducting many experiments, with several parameters is often unfeasible. Because of all these reasons, experiments in LSDC often rely on simulations.

Today we find many simulation approaches for LSDC. Most of them objective specific architectures, such as cluster, grid or volunteer computing. Each simulator claims to be more adapted for a particular research purpose. Nevertheless, those simulators must address the same problems: modeling network and managing computing resources. Moreover, they must satisfy the same requirements providing: fast, accurate, scalable, and repeatable simulations. To match these requirements, LSDC simulation use models to approximate the system behavior, neglecting some aspects to focus on the desired phenomena. However, models may be wrong. When this is the case, trusting on models lead to random conclusions. In other words, we need to have evidence that the models are accurate to accept the conclusions supported by simulated results. Although many simulators exist for LSDC, studies about their accuracy is rarely found.

In this thesis, we are particularly interested in analyzing and proposing accurate models that respect the requirements of LSDC research. To follow our goal, we propose an accuracy evaluation study to verify common and new simulation models. Throughout this document, we propose model improvements to mitigate simulation error of LSDC simulation using SimGrid as case study. We also evaluate the effect of these improvements on scalability and speed. As a main contribution, we show that intuitive models have better accuracy, speed and scalability than other state-of-the art models. These better results are achieved by performing a thorough and systematic analysis of problematic situations. This analysis reveals that many small yet common phenomena had been neglected in previous models and had to be accounted for to design sound models.

Contents

1 Introduction **12**

I State of the Art **16**

2 Performance Evaluation in Large-Scale Distributed Computing **18**

2.1 CPU Performance Evaluation . . . 21
2.1.1 Benchmarking . . . 22
2.1.2 Profiling and Tracing . . . 23
2.1.3 Simulation . . . 23
2.1.4 Emulation . . . 23
2.2 Network Performance Evaluation . . . 24
2.2.1 Benchmarking . . . 24
2.2.2 Simulation . . . 25
2.2.3 Emulation . . . 25
2.3 Performance Evaluation in LSDC . . . 27
2.4 Experimental Platforms . . . 27
2.5 Cluster Computing . . . 28
2.5.1 Benchmarking . . . 28
2.5.2 Simulation . . . 29
2.5.3 Emulation . . . 30
2.6 Grid Computing . . . 30
2.6.1 Benchmarking . . . 31
2.6.2 Simulation . . . 31
2.6.3 Emulation . . . 32
2.7 Peer-to-Peer Computing . . . 32
2.7.1 Benchmarking . . . 32
2.7.2 Simulation . . . 33
2.7.3 Emulation . . . 33
2.8 Volunteer Computing . . . 33
2.8.1 Benchmarking . . . 34
2.8.2 Simulation . . . 34
2.8.3 Emulation . . . 35
2.9 Conclusion . . . 35

3 Models Underlying Performance Evaluation Tools **38**
3.1 Quality Metrics of a Model . . . 38
3.2 CPU Models . . . 39
3.2.1 Fluid Models for CPU Bound Applications . . . 39
3.2.2 Fluid Models for Memory Bound Applications . . . 40
3.3 Network Models . . . 40
3.3.1 LogP and Spinoffs . . . 41
3.3.2 Single Link Analytic Models . . . 42
3.3.3 Store & Forward . . . 43
3.3.4 Wormhole . . . 44
3.3.5 Bandwidth Sharing Models . . . 44
3.4 Application Models . . . 45
3.4.1 Shared Memory Applications . . . 45
3.4.2 Message Passing Applications . . . 46
3.5 Conclusion . . . 48

II Context **50**

4 SimGrid **52**
4.1 History . . . 52
4.2 SimGrid in a Nutshell . . . 53
4.2.1 Platform Description . . . 54
4.2.2 Application . . . 56
4.2.3 Deployment . . . 57
4.2.4 Availability Traces . . . 58
4.3 SimGrid Architecture . . . 59
4.3.1 Simulation Kernel (SURF) . . . 60
4.3.2 Concurrent Process Management (SIMIX) . . . 60
4.3.3 Application Programming Interfaces (GRAS, SMPI, SimDAG, MSG) . . . 60
4.4 The Simulation Process . . . 62
4.5 Conclusion . . . 64

5 Models Implemented in SimGrid **66**
5.1 SimGrid Fluid Models . . . 66
5.1.1 CPU Fluid Model . . . 68
5.1.2 Network Fluid Model . . . 69
5.2 Max-Min Fairness . . . 70
5.2.1 Optimization Problem . . . 71
5.2.2 Algorithm . . . 71
5.2.3 Implementation . . . 72
5.3 Duality Models . . . 74
5.3.1 Optimization Problem . . . 74
5.3.2 Solution . . . 74
5.3.3 Lagrangian Optimization . . . 75

5.3.4 Implementation . . . 77
5.4 Packet-level Discrete Simulation (GTNetS) . . . 77
5.4.1 Protocol Stack . . . 78
5.4.2 Implementation . . . 79
5.5 Conclusion . . . 79

III Model Evaluation and Improvements 80

6 Simulation Accuracy Improvements and Results 82
6.1 Performance Evaluation Approach . . . 82
6.1.1 Comparison Point . . . 84
6.1.2 Error Metric . . . 85
6.2 Accuracy Evaluation of CPU Model . . . 86
6.2.1 Quantitative Analysis . . . 86
6.3 Accuracy Evaluation of the Max-Min Network Model . . . 88
6.3.1 Evaluation of the Single Link Model . . . 88
6.3.2 Evaluation of the Max-Min Bandwidth Sharing Model . . . 94
6.3.3 Evaluation of Bidirectional TCP Connections . . . 99
6.3.4 Evaluation with Random Network Topologies . . . 107
6.4 Accuracy Evaluation of Duality Models for TCP/Ethernet . . . 113
6.4.1 Evaluation with Random Network Topologies . . . 113
6.5 Conclusion . . . 116

7 Simulation Speed Improvements and Results 118
7.1 Performance Evaluation Methodology . . . 118
7.1.1 Scalability Parameters . . . 119
7.1.2 Workload Parameters . . . 119
7.1.3 Speed Measures . . . 119
7.2 Speed Evaluation of CPU Models . . . 120
7.2.1 Naïve Implementation . . . 120
7.2.2 Reducing Cost per Iteration . . . 121
7.2.3 Analysis . . . 124
7.3 Speed Evaluation of Network Models . . . 126
7.3.1 SimGrid vs. GTNetS . . . 126
7.3.2 Comparing SimGrid's Models Performance . . . 128
7.3.3 Speeding SimGrid's Network Models . . . 130
7.4 Conclusion . . . 132

8 Conclusion 134
8.1 A "Historical" Perspective . . . 134
8.2 Contributions . . . 136
8.3 Perspectives . . . 137

Chapter 1

Introduction

Today many applications in science and industry force the boundaries of computing power and storage. As common examples we can cite weather forecast, climate and global change, aeronautics and biotechnology [37]. To fulfill this increasing demand, one possible solution is to harness the computing power of many commodity resources conceiving a Large-Scale Distributed Computing (LSDC) environment [30, 40, 9]. The advantages of doing so are many. First, such environments rely on commodity resources reducing the cost when compared to other solutions. Second, they are flexible, enabling users to match different design goals. Third, LSDC environments have theoretically unlimited scalability. For those reasons, such environments are popular nowadays [81].

One problem arising with the appearance of LSDC systems is the performance evaluation of applications running on such environments. To investigate LSDC systems, specialists need to validate alternatives proposing improvements. Moreover, they need to vary several system parameters, such as input data and communication pattern, observing the resulting output. Yet conducting experiments in such environments is slow, due to the time consuming nature of applications that run upon them. Therefore, trying several system settings is often unfeasible. Moreover, we can identify at least three factors that increases the complexity of conducting experiments in LSDC environments:

Heterogeneity: resources are intrinsically heterogeneous, i.e., resources with different characteristics cope with each other. Due to the number of possibilities each system becomes an unique combination of software and hardware. Consequently, it is hard to reproduce the experiments of others.

Volatility: LSDC systems deal with a huge number of devices, so, it is very likely that resources fail or go off-line. Likewise, resources can abruptly rejoin the system. Hence, it is hard to repeat experiments since the system configuration is very likely to change from one execution to another.

Interference: such environments are shared between applications, or even by processes of the same application. Therefore, interference affect the overall system behavior hindering the development process.

As a result, most studies on LSDC are based on simulation.

We may highlight several advantages of using simulation to assist LSDC research:

Speed: simulations can enable experimentation of many parameters in feasible time.

Scalability: the number of resources can be extrapolated to verify hypothetical scenarios.

Repeatability: with simulation we can repeat experimentation.

Today we find many simulators for LSDC research. Most of them often aim at a punctual research target and are abandoned after the simulation results were published [36, 87, 100]. Some authors [36, 100] claim that instead of using existing simulators it is more adapted to build a new one that can match exactly their needs. However, most LSDC simulators rely on similar needs: they need to estimate computing and communication behavior. Even if they target a different set of users or programming interface, LSDC simulators are expected to be fast, scalable and reproducible. To achieve this objective, simulation relies on modeling, replacing real communications and computations by models approximating the system behavior.

Models are an understanding of the system. Once extracted, these ideas can be used to estimate the system behavior. Unfortunately, to match the speed and scalability constraints of LSDC research, models have to neglect some phenomena to focus on the most important characteristics. From that arises the need of choosing what to consider and what to neglect in a model. Besides speed and scalability, another important issue in model conception is accuracy. Generally, the accuracy of models is limited to a given range. So, we must address the accuracy of models to assess the correctness of a simulator. If a model is ever proved wrong, all conclusions based on simulations building on this model cannot be trusted. Hence, the accuracy evaluation of models is of extreme importance to mature the use of simulation in LSDC research. Unfortunately today, accuracy studies are rarely found. The few that exist focus on simple scenarios [36, 100].

The objective of this thesis is to conduct a comprehensive accuracy study of models designed for LSDC research. Here, we verify the accuracy of common and new models using SimGrid [24] as case study. SimGrid is a simulator that differs from others since it proposes a generic framework for LSDC research. Moreover, the validation issue has been a major center of interest since several years in the SimGrid project. Such concern allied to good software quality has enabled SimGrid to start becoming a standard in the LSDC research field with more than 53 papers published[1] [79]. Therefore, SimGrid is the perfect environment to put in practice new models and evaluate their performances. Moreover, SimGrid is the only simulator of our knowledge that has a comprehensive accuracy study of network models [34]. To evaluate accuracy, we first reiterate the conclusion of this previous work, completing with the implementation of new models. Their work is limited to network models, here we aim at presenting a comprehensive accuracy study evaluating network and computing models adapted for LSDC. Moreover, we do not limit our studies to accuracy evaluation. We also propose several improvements to mitigate simulation error when possible. Furthermore, we evaluate the speed and scalability of simulation models to verify that our improvements have minimal effect on the simulation requirements.

The remaining content of this thesis is organized as follows:

Chapter 2: Performance Evaluation in Large-Scale Distributed Computing

We present in this chapter the state-of-the-art of performance evaluation in LSDC research. In this chapter we introduce several concepts such as modeling approaches and research goals. Next, we review most popular performance evaluation approaches including the most common LSDC simulators.

Chapter 3: Models Underlying Performance Evaluation Tools

In this chapter we present the state-of-the-art of models available for computation and communication. In the discussion we justify why some known modeling approaches are unsuitable to LSDC. At the end, we present the modeling techniques used in most simulators.

[1] Without counting papers co-authored by a SimGrid developer.

Chapter 4: SimGrid

Since our experiments rely on a case study, in this chapter we present in detail the SimGrid simulation framework. Here, we avoid entering into details of simulation models. Models are of extreme importance to understand the contributions of our work. For this reason, we dedicate one entire chapter to them.

Chapter 5: Models Implemented in SimGrid

In this chapter, we conduct a detailed discussion of the models evaluated and proposed during this thesis. These models are essential to understand the performance improvements proposed on the following chapter.

Chapter 6: Simulation Accuracy Improvements and Results

This chapter presents the main contribution of this thesis consisting of the accuracy evaluation of models. Here, we exhaustively evaluate simulation accuracy under specific and general setting, pointing out improvements and verifying their effectiveness.

Chapter 7: Simulation Speed Improvements and Results

In this chapter, we evaluate simulation scalability and speed to certify that the previous improvements have minimal side-effect in the simulation requirements. We also propose some improvements that can speedup significantly simulations with SimGrid.

Chapter 8: Conclusion

Finally, we present a conclusion detailing the main contributions of this thesis and some perspectives of future work.

Part I

State of the Art

Chapter 2

Performance Evaluation in Large-Scale Distributed Computing

One of the basis of science is to build upon the knowledge of others but also to question this knowledge. Therefore, reproducing the results of others is a major requirement and, like any other experimental scientist, computer scientists need to reiterate and check previous result. Yet, computer technologies have evolved extremely fast in the last decades. Computers have evolved probably faster than any other kind of machine, which easily calls into question previous work. This constant evolution of technology makes performance evaluation a crucial issue in computer science.

In the last decades, this rapid evolution has lead to tremendous breakthroughs in many fields of science, such as biology, chemistry, or astrophysics, and even in our every-day's life. Such breakthroughs have required infrastructures capable of delivering a huge amount of processing power and storage. Most infrastructures relied on harnessing several computing units through a communication infrastructure. There has been several kind of such platforms: parallel machines, clusters, grids, volunteer computing systems, and more recently clouds. We will refer to such platforms under the term *Large-Scale Distributed Computing* (LSDC for short) systems, as previously stated in the introduction.

Due to the *Distributed* nature of LSDC systems processing units are spread over different places and communication relies on complex networks. In such systems the network affects dramatically the overall performance and has to be considered in performance evaluation. Furthermore, the large scale nature of such systems leads to the following three characteristics:

Heterogeneity : the larger the scale of the system the more difficult it is to assemble exactly the same devices. So, to achieve large-scale it is often needed to incorporate different processing units, memories, network, computing platforms and connections with heterogeneous characteristics. This heterogeneity hinders the development of LSDC systems since each environment combines devices in a different way. Consequently, each environment becomes unique, increasing the doubtfulness of relying on performance evaluation experiments of others.

Interference : such large-scale systems are often shared by many users that often deploy applications simultaneously. When doing so, it is very likely that their applications compete for resources. Therefore, the application behavior will not be the same as when it is running alone in a dedicated environment. Moreover, since resources are shared in LSDC, even two process belonging to the

same application interfere with each other. Hence, interference affects the application behavior in LSDC and should be controlled or at least taken into account in performance evaluation.

Volatility : at such a scale, the probability that individual components fail or disconnect temporarily is high. Therefore, the system must deal with that, continuing to work even when resources are temporarily unavailable. This characteristic, known as volatility, hinders the performance evaluation in LSDC because the system becomes unstable and repeating experiments is only possible, when possible, targeting a small controlled part of the system.

Therefore, due to heterogeneity, interference, and volatility it is hard to propose and reuse performance evaluation approaches in LSDC. Yet, scientific research in LSDC, as any experimental science, should rely on repeating experiments. A first step to reproduce experiments is to have a common or at least repeatable performance evaluation approach. Unfortunately, until today we lack of consensus on which methodology is the most adaptable. Most published work propose solutions specifically targeting one LSDC research branch, like OptorSim [11] that aims for volunteer computing, or ProtoPeer [36] that focus on peer-to-peer computing. These approaches are indeed innovative and bring the discussion of conducting performance evaluation in LSDC. Yet, most of them are short lived and are abandoned once the results are published. The main reason for that is that the main goal of such works is generally focused on quickly assessing that a given approach is better than another one. Therefore, once the interest of the authors shifts to another area, the prototyping code they used can rarely be reused in another context, even if it looks very close. Nevertheless, among those approaches we can distinguish some similarities observing three distinct methodologies: real experiments, simulation and analytic modeling.

Real Experiments : a myriad of distributed LSDC environments like clusters, grids and clouds exist and are ready to use today. These platforms support the development of distributed applications providing an environment to test the feasibility of such systems. But from an LSDC scientist's perspective those platforms are very limited. Most platforms are not customizable, users generally do not have access to low level parameters such as operating system or network topology. Consequently, even when volatility is not a problem it is hard to control interference and heterogeneity. As a consequence, most part of those platforms available today aim at providing processing power for other causes unsuitable to run experiments for their own designers. Recently, some projects appeared to provide LSDC platforms for LSDC experimentation. The two main projects that follow this idea are Grid5000 [40] and PlanetLab [9]. These platforms intend to be customizable in such a way that operating system and routing is more controllable. Nevertheless, to execute experiments one must have access to the platform which is sometimes complicated to achieve. Even when one gets to access such platforms, real experiments are still hard to configure and control. In addition, real experiments are limited to available resources, which may be a real issue when performing a scalability study for example. For those reasons, real experiments are being used today but they are hardly repeatable and controllable.

As an alternative to real experiments one can use model based approaches that imitate the behavior of the system under study, focusing on the important parts. There are mainly two types of model based approaches: simulation and analytic modeling. It is often possible to combine such approaches to conduct performance evaluation.

It is common sense to trust real experiments, even though the chosen experimental framework may introduce severe bias to the performance evaluation study. When using a model based approach, bias

are even more likely to occur (in the model itself, in its instantiation, or when using it to extrapolate the behavior of the system). Therefore, when using model based approaches, one needs to evaluate their accuracy and confront them against other approaches before they can be trusted.

Analytic Models establish a relationship between the system parameters and its expected output in a tentatively explanatory way. These approaches, are based on a set of math expressions that measure and imitate the system behavior. A key point when designing analytic models is decide of the tradeoff between accuracy and complexity. It is generally impossible to account for the whole complexity of a system through a simple set of equations. Most of the time, some phenomenons have to be ignored to come up with a simple model of the main characteristics of the system.

It is also generally accepted that the more detailed and precise a model, the more complex to instantiate it becomes. Therefore, coming up with simple and accurate models is for simple phenomenon but becomes hard for more complex phenomena with enough accuracy. For those reasons, it is often difficult to reflect the system behavior solely by a set of equations without making some unrealistic assumptions, especially when the system under study involves a wide range of parameters. The complexity and the number of parameters of LSDC systems makes it hard to analyze their behavior though analytic models. In such situations, one possible alternative is to resort to simulation approaches.

Simulation mainly differ from the previous approach by the usage of the model. In simulation, models are implemented with a computer program that computes or solve the corresponding equations or dynamic which would be impossible to do by hand. Hence, most simulators implement rather complex models and rely on rather fine-grain modeling of the system under study. Sometimes, the level of detail of this process is customizable but it directly influences the complexity of the simulator. The more detailed, or fine grained, a simulation is the slower it is, and conversely. At the same time more detailed simulations are expected to produce more accurate results while less detailed simulations are expected to give rough approximations. Being implemented with a deterministic computer, a great advantage of simulation over real experiments, is to provide repeatable performance estimations. In addition, simulations are also faster and more customizable than real experiments. Yet, just like any model based approach, the accuracy of simulation results must be confronted to real experiments before being trusted.

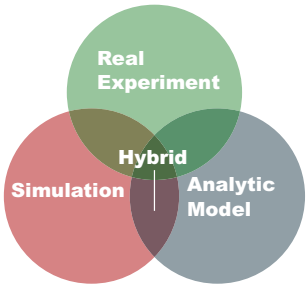


Figure 2.1: Performance evaluation approaches for LSDC research.

Hybrid approaches are also commonly found today. These approaches combine two or more of the three methodologies presented herein, as depicted by Figure 2.1. Emulation is an example of hybrid performance evaluation, which combines the use of real experiments (through benchmarking)

with a model based approach, putting real software or hardware pieces together with a simulator or analytic model or both. Some scientists tend to think that since a real piece of the system is used in emulation this approach is as trustful as real experiments. However, the approximative parts play an important role and emulations have the same drawbacks of any model based approach. So, emulated results need to be verified before trusted.

Therefore, due to the complexity of LSDC systems, research in LSDC often resorts to the combination of the basics performance evaluation approaches: real experiments, simulation, or analytic modeling. Hybrid approaches have proven to be more adaptable to produce repeatable, controllable and fast experiments with a customizable level of detail. Using such an approach users can to adjust the level of detail to use for each part of the system, depending on their relevance for their study. Consequently, hybrid approaches are often found in LSDC scientific literature.

Selecting the right level of detail is highly dependant from the LSDC environment and the application. For sake of clarity, we decided to present the related work of this thesis in two separated chapters. To start, in this chapter we review performance evaluation approaches in LSDC and several related communities presenting an overview of real experiments, simulation and emulation. Our goal here is to justify the existence of SimGrid and why we believe this simulator is a candidate to become a performance evaluation standard in the field. Yet, this chapter ignore details of the analytic models behind SimGrid-based simulations. Analytic modeling is a too extensive subject and is directly connected to the goal of this thesis. As so, we decided to present analytic models in the next chapter to improve readability.

LSDC systems can be seen as the combination of a platform and applications. As a first approximation, the platform is composed by network resources that provide the communication infrastructure and CPU resources that provide computing power.

The application is the active part of the environment that the user customizes to exploit these resources. It is thus the responsibility of the analyst to define the workload and input parameters of the application. It is also the responsibility of the analyst to define the metric he/she will use to measure performance when using a system. To cite a few, the metric could be speedup, makespan, response time, resilience, fairness, cost, energy consumption, ... Evaluating performance in any LSDC environment is hence the problem of evaluating application and platform and their interactions through some metric under a given workload. Several references are available to evaluate network and computational resources independently of each other. The new challenge, in LSDC, is to evaluate both together. Therefore, before reviewing LSDC performance evaluation approaches (i.e., CPU, network and application together), we first briefly review performance evaluation approaches that are specific to CPU performance evaluation and to network performance evaluation. By reviewing this work, we aim at presenting why those ideas are often unadapted to be directly used for LSDC environments, but how they can inspire LSDC analysts. For each area, we classify the related work along benchmarking (real experiments), simulation, and emulation (hybrid approach).

2.1 CPU Performance Evaluation

In this section, we present several approaches used to evaluate the performance of a single CPU. These approaches, as we will see later in this chapter, influence in great part some of the most commonly used performance evaluation approaches for LSDC. There are two main approaches based on real

executions. The first one, Benchmarking, aims at evaluating performance, while the second one, tracing and profiling, aims more at understanding and explaining performances.

2.1.1 Benchmarking

Benchmarking is a performance evaluation technique where the performance of a system is evaluated by running programs that are considered as representative of a given workload. Depending on the complexity of the program we can distinguish between at least four different types of benchmarks.

Real program benchmarks are applications that reflect specific usage cases. The target is to compare two machines by running the same application with the same workload to compare their performance under the same measure. In practice this is done using mostly any software used in day-by-day life from computer games to desktop applications like text editor, web browser, and so on.

Synthetic program is an old approach [65] to explore a wide range of operations common to many application. In practice, synthetic programs are created profiling several applications reproducing the amount each operation is executed in a new program. The resulting application is called a synthetic program because it aim is deviated from normal programs where the goal is to implement an algorithmic solution. Instead, synthetic programs goal is to measure performance trying to cluster several usage cases in a single piece of software. As an example of synthetic program we have STREAM [84] which aims at saturating memory operations to measure memory throughput.

Kernel is based on several basic code that is often used as part of application like, sort, matrix algebraic operations, list control, among others. As an example the Linpack benchmark uses kernels from the BLAS [14] (Basic Linear Algebra Subprograms) library. Another example is CPU2006 [85] that evaluates several kernel to qualify CPU performance using a selection of common used operating system applications.

Micro-Benchmarking is the technique of measuring execution time for small pieces of code. Using this approach is possible to characterize commonly used operation by CPU clocks needed to run them. This approach give a fine grain evaluation of the system presenting a detailed description of performance: context switching, disk performance, memory latency, memory bandwidth, and so forth. One example of micro-benchmarking implementation is Lmbench [75]. Lmbench propose a series of tests to compare several Unix system implementations characterizing the performance on common POSIX system calls. A problem that arises in practice with micro-benchmarking is that it is difficult to measure small operations when there is code optimizations or cache effects or both. In these cases benchmarking is rarely enough to indicate the system limitation.

A common estimation to define CPU processing power is processing peak, in Flops (floating point operations per second) or memory achieved throughput, in B/s (bytes per second). It is to see that those measures, are high level estimations of CPU performance that are mostly helpful from an end user perspective.

2.1.2 Profiling and Tracing

A common practice to evaluate performance of CPU resources is to use profiling tools. Profiling tools access hardware performance counters found in most part of microprocessors in production today. Hardware counters expose information about the hardware such as, cache misses and number of floating point instruction per second. With these informations it is easier to debug, benchmark and understand the performance bottlenecks. However, each vendor implements a different set of hardware counters and profiling varies from each microprocessor architecture. A general profiling approach is the Performance API (PAPI). PAPI [88] provides profiling for several processors with a common interface. Profiling tools are interesting to detect low-level phenomena (such as trashing, cache effect), however, this type of measurement requires access to hardware counters. An alternate method, to evaluate the performance of the system without concern of access restrictions is to use simulation.

2.1.3 Simulation

Since the first integrated circuit processors were designed, computer architecture researchers perceived that design was an expensive and time consuming part of micro-processors development cycle. From that necessity, they looked forward to automate the different stages on micro-processor design. Today, a common approach to design a micro-processor behavior is to start describing it with a hardware description language (HDL) using intensively simulation to debug and evaluate system performance. Due to that microprocessor design counts on a multitude of computer architecture simulators [97]. Among those architectural simulators we can distinguish at least three families.

Cycle-Accurate simulator detail the signals of integrated circuits at each clock tick. Although such approach easies the process of debugging and evaluating hardware, it simplifies the studied system assuming that signals change instantaneously, neglecting some known phenomena such as delay and drift. In spite of useful and very accurate from a micro-processor design perspective such models generally aim at providing high detailed simulations. So, normally simulations are slow. Examples of cycle-accurate simulators are ModelSim©[1] and FaCSim [1].

Instruction-Driven simulator is a name to define those simulators that roughly approximate the number of cycles taken by each CPU instruction. This approach is more high-level and hence less accurate than cycle-accurate simulators being in counter part faster than cycle-accurate simulators.

Full system simulators are used to model the system as a whole with hardware devices and software. Those approaches are very detailed and enable users to run even operating systems on top of a simulated processor. Avoiding to port programs to run inside the simulated environment. Examples of full system simulators are Simics©[2] and SunFlower [91].

In spite of useful all computer architecture simulators are too detailed and hence hardly scalable to match LSDC constraints.

2.1.4 Emulation

CPU emulation is done through the implementation of a software that is able to run a code on a machine differing from the one it was originally conceived to work on. This is achieved through virtualization

[1] ModelSim is a trademark of Mentor Graphics.
[2] Simics is a trademark of WindDrive.

techniques which were first proposed in the 60's [17]. Originally, this technique was meant to cope with the inability of old operating systems to treat multiple users and multiple programs. The development of operating system finally solved this issue using other techniques but the work on virtualization has been revived in the end of the 90's with projects like VMWare©[3] or Xen [98].

These systems enable the user to run guest operating system from different manufacturers so that they can access hardware resources creating the illusion of a dedicated machine. Recent developments have enable to decrease the overhead and have made this technique viable for a large scale usage, e.g., in cloud computing centers.

Yet, even though this technique can be used to fake resource characteristics, it was not developed with performance evaluation in mind. For example, it is rather hard to claim that the performances obtained through an emulator running on architecture A will be representative of those that would have been obtained on architecture B, even if the emulator tries to mimic the characteristics of B as precisely as possible. Such approach suffers from the same validity issues as model based approaches.

2.2 Network Performance Evaluation

In LSDC, the performance of network resources plays an important role since it is the is used for data movements. Network is thus potentially a performance bottleneck and correctly evaluating its behavior is essential to achieve reliable performance estimations in LSDC. Again, we classify our review of performance evaluation techniques depending whether they are based on real experiments, on models, or use a hybrid approach.

2.2.1 Benchmarking

Networks are complex systems that generally involve many resources (links, hosts, routers) connected through a given topology and using a set of protocols. Protocols are used to guarantee a series of properties such as, congestion avoidance, routing and error checking.

Typical network users wants seek to obtain the maximum possible throughput so that the time to send or receive messages is minimized. Consequently, network users are generally interested in evaluating the throughput of network connections. For evaluating network throughput performance several benchmarks were proposed and are available today. We cite few:

NetPerf [62] evaluates the bandwidth throughput between two peers by streaming a huge amount of data for a timed duration. Knowing the time and the amount of data transfered NetPerf is able to determine a rough estimation of connection throughput.

NetSpec [2] was proposed in 1998. The approach proposed by the Network Weather Service (NWS), is slightly different of similar projects because it aims at monitoring the state of several network devices online. Therefore it cannot incur a huge network load all the time and resorts to less intrusive techniques (small probes and re-calibration) to get an estimation of the current and future network performances.

Pathchar [68] is a tool from Van Jacobson. Just like traceroute, Pathchar is able to detect the several hops involved in a path used for communicating. Besides that, Pathchar can also estimate the

[3] VMware is a trade mark of WMware Inc.

bandwidth, delay, average queue and loss rate of each hop in the path between a source and its destination. Although enabling to estimate network performance, Pathchar and other bandwidth measure tools such as Pathrate [70] and Pathload [69] have questionable reliability. As pointed by Jain *et al.* [44], several flaws are found in such performance evaluation approaches.

The software (protocol) used in the network domain plays an extremely important role in the overall performance of the system. Therefore, many researches are interested in evaluating the performance of several network protocol aspects. However, to conduct such fine grain analysis, high level approaches such as benchmarking are unadapted. Furthermore, the complexity of theses systems makes it very hard to finely observe or control. Due to that, in the last decade many network protocols advancements were proposed through simulation.

2.2.2 Simulation

Due to the difficult of extrapolating and debugging network performance aspects, packet-level simulators became popular in the mid 90's as a repeatable and configurable method to conduct network evaluation and research. Network packet-level simulators are generally discrete event simulators, where the protocol stack is modeled as a state-machine. The protocols are hence implemented as a set of waiting queues to go from one stack level to another following a given algorithm. Such simulators detail high-level protocols imitating the entire protocol stack and the waiting queues involved in the transmission process.

Among several network simulators that were popular, we find NS2 [42]. Designed in 1989, NS2 was the successor of the REAL simulator. Since then, NS2 was used to propose several protocols and algorithms improving network performance. Nevertheless, many issues were soon criticized in NS2 like the lack of modularity in the code that made difficult the process of updating the simulator. Moreover, the nature of NS2 simulations being written in Tcl/Tk, an interpreted language, seriously affected simulation performance.

In 2003, another packet-level simulator that gained in popularity is GTNetS [74]. GTNetS profits from C++ modularization to enable users to customize the simulator, adding internal functions and new modules when needed. Besides, GTNetS was proved to be more scalable than NS2, simulating hundred thousands of nodes and simultaneous flows in reasonable time. Nevertheless, GTNetS development is discontinued today. Indeed, in 2008, NS3 [90] appeared to replace NS2. This simulator results from a joint effort of the whole networking community, including the former developers of NS and GTNetS. This completely reborn network packet-level simulator is unfortunately not compatible with its predecessor NS2. Being natively written in C/C++, NS3 benefits from the modularity and scalability of GTNetS.

2.2.3 Emulation

An alternative approach to network simulation is network emulation. There are two main emulation approaches: hardware-based or software-based.

The principle of hardware-based network emulation is to slow-down network components by introducing delays or reducing network bandwidth. Many network emulation approaches follow this idea, e.g., DummyNet [21] or NetEm [63]. These two projects were created to provide a configurable way of artificially tuning network bandwidth and latency. Although this emulation uses real hardware, the emulation itself is implemented in the software level, where new queues are introduced to force pacing the

transmission of packets. This enables to tune the available bandwidth of a given link. A critical point of emulation is that the realism of introducing such algorithms is often unquestioned. People, tend to think that since they are using the real device this approached is more accurate than simulation. To illustrate how this first impression is wrong let us take a look inside the way NetEm emulates latency.

To increase the latency of a network connection NetEm proposes the introduction of a delayed queue. This queue is served with packets that are sent before dispatching them through the network link. Therefore, the algorithm implemented by NetEm guarantees latency delay by freezing the packet in this queue in the desired amount of mile-seconds. However, this queue has a limited size. When the queue is full packets are dropped randomly from this waiting queue. This example illustrates that network emulation may be very intrusive under specific workload and do not necessarily reflects what would happen in a real environment.

The principle of software-based emulation is to use real software with a model of the hardware. Packet-level network simulators reproduce the mechanic of packets transiting from one level to another of the protocol stack. To test those protocols, specialists need also depend to model external factors, such as the operating system. Implementation of real protocol stack inside simulator is time-effort and error-prone and still may not be faithful to real implementations. An altertive, to direct achieve a real protocol stack is to connect a real TCP protocol implementation on top of a simulator. Recently, the NS-Cradle [46], implements this idea enabling users to simulate network low level aspects while a real protocol stack is used for each simulated host. This approach is called software-based emulation because it uses real software upon a model of the hardware. Using this approach, one can reproduce the behavior of a specific protocol inside the simulator.

Another interesting approach to conduct network experiments that involves emulation is NS-2 distributed clients [50]. This approach proposes to distribute the simulation on several real world hosts. Them, each real host communicate to an EmuHost (Emulation host) that will instead simulate protocols and network stack using NS-2. To remain simple each connection with the EmuHost is made through an UDP channel. This approach is interesting to evaluate real application upon real hosts when those application are too complex to be ported inside NS-2. Nevertheless, such approach is very intrusive and results point that the EmuHost performance affect severely the network characteristics. Moreover, such approach introduces some of the real world indeterminism inside the simulation, due to that simulation are no longer as repeatable and reliable as they were before.

In this section, we have reviewed network performance evaluation approaches. There are three main approaches: benchmarking, simulation and emulation. The advantage of benchmarks is that it quantifies the performance of the system. Nevertheless, benchmarks are real experiments hindering reproducibility or performance extrapolation for other scenarios. For this reason, many research in the network field recalls to simulation and emulation.

As we can observe by the reviewed projects, packet-level simulators tend to be very precise evaluating the network packet per packet. As it is expected fine grain simulation induces to high computation cost and scalability becomes an issue. This issue is particularly undesired in LSDC because networks in this context are intrinsically large. As an alternative to overcome this problem the MaSSF (MaSSF is pronounced as *massive*) network simulation proposes to use several simulation hosts [53]. Those hosts can be used either for application execution or as network simulation engines. While application execution hosts are used to run a slightly modified network application, the network simulation engines reconstruct the simulated network. This is done through routing rules that make the behavior of the real network reflects a different simulated one. This approach is between emulation and simulation since

real transfers are used but the path they follow uses a simulated network approach. Using this approach MaSSF authors can scale their simulator up to 20,000 routers and 10,000 hosts when using 90 nodes as simulation engines and 7 nodes for application execution.

Notably, after reviewing performance evaluation approaches in the network and computer architectures we can state that those two communities are indeed related to distributed computing. In LSDC the focus is to produce experiments stressing the number of CPUs while using a network as communication infrastructure. Therefore, CPU and network usage are the subjects of investigation in LSDC performance evaluation. Moreover, due to the large-scale of such environments the network simulation must be fast and scalable to be used in our context. Unfortunately, it is hard to port network specific proposed approaches to the LSDC field because their aim restricts the applications instantiation or are too slow or both. We next present several approaches proposed for performance evaluation in LSDC. These approaches are similar to a combination of CPU and network performance evaluation. However, we can state the advantage of focusing in speed and scalability since in these approaches the problem is perceived as a whole.

2.3 Performance Evaluation in LSDC

Experimental research in LSDC is relatively new and performance evaluation in this context has not yet reached a mature state. To propose solutions and improvements LSDC researchers face various complicated factors such heterogeneity, volatility and interference. Besides that, several platforms are available and experimentation depends on the objectives of each platform. Now, the LSDC research domain can be basically divided in at least four sub communities:

Cluster Computing : several workstations connected through a controlled network;

Grid Computing : several clusters interconnected;

P2P computing : random computing peers and some servers relying on wide range networks;

Volunteer computing : use volunteer computers and communication is mostly done using the Internet.

Today we lack of an unified approach for experimental research in LSDC. Each sub community has a set of available tools. Due to that, reuse of ideas is rare and most part of tools are short lived ad-hoc solutions. In the middle of several short-lived tools, one that resisted through the proof of time is SimGrid [24]. SimGrid is more than 10 years old. Moreover, SimGrid is far away of being ad-hoc and has successful usage in grid computing [51], cluster computing [26] and volunteer computing [29]. In this section, we start presenting some experimental platforms proposed for LSDC research. These platforms are computing environments available for deploying applications. After that, in this same section, we present several model based approaches focusing on specific communities to present several ideas for modeling and evaluating performance of LSDC.

2.4 Experimental Platforms

The most famous experimental platform today is clearly PlanetLab. PlanetLab is a world wide project spread over 516 sites and composed by 1132 nodes [9]. The aim of PlanetLab is to provide a world

wide testing platform for services and peer-to-peer computing. PlanetLab uses virtualization so different users can share nodes having some isolation. This platform is useful to conceptually prove that P2P systems are feasible. Nevertheless, the machine is constantly shared and interference of other users compromise repeatability.

Another experimental platform for LSDC research is Grid5000. Grid5000 is a french government initiative to cluster thousands of machines spreed over 9 geographical sites in France and Brazil [40]. To enable an experimental environment Grid5000 propose a complete customization of resources. That means, the user is able to use whatever operating system he wants and is also free to emulate network environments or CPUs through the usage of virtual machines. To provide such features, in Grid5000, the state of machines can be restored or deployed so that the user has complete access (with administrator's privileges) to each host. Like this, Grid5000 provides a controllable environment where the user has the possibility to completely customize each machine. LSDC research this is an initiative to provide to investigators a platform to experiment new protocols, middle-wares and applications. Nevertheless, Grid5000 is controllable in a single node scope it is not controllable at global platform scale. Notably, other users experiments may interfere with each other due to shared resources such as network and NFS. Another negative point of Grid5000 is that it is not an open project and access to Grid5000 must be granted to use it.

Platforms for real experiments are found but due to interference and shared resources repeatability is infeasible in such environments. Besides that, real experiments like PlanetLab and Grid5000 are often time consuming and scalability is limited to the available amount of computing nodes. For these reasons, model based approaches are often used as an alternative by the LSDC research community. We next list several examples of model based approaches available for LSDC research classified by target community.

2.5 Cluster Computing

A way of achieving large-scale distributed computing without incurring in the high costs of a supercomputer is to use cluster of workstations. Clusters are composed by "out of the shelves" computers as an alternative to obtain a large-scale distributed computing environment. This concept started to gain popularity with the project Beowulf in 1993, when Donald Becker and Thomas Sterling began sketching the outline of a commodity-based cluster system designed as a cost-effective alternative to large supercomputers [94]. The most innovative part of the cluster computing concept was the idea of using desktop machines that were chip. By consequence, cluster gained in popularity being much more affordable than super computers. With the gain in popularity of those systems a set of performance evaluation approaches were proposed.

2.5.1 Benchmarking

To evaluate the performance of distributed applications several benchmarks were proposed. Among these benchmarks we see often in the literature references to PARSEC [13]. PARSEC is a benchmark for memory bound and CPU bound applications. This benchmark aim at quantifying the performance of a shared memory systems.

Another benchmark used by industrial and business companies is the LAPACK benchmark. LAPACK is based on linear algebraic matrix computation and was ported to run efficiently in vectorial and shared

memory machines. Due to its applicability in many scientific domains this benchmarks was used to produce ScaLAPACK which is a parallel version of LAPACK for distributed systems. ScaLAPACK is very famous today being used by the TOP500 [81] website which maintains a list of worldwide most powerful machines. Nevertheless, SCALApack is very specific to the scientific community and its usage is questionable. Moreover, some manipulation such as checkpointing is necessary to run SCALApack in huge machines.

A different benchmark for parallel computing architectures is NAS. As cited in their own website [61]: "The NAS Parallel Benchmarks (NPB) are a few set of programs designed to help evaluate the performance of parallel supercomputers. The benchmarks, which are derived from computational fluid dynamics (CFD) applications, consist of five kernels and three pseudo-applications. " NPB appeared as a more general solution since it gathers several scientific applications (such as, fast Fourier transform and linear algebra) in contrast with ScaLAPACK which is dedicated to linear algebra.

The performance evaluation of cluster computing often rely on benchmark techniques. Therefore, since there are a multitude of possible configurations it is difficult to propose an unified performance evaluation approach and benchmarks are always limited to the target application used. However, benchmarking is useful to characterize the system performance. Nevertheless, many authors, notably the scheduling community, are interested in evaluating the system under several settings to verify its performance. In these cases, control over the platform, repeatability and speed are necessary. Due to that, many research on cluster computing often use simulation approaches.

2.5.2 Simulation

A current snapshot of simulation for cluster computing is several simulation tools that aim at the same goal. Unfortunately, due to the vast number of existent tools we are unable to present all of them. So, for the sake of brevity, we present only a few consolidated simulators that we believe can be a representative sample of current available tools.

LogGOPSim

LogGOPSim [39] was recently introduced in 2009 to provide scalable simulation of MPI applications. Communication estimations are obtained though an algebraic model called LogGOPS combined with a MPI event list. In spite of the authors claim that LogGOPSim can scale up to 8 billion processors, at the end, using a realistic test case, they are unable to scale more than 28,000 processors.

The application's behavior is achieved profiling an actual MPI code, hence, the application must be already working to obtain relevant data. Therefore, the instantiation of the platform is also done with the help of profiling, using Netgauge's LogGP. A negative point of relying on profiling tools is that access to the actual machine and application is mandatory. More over, calibrated parameters bind application and target platform in such a way that extrapolation is hard and lack of semantic.

Their accuracy evaluation is done using two different clusters as test bed. To test accuracy the author's compare each cluster at time, comparing the simulation time when using 2 real nodes against 2 simulated hosts. Afterwards, they extrapolate from this simple 2-nodes scenario to 24,000 nodes without even questioning the effect of this extrapolation in the overall accuracy. As already presented in previous works SimGrid [29] scalability is much better without of a network contention model and SimGrid can also simulate directly a MPI application with SMPI. This approaches is not yet mature enough to be compared with SimGrid under the same settings.

DIMEMAS

Another approach that aims at providing cluster programming interfaces such as MPI and PVM is DIMEMAS [28]. DIMEMAS is a promising simulator as stated in their web site, [28]:

"*DIMEMAS reads in the traces from MPIDTrace to do calculations/simulations on the application's performance. Several message-passing libraries are supported, including PVM, MPI, and PARMACS. A linear performance model is used for communication, but some nonlinear effects, such as network conflicts, are taken into account...*"

Unfortunately, DIMEMAS source code is proprietary and we were not able to assess and reproduce their results.

2.5.3 Emulation

From the lack of reliability of simulation approaches some tools recall to the use of real hardware or software or both to model cluster computing architectures. Among those approaches one that is popular is BigSim. The proposal of BigSim [100] is to provide a Charm++ and MPI parallel simulator that can be itself parallelized. BigSim, uses an emulated environment simulating many logical processors into one. As result, BigSim, have to exchange messages over time to guarantee the coherence of logical processors and real processors. In spite of the overall behavior of the application being deterministic this approach can hardly be used for debugging purposes since the order of events in distinct logical processors is non-deterministic.

Another weak point of BigSim, is the validity test scenario that are either naïve, no confidence interval and statistic analysis is made, or limited, the time of simulations were in the order of one second and only using four scenarios. In a real test case, BigSim, can scale up to 64 thousands (targeting BlueGene/C) processors but the time to simulate is very near real execution.

To mitigate the time-to-simulation of BigSim the parallelization present linear speed up to 200 processors. The authors of BigSim came up recently with a network simulator, called NetBigSim, to simulate environments and applications of network contention. However, NetBigSim was yet unavailable at the time this documment was written. For the sake of brevity we do not show other emulation approaches. However, we see already with this example that emulation leads to a fake trusting felling that since real pieces of software and hardware are used this approach does not depend on a model, and so, does not depend on the precision of the model. Nevertheless, this statement is not true, when a real software or hardware is artificially slow-down its real behavior becomes biased.

2.6 Grid Computing

A wide range of technologies to exploit computation resources have emerged in the last decade and the concept of *computational grid* have been recently proposed [33]. The main objectives of a *computational grid* is to share large-scale resources among research centers and to harness their computational power. Grid computing first intend was to produce the software necessary to interconnect clusters aiming at bringing performance to a next level. In grid computing an extra overhead is introduced to manage and deploy applications. Commonly, a piece of software called middleware is responsible for providing access to the user to the shared resources providing resources reservation, ease launching application

and collecting results, task migration, scheduling and checkpointing. When providing those features a grid infrastructure is supposed to hide low-level aspects from the user being as transparent as possible.

2.6.1 Benchmarking

In grid computing the problem of benchmarking is more complex in comparison with dedicated clusters. On the grid, the middleware must also be evaluated since great part of the complexity and time spend running an application is concentrated there. Due to that normal benchmarks approaches are less suited to evaluate the performance of cluster environments and there is not yet a consolidated approach for evaluating grids. Motivated by this problematic some researchers recently proposed GridBench [92]. GridBench proposes a framework for conducting experiments, measure the performance and interpret the results on the grid. With GridBench, users can describe system performance in various levels, so that the performance can be detailed per site (cluster or SMP), per virtual organization or middleware. Although useful, benchmarking in grid computing is much more complex than in dedicated environments, due to that most part of performance evaluation approaches in the field rely on simulation.

2.6.2 Simulation

Analogous to cluster computing the grid computing community has several simulation tools that were proposed. Unfortunately, again we do not have enough space to present all these approaches. Moreover, such approaches are often short-lived and very similar. For these reasons, it is out of scope of this thesis to present them all. We will next present some representative simulators which had a bigger life time and a relatively big number of users from outside the project.

OptorSim

OptorSim [11] is a simulator written in Java launched as part of the European Data Grid (EDG) project. OptorSim was created as a specific tool to evaluate dynamic data replication strategies on the grid infrastructure. Modelling separately platform and applications it uses a contention aware network model. Nevertheless, the contention model implemented by OptorSim is not assessed in an accuracy study. Moreover, OptorSim scalability is poor, a few thousands processors and today the OptorSim project is apparently discontinued. In their SourceForge website the last files date from Mars 2008. Besides that, they explicitly mention that new releases are not expected since the project maintainers are now assigned to different projects.

GridSim

From all projects presented in this chapter, GridSim, is the only one that has evolved similarly as SimGrid. GridSim, is a simulator written in Java which first intent was to provide an environment for grid economy experiments. The last advances in GridSim design target to transform it in a more generic approach. Although a similar goal of GridSim and SimGrid, i.e., large-scale distributed computing research, their authors apparently neglected the fundamental aspect in the LSDC context: scalability. In spite of providing a set of built-in functionalities GridSim scalability is limited to a few 1,000 of processors [24]. Moreover, SimGrid provide an extensible API were extra functionalities can be easily implemented.

SimGrid

SimGrid was born in 1999 with the specific goal of providing a reproducible environment for experimenting scheduling algorithms in heterogeneous grid platforms [24]. Since then, the project evolved to a generic framework for large-scale distributed computing research. Today SimGrid is fast becoming *the* state-of-the-art simulator for LSDC systems. The proof is the number of publications from non-developer authors, 54 since it first version in 2000 [79]. SimGrid was already used fomenting experimental research on different LSDC sub communities like grid computing [10], P2P computing [29] and cluster computing with the add of the SMPI module recently. Moreover, available scalability studies [24], show that SimGrid is the most scalable simulator available today deploying more than 100,000 processors.

2.6.3 Emulation

Emulation can be also used to evaluate the performance of grid computing environments. Nevertheless, the literature of available performance evaluation approaches for grid computing is not as vast as the one found for simulation approaches. This remark made, we are still able to find projects that focus in grid emulation such as MicroGrid. MicroGrid is strongly based on the MaSSF simulation framework so that a different network is simulated using several hosts as network simulation engine. Besides that, MicroGrid introduces the possibility to use several virtual machines in one application so one is able to have a set of simulated hosts and even one instance of MaSSF network engine running on the same machine. Using this approach the computation is emulated through virtual machines while the network is simulated/emulated by those several MaSSF running simultaneously [54].

2.7 Peer-to-Peer Computing

The goal of Peer-to-peer (P2P) systems is to share resources such as, network bandwidth, processing power and storage. The only requirement for that is to be interconnected through a network and install a P2P client. Moreover, depending on the P2P system it might require a server responsible for keeping track of available resources and their location. A problem that arise from P2P system is related to selfish users that just want to consume resources without providing any in counterpart. At this point, sharing policies must be implemented to privilege non-selfish users giving more credit to those who share more.

A multitude of P2P protocols gained in popularity due to the sharing nature of mankind. Most part of these P2P systems at beginning provided a distributed network to share music, books and files like Napster, Gnutella, eDonkey, Kaaza, Torrent. Distributed computing researchers soon observed a potential for computation in such environments. So, aiming at providing computational power P2P computing environments started to appear. Among others one project that survived the test of time is OurGrid [66, 7]. OurGrid is a Brazilian project to provide a free grid for every one that needs it.

2.7.1 Benchmarking

If benchmarking is hard in grid computing, in P2P computing it is even harder. In P2P systems heterogeneity and volatility are beyond control and testing the system performance under certain criteria is difficult due to the absence of one centralized point. Nevertheless, some benchmarking specific approaches were proposed there is no unified approach [64]. Moreover, P2P systems performance is

directly connected to the number of users that join the system and this last is a variable completely unknown. For all these reasons benchmarking is not adapted to the performance evaluation of P2P computing environments and P2P computing research often boils down to simulation and emulation.

2.7.2 Simulation

In P2P systems the aim at performance evaluation seems to be less important than having proof of concepts. ProtoPeer [36] provides a toolkit for simulation and deployment of distributed computing applications. A main feature of ProtoPeer is the possibility to run the same code using the simulator and a real P2P environment. To do so, it implements policies easing the implementation of communication primitives. Today, ProtoPeer scales up to 20,000 processors and as most of the related work presented here it is probably being discontinued. The last post in the project blog dates from Mars 2009 (almost two years ago) when their first publication appeared [72].

2.7.3 Emulation

Contrary to grid and cluster computing, in the P2P community a strong appeal to emulation. Among several approaches we present here MACE and Splay which are representative examples of the most common emulation techniques in the field.

MACE

The MACE [48] project is a redesign of the MACEDON project aiming at simplifying the development, research and deployment in distributed systems [56]. One new aspect, introduced by MACE is the ability of model checking enabling the user to verify properties of the application such as liveness. Differently, of many related work MACE is still under development. MACE main focus is to add debugging functionalities to distributed environments and hence, unsuited for performance evaluation.

Splay

Splay [52] also proposes an environment to implement and deploy distributed LSDC applications. Although different from MACE, Splay furnishes access to real platforms such as PlanetLab and ModelNet. With access to the platform the user is able to deploy the application and rapidly test it on the real world. Nevertheless, as MACE, Splay lacks of performance estimation approach. As so, Splay suffers from the problems of real execution experiments, i.e., performance estimations can only be assessed when the system is in production.

2.8 Volunteer Computing

Volunteer computing, consists in exploiting idle cycles of workstations to achieve high performance computing environment. Indeed, nowadays almost all universities and enterprises are equipped with many computers that are idle in a considerably part of the time (at lunch time, when the university is closed, in vacations, and so). In volunteer computing, users have the possibility to do some useful computation while their desktops are idle. The principal is simple, the user that wants to join the computing environment install a client. This client is hence able to detect when the processor goes idle (like a screensaver

program) searching at this point for useful computation to do. As result, volunteer computing makes possible to profit from unused resources compounding a LSDC environment.

The oldest project on volunteer computing is BOINC (Berkeley Open Infrastructure for Network Computing) [5]. Launched in May 1999, BOINC has harnessed over two million years of CPU time from over 5.3 million volunteers spread across 226 countries. Nevertheless, BOINC systems are rarely available for experimentation and deploying a test set for debugging purposes is possible at best at moderate, non-representative scale. Examples of projects that use BOINC are: FightAIDS@Home [95] and SETI@Home [96]. An interesting method put in practice by BOINC is a credit system. To motivate users to continue to share resources BOINC put in practice a simple game mechanic rewarding with virtual credits users as they return processed packages. This mechanism make users connected to the project they donate CPU so they feel like part of the scientific effort and compote between them to receive more credits.

2.8.1 Benchmarking

Volunteer computing systems are more centralized than it might appear. In spite of clients being world widely located the servers are just a few creating a minimum centralization. As so, volunteer computing designers are easily able to access the amount of delivered power by the environment. Consequently, the interest in benchmarking for volunteer computing systems is limited to estimate the processing power of clients with scheduling purposes.

2.8.2 Simulation

From several volunteer computing environments one that is widely known and used in BOINC. Followed by BOINC popularity several simulation approaches were proposed to model and evaluate its performance. Here we present several BOINC related simulators which are representative of others volunteer computing simulators.

SimBA

SimBA [30] is a custom simulator for BOINC proposed by Estrada *et al.* to investigates threshold-based scheduling policies. SimBA is a discrete-event simulator that models hosts as a set of finite-state automata. The parameterization of the automata describe several characteristics (e.g., compute rate, error rate, timeout rate). Moreover, the parameters model the failure or success of each task using a uniform probability distribution. To model volatility, SimBA uses a Gaussian probability distribution. One negative aspect of SimBA is that the simulation models implemented inside it are too simplistic and its accuracy was only superficially evaluated.

SimBOINC

Kondo *et al.* implemented a simulator that uses SimGrid as a back-end called SimBOINC [78, 20]. SimBOINC uses a modified version of the BOINC components, relying on the original source code. This technique can be seen as an emulation approach since BOINC clients are running the real code with a few modifications. One weakness of this simulator is that it is hard to maintain since the modified BOINC source code is constantly being developed. One advantage of the SimBOINC in comparison to SimBA

is that it allows for the performance evaluation of a complete BOINC system with many projects and many clients.

2.8.3 Emulation

As we already saw for other domains volunteer computing also follows up a series of emulation approaches. To keep coherent we particularly focus on BOINC related emulation approaches since BOINC is one of the most popular volunteer computing environment.

BOINC Client Emulator

The BOINC Client Emulator (BCE) [15] was developed as part of the BOINC project proposing an emulated client that runs directly the BOINC client source code. As so BCE is expected to accurately represent the scheduling policies implemented inside the BOINC clients. However, the BOINC Client Emulator lacks of an accuracy study.

emBOINC

After proposing SimBA, Estrada *et al.* developed a new BOINC emulator called EmBOINC [31]. To achieve EmBOINC they modified the BOINC server source code so that it could run inside EmBOINC. This approach is complementary to the BCE. Using emBOINC one is able to simulate a server that can handle multiple clients sending fictitious or real requests to a server based on real BOINC source code. These modifications were integrated in the BOINC source code. Recently, SimGrid was compared to a series of BOINC simulators beating all of them in scalability, speed and configuarability [29]. Moreover, SimGrid can be used to simulate an arbitrary number of BOINC clients and servers at the same time which is now only possible with SimBOINC.

2.9 Conclusion

In this chapter we listed related work of performance evaluation approaches used in LSDC and related domains. Our aim was first to point out the need for an unified method for performance evaluation in LSDC. A second goal was to present that using specific low-level performance evaluation tools (such as, packet-level and CPU's cycle accurate simulators) we can hardly achieve speed and scalability to match the performance evaluation constraints of LSDC.

Further, we presented, in this chapter a set of available tools designed for performance evaluation of LSDC. As presented, most tools have a short life time. Conceived aiming at investigating specific phenomena, those simulators are often used only by their own developers being discontinued after the main research goal is achieved. Moreover, being too specific or too simplistic they rarely aim at proposing an unified approach for performance evaluation in LSDC in spite of presenting roughly the same ideas over and over again. Differently, SimGrid evolved to a generic experimentation framework for research in LSDC. SimGrid enable to conduct fast performance evaluation of LSDC systems.

Today, the gap to fill to consolidate SimGrid as the choice for performance evaluation in LSDC is a comprehensive accuracy study. However, to be comprehensive we need to look at SimGrid internals and compare it modeling choices with other approaches to justify that the choices made in SimGrid

design are the most adaptable ones. In the next chapter we focus on the internal parts of SimGrid and related approaches aiming at showing that SimGrid respect the trades to become the *the* facto tool for performance evaluation of LSDC environments.

Chapter 3

Models Underlying Performance Evaluation Tools

In the last chapter we presented various performance evaluation approaches for LSDC research and related domains. At the end, we were able to conclude that great part of the proposed solutions are model based approaches, such as simulation and emulation. By model based approaches we understand those approaches that replace a real component by a model that imitate its behavior. Therefore, introducing a model one achieve control over the experiment parameters and gain in speed. Nevertheless, when a model replace a real component an error is introduced. In spite of most part of LSDC researchers being aware of that, many of them conduct superficial accuracy studies or not at all. Hence, the study of simulation accuracy is still an open research question in the LSDC domain. Contrary of most work until know published in LSDC domain, our goal with this thesis is to present a comprehensive accuracy study of the SimGrid framework. With that we look forward to evaluate that the simulator guarantees a certain level of realism and, at the same time the speed of simulations is not compromised by that fact.

In essence, most works rely on model based approaches because it is difficult and time consuming to do real experiments. Due to that, most approaches try to simplify the studied system replacing real components by models. Therefore, using models is atractive because we can speed the response time of the simulator. However, speed comes with the price of introducing some error and it is of last importance to analyze the value of this error in the overall accuracy. The goal of this thesis is to provide accuracy, fast and scalable models for simulation. Consequently, evaluating the realism of models and speed is mandatory to achieve the goal of this thesis. To achieve that last, in this chapter we analyze several models that were (and most part of them still are) used to model the basic components of CPU resources and network resources. So, both resources are the basic building blocks of any LSDC environment.

3.1 Quality Metrics of a Model

A model is based on a set of assumptions. To evaluate if these assumptions are good or not we have several possible quality metrics.

Accuracy : models need to be confronted with other approaches to be trusted. Consequently, the error introduced by the simulation need to be known before trusting in simulated results.

Speed : particularly LSDC applications may take hours, days or even weeks to run. Due to that, models adapted to LSDC must be fast so one is able to explore different alternatives in feasible time. In other words, the faster the model is the better it is.

Scalability : in LSDC we are interested in going large-scale, or in other words, to stress the number of processing units simultaneously working. So, the scalability is also of mandatory importance and models that scale better are preferable.

Instantiation : As we will see in this chapter some models use parameters that are hard to obtain in practice or even that do not have a particular meaning in reality. Since we expect our model to be realistic we also expect that users instantiate those models with parameters that are realistic. Consequently, simpler models are preferable than complicated ones. Hence, we look forward to have just a minimum set of parameters figuring only those that have an actual connection with the modeled reality.

Repeatability : Models have a significant advantage to reproduce results. Particularly, in LSDC, due to heterogeneity, volatility and interference, it is hard to repeat experiments in practice. For that reason, one of the main advantages of using models in our context is to reproduce results. For that reason, a good model in context of LSDC must guarantee repeatability.

Implementability : Some models may look logical with realistic parameters and so on. However, some models are hard to be effitiently implement in practice. This metric is related to the possibility of efficiently implement a model.

Following we will present many models used for CPU, network, and both used together from an application perspective. We follow this same order to increase readability. So, next, to start we evaluate the set of model available for modeling CPU resources.

3.2 CPU Models

Since the beginning of computer science some models were proposed to understand CPU behavior. Among many models, one that gained in popularity the model proposed by Von Neumann. This model separates the memory used for executing the instructions from the memory used to store data. With this separation it was simpler to propose abstract programming concepts such as functions and objects. Nevertheless useful, we are not interested in those type of models on LSDC, the reason for that is that such models are too detailed to be fast and scalable to match our requirements.

3.2.1 Fluid Models for CPU Bound Applications

Another approach to model the CPU is to evaluate it as a computing unit associated with a working rate. In this case, the CPU is seen as a producer of processing power while the application consumes it. A simple model that uses this idea is described in Equation (3.1). In the model of Equation (3.1) we know before hand the amount of work that one application A requests, denoted by Work_A. Since CPU

processing power working rate is also known, we can estimate the time to finish the application dividing the amount of work-to-do by the CPU working rate.

$$T = \frac{\text{Work}_{\text{A}}}{\text{CPU}_{\text{working rate}}} \tag{3.1}$$

If we classify this model using the metrics that are relevant in the LSDC scope we have:

Accuracy : unknown, probably will be low, need to be verified, assumes several simplifications, for instance, assumes that the application is CPU bound;

Speed : very fast;

Scalability : promising;

Instantiation : the work-to-do associated to the application need to be measured;

Repeatability : possible;

Implementability : straight forward.

This model rise some questions about its accuracy and instantiation. Well, the accuracy problem is an open question that is addressed in this thesis. However, a possible solution for the instantiation problem is to run simple benchmarks that can give an overall performance of the system. However, sometimes, such model may be inaccurate when the application is memory bound. In such cases a possible approach is to use fluid models that take the memory hierarchy into account.

3.2.2 Fluid Models for Memory Bound Applications

One model that take the memory hierarchy into account is proposed by Snavely *et al.* [83]. This model is based on tracing and profiling the program memory access. After that, a model is extracted to estimate the processing time of different applications on different CPUs. Nevertheless, this approach is based on many parameters that need to be measured before extending the model. Moreover, in their work the authors extrapolate their measures to different processors and application. It is difficult to trust in such extrapolations since measured traces are dependent of low-level aspects such as, cache access pattern, cache size, cache coherence policy and so on. So, in spite of accurate, such approaches have to be tuned in a case-by-case basis. Moreover, the instantiation of this approach is complicated since one needs to trace a certain number of parameters and tune the application.

3.3 Network Models

Since a long time network models are being studied to estimate the time taken to transmit data. In LSDC systems data travels often using wide area networks such as the Internet. When this is the case, the instability of peers and the possible losses in transmission are present and must be treated. So, to cope with those problems a possible alternative is to use handshake protocols that guarantee congestion avoidance, error control, and serialization of packets like TCP. TCP, the Transfer Control Protocol, was invented as an initiative to share the network among different users so that each one is able to use the

network simultaneous even when congestion is present. By consequence, TCP is the worldwide used when congestion in the network is detected.

For those reasons, in this section, we present many methods of modeling network. Therefore, we start by presenting the simplest network model possible followed by some slightly complex one's.

3.3.1 LogP and Spinoffs

Also motivated by the lack of communication cost in PRAM the LogP models emerged. In LogP [27], the machine is described by a set of parallel processor plus other three communication parameters.

latency (L) : a constant time to send a small message;

overhead (o) : the time spend processing a message before sending or after receiving it;

gap(g) : a waiting time between to consecutive transmissions.

From those parameters the name of the model, LogP, comes from. Based on these three parameters the LogP model can efficiently predict the time needed to send small messages considering an accurate level of detail in communication as presented in Figure 3.1.

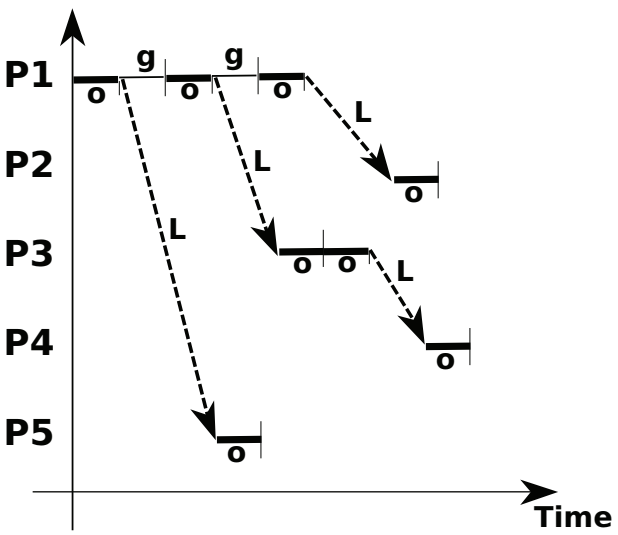


Figure 3.1: In the LogP model communication is more detailed. Before sending or receiving a message one waits for the overhead (**o**), the time taken to transmit is dictated by the latency (**L**) and between consecutive transmissions a gap (**g**) is respected.

The LogP model is inspired by cases where several small messages are sent generating few contention. As consequence, those models present some limitation for long messages. To be useful even for big messages the LogP model have had some extensions like: LogGP [4] and LogGPS [41]. Both extensions use the idea of network bandwidth (**G**) so that the time taken to transmit long messages depend on the throughput of the network link. Another weak point of LogP models are that such models are very homogeneous so that the same parameter L, o, g are used for each processor on the system. One way of countering this limitation is to introduce the usage of a vector of values for each processor on the system. However, when doing this the model become so complex that it becomes hard to instantiate.

3.3.2 Single Link Analytic Models

To understand network mechanics analytic models were proposed since 1994 [38]. In his work, Roger W. Hockney, proposes a simple fluid model to approximate transfer time ignoring contention for a single link. The Hockney model state that a network connection has a transmission capacity rate and latency. The capacity is an upper bound of available bandwidth while latency is the time needed to arrive from one peer to another making an analogy to fluids inside pipes. Knowing the bandwidth, latency and the amount of data to transfer the model states that the time to finish the transmission is roughly approximated by $\frac{S}{B} + L$ as presented in Equation (3.2). Where B is connection bandwidth, S is the number of bytes to transfer, and L the connection latency capturing the basis of a network transmission from an end-user's perspective.

$$T = \frac{S}{B} + L \tag{3.2}$$

Being fast, the Hockney model is useful to predict the time to finish a download when contention is absent. Nevertheless, contention is one of the most common phenomena in the network domain and even for a single link complex analytic models need to be added to the Hockney model so that the bandwidth reflects the sharing of TCP. We present an example of analytic model that cope with contention for a single link [67] in Equation (3.3).

$$B = min\left(\frac{W_{max}}{RTT}, \frac{1}{RTT\sqrt{2bp/3} + T_0 \times min\left(1, 3\sqrt{3bp/8}\right) \times p\left(1 + 32p^2\right)}\right) \tag{3.3}$$

Where:

- W_{max}, maximum size of the advertising window;
- RTT, round trip time;
- $T0$, TCP average retransmission timeout;
- p, packet loss rate;
- b, number of packets confirmed per ack.

As one can notice, in this model the bandwidth allocated to the flow depends on more parameters. Indeed, two relevant parameters are the round trip time (RTT) and maximum advertising window (W_{max}). In practice, those two parameters act as a lower bound for the bandwidth due to the protocol characteristics. TCP limits the maximum amount of data a sender can transmit (W_{max}) until a confirmation ack is received (RTT). So, when this is the case the time waiting for acks eventually limits connection throughput by the factor of these two parameters $\frac{W_{max}}{RTT}$. A drawback of this model is its parameterization since some parameters are in practice very hard to obtain, such as packet loss rate (p) and number of packets confirmed per ack (b). Moreover, this model is too much detailed for one link creating serious doubts about it extensibility for several links, i.e., in a typical LSDC system usage case.

Contention models, like the one we just described, give useful insight of the direction to improve estimations accuracy when the bandwidth allocated to each flows is known, Equation (3.4). Indeed,

their main contribution is the idea of using the Hockney model as basis focusing on the problem of bandwidth sharing from an isolated perspective.

$$Time = L_c + \frac{Size}{\min\left(B_c, \frac{W_{max}}{RTT}\right)} \tag{3.4}$$

Where:

L_c : sum of latencies from all links that connection c pass through L_c;

B_c : throughput allocated to connection c;

W_{max} : maximum size of the advertising window;

RTT : round trip time.

The advantage of this last model is to profit from the better parts of the others presented. However, this model only works for a single link if the bandwidth share is known before hand. To remain functional, such a model need to incorporate a network contention model. To fill that gap, many authors proposed ideas of bandwidth sharing on several links. In this context, we present next several techniques to model multiple links and some proposed network sharing models evaluating their advantages and flaws.

3.3.3 Store & Forward

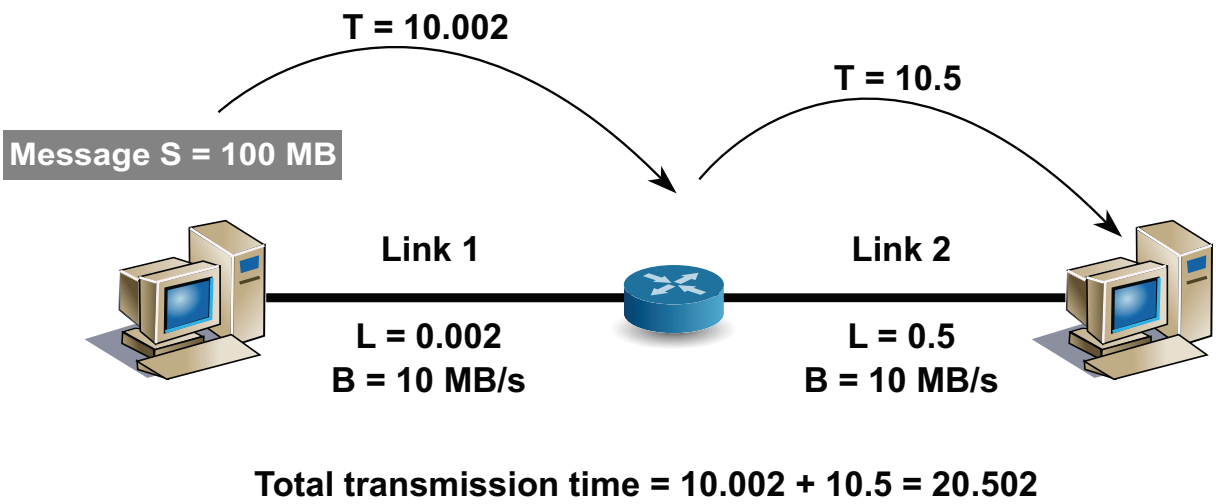


Figure 3.2: Store & forward approaches does not reflect the behavior of TCP for multiple links even when contention is neglected.

One alternative to model multiple links is to think at each link as an exclusive access resource. In this idea, a resource is used by one connection at time so that the time to pass through a link is computed one at time. For instance, in a store & forward approach to transmit 100 MB passing through two links one need to pay the price to pass through each link at time as illustrated by Figure 3.2. However, this model works very different from the way the TCP protocol works. In first instance, TCP avoid to make a peer which is in the middle of a transmission to receive the entire message before sending to its final destination. Second, this model sees a message as a single unit of data while in practice the message is split in smaller pieces. As so, store & forward approaches are not adapted to model TCP behavior even when contention can be neglected.

3.3.4 Wormhole

A better approach to deal with multiple links is to split a message in a set of smaller packets. Indeed, this idea reflects roughly the way that the TCP protocol will treat messages. In Figure 3.3 we have the same example previously presented in Figure 3.2, although, now the message is split in many smaller packets. After that, a similar store & forward approach can be used to each smaller package. Eventually, when the transmission reaches steady-state the resulting throughput of flows using this approach will be close to the TCP achieved bandwidth.

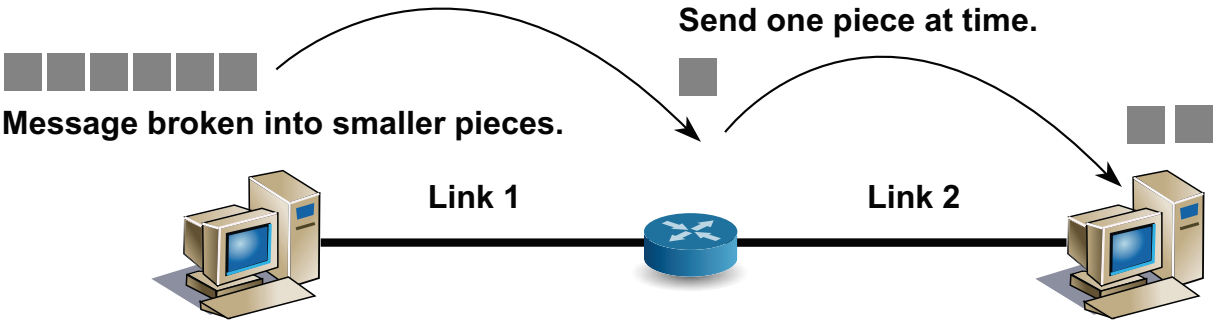


Figure 3.3: Wormhole models are commonly used on LSDC simulator, however such models are not scalable generating as many events as a network packet-level simulator.

Several LSDC simulators rely on the wormhole strategy such as GridSim 4.0 [16], SimGrid until 2002 and most ad-hoc simulators. However the protocol fragmentation mechanism is more complex than this. In practice, TCP has finite waiting queues and managing such approach becomes as complex as reinventing packet-level simulators such as GTNetS, NS-3, and so. Due to that, results using wormhole are as time consuming as packet-level simulators without being as accurate as packet-level simulators. Moreover, the use of this approach is too slow to model several links in LSDC systems. A better approach, when such level of detail is needed would be to use a packet-level simulator directly. However, such approach will impact in scalability.

A better compromise of this detailed model are the use of fluid models that propose modeling the entire network. We next observe such models.

3.3.5 Bandwidth Sharing Models

Macroscopic modeling TCP can be harder than it seems at first sight. In practice, several works propose models to TCP relying in a macroscopic approach. Therefor, all such models rely on some analytic description that extracts the fairness of TCP when connections reaches steady-state. As so, such models rely on a mean behavior without considering the slow-start phase.

A typical model that rely on this idea is the Max-Min model [57]. This model introduces the problem of modeling the network and a set of links with limited capacity working rate (the bandwidth) and that many flows compete for the access of those links. In this last the author's conclude that a fairness problem that approximates TCP behavior is to maximize the minimum bandwidth received for each link. So, this model is also referenced as Max-Min where the maximum throughput can achieve by less privileged flows. In other words, the congestion control of TCP tries to guarantee the best minimum bandwidth for every flow in the system.

A work that follows up this line of thought was presented by Steven H. Low in 2003 [55]. In his work, Low studied several implementations of TCP, the Internet Transfer Control Protocol, expressing mathematically the laws of fairness behind specific algorithm implementations. As so, Low see the fluid model from a more complex perspective as a general optimization problem. Low also proposes that the fairness of TCP may completely change from implementation to implementation. Consequently, Low proposes several fairness functions presenting some directives of how implement it.

We believe that bandwidth sharing models are ideal for LSDC simulation because they are fast, scalable, parameterizable, repeatable and easy to implement. However, we still need to verify their accuracy to determine if they are precise enough in LSDC. Since the goal of this thesis is to verify this property we will explain such an approach in details in Chapter 5. Nevertheless, for the sake of completeness we explained here the concept to be able to justify this choice and also to be able to compare it with other approaches.

3.4 Application Models

Since the beginning of parallel computing many authors started to propose methodologies to evaluate applications. Those, so called application models, emerged as an effective way of analyzing complexity from a programming perspective. However, those models, have are too abstract often relying on unrealistic assumptions. Moreover, such models are not meant to produce measures of application performance but instead to understand its behavior when increasing the number of processors.

One intuitive application model consists of evaluating the application as a set of operations or instructions. In this case CPU resources are seem as consumers of instructions, so, the rate at which a CPU consume instructions is the CPU working rate. Working rate based models, like this, are straight forward to conceive. Nevertheless, a possible implementation requires to execute the application to calibrate the model with a reasonable working rate. From these informations, amount of work to do and working rate, one can obtain completion time estimations by dividing the number of work to do by the working rate. A model like this is easy to implement, fast, and indeed realistic for a single CPU if the application number of operations is known before hand, and if the CPU working rate is characterized using a similar application. However, such models are too simplistic to characterize the performance of distributed applications since it does not consider the time penalties needed to exchange messages.

3.4.1 Shared Memory Applications

Another possible approaches focusing on parallel machines is to estimate performance on multiple computing units using abstract analytic models [86]. Mainly three models exist and their derivations: PRAM, BSP, LogP.

PRAM

The PRAM acronym stands for Parallel Random Access Machine. This model was introduced to analyze the complexity of parallel applications. In PRAM, all processors access memory homogeneously like in SIMD (single instruction multiple data model) relying on an infinity shared memory. The idea of PRAM is to propose a model where assumptions such as memory and number of processors can be easily extrapolated. Such assumptions are interesting to evaluate algorithms extracting their asymptotic

behavior as a function of problem size and number of processors. Typically, using PRAM one is able to estimate theoretical speedup and efficiency.

To coordinate memory access the PRAM model propose several approaches specifying the way the memory can be accessed. Generally, in shared memory system, the memory is either concurrent (simultaneous access is possible) or exclusive (simultaneous access is forbidden) for either reading or writing. Generally a PRAM model may follow one of the following specification: EREW (stands for exclusive read and exclusive write), CRCW (concurrent read and concurrent write), or CREW (concurrent read and exclusive write).

The PRAM model presented itself very efficient to model and understand the characteristics of parallel algorithms. With PRAM one cans reason, compare, analyze and design algorithms. However, PRAM models are over-simplistic ignoring synchronization and communication times. As so, such models are unrealistic in particular in the domain of LSDC where communication and synchronization is so important. In spite of being a good model the PRAM model has a series lack of realism as stated before [27]:

"*The PRAM is the most popular model representing and analyzing the complexity of parallel algorithms. The PRAM model is simple and very useful for a gross classification of algorithms, but it does not reveal important performance bottlenecks in distributed memory machines because it assumes a single shared memory in which each processor can access any cell in unit time. In effect, the PRAM assumes that interprocessor communication has infinite bandwidth, zero latency, and zero overhead...*"

PEM

Recently, some authors proposed an extension to the PRAM model so that it can cope with the memory hierarchy. The Parallel External Memory (PEM) [8] model proposes a two level hierarchy so that process have an external memory (main memory) that is shared by all processor and each processor has a dedicated faster memory (cache). Like in PRAM those models have the possibility to be customizable so that one is able to decide the policy when accessing memory, CRCW, CREW or EREW. In the case of exclusive read and write simultaneous access to the main memory to the same block are handled using a conflict resolution strategy. The PEM model is an extension of the PRAM model aiming at providing complexity analysis mechanisms for parallel algorithms. As so, the PEM model relies on parameters that are far from the desired level of realism necessary to evaluate LSDC system's performance.

3.4.2 Message Passing Applications

To overcome the fact that PRAM models took for granted communication the BSP (Bulk Synchronous Parallel) model has a main concern in communication and synchronization [82]. In such model the application is a set of supersteps. A superstep is a set of completely parallel computation followed by communication and a synchronization point. As so the software execution goes in cycles of many supersteps alternating from computation, communication and synchronization. Indeed, many iterative methods follow this idea. Numerical methods are an example of application that can fit this behavior. For instance one equation system is split in many processor that compute each one a part of the approximated eigenvector of the next iteration. After computing, the process need to exchange their partial eigenvectors so that each process have a consistent copy of the complete eigenvector. As an iterative method this cycle must be iterated many times until the eigenvector solution is achieved.

In spite of the last example being a precise one, indeed, many other applications follow this behavior. Some examples are: iterative methods for solving equations (as mentioned before), natural science

simulation, genetic algorithms, divide conquer strategies and so on. The main advantage of this model, instead of PRAM models, is the communication part, ignored in most PRAM models. Due to that, the BSP algorithmic complexity of an application is often split in two, analyzing computation and number of message exchanged separately per superstep.

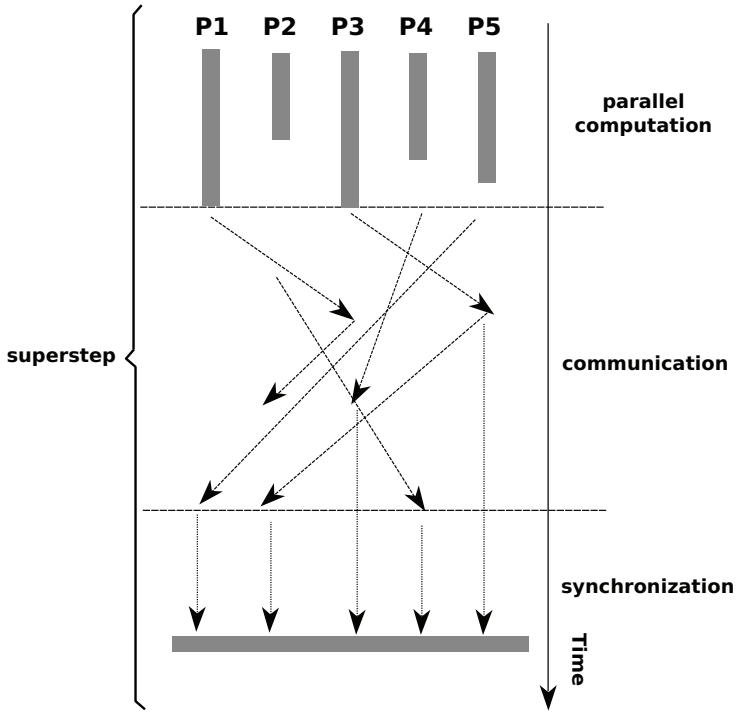


Figure 3.4: A superstep for the BSP model is a set of parallel computation followed by one communication and one synchronization phase.

Although being more realistic than PRAM models, BSP models are also abstract assuming that all processors are homogeneous and that there are no contention issues at network level. By consequence, BSP models assume that the network transmission have reached steady state and that the slow start phase can be unconsidered. To resume, BSP models gave also another step further showing that communication cannot be all the time neglected while evaluating parallel applications' complexity. However, such models are still high level models making some naïve assumptions about the processors characteristics. This compromise their use in LSDC systems.

To resume we saw, this section, that the PRAM model, assumes an infinity number of processors neglecting completely network latency and contention. The Bulk Synchronous Parallel model, BSP, present a similar approach but using finite resources and adding the idea of communication and synchronization steps. Therefore, the BSP communication cost is mostly estimated based on the number of messages per step neglecting propagation effects. The LogP model, use more detailed information to accurate estimate communication cost. As we can observe models like PRAM, BSP or LogP are repeatable and useful to understand the limits of parallel algorithms. However, they are not adaptable to LSDC because they lack of detail when defining volatility and heterogeneity which are essential in LSDC. For that main reason many of the available simulators today recall to alternate models modeling network and computation separately. Following, we visit the set of available models available today to estimate network or computational time.

3.5 Conclusion

In this chapter we focused on the internal parts of the performance evaluation approaches proposed for LSDC. In spite of most approaches are based in analytic modeling, some models resort to discrete-event simulation. Those models that use pure simulation tend to be inaccurate and too slow to satisfy the scalability of LSDC systems. Contrary, approaches that use fluid models assume that the application reaches steady-state and can be used to approximate complicated phenomena such as communication cost and computation cost.

In this chapter most part of LSDC performance evaluation tools use either wormhole or constant models for network. Although useful, both models are either too slow or inaccurate to be used in LSDC where interference and hence contention play an important role. Therefore, we can detect a main flaw in the various LSDC simulators or performance evaluation approaches: most part of tools propose a solution to the problem without questioning accuracy of the basic models. To continue to fill this gap in the next chapter we detail SimGrid.

Part II

Context

Chapter 4

SimGrid

In the previous chapter, we presented an overview of performance evaluation approaches for LSDC systems. As we saw, those approaches can be either based on models or analytic. Moreover, we presented that a preferable in-between solution is to combine analytic models with discrete event simulators. Following this idea, one tool for simulating LSDC systems is SimGrid.

SimGrid is a generic framework for performance evaluation on LSDC research. It has been designed since 1999 and today counts on several users from different LSDC communities. One of the main reasons for the SimGrid's success is that it gathers several models adapted to design the characteristics of LSDC system. As so, SimGrid is versatile and flexible, enabling users to customize the desired level of heterogeneity, interference and volatility. Moreover, SimGrid is open source and users are welcome to add their own modules. Like that, parts of the simulator can be reused while other parts can be adapted to reflect specific needs. SimGrid has also good scalability in comparison to similar tools available today. For those and other reasons that will be presented in details during this chapter, SimGrid is becoming a standard for building LSDC simulators.

4.1 History

SimGrid started to be developed in 1999 when Henri Casanova received Arnaud Legrand (a student from Lyon) to code a simple simulator for testing scheduling algorithms on the grid [79]. At that time, Henri was working as a post-doc researcher in the AppLeS research team in the Computer Science and Engineering Department at the University of California, San Diego. Henri fast understood that this specific simulator would be helpful for other Grid researchers. So, after that Legrand had left, Casanova created a package distributing this simulator with the name SimGrid. This was the first official appearance of SimGrid in 2001 [23].

Later in that same year Legrand started a Ph.D. at ENS Lyon (Ecole Normale Supérieur de Lyon) on the studying of scheduling algorithms in heterogeneous platforms. Motivated by the need of a more generic application description he started to rewrite the SimGrid's interface. As result he coded MSG (Meta-SimGrid) adding a top layer API to the original kernel. The main features integrated with this module was the ability to manage concurrent communication and computation enhancing interference realism. Two years later, in 2003 Loris Marchal, detected some flaws in the contention model first designed. Consequently, in this same year, Marchal with the help of Casanova, modified the original simulation

kernel implementing a more realistic communication contention model [25]. Those modifications were available in 2003 with SimGrid's version 2 [51].

Two years later, in 2005, many features were incorporated in SimGrid. A main contribution was GRAS, implemented by Martin Quinson during his Ph.D. at ENS Lyon [73]. GRAS aim was to provide a common interface to develop and simulate distributed applications. Meanwhile, Legrand cleaned and improved the simulation kernel gaining in speed and modularity. As result of his work SimGrid performance was greatly improved from the first simulation kernel. Moreover, modularizing the code he made easier to other users to add custom simulation modules.

In 2007, Bruno Donassolo advised by Legrand factorized the MSG code. At that time, Donassolo was an undergraduate student from UFRGS/Brazil abroad in Grenoble University through an interchange program. As result of his work he separated in two the MSG interface. One part implementing support for concurrent process operating system functionalities (SIMIX), and another keeping backward compatibility with the MSG interface. All those improvements are now available since 2009 when SimGrid version 3 [24] was released.

Today SimGrid is officially supported by the french government through the USS-SimGrid project [80] (ANR 08 SEGI 022). Being an active project, several features are under development, to cite a few:

- Tracing and visualization features are under development by Lucas Schnorr.
- Improvements of platform file format to minimize the size of routes were recently added by Pierre Navarro and David Gonzalez.
- An interface to simulate collective and one-to-one MPI communication primitives were implemented at Nancy by Pierre-Nicolas Clauss and Stéphane Genaud.
- Model checking and better thread management are being developed at Nancy by Cristian Rosa and Martin Quinson.
- Validation of simulation accuracy, which was started by Kayo Fujiwara and is the subject of this thesis.

This discussion is a preliminary introduction to the SimGrid that demonstrates one of the strongest points of this simulator, the people that work dedicated to make it even better. Moreover, making reference to those people we hope to present the amount of effort that is invested in SimGrid which is in great part the responsible for its success. This discussion is not meant to be comprehensive but to present an overview of the SimGrid evolution from a historical point-of-view. Many people that contributed, or is today working with SimGrid are not mentioned here, to a complete list the interested reader is invited to consult the references [24, 80, 79]. We next present how SimGrid works.

4.2 SimGrid in a Nutshell

A common miss-conception is to believe that SimGrid is a tool dedicated for grid computing. After a short historical overview, it becomes clear that the name SimGrid comes from its past and is no longer suited. SimGrid has evolved into a generic simulation framework for LSDC research. The proof is that SimGrid has today successful usage cases in different LSDC domains such as cluster computing, grid computing and volunteer computing [24, 77, 29].

Different LSDC communities have different goals. Volunteer computing researchers, for example, do not aim at the same performance measures as P2P researchers and must adapt or rebuilt a simulator that fit their needs. In spite of the desired performance measure we can state that from these several communities there are still a huge set of common goals. To start, all distributed computing environments have to deal with interference, volatility and heterogeneity. However, each type of platform presents a different trend in these three requirements. To exemplify, in volunteer computing resources are very volatile and heterogeneous. Nevertheless, in cluster computing resources are more homogeneous and less volatile. In LSDC it is necessary to achieve the right level of heterogeneity, volatility and interference. Today, several performance evaluation solutions aim to provide specific simulators adapted to one specific community. Nevertheless, different LSDC communities have several ideas and aspects in common that needed to be modeled. For that reason SimGrid aim at proposing a general framework for the performance evaluation of LSDC systems. In SimGrid a simulator is built defining the platform characteristics and applications using two abstract entities.

Resources : are all hardware parameters used to describe the platform such as network links, routers and processing units that enable computation and communication.

Actions : are software parts that consume resources. In LSDC research we are interested in two types of actions.

- **Computation** : actions that consume processing power. Therefore, representing a real application workload to be executed in a computing node.
- **Communication** : actions that consume network bandwidth representing data flowing from one source to a destination.

To instantiate a SimGrid simulation we need to define several characteristics.

Platform : a description of computational resources and network resources.

Application : the active agent during simulation in SimGrid is the application real code that runs using trapped simulation functions, as so, communications and computations are translated to simulated actions that are estimated by the simulation kernel.

Deployment : a mapping of system initial state placing each process in a particular simulated host. Like that the simulation kernel will be able to know which resources will be affected by transmission and executions when the simulation starts.

Availability Traces : a series of states to shape the availability of resources over time.

To present how the simulator works in details we next explain each one of the parameters listed above. To improve readability we will use an example showing how to define platform, application, deployment and traces using a small platform and a simple application.

4.2.1 Platform Description

LSDC platforms combine a set of computational and network resources together. Such resources provide computational power and the communication infrastructure composing the essential parts of LSDC systems. SimGrid uses a XML file [79], also called platform file, to describe the set of resources.

The motivation of using one independent file to describe each platform is to enable researcher to reuse platforms proposed before. The platform file is parsed when a simulation starts translating a platform description inside the simulator kernel. The platform describe a set of resources that, at runtime, will be used by the application. In a platform perspective there are mainly three types of resources to describe: hosts, links and routers. In the XML representation each resource is described by one XML tag as follows.

- `<host id="name" power="P"/>`
  host tag helps to describe a simulated host. Each host has a name associated to be used during the simulation. So, with the host name, can reference a given host to assign tasks to it or to address messages to it. To provide realistic simulation each host has also a computational power associated, which is the processing power of the host. This power is an upper bound of processor working rate expressed in Flops (millions of floating point operations per second) which is a measure that can be obtained through the usage of benchmarks.

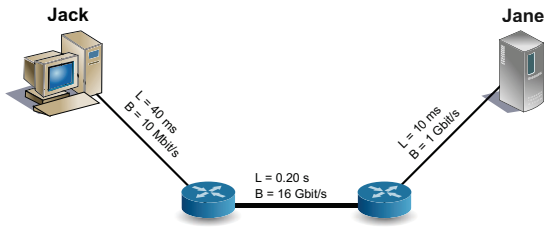
- `<link id="name" bandwidth="B" latency="L"/>`
  The **link** tag helps to declare one network link. Each link has one identification name that will be used later for routing purposes. Using this tag links are described one-by-one easing the description of heterogeneous networks. Furthermore, each link has a bandwidth capacity and latency associated to it. The bandwidth is the maximum transfer rate of this link, meaning that the transfer rate of flows sharing this link cannot exceed this value. Latency is a time to fill the link with data, a delay associated to the link, a time that even small transfers, of insignificant size, will have to consume to travel through that link. Bandwidth and latency are both parameters obtained from the device specification and those two are expressed in B/s (bytes per second) and seconds.

- `<router id="name"/>`
  The **router** tag declares network routers. Routers have a name associated for routing purposes, being exclusively used to join two or more links. So, those resources are like brainless hosts that are unable to execute computational tasks.

Besides the creation of several resources the platform file also carries information about the routes. Those routes are paths used to transmit data between two hosts playing an important role to define contention paths. In SimGrid, a route is a sequence of network links (identified by the identification name in their link tag) in the order they must be used when transmitting from a source host to a destination host (both identified by their identification name specified in their respective host tag). The format to declare a route in SimGrid is this:

- `<route src="sid" dst="did"><link_ctn id="lid"/>* </route>`
  The **route** tag encloses a set of links to communicate data from one host *sid* to another *did*. Inside the route tag several **link_ctn** tags associate the links that must be used when communicating from one **src** to **dst**.

Figure 4.1 shows an example of SimGrid's platform file. In this example, a simple topology composed by two hosts (Jack and Jane), two routers and three links is drawn with its respective XML description aside. As can be seen, in this example the platform is highly heterogeneous: each host has a different processing power and each link has a different latency and bandwidth.



SimplePlatform.xml

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "simgrid.dtd">
<platform version="2">
  <AS  id="AS0"  routing="Full">
    <host id="Jack" power="2000000"/>
    <host id="Jane" power="8000000"/>

    <router id="routerNextToJack"/>
    <router id="routerNextToJane"/>

    <link id="linkJack"    bandwidth="1250000"    latency="0.04"/>
    <link id="linkJane"    bandwidth="12500000"   latency="0.01"/>
    <link id="linkRouters" bandwidth="2000000000" latency="0.20"/>

    <route src="Jack" dst="Jane">
      <link_ctn id="linkJack"/>
      <link_ctn id="linkRouters"/>
      <link_ctn id="linkJane"/>
    </route>
  </AS>
</platform>
```

Figure 4.1: SimGrid's platform file.

In Figure 4.1 is presented how the platform description provided by SimGrid is customizable, enabling to model contention and heterogeneity at the desired level of detail. However, one of the problems with this format is that each pair of routes must be detailed. This characteristics hinders the description of platforms leading to stressful memory usage and parsing time during initialization. To solve that issue a new hierarchical platform routing was recently implemented in SimGrid. In this new platform description routes are described as many autonomous systems (AS), much like network routing algorithms. In the example of Figure 4.1, we have only one AS, for that reason the route is fully detailed.

4.2.2 Application

A distributed application defines the active parts of any LSDC system. It is responsible for creating communication and computational actions during runtime and is often the object of research because changing and fine tuning the application is much more affordable than modifying hardware design. So, in the SimGrid context the application is responsible for producing several simulation actions. Those actions are fake transmission or computation that will allocate simulated resources when several actions share the same resources. Follow we will continue our example proposing an application that can run over the platform previously presented.

An important aspect when defining an application is to provide users with several programming interfaces. Indeed, LSDC is a vast domain. Using low level programming languages, like C/C++ one can achieve enhanced performance. Users that prefer Java or Ruby are also able to use SimGrid because it supports many programming languages: C/C++, Java, Ruby and LUA. The detail of those different SimGrid API will be presented later in this chapter when presenting the SimGrid architecture for now

we will focus on using MSG. MSG is a versatile SimGrid API providing a set of sending/receiving (synchronously and asynchronously) primitives. Besides that, MSG also provides an interface to manage tasks (create, destroy, resume) and simulate computation. An in depth detail of MSG interface and the other APIs available in SimGrid will be given later in this chapter.

Figure 4.2 we illustrate the source code, in C, of an application description using the MSG API. In this example, for the sake of clarity, we divided the application in three files. The main file, SimpleExample.c, has the declaration of two functions sender and receiver. The function sender, implemented in file sender.c, creates one task using the `MSG_create_task` that has no computation cost and 200 Bytes as communication cost. After that, the sender function only send this task to another process using `MSG_task_send`. Details of how the sender function is instantiated will be presented with the deployment file. The receiver function is implemented in the receiver.c file. This function receive a task using `MSG_task_receive`, this call blocks both sender and receiver, establishing a synchronization point between two process.

SimpleExample.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "msg/msg.h"

int sender(int argc, char *argv[]);

int receiver(int argc, char *argv[]);

MSG_error_t test_all(const char *platform_file,
                     const char *application_file);


MSG_error_t test_all(const char *platform_file,
                     const char *application_file)
{
  MSG_create_environment(platform_file);
  MSG_function_register("sender", sender);
  MSG_function_register("receiver", receiver);
  MSG_launch_application(application_file);
  return MSG_main();
}

int main(int argc, char *argv[])
{
  MSG_global_init(&argc, argv);
  if (argc < 3) {
    printf("Usage: %s platform_file deployment_file\n", argv[0]);
    exit(1);
  }
  return test_all(argv[1], argv[2]);
}
```

sender.c

```c
int sender(int argc, char *argv[])
{
  m_task_t task = NULL;
  char tname[64] = "Task";
  task = MSG_task_create(tname, 0, 200, NULL);
  MSG_task_send(task, argv[1]);
  MSG_task_destroy(task);
  return 0;
}
```

receiver.c

```c
int receiver(int argc, char *argv[])
{
  m_task_t task = NULL;
  MSG_task_receive(&(task), argv[1]);
  printf("At simulation time %f receiver %s got a task\n",
         MSG_get_clock(),
         MSG_host_get_name(MSG_host_self()));
  MSG_task_destroy(task);
  return 0;
}
```

Figure 4.2: A simple SimGrid application model.

To the main program, remains initialization routines such as parsing platform file and deployment file. This last is detailed next.

4.2.3 Deployment

The deployment file initializes the application mapping which process are running at which hosts when the simulation starts. In SimGrid the deployment is typically described by an XML file, the deployment file.

A deployment file has two purposes. First it server to attach process to simulated host. Second, the deployment file helps to specify several parameters needed by each application instance. To illustrate a deployment file we present an example of deployment file in Figure 4.3. This example copes with the application presented in Figure 4.2. This deployment will create only one process sender and one

SimpleDeployment.xml

```xml
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "simgrid.dtd">
<platform version="2">

  <process host="Boli" function="sender">
      <argument value="1"/>
  </process>

  <process host="Hori" function="receiver">
    <argument value="1"/>
  </process>

</platform>
```

Figure 4.3: A deployment file for the SimpleExample.c application.

process receiver respecting the platform of Figure 4.1 and the application of Figure 4.2. This deployment example creates a sender process in host Jack passing as argument the message identifier 1. This identifier will be used to identify the message on the receiving peer. To complete the application deployment a receiver process is attached to host Jane using the same number as identifier.

Now that we now how to instantiate a SimGrid simulation it is time to talk about volatility. In SimGrid volatility can be described through availability traces files. Those traces are used to model volatility changing the availability of resources during the simulation. We will next continue to increment our example assigning availability traces.

4.2.4 Availability Traces

For some LSDC communities it is necessary to model resources that leave the system abruptly or change their availability over time. For examples, in cloud computing and volunteer computing resources are available but so abundant that the probability that one resource fails or disconnect cannot be neglected. In those systems many administrative domains warn users before rebooting or disconnecting a machine is infeasible. Moreover, in volunteer computing volunteers may decide the amount of processing power to share, making available only a percentage of the overall processing power. So, to cope with LSDC communities that care about volatility SimGrid provides a method for users to define the availability of resources over time.

Generally, availability traces are histograms that present the available amount of resource over time. SimGrid relies on textual traces that produce availability events during simulation. In SimGrid, an availability histogram is described textually. As illustrated by Figure 4.4, the trace file describes an availability histogram of the resource. In this example, when the simulation starts, at time 0, availability is set to 100 %. After that, the next traced event is when the simulation clock advances 10 seconds, at this time the trace file specifies that the resource availability is reduced to 10 %. At 20 seconds the availability is increased to 50 % and so on until the end of the file. When the end of the availability file is reached, the user may specify a periodicity parameter. This parameter defines a cyclic behavior, like that the same histogram file is replicated after 10 seconds as depicted by Figure 4.4.

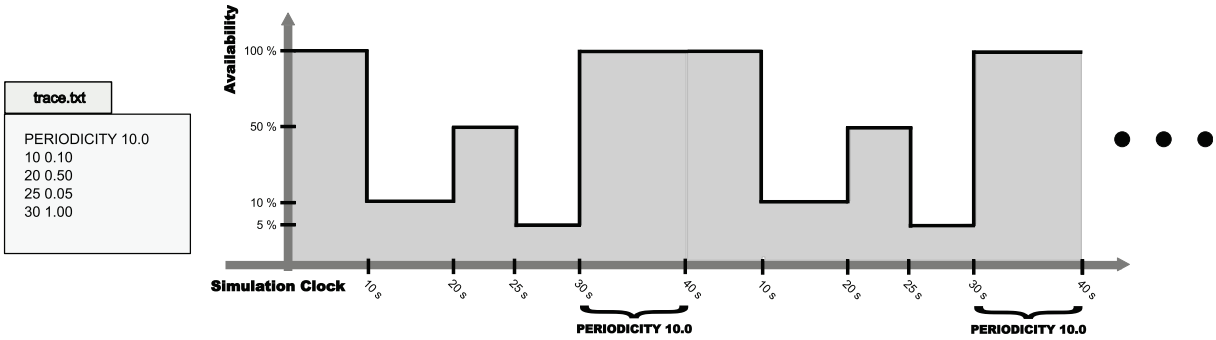


Figure 4.4: SimGrid's availability trace example.

The availability trace file is connected to resources using the platform description. For instance, if we want to shape availability of host Jack with the previous example we would insert in the host tag the information of the trace file like this:

```
<host id="Jack" power="2000000" availability_file="trace.txt"/>
```

Therefore, during execution the CPU power of host Jack would follow the availability dictated in file `trace.txt` presented in Figure 4.4. In this section, we presented how to instantiate a SimGrid based simulator throughout a simple example. We next present the SimGrid available internal architecture.

4.3 SimGrid Architecture

As we saw, a wide variety of LSDC communities can use SimGrid. Those communities have different goals and SimGrid can be adapted to them because it is flexible and modular. This capability is reflected by the SimGrid internal layered structure which is organized in three abstractions levels as presented in Figure 4.5.

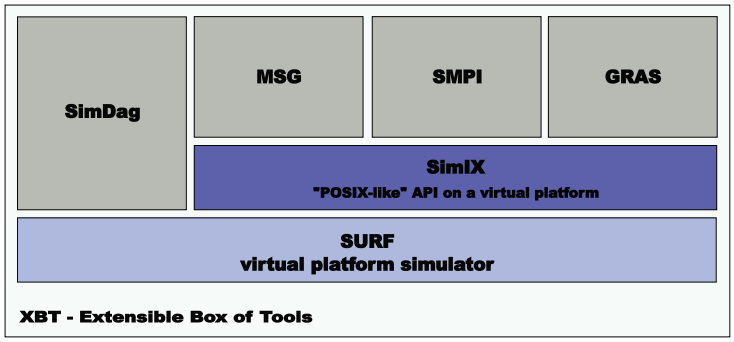


Figure 4.5: SimGrid architecture.

In Figure 4.5, we present the simulator main layers.

SURF : the simulation kernel itself. SURF is responsible to predict the necessary time to finish application actions such as communication and computation.

SIMIX : provides operating system like support. SIMIX is able to coordinate several threads orchestrating simulation events in a deterministic manner.

API : provides several primitives to insert action inside the simulation kernel. It is used in high level to define an application model so that the user can use communication primitives such as send and receive very close to a real distributed application. The programming interfaces available here are DAG, MSG, SMPI, GRAS Figure 4.5.

Aside with those three main layers there is the Extensible Bench of Tools (XBT). This software toolkit provides common programming routines and data types that are not natively implemented in C like hash tables, dynamic arrays and linked lists. Moreover, XBT is also responsible for providing memory management, logging, and software exceptions. XBT is mainly a development set of tools to help programming and so it is out of the scope of this document.

4.3.1 Simulation Kernel (SURF)

In the present work, we are interested in evaluating estimations produced by SimGrid. This estimations are obtained when running an application over a simulated platform. The piece of software responsible for the simulation of communication and execution is called SURF.

SURF works with resources that are shared over actions, those resources are generically seem as provider of working rate. Physically resources have constraints that limit the maximum throughput achieved by the actions running over it. Basically in SURF there are two types of resources: CPU resource and network resources. CPU resources are limited by the maximum computing power, while network resources are limited by link bandwidth and latency. As so, SURF, implement several models adapted to different types of resources.

Based on several actions the simulation kernel is capable of estimating the next action to finish. Nevertheless, some actions may share the same resources. At this point, the kernel models must implement sharing policies to reflect the desired level of interference. So, the simulation kernel is the most important module for the present work once it is direct connected to the simulation accuracy.

4.3.2 Concurrent Process Management (SIMIX)

So the simulation kernel SURF works as an oracle capable of answering the next action to finish. However, we need another module to manage many concurrent process simultaneously. In SimGrid SIMIX is responsible for creating one thread per process which runs real programing code that interface with the simulation kernel through simulated functions.

SIMIX works very similar to a light-weight operating system. Instead of running concurrently several threads, SIMIX orchestrates the threads in a deterministic way. This is desired to guarantee that the simulation is repeatable. Similar to an operating system SIMIX provides basic system calls that can be used to the several programming interfaces available in SimGrid.

4.3.3 Application Programming Interfaces (GRAS, SMPI, SimDAG, MSG)

SimGrid has an extensible programming interface. User's are invited to contribute and implement new API if needed. However, SimGrid has already four APIs described hereafter.

SimDAG

A simple way of describing a parallel application is to design a graph where the dependencies among all tasks are known. One method of describing this dependencies is to use a Direct Acyclic Graphs (DAG, for short). DAG, acronym come from Direct Acyclic Graphs (DAG). In a DAG each task is a node and directed edges are used to represent the dependencies between tasks as presented in Figure 4.6 where five tasks, are represented as five nodes numbered from 1 up to 5. In Figure 4.6 a dependency of tasks 2, 3 and 4 on task 1 is represented by three edges meaning that before computing tasks 2, 3 and 4 the result of task 1 need to be available.

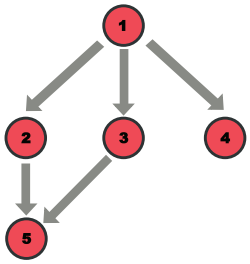


Figure 4.6: A task direct acyclic graph (DAG) example.

The DAG interface offers a direct API to describe a DAG as follows.

`SD_task_create(name, data)` : create a task adding it to the DAG.

`SD_task_dependency_add(src,dst)` : create a dependency of tasks src and dst. Where dst depends on src.

`SD_task_dependency_remove(src,dst)` : same as above but instead of creating a dependency it removes an existing one.

`SD_task_schedule` : schedule tasks.

The DAG interface is limited to a DAG structure. This structure is based on knowing before-hand task dependencies. Unfortunately, the dependency of tasks is often unknown for real applications and sometimes a more detailed application interface must be furnished.

MSG

An intuitive programming approach for LSDC environment is to create tasks and use them to transmit work. Hence, when a task is received a process is able to decide which action to take based on its content. A SimGrid programming interface that follows this idea is MSG (Meta SimGrid). The MSG interface is the oldest SimGrid API aiming at providing to the specialist complete control over the simulation actions. As so this interface is able to provide several primitives for executing, sending and receiving tasks synchronously and asynchronously. The MSG interface offers several primitives.

`MSG_task_create(name, comp, comm, data)` : create a task associated with a certain number of floating point operations to execute (**comp**) and an amount of byte to be transmitted (**comm**). Optionally one task can be associated to a name or data.

`MSG_task_execute (m_task_t task)` : simulate the time to compute a given task respecting the computing power of the machine and the number of floating point operations associated to the task.

`MSG_task_send (m_task_t task, const char *alias)` : simulate sending a task synchronously through the channel identified by **alias**.

`MSG_task_recv (m_task_t task, const char *alias)` : simulate receiving a task synchronously through the channel identified by **alias**.

MSG is suitable for most applications enabling to detail communications and computations inside the simulation kernel. Nevertheless, the MSG interface is far from a real application programming interface. We next present GRAS which is offers a common programming interface to simulate and implement the application using the same code.

GRAS

GRAS is an acronym for *Grid Reality And Simulation*. GRAS provides a way of porting prototyped code to use almost the same code, with a few modifications, as a real application. GRAS was implemented by Martin Quinson and its main motivation was to avoid throw-away simulations that are hard to be ported as real code. Therefore, a GRAS application can work either on top of SimGrid to be used when debugging source code or directly linked with the operating system communication library.

SMPI

SMPI is a simulation interface that cope with the MPI standard. Similar to GRAS, with SMPI one can shorten the time to develop distributed applications since the same source code can be used in both real and simulated environments.

The main difference of the simulated code produced with SMPI is that communication primitives instead of executing a real transmission are translated in simulation actions. As presented before, actions replace a real transmission by a fake simulated call. With the transmission information SURF can estimate the time needed to finish the transmission.

4.4 The Simulation Process

Well, now we have the knowledge of SimGrid internal modules it is time to see how the simulation process works. A SimGrid based simulator start by parsing the platform file to map the user defined resources into SURF. This process will build a representation of the platform topology (i.e., the set of available resources, i.e., the computational hosts, routers and network links) inside the simulation kernel, also creating communication routes, i.e., the paths to use when transmitting data. Next, during initialization, several threads, precisely one for each process, are created upon SIMIX. This process is made parsing the deployment file. At this point, one SIMIX thread is created to run the code defined by each user. An example of how the final state of SimGrid modules after initialization appears in Figure 4.7.

After that the simulator is initialized SIMIX will orchestrate sequentially the threads unblocking one-by-one as the simulation flows, so that the behavior of the program is completely repeatable. The API used (SimDag, MSG, SMPI or GRAS) provides a series of simulation primitives that will replace the real communication and computation routines. Those primitives trap the user code blocking it until a

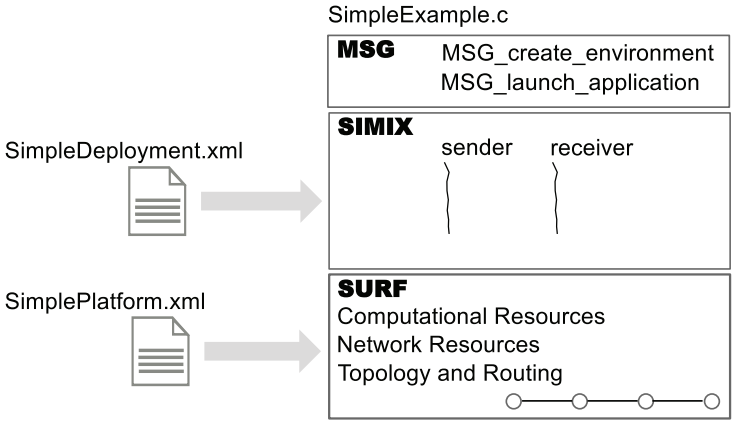


Figure 4.7: SimGrid internals after initialization. One actual thread is created in SIMIX for each user process. A representation of the network topology is created inside the simulation kernel.

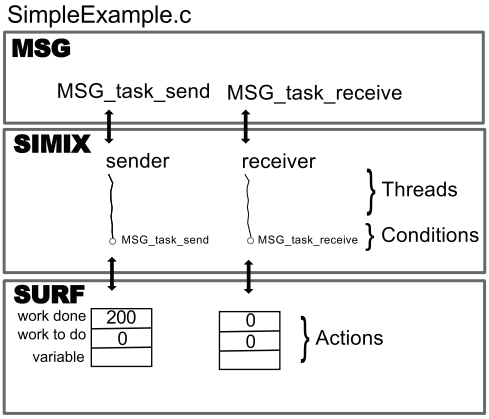


Figure 4.8: Internal simulation workflow. Actual process are trapped by simulation actions (fake computation or communication). At this point, the simulation kernel must decide based on a model which action finishes first.

simulation event wake them up. When a process reaches one of those simulation primitives, SIMIX main thread gain control and must find the next process which is not yet waiting for a simulation event. Eventually, all simulation process will be waiting for a simulation primitive to complete. When this is the case SIMIX will ask to the simulation kernel SURF which is the next action to run as depicted by Figure 4.8.

Each time an action is created into SURF it has associated to it an amount of work to do and an amount of work done. Inside SURF the sharing of actions (communications and computations) that concurrently use the same resources are solved by a model. SimGrid provide several models that are responsible for estimating the completion time of communication and computing actions, we will detail SimGrid models in the next chapter. Following, based on the model of choice SURF is able to return to SIMIX a list of actions that finish first. At this point, SURF is also responsible for updating the amount of work done by each action.

This simulation process repeats until all threads have finished or when the simulator detects that after one iteration the simulation does not evolved. This last situation is particularly useful for debugging applications easing the process of detecting deadlocks or process still running after the application finishes.

4.5 Conclusion

In this chapter, we presented a short history of SimGrid and its architecture. SimGrid is a general simulator that aims at providing the common features needed to simulate LSDC environments. In LSDC resources are heterogeneous and volatile. Moreover, interference plays an important role since resource are commonly shared. Consequently, it is hard to conduct experiments and most research in the field relies on simulation. SimGrid is a tool that gathers several features desired. SimGrid is fast and has incomparable scalability. Enabling investigators to try several parameters and strategies in feasible time.

A common miss conception is to believe that SimGrid is dedicated to Grid computing. As presented here, SimGrid is a general framework to manage simulations of most common kinds of LSDC systems. This is possible, because SimGrid model the main actors of a LSDC environment independently. Enabling investigators to customize the right level of heterogeneity, volatility and interference adapted to his needs.

SimGrid is composed by three layers: the API, SIMIX and SURF. A simulator running on top of SimGrid must have a description of the platform, though an XML file and an actual code using one of the available APIs. When the simulation starts, the platform file and deployment file are parsed. At this point, actual process are created. The process management is done by SIMIX which is responsible to orchestrate process execution in a deterministic way, i.e., guaranteeing repeatability.

In a SimGrid simulation a series of actions take place. Actions, represent fake communications or computations that replace actual execution by an amount of work to compute or transmit. Based on this information, the simulation kernel (SURF) is responsible to compute the time duration of actions. SURF makes estimations of the time needed to finish actions using models. Models are used to fast estimate the behavior of the real system. In the previous chapter, we saw some of the models that are commonly used in LSDC experiments. Therefore, those models must be reliable to guarantee simulation accuracy is acceptable. Consequently, the models implemented in SimGrid are the main focus of this thesis. In the next chapter we will present SURF models.

Chapter 5

Models Implemented in SimGrid

A simulator is essentially the implementation of models enabling to estimate the system behavior. In LSDC, models are expected to provide heterogeneity, interference and volatility. Moreover, models are expected to be fast, scalable, accurate and repeatable. To match these requirements, models adapted to LSDC need to neglect some aspects. Hence, models are approximative and the accuracy trends of models need to be addressed before trusting in simulated results. Therefore, the accuracy of simulators depend on the accuracy of the models implemented internally.

In the last chapter, we saw that the simulation process using our case study, SimGrid, is, as expected, based on models. Precisely, at each iteration, the simulator needs to determine the earliest action to finish. To do so, in each simulation step, SimGrid uses a model to estimate communication and execution time. Nevertheless, in the last chapter, we avoid detailing these models. We decided to present models in a dedicated chapter because models are a main contribution of our work. Moreover, here we present also the models implemented during this thesis to improve simulation accuracy.

In this chapter we introduce four models adapted to LSDC research. We start by explaining and presenting the general characteristics of models. To structure the text, we decided to present the content progressively. So, we present CPU models, which are simpler to explain, switching next to network models.

5.1 SimGrid Fluid Models

Reviewing related work, in Chapter 3, we saw that fine grain models, such as packet-level or wormhole are too slow to be used in LSDC systems. In such systems, fluid models are preferable to match speed requirements. The most basic fluid model consists of associating to each **resource** a given *working rate* capacity. This work rate capacity determines the maximum amount of work that the resource can do in a given duration. For instance, a possible network link work rate is Bytes per second (denoted B/s). Meaning that a 10 B/s link can transmit, at maximum speed, 10 Bytes at each second.

Following this idea, the application creates **actions** that need to consume resources. Thus, we can say that an action has an *amount of work* associated to it. For instance, when an application want to transmit 100 Bytes, we create one action associating 100 Bytes of work to it. Therefore, to estimate the duration of actions we divide the *amount of work* (of the action) by the resource *working rate*, as appears in Equation (5.1).

Sometimes, due to initialization and other phenomena, we want to assign a minimum duration to actions. This means that even actions with zero work to do must respect a minimum duration. Such behavior, for example, is justified for network resources when transmitting zero sized messages may cost a certain time. This duration, called latency, is also introduced in the model of Equation (5.1).

$$T = \frac{W}{B} + L \tag{5.1}$$

Where:

W is the amount of work to do of the action.

B is the working rate of the resource.

L is the latency.

This first fluid model is useful when resources are stable. However, LSDC systems are volatile. Therefore, the availability of resources may change. To cope with this characteristic, in SimGrid, we can associate one availability trace per resource. Those traces provide availability histograms that define a percentage of resource available over time. The model presented in Equation (5.1) need to be extended to work with histograms. To estimate actions finish time, considering these variations, we need to find the moment where the resource produces the necessary amount of work to finish a given task. This idea is formally presented in Equation (5.2), where T is the time to finish the action obtained when the integral of working rate (B) about time (t) is equal to the amount of work to do (W). We assume that the latency duration last until start.

$$\int_{start}^{T} B(t).dt = W \tag{5.2}$$

In SimGrid the working rate model is the basic fluid model. This model is generic enough to model network and CPU incorporating new models that assume that resources produce working rate and action consume this work rate, such as, disk, electrical power, and so. Yet simple, this fluid model consider volatility (through the usage of histograms) and heterogeneity (through a per resource definition of working rate). Nevertheless, this model lacks of interference support, which is a key requirement in LSDC research since resources are shared among different applications, and even among the processes of a same application.

In LSDC systems, interference is expected when messages are simultaneously transmitting or two programs are concurrently running on the same host. In such cases, additional mechanism must be added to the basic fluid model. Intuitively, we can divide resource working rate to several actions simultaneously using it. This idea implies in associating to each action a share of working rate. We call a sharing policy the rules that dictate how the sharing is done. This idea is incorporated in Equation (5.3).

$$T = \frac{W}{\phi} + L \tag{5.3}$$

Where:

L is a latency.

W is the amount of work to do of the action. of the action.

ϕ is a policy to determine the working rate share

The sharing policy is hence responsible to determine the amount of working rate assigned to each action. Nevertheless, resource maximum working rate capacity must be respected. As so, we need to define a set of constraints to guarantee that the sum of allocated working rate for each action is respected. In Equation (5.4) we define the feasibility constraints of the last model. Here, the working rate of each action i is denoted by ρ_i. To respect the working rate capacity of resources, the sum of working rate from actions using the same resource j, cannot exceed the working rate of the resource B_j.

$$\forall_j, \sum_{\substack{i \text{ such that} \\ \text{action } i \text{ uses resource } j}} \rho_i \leq B_j \tag{5.4}$$

5.1.1 CPU Fluid Model

Particularly for CPU, a simple fluid model is justified. This is possible, because most part of CPU timesharing algorithms result in a fair slice of CPU power to each process. In the long run, this behavior is equivalent to equally divide the processing power among several process running simultaneously on the same CPU.

Problem

Let $W = \{W_1, W_2, \ldots, W_p\}$ be the set of actions and the work associated to each one (W_i), and $CPU = \{CPU_1, CPU_2, \ldots, CPU_n\}$ the set of CPUs with their respective maximum working rate capacity (CPU_i). Our goal is to find the effective computing rate (ρ_i) associated to each CPU action.

Feasibility Constraints

In SimGrid, the default computing model assumes that actions are atomic units of work. As such, CPU actions cannot use multiple CPUs. Assuming this, the basic fluid model feasibility constraint is enough to model CPU resources, Equation (5.5).

$$\forall_j, \sum_{\substack{i \text{ such that} \\ \text{action } i \text{ uses CPU } j}} \rho_i \leq CPU_j \tag{5.5}$$

Following this idea we can conceive a very simple and fast model that is share and maximize CPU usage. The idea is to divide in equal parts the available CPU working rate among actions that are concurrently executing on the same host. This idea is formalized as the working rate (CPU_j) being

equally divided by the number of action (n_j) running on the same CPU. With this we obtain the share of computational power that will be assigned for each action (denoted by ρ_i) presented in Equation (5.6).

$$\rho_i = \frac{CPU_j}{n_j} \tag{5.6}$$

Where:

n_j is the number of tasks sharing the CPU_j

CPU_j is the working rate of the CPU_j

With this model we are can estimate the finish time of one action dividing the amount of work to execute, by the action working rate as obtained through Equation (5.6). This similar principle can be also used to estimate the finish time of an action when the resource availability changes using the basic fluid model.

This model is designed to be realistic, fast and is justified when tasks are large enough to reach the stability of the system, i.e., steady-state. Due to the nature of this model, low level phenomena, such as cache initialization, are neglected. This model is feasible for CPU resource mainly because CPU action in SimGrid are unable to run on multiple hosts of a time. As we will see next, this assumption is not valid for network models and for this reason modeling network resources must resort to more sophisticated sharing policies.

5.1.2 Network Fluid Model

In LSDC systems applications use the network to communicate with remote CPU's. This interference plays an important role in the system's overall performance. Interference of concurrent flows accessing network resources impact directly in the system performance. Consequently, modeling network is necessary to obtain realistic simulation results and the accuracy is dependent of the model of choice. Therefore, to manage scalability and speed, the network models implemented in SimGrid follow the same fluid principle presented for CPU modeling.

Problem

Let $L = \{L_1, L_2, \ldots, L_p\}$ be the set of links, and $F = \{F_1, F_2, \ldots, F_n\}$ be the set of transmissions, here also called flows. Each link L_j that has a limited working rate capacity B_j, here also called bandwidth. The goal is to find the effective transmission rate ρ_i associated with each flow F_i.

Feasibility Constraints

The feasibility constraints of the network model are slightly more complicated than the CPU feasibility constraint. Now, additional constraints are necessary.

i A flow is limited by a maximum transfer rate, defined by T_i: $\forall_i, \rho_i \leq T_i$;

ii The sum of the effective bandwidth associated with all flows that use link j must not exceed the link capacity: $\forall_j, \sum_{\substack{i \text{ such that} \\ \text{flows } i \text{ uses link } j}} \rho_i \leq C_j$.

The first constraint (i) add the flow control of the transport protocols. Flow control mechanisms limit the maximum transfer rate depending on the peers specific parameters such as the window size and round trip time. The second constraint (ii) states that the sum of effective bandwidth associated with each connection cannot exceed the link work rate capacity C_j.

This model presents one extra constraint per flow. It is useful for modeling particularly TCP. The protocol TCP establishes that confirmation packets (ack) must be sent periodically to avoid overflowing a slower receiver. As so, TCP protocols limit the connection when it reaches the maximum window size (W_{max}). Which is the maximum amount of data one is able to send before receiving an ack. Since the time to send a packet and receive back its ack (the round trip time) can be roughly approximated by twice the **latency of the flow**. We can define T for the TCP protocol as presented in Equation (5.7) where L_i is the sum of latencies of all links that flow i pass through, i.e., $L_i \sum_{\substack{i \text{ such that} \\ \text{flows } i \text{ uses link } j}} L_j$.

$$\rho_i = \frac{W_{max}}{2\ L_i} \tag{5.7}$$

This network model resembles the CPU model. As in the CPU model, this model also assumes that transmitted data is long enough to reach steady-state, ignoring throughput oscillations in the beginning of transmissions. In spite of this model being very similar to the CPU model we are unable to solve it using the same simple approach we used before. This happens because in the network model one action can use concurrently multiple resources. Therefore, dividing the resource working rate by the number of flows will give several choices of achieved bandwidth. As so, this algorithm is insufficient to model network behavior. Nevertheless, OptorSim [11] define a model based on the simple CPU allocation. The model in OptorSim defines the allocated bandwidth for each flow (ρ_i) as $\rho_i = \min_{\forall \text{link j used by flow i}} \left(\frac{C_j}{n_j}\right)$, where n_j is the number of flows simultaneously sharing the same link. Nevertheless, we can prove that this approach is ineffective even in simple scenarios through a simple example.

Considering the platform presented in Figure 5.1 two communications are made simultaneously. On the top, host Jack is sending a message to Jane. On the bottom, Ryan is sending a message to Rachel. All links have a bandwidth of 100 except the link used to connect Jane to the central link which has bandwidth equal to 10. Following the model presented by William H. Bell *et al.* [11] the allocated bandwidth to the flow from Jack to Jane is 10 because $\min\left(\frac{100}{2}, \frac{100}{2}, \frac{10}{1}\right) = 10$. This value is correct since the top flow is indeed limited by the link with lowest capacity. Although, the resulting bandwidth of the other flow will be 50 because $\min\left(\frac{100}{2}, \frac{100}{2}, \frac{100}{1}\right) = 50$. Here, there is clearly a problem since only 60 % of the capacity of the middle link is used. In other words, the final bandwidth of the flow from Ryan to Rachel is penalized with no plausible justification. So, this model is in practice easy to implement but, as we just presented with a counter example, it is unrealistic even in simple scenarios. Nevertheless, in LSDC systems applications share the network, and a policy that can cope with contention is necessary. Therefore, we present in the next section network contention models.

5.2 Max-Min Fairness

We presented the Max-Min fairness during Chapter 3. Particularly in the network domain, Max-Min fairness is useful to represent congestion avoidance mechanism penalizing greedy flows to avoid congestion. In Max-Min fairness congestion is avoided guaranteeing a minimum bandwidth for each flow so that the network can scale up to a huge number of simultaneously transmitting flows.

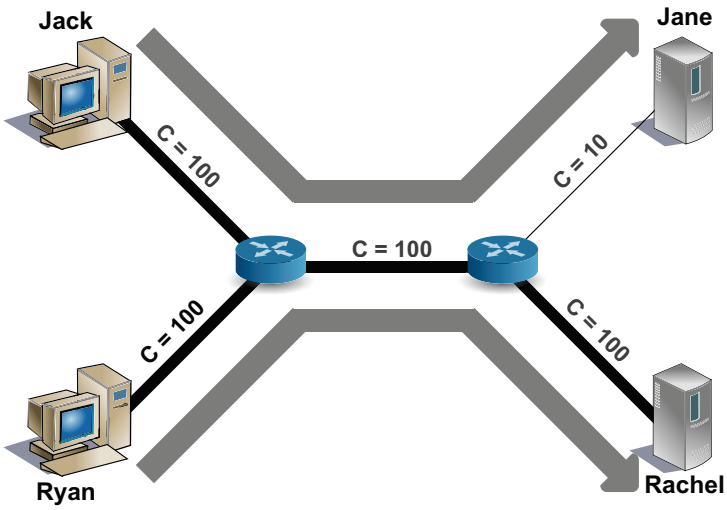


Figure 5.1: Example of how the simple sharing model used in OptorSim [11] is incapable of capturing the network behavior. In this example the top flow gets a working rate of 10 while the bottom flow gets a working rate of 50. At the end we have that only 60% of the central link is used.

The Max-Min model is based on the congestion avoidance algorithm of TCP. The TCP protocol, to guarantee a minimum bandwidth to each communication, reduces the throughput of connections whenever congestion is detected. Consequently, an intuitive fairness problem that approximates this behavior is to maximize the minimum bandwidth received for each link [57]. This model is forth referenced as Max-Min because it maximizes the throughput of minimum, i.e., less privileged, flows.

5.2.1 Optimization Problem

The Max-Min fairness is hence formally presented in Equation (5.8). In this formula, it is possible to consider a weight associated to the throughput achieved by each action.

$$\max \min \left(w_i \rho_i \right) \tag{5.8}$$

A possible algorithm to solve the Max-Min fairness is recursive. When the system reaches the equilibrium, increasing any flow bandwidth ρ_f result in decreasing other flow ρ'_f with $\rho_f > \rho'_f$. A bottleneck link is saturated when the sum of throughput of all flows that use it reaches the capacity limit, i.e., $\sum_{f' \in l} \rho_{f'} = C_l$. Considering that each **flow** f has at least one bottleneck link **link** l. We can recursively find flow throughputs by finding bottleneck links as appears in Equation (5.9). In the next section we present some efficients algorithms to solve Max-Min algorithm in practice.

$$\begin{aligned} \forall f, \ \ \exists l \in f, \ \ \sum_{f' \ni l} \rho_{f'} = C_l \\ \rho_f = \max \left(\rho_{f'}, f' \ni l \right) \end{aligned} \tag{5.9}$$

5.2.2 Algorithm

The most intuitive algorithm to find the Max-Min fairness solution is known as **bucket filling**. In the bucket filling the goal is to increase the bandwidth of each flow by a small constant value, this idea

appears in Algorithm 5.1. Like making an analogy that each flow is a bucket and we try to fill them with a tiny amount of water per iteration. Following this analogy at each iteration we add a fixed small amount of bandwidth for each flow and check whenever a link becomes saturated. When a saturated link is found we fix the bandwidth of all flows using it and update the system. This iterative method goes until all the used links become saturated.

Algorithm 5.1 Bucket filling algorithm to achieve Max-Min fairness

repeat
  for each unfixed flow ρ_i **do**
    {*increase bandwidth by a constant value*}
    $\rho_i = \rho_i +$ constant
  end for

  for each constraint **do**
    {*fix flows when the constraint is saturated*}
    if $\left(\sum_{\substack{i \text{ such that} \\ \text{flows } i \text{ uses link } j}} \rho_i - C_j\right) <$ constant **then**
      fix flows that pass through link l
    end if
  end for
until There are unfixed flows

Therefore, the bucket filling algorithm need to be well parameterized to find a good tradeoff between the amount of bandwidth to update at each iteration, and the number of iterations. So, a small update value per iteration will give a very precise solution but it will take several iterations to reach equilibrium. Otherwise, using a too big step will leave some unused bandwidth. So, the algorithm complexity is hard to predict because it depends on many problem characteristics. For those reasons, the bucket filling is inefficient. In SimGrid we use an alternative solution that is presented next.

To solve the Max-Min fairness inside SimGrid we prefer to use a more efficient solution. Instead of using the bucket filling algorithm we can achieve a solution much faster using the following approach. We find which is the bottleneck link of the system, i.e., the link where the capacity of the link divided by the number of flow sharing it is minimal, find $l, \min\left(\frac{C_k}{n_k}\right), k \in L$. This idea is described in Algorithm 5.2. Once found the bottleneck link, we fix the achieved bandwidth for each flow using the link. After that, we remove the fixed flows from the system. When removing a flow from the system we take care of updating the available capacity of links where the flow pass through modifying the feasibility constraint.

The main advantage of this efficient algorithm in comparison to the bucket-filling is that it is directly dependent on the size of the Max-Min system. As so, this algorithm eliminates the indeterminism of bucket-filling. Moreover, using this efficient algorithm one is able to produce very precise results since there is no more the idea of constant increments. In theory this algorithm presents already good performance although, it requires intensive manipulation of the linear program. As consequence its implementation must deal with such problems. We next enter in details of the implementation of the Max-Min.

5.2.3 Implementation

The bottleneck link strategy, presented in Algorithm 5.2, is implemented inside SimGrid with the help of a sparse data structure. Due to the characteristic of this algorithm it is necessary to loop in the links

Algorithm 5.2 Efficient algorithm to achieve Max-Min fairness

repeat
  {*find the bottleneck link*}
  find l which satisfies $\min\left(\frac{C_k}{n_k}\right), \forall k \in L$

  for each flow ρ_i that pass through l **do**
    {*fix flows when the constraint is saturated*}

    fix flow bandwidth as $\rho_i = \frac{C_l}{n_l}$

    for each link j which flow i pass through **do**
      {*update constraints used by flow i*}
      $C_j = C_j - \rho_i$
    end for
  end for
until There are unfixed flows

used by an action and vice-versa. Thus, we need a data structure that eases to explore the links related to one flow, as well, as the opposite. Thus, we have a data structure which considers three abstractions:

Constraints: representing the link properties such as bandwidth capacity, latency, and so on.

Variable: to store the allocated throughput of each flow.

Elements: joint pieces to connect variables with constraints.

Figure 5.2 illustrates the data structure used to store the Max-Min system. This structure easies looping over the variables of a given constraint or the opposite.

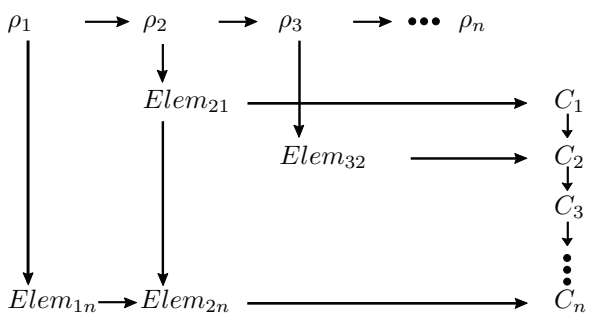


Figure 5.2: Data structure used to store the Max-Min system. Besides the list of constraints and variables there are several elements connecting the constraints used by one specific variable.

A main advantage of Max-Min fairness is its implementability since the presented algorithms are straight forward to code. Moreover, this algorithm has an uniform complexity depending only on the size of the Max-Min system itself. Nevertheless, in a preliminary accuracy study, Fujiwara *et al.* [34, 35] stated that Max-Min is unsuitable for modeling TCP. Their assumption is based on the work proposed by Low [55] that states that TCP sharing policy follows different objective functions. We propose models based on this work next.

5.3 Duality Models

An alternative to Max-Min fairness is the duality models proposed by Low [55]. This work bridges the law that rules the evolution of TCP protocols with Lagrangian optimization problems. Consequently, equilibrium is reached depending on the objective function optimized. Yet, he proposes that the fairness that reflects the variations of TCP protocols are better approximated by non-linear optimization problems. At the end, he proposes an algorithm to achieve a solution based on solving the dual Lagrangian problem. We next present the implementation of such models.

5.3.1 Optimization Problem

Our problem can be expressed as an optimization problem where we want to maximize $f(\rho) = \sum_{i=1}^{n} f_i(\rho_i)$ where ρ_i stands for the effective bandwidth associated with each flow F_i. We name the solution vector of the problem ρ^*. Most part of network protocols can be written as an optimization problem [55], when the connection reaches steady-state. Our objective using this modeling technique is to provide an engine that may easily extend new protocols as well as model connections using different transport protocols[1].

The optimization problem can be written under the constraints that appear in Equaiton (5.10).

$$
\begin{aligned}
&\text{MAXIMISER} \sum_{i=1}^{n} f_i(\rho_i), \\
&\text{UNDER CONSTRAINTS} \\
&\begin{cases}
\text{(5.10a)} \quad \forall L_j \in L : & \displaystyle\sum_{\substack{i \text{ such that} \\ \text{flows } i \text{ uses link } j}} \rho_i \leq C_j & \text{(Links constraints)} \\
\text{(5.10b)} \quad \forall F_i \in F : & \rho_i \leq T_i & \text{(RTT constraints)} \\
\text{(5.10c)} \quad \forall F_i \in F : & \rho_i \geq 0 & \text{(Feasibility constraints)}
\end{cases}
\end{aligned}
\tag{5.10}
$$

Low also presents that the optimization functions behind well known TCP implementation, such as, TCP Reno and TCP Vegas follow a different objective function f_i. In the case of Vegas the objective function is $\ln$ while for Reno it is $\arctan$. So, we look for a solution to the problem independent of the objective. With that we aim at proposing a generic model that can be easily adapted for new protocols that may appears. We next present how to solve this optimization these problems.

5.3.2 Solution

We present a method to approach the values of ρ^* based on theorems from the Lagrange Optimization [12]. This approach is supposed to work if f holds the following property:

$$\forall_i : \textit{exists the inverse of } (f_i)' \tag{5.11}$$

The property presented in Equation (5.11) says that for each $(f_i)'$ the inverse of the partial differential equation exists. This inverse function is essential to proceed with the method because its used to obtain the value of ρ after introducing the Lagrange multiplier. If the problem (5.10) hold the property of Equation (5.11) then the method proposed thereafter can be used to approximate the solution.

[1] Routers use advanced queue management (AQM) such as, RED, REM, DropTail.

5.3.3 Lagrangian Optimization

We will denote by f^* the optimal value of the objective function f for the problem (5.10).

Lemma 1. *If for each i, we have $T_i > 0$ and for each j we have $C_j > 0$, then the problem* (5.10) *has a solution and $f^* \in \mathbb{R}$.*

We will assume in the following that $T_i > 0$ and $C_j > 0$. Let us now define the *Lagrangian Function* $\mathcal{L}$ associated to the problem (5.10).

Definition 1 (Lagrangian Function)**.** *The Lagrangian function $\mathcal{L}$ associated to the problem* (5.10) *is:*

$$\mathcal{L}(\rho, \lambda, \mu) = \sum_{i=1}^{n} f_i(\rho_i) - \sum_{j=1}^{p} \lambda_j \cdot \left(\sum_{\substack{i \text{ such that} \\ \text{flows } i \text{ uses link } j}} \rho_i - C_j \right) - \sum_{i=1}^{n} \mu_i \cdot (\rho_i - T_i)$$

The objective here is to use a shadow price approach relaxing constraints C_i and F_j, capacity and maximum transmission rate. In Equation (5.12), the Lagrangian function is rewritten in this perspective. The same equation may be presented as in Equation (5.13).

$$\mathcal{L}(\rho, \lambda, \mu) = \sum_{i=1}^{n} f_i(\rho_i) + \sum_{j=1}^{p} \lambda_j C_j + \sum_{i=1}^{n} \mu_i T_i - \sum_{i=1}^{n} \rho_i \underbrace{\left(\left(\sum_{\substack{j \text{ such that} \\ \text{link } j \text{ is on flow } i}} \lambda_j \right) + \mu_i \right)}_{\text{Total price to pay to use the corresponding links.}} \tag{5.12}$$

$$\mathcal{L}(\rho, \lambda, \mu) = \sum_{i=1}^{n} f_i(\rho_i) - \sum_{i=1}^{n} \rho_i \left(\sigma_i + \mu_i\right) + \sum_{i=1}^{n} \mu_i T_i + \sum_{j=1}^{p} \lambda_j C_j$$

Where:

$$\sigma_i = \left(\sum_{\substack{j \text{ such that} \\ \text{link } j \text{ is on flow } i}} \lambda_j \right) \tag{5.13}$$

Following the concept of Lagrange Multiplier is needed to achieve the dual function.

Definition 2 (Lagrange Multiplier)**.** (λ^*, μ^*) *is a Lagrange multiplier for problem* (5.10) *if and only if:*

$$\lambda^* \geq 0, \mu^* \geq 0, \text{ and } \sup_{\rho \geq 0} \mathcal{L}(\rho, \lambda^*, \mu^*) = f^*$$

Which will lead us to the dual functional d.

Definition 3 (Dual functional)**.**

$$d : \begin{cases} \mathbb{R}^{2N} & \mapsto]-\infty, +\infty] \\ (\lambda, \mu) & \to \begin{cases} \sup_{\rho \geq 0} \mathcal{L}(\rho, \lambda^*, \mu^*) & \text{if } \lambda \geq 0 \text{ and } \mu \geq 0 \\ +\infty & \text{otherwise} \end{cases} \end{cases}$$

The Slater conditions [12] tell that as soon as there is a feasible ρ (which is the case here since for each i, $C_i > 0$ and $T_i > 0$), the set of Lagrange multipliers is compact and non-empty. The following theorem is a consequence of these conditions.

Theorem 1. *A vector (λ^*, μ^*) is a Lagrange multiplier if and only if it is an optimal solution of the dual optimization problem:*

$$\min_{\substack{\lambda \geq 0 \\ \mu \geq 0}} d(\lambda, \mu) \tag{5.14}$$

Last, we have the following theorem

Theorem 2. *Let (λ^*, μ^*) be a Lagrange multiplier. ρ^* is an optimal solution to problem* (5.10) *if and only if*

$$\begin{cases} \text{(5.15a)} & \mathcal{L}(\rho^*, \lambda^*, \mu^*) = \sup_{\rho \geq 0} \mathcal{L}(\rho, \lambda^*, \mu^*) \\ \text{(5.15b)} & \rho^* \textit{ is a valid solution of } (5.10), \\ \text{(5.15c)} & \forall j, \dfrac{\partial \mathcal{L}}{\partial \rho_j}(\rho^*, \lambda, \mu) = 0 \end{cases} \tag{5.15}$$

The condition (5.15a), ensures that the value of $f(\rho^*)$ is the optimal one and, condition (5.15b), ensures that ρ is a solution of Problem (5.10). Finally, condition (5.15c), states that the optimal solution in ρ is always a maximum point of f.

Subproblem 1, Compute the Maximum of the Function $\mathcal{L}(\rho, \lambda, \mu)$

Optimal solution is also the maximum of f we may express the value of ρ as a function of λ and μ as seen in Equation (5.16).

$$\begin{aligned} \forall i, \tfrac{\partial \mathcal{L}}{\partial \rho_i}(\rho, \lambda, \mu) = 0 \iff & \quad f_i^{'}(\rho_i) - (\sigma_i + \mu_i) = 0 \\ \iff & \quad f_i^{'}(\rho_i) = (\sigma_i + \mu_i) \\ \iff & \quad \rho_i = (f_i^{'})^{-1}(\sigma_i + \mu_i) \end{aligned} \tag{5.16}$$

Since the objective function is explicitly invertible we can compute directly the value of ρ_i at each iteration, knowing the set of objective functions f_i'.

Subproblem 2, Compute λ and mu

Hence, using Equation (5.16), the dual problem may be written depending on λ and μ variables, presented in Equation (5.17).

$$d(\lambda, \mu) = \sum_{i=1}^{n} f_i\left((f_i^{'})^{-1}(\sigma_i + \mu_i)\right) - \sum_{i=1}^{n}\left((f_i^{'})^{-1}(\sigma_i + \mu_i)\right)(\sigma_i + \mu_i) + \sum_{i=1}^{n} \mu_i T_i + \sum_{j=1}^{p} \lambda_j C_j \tag{5.17}$$

The objective here is to find the solution of Equation (5.17) such that function $\forall \mu_i, \frac{\partial d}{\partial \mu_i} = 0$ and $\forall \lambda_k, \frac{\partial d}{\partial \lambda_k} = 0$. The known way of solving the problem of Equation (5.17) is to inverse the partial differential equations related to μ and λ.

$$\begin{aligned}\frac{\partial d}{\partial \mu_i}(\lambda,\mu) =&((f_i^{'})^{-1})'(\sigma_i+\mu_i)\times f_i^{'}((f_i^{'})^{-1}(\sigma_i+\mu_i)) - (f_i^{'})^{-1}(\sigma_i+\mu_i)\\ &-\left[((f_i^{'})^{-1})'(\sigma_i+\mu_i)\right]\times(\sigma_i+\mu_i)+T_i\\ =&((f_i^{'})^{-1})'(\sigma_i+\mu_i)\times(\sigma_i+\mu_i) - (f_i^{'})^{-1}(\sigma_i+\mu_i)\\ &-((f_i^{'})^{-1})'(\sigma_i+\mu_i)\times(\sigma_i+\mu_i)+T_i\\ =&-(f_i^{'})^{-1}(\sigma_i+\mu_i)+T_i \end{aligned} \tag{5.18}$$

Therefore, after some algebraic simplifications, Equation (5.18), we can compute the values of μ and λ using just the inverse of the dual partial differential. We choose to use a numerical dichotomy to approximate the solution for two reasons.

Since the interval is divided by two at each iteration a few steps are enough to get a good approximate answer (error less than 10^{-6});

Since f is an increasing and convex function the method is granted to converge;

Equation (5.18) show the general form of the partial differentials equations for all μ_i and Equation (5.19) shows for λ_k.

$$\frac{\partial d}{\partial \lambda_k}(\lambda,\mu) = \sum_{\substack{i \text{ such that}\\ \text{flow } i \text{ is on link } k}} \left(-(f_i^{'})^{-1}(\sigma_i+\mu_i)\right) + C_k \tag{5.19}$$

Equation (5.18) shows that the differential equations are composed by the same terms as Equation (5.19) $(f_i^{'})^{-1}$. Hence, this easies to update this model to reflect new objective functions. Consequently, we can minimize d about λ and μ using a gradient with optimal step for each directions. Nevertheless, to obtain this optimal step we use a dichotomy approach.

5.3.4 Implementation

To implement these last models we used the same sparse data structure previously implemented for Max-Min. Next, we present in Algorithm 5.3 the solution for the duality models based on Lagrangian optimization.

The model presented throughout this section was entirely implemented inside SimGrid during this Ph.D. thesis. We have at least two reasons for implementing such a model. First, in the preliminary work of Fujiwara *et al.* [34, 35], she points that those models seem to be more realistic and could improve significantly the simulator performance. Second, those models are solved independent of the objective function. As such, one is able to easily update the simulator with new models when the objective function is invertible, continuous and derivable.

5.4 Packet-level Discrete Simulation (GTNetS)

As an alternative to the fluid models already implemented inside SimGrid we have the possibility to use a packet level simulator called GTNetS. Packet level simulators as GTNetS were already presented in

Algorithm 5.3 Algorithm to solve models based on Langrangian multipliers

Start with initial values for λ and μ
repeat
  for each variable λ_k **do**
    {*Using dichotomy*}
    find λ_k such that $\frac{\partial d}{\partial \lambda_k}(\lambda, \mu) = 0$
    Each step of the dichotomy requires to compute, $\frac{\partial d}{\partial \lambda_k}(\lambda, \mu)$, which is done using Equation (5.19) and (5.16)
  end for

  for each variable μ_k **do**
    {*Using dichotomy*}
    find μ_k such that $\frac{\partial d}{\partial \mu_k}(\lambda, \mu) = 0$
    Each step of the dichotomy requires to compute, $\frac{\partial d}{\partial \mu_k}(\lambda, \mu)$, which is done using Equation (5.19) and (5.16)
  end for
until d cannot be improved
Once found the optimal λ, μ, we can compute the corresponding optimal ρ using Equation (5.16).

details during Chapter 3. The integration of GTNetS inside SimGrid was first proposed by Fujiwara *et al.* [34, 35] and was maintained by us during this thesis.

5.4.1 Protocol Stack

Modern network protocols are organized as a stack of protocols which provide at a given level the needed functionalities. As so, the software and hardware network modules have to play many roles during packet routing and transmission. Consequently, each host has a protocol stack representation and each time one packet arrives or lives from a network node it must pass through all the protocol layers. Similarly, when the packet arrive to its destination it must pass through the same protocol stack in the inverse order. The protocol stack along with a simple network representation is illustrated by Figure 5.2.

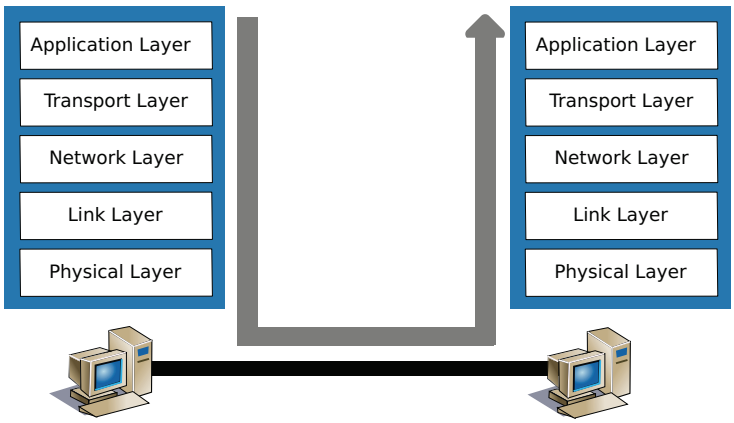


Figure 5.3: Network protocol stack example. The flow must pass through all levels before being send and received. A packet-level simulator does the network modeling with a fine grain level of detail.

GTNetS splits messages in many packets and treats each one as a single work unit. As so, GTNetS send a message in a fine grain approach where events are packets travelling between two network layers until it reaches the network card, the wire, and so on. This approach is completely different of the SimGrid fluid model. For this reason, some adaptation is needed to get GTNetS coping with the fluid model of SimGrid. To do that we need to find a way that GTNetS could return the time to finish flows as SURF expects.

5.4.2 Implementation

In SimGrid the simulation kernel works as an oracle that find the first finishing communication action. As so, the simulation kernel has to control a list of active actions to compute the next to finish. To make GTNetS cope with SimGrid, Fujiwara patched GTNetS code. Nevertheless, this patch does not alter deeply the behavior of the simulator. What this patch does is to add the ability of running the simulator until the earliest transmissions finish. Also this patch adds the ability to run the simulation for a given duration. To summarize the GTNetS patch implemented by Fujiwara adds two important functionalities into GTNetS.

run_until_next_flow_completion run until the next flow completes returning the time to the earliest communication events plus a list of communication events.

run_until run the GTNetS simulation until a given duration.

One of the main challenges of implementing such mechanisms was to cope with the ability of foreseeing simulation events. In practice GTNetS is made in a way that the simulation clock can only advance. So, Fujiwara modified GTNetS to make a process copy saving the simulation state to resume it later. With that copy in hand, the simulator can run to discover the next flows finish.

5.5 Conclusion

In this chapter, we presented several models implemented inside SimGrid. As we saw great part of the models current available use a fluid model. The motivation for using fluid models is to achieve speed and scalability that can match with LSDC requirements. Most part of SimGrid models were developed over the years receiving intensive feedback from SimGrid users. Nevertheless, some models, specifically the duality models were implemented during this thesis because we believe that those models improve the accuracy results obtained previously by Fujiwara *et al.* [34].

The accuracy of those models is justified only when the system reaches steady-state. Therefore, such models are expected to be accurate for a minimal amount of work to do. As we saw, besides the fluid models SimGrid has also the possibility to incorporate a packet-level network model. This model was introduced by Fujiwara as an alternative to validate the simulation accuracy [34]. We present, in the chapter that follows, the accuracy study of these simulation models.

Part III

Model Evaluation and Improvements

Chapter 6

Simulation Accuracy Improvements and Results

Many works in the domain of large scale distributed computing (LSDC) systems today are supported by simulation. However, as presented in Chapter 2, there is often a lack of accuracy studies under relevant settings.

An inaccurate simulator produce meaningless results and all reasoning taken from inaccurate simulations are wrong. For that reason, estimating simulation accuracy is needed before trusting simulation results. As discussed before, the main goal of this thesis is to evaluate the accuracy and speed of simulation to prove that this approach is suited for conducting research in LSDC. Therefore, in this chapter, we address a comprehensive accuracy study of simulation models available in SimGrid.

This chapter is organized as follows. Section 6.1 presents a general discussion of our evaluation approach linking the present work and previous research. Sections 6.2 and 6.3 show accuracy analysis for CPU and network models. Section 6.4 present a general discussion of network model validity when compared to other fluid approaches. Finally, in Section 6.5, a brief summary and conclusions are depicted.

6.1 Performance Evaluation Approach

A typical user of LSDC is interested in evaluating system performance varying several parameters. Consequently, analyzing several alternatives is possible at best partially in a non-representative scale of the target system. Moreover, such environments are difficult to control, low-level aspects depend on user privileges that are often hard to obtain in such platforms due to security or administrative constraints. Therefore, the user is often limited by the platform being unable to evaluate the system with different parameters. Another drawback, is the time needed to obtain results. In such systems, applications are computational intensive and take time duration to produce results, so evaluating several alternatives is barely possible in feasible time.

To provide faster results a simulator approximates the behavior of the real system using models that replicate the system behavior without incurring in the computational cost of doing the real execution. For that reason simulation models in this context need to be fast, enabling the evaluation of several

parameters in feasible time. At this point, we need to assess those approximative model certifying that they respect the real system behavior.

Nevertheless, lack of accuracy on models may lead to fake conclusions. Furthermore, the model quality depends of its usage. Sometimes a good model that is accurate for a certain reality (the reality it was meant for) is inappropriate for other purposes. A constant model is suitable to estimate transmission time for CPU bounded applications. However, this is not true for network bounded applications. A good simulation model must be adapted to its usage, as well, a comprehensive accuracy study have to rely on relevant settings.

Empirically, to verify if a piece of software is working, we compare its output with the expected behavior, an output that we know is correct. If the software output matches the expected behavior we conclude that its working. Otherwise, we investigate the reasons of unexpected behavior. This last example is also called black-box test [60]. In a black-box test the system is seen as a *black-box*, no knowledge about the system internals is used in the test. The only information available is input parameters, output and expected behavior. To summarize the testing process, there are four important things to determine for a test plan conception.

Configuration: a set of input parameters.

Measure: a set of output parameters.

Expected behavior: the correct output.

Metric: a definition of distance between measured and expected behavior.

In accuracy evaluation we follow more or less the same idea of a black-box test. Our goal is to measure simulation accuracy, but for doing that we need to configure the simulator; run a simulation to measure some relevant aspects; and, at the end, we need to have access to a comparison point to finally measure simulation accuracy using a metric. In LSDC research, the simulator is typically configured with an application and a platform. The application determine the behavior of processes while the platform is a description of resources (such hosts, network links, routers, routes). Following, after running a simulation we measure the simulation results and compare those with the expected behavior using a distance, or metric.

Defining input and output depend on the evaluated subject. To evaluate network, a probable input choice would be network description, number of transmission or flows size. Examples of observed results, in network context, would be flows throughput, response time, and transmission time. Configuring and measuring a network experiment is completely different of evaluating CPU performance. In the CPU context, one is rather interested in setting processing power, and observe at the end CPU usage or throughput. Defining parameters and measures appropriated to each accuracy evaluation scenario depends on its context. In this chapter, we will define input and output when detailing the executed experiments.

To conduct accuracy studies coherently we need to first define the common points. To be comprehensive an accuracy study must be clear about the choices made. The accuracy is roughly a quantitative metric at which the simulated results are distant from the expected behavior. Nevertheless, selecting a metric, and a comparison point are of extreme importance to conduct a reliable accuracy study. Particularly, those choices are sustained until the end of this section. So, before conducting the accuracy study, we clarify our choices for comparison point and metric.

6.1.1 Comparison Point

In SimGrid, we have an hybrid simulation approach. Analytic models are used to design low-level resource sharing and consumption while the overall simulation accounts in discrete-event. As stated by Jain [45], analytic results are doubtful until confronted with a different approach. So, to prove the accuracy of analytic models we need to extensively evaluate them with a comparison point to assess if the model respects the expected behavior.

In SimGrid there are mainly two model types: network and CPU. A typical usage rely on the network model to estimate transmission cost, and on the CPU model to estimate execution time. Eventually, those models cope with each other in the simulation process however, they have completely different instantiations. CPU models consume processing power while network models consume latency and bandwidth. Moreover, network models care about topology while for the CPU models topology details are disposable. Consequently, to extensively test each model under relevant setting we propose to evaluate them separately, respecting their different nature.

CPU

To evaluate the CPU models we propose to use real experiments because one single host is controllable (we can avoid interference of other jobs and users.) Furthermore, real experiments, in this context, are roughly repeatable but since in our context the CPU models are coarse grain bias can be avoided using simple statistics. Anyway, analyzing the behavior in one single processor is feasible using relevant settings and so we use real experimentation for evaluating SimGrid's CPU model. As we will now explain, the situation for the network evaluation is slightly more complex.

Network

In the network context real experiments give a limited proof of concept that simulations are reliable. In networks, measuring achieved bandwidth and latency is hard due to several phenomena [68, 70, 69]. Last, real experiments of large-scale networks are limited to the available settings, time consuming, and difficult to repeat (due to the interference of other process). In this context some authors claim that emulation is the key, bridging the gap between the real and the controllable, however, doing emulation without questioning its validity range is naïve once emulation is also based on models. So, at the end, emulation has the same accuracy issues as a simulation being many times slower.

Another possible solution to compare network models is to use different models whose validity has already been assessed. For instance, packet-level simulators, such as the Network Simulator NS-2 [89], NS-3 [90] or Georgia Tech Network Simulator (GTNetS) [74], imitate the entire protocol stack using a discrete-event approach. Packet-level simulators reproduce the low-level mechanics of packet flow on the network protocol stack (such as, packing, unpacking, retransmission) from physical layer up to the application layer and vice-versa. Unquestionably, the advantage of using these models is to control several heterogeneous network settings. Hopefully, with controllable settings and being able of reproducing results we can propose a representative model validity study. Besides that, GTNetS is already integrated in SimGrid, thanks to the work of Fujiwara *et al.* [34], making its use straightforward. For these reasons, in the rest of the accuracy evaluation study of network models we use GTNetS as comparison point.

6.1.2 Error Metric

In LSDC we are interested in the evaluation of resource usage and time. Resource usage is expressed, for instance, in B/s for bandwidth usage, or MFlops/s for processing power. We also may want to evaluate time indexes like makespan, or response time. As noticed, we may measure, several output parameters using real numbers, however, those numbers can be in a different scale. In this case, an appropriate measure is a vector of real numbers $x \in \mathbb{R}^n$. Consequently, the expected output, will be also a vector of real numbers $y \in \mathbb{R}^n$ obtained running the same experiment with the comparison point of choice.

Nevertheless, due to having different scales, it is important at some point to normalize accuracy results to compare different measures that may vary in scale and order of magnitude. As presented, network usage in B/s and number of processed tasks are completely different measures varying significantly in scale and domain. A common practice to solve this problem is to use scale-free metrics. Scale-free metrics are those that respect the property of Equation (6.1) and in essence are comparable even in completely different units or orders of magnitude.

$$d(a.u, a.v) = d(u, v) \quad a \in \mathbb{R} \tag{6.1}$$

To compare both vectors x and y, it is necessary to propose a metric, or distance. A distance, denoted by d, must be defined in a metric space mapping two resulting vectors, into a single real number $d(x, y) : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ respecting four axioms.

1. $d(x, y) \geq 0$ (non-negativity)

2. $d(x, y) = 0$ if and only if $x = y$ (identity)

3. $d(x, y) = d(y, x)$ (symmetry)

4. $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)

In spite of the properties presented before being basic they are often neglected in accuracy studies. Fujiwara *et al.* [34] measure the error of two estimated values x_i and y_i as a ratio of both $\frac{x_i}{y_i}$. The ratio metric has the advantage of being scale-free. However, this measure disagree with three axioms: $\frac{x_i}{y_i} = 1$ when $x_i = y_i$, $\frac{x_i}{y_i} \neq \frac{y_i}{x_i}$, and $\frac{5}{1} > \frac{5}{2} + \frac{2}{1}$. Other authors propose a similar metric $\frac{|x_i - y_i|}{x_i} \times 100$ [19, 22]. Nevertheless, this metric is invalid since it fails to satisfy symmetry and triangle inequality.

Here we propose the error metric, denoted by ϵ, as a log based distance presented in Equation (6.2). This distance is adapted to accuracy evaluation because it respects a metric space's constraints. The reason of using log is to have a proper and at the same time scale-free error metric.

$$\epsilon(x_i, y_i) = |\log(x_i) - \log(y_i)| \tag{6.2}$$

Therefore, the log metric is useful for a single error comparison. However, very often we are interested to compare the entire vector of simulated results x with the expected behavior y. For those cases we propose two metrics the maximum error, denoted by ϵ_{max}, and the mean error, denoted by ϵ_{mean}. To improve models behavior, we focus on worst cases. Using the maximum worst cases are translated to one absolute value, as in Equation (6.3). Maximum error enable us to identify and propose improve-

ments where the model is incorrect. Another advantage of using max is that it is capable of putting in evidence worst cases even when the maximum error is hidden among several good results.

$$\epsilon_{max}(x, y) = \max(\ |\log(x_i) - \log(y_i)|\ ) \tag{6.3}$$

Nevertheless, sometimes we aim at presenting, as a proof of validity, mean error to state that overall estimations behave like expected. To do so, we define ϵ_{mean} in Equation (6.4). At the same time, the mean error hides the maximum, when the worst case is a punctual case, proving that the simulator accuracy is indeed globally acceptable.

$$\epsilon_{mean}(x, y) = \frac{\sum_{i=1}^{n}(\ |\log(x_i) - \log(y_i)|\ )}{n} \tag{6.4}$$

Another positive aspect of this error is that it carries the percentage of discrepancy. However, to extract that information, which is directly obtained with the ratio, we must use Equation (6.5). It is useful to state how many times the model is distant from the comparison point in percentage.

$$\epsilon_{ratio}(x_i, y_i) = \exp(\ \epsilon(x_i, y_i)\ ) + 1 \tag{6.5}$$

To conclude, using the proposed distance we can conduct a detailed analysis respecting our design choices. Following we present a comprehensive accuracy study with a series of instantiation scenarios.

6.2 Accuracy Evaluation of CPU Model

As stated before the CPU behavior in SimGrid is based on a fluid model where the CPU is described as a work rate and the task is defined by an amount of flops (floating point operations) to execute. Accuracy is justified when the application is CPU bound, i.e., there is enough work available to reach steady-state.

6.2.1 Quantitative Analysis

To verify CPU model accuracy we opted for a real execution of a simple algebraic benchmark. This benchmark is based on floating point operations using a square matrix of fixed size 300x300. The kernel computation intensive code appears in Algorithm 6.1.

Algorithm 6.1 Simple benchmark computational kernel

```
for all i, j ∈ {2..300} do
  if (α%3) == 0 then
    α = α + M(i − 1, j)
  end if
  M(i, j) = M(i − 1, j − 1) + α
end for
```

The computation kernel in Algorithm 6.1 is conceived to force concurrent access if the matrix is shared. In a first experiment, we try one dedicated matrix for each thread to verify if the sharing follows the model presented before. To do so, we will use a timed application that executes the computation kernel many times until a timer reaches a given amount of seconds. During execution, each time the computation kernel is entirely executed a counter is incremented. At the end, throughput can be estimated out of the number of floating point operations in the computational kernel loop.

Tests were executed in a 32bit Intel Pentium 4 CPU at 3 GHz with 2 MiB of cache and 1 GiB of RAM. To guarantee that the CPU is dedicated during experiments some network services and Xorg window system were powered off. Figure 6.1 presents the results of the average throughput per thread (top graph) and the sum of the achieved throughputs (bottom graph). The usage of CPU power is always very near the available. Indicating the overhead introduced by context switching can be neglected. To estimate available CPU power we use the result measured with a single thread. In this specific case a 100 % of CPU usage is approximately 3 MFlops (millions of floating point double precision operations per second) because the number of kernels executed with a single thread in 10 seconds is 243 and one kernel execution takes 119200 floating point operations.

The graphs of Figure 6.1 show that the model is adaptable to coarse grain simulations for this type of machine since the degradation of context switching is insignificant. Even so, the sharing shows to be also accurate, confirmed by the top graph. In Figure 6.1 the top graph includes the average throughput per thread. We plotted the confidence interval (with $\alpha = 0.05$) enclosing each point in the graph. Confidence interval is invisible because it is too small. Conclusions lead that if the CPU power is well calibrated the model is accurate for independent memory access applications.

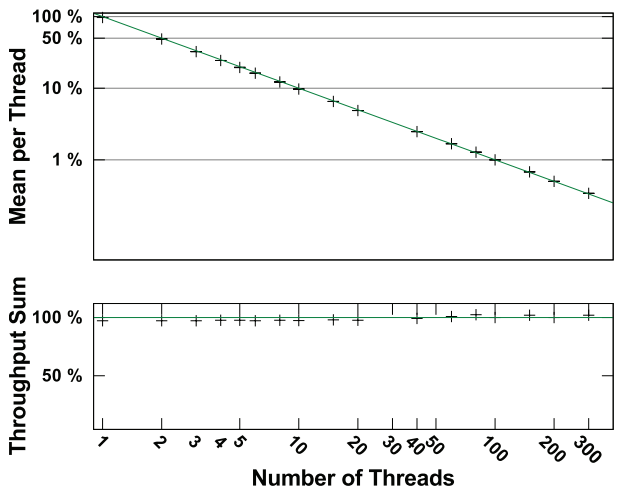


Figure 6.1: CPU model (continuous line) in comparison with real experiments (crosses) when varying the number of concurrent threads. The simple CPU model is accurate to model processing time even when the host is shared by several **independent** threads.

Now that we are sure about the CPU model in a simple scenario we would like to try to use it in a slightly more complex test case. Following, we design a test case using a single shared matrix of size 300×300. Like this, one can verify if the use of non independent tasks impacts in the computation model. So, in Figure 6.2 we depict the same result as before with mean throughput per thread (top graph) and sum of thread throughputs (bottom graph) when varying the number of threads. Observing the figure we can see that these tests present significantly more variation. To assure concurrent access we start all threads at the same time. Curiously, the achieved throughput is much more important than before, 820 MFlops instead of 3 MFlops. We believe this happens because cache memory is much more consistent in a good part of time while before the cache memory would be switched each time one thread context changed. This leads to think that the processing power of a machine can be better modeled if characteristics of the application are taken into account. Nevertheless, if the processing power is well instantiated a good accuracy can still be verified.

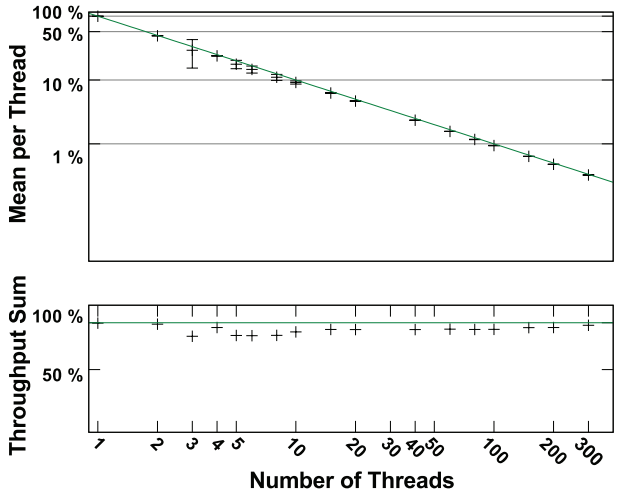


Figure 6.2: CPU model (continuous line) in comparison with real experiments (crosses) when threads are **sharing the same memory space**. The simple CPU model is accurate, however, it is yet slightly less accurate than with independent threads.

Indeed, the CPU model despises low level phenomena such as cache access. Even so, applications specific characteristics can be calibrated using as CPU overall power the throughput achieved with a single instance of the application. The main advantage of this model is scalability due to low complexity it can effectively be used to simulate many of hosts at same time. Nevertheless, the simulation scalability is also dependent on the network model. This subject will be discussed in the next section.

6.3 Accuracy Evaluation of the Max-Min Network Model

Our network evaluation approach is based on the previous work presented by Fujiwara *et al.* [34]. We compare estimations obtained with SimGrid versus a different performance estimation approach. For the network model accuracy evaluation we use GTNetS which is a packet-level discrete event simulator that aims at producing very accurate results for the network research community.

6.3.1 Evaluation of the Single Link Model

Model

As detailed in Chapter 5 the basic network model in SimGrid defines the estimated communication cost T as in Equation (6.6).

$$T(L, \phi, S) = L + \frac{S}{\phi} \tag{6.6}$$

Where:

L is the latency associated to the TCP connection

ϕ is the transfer rate

S is the amount of data to transfer

TCP is a handshake protocol where often connections are limited by the round trip time. So the transfer rate ϕ is the minimum of the **nominal bandwidth**[1] B and $\frac{W}{RTT}$ where RTT the round trip time. The W is called the TCP maximum window and it is 16 or 32 bit integer depending on the protocol version implemented [43]. Today many known Linux distributions use about 128 KiB (131071 Bytes) as default value for the maximum TCP window. The model assumes that a good round trip time RTT estimation can be $2L$, i.e., twice the latency. To cope with this constraint SimGrid's network model is extended to the one presented on Equation (6.7).

$$T(L, B, S) = L + \frac{S}{\min(B, \frac{W}{2L})} \tag{6.7}$$

Where:

L is the latency associated to the TCP connection

B is the nominal bandwidth

S is the amount of data to transfer

W is a constant maximum TCP window

It is already possible to remark that the communication overhead is neglected in this model. In practice nominal bandwidth is never entirely used for transmitting application data. The nominal bandwidth is a transfer rate of the link itself but substantial traffic is exchanged to guarantee quality (to avoid congestion, routing purposes). Hence overhead traffic will certainly consume part of the connection throughput.

Quantitative Analysis

To verify if the congestion window assumption holds we define an experiment describing the time as a linear function of message size, latency and TCP window. For this we use one single flow through one single link. The flow size varies with 5 points linearly distributed $S \in$ [10 MB, 50 MB], latency was varied with 5 points linearly distributed $L \in$ [0.1 s, 0.5 s] and bandwidth was fixed B = 1 Gbit/s. With a maximum latency of 0.5 seconds a maximum window greater than 125 MB would be necessary to reach the maximum available bandwidth of 125 MB/s (= 1 Gbit/s) link rate, i.e., we assure that the flow is limited by latency and the maximum TCP window. Likewise we varied the TCP maximum congestion window with 14 points linearly distributed $W \in$ [500 B, 131071 B]. We used the assumption that transmission time is linear function of the maximum window W as presented in Equation (6.8) where the throughput obtained, when limited by the latency, is a function of the maximum window ($f(W)$) divided by the flow latency.

$$T(S, L, W) = L + \frac{S}{\frac{f(W)}{L}} \tag{6.8}$$

[1] Here we adopt the nominal bandwidth as been the one written on the standard specification, i.e., 100 Mbit/s (for ethernet), 1 Gbit/s (for fast ethernet), 56 Kbit/s (for a modem) and so on.

Normalizing with respect to S we used Equation (6.9) to obtain the results that appear in Figure 6.3, considering that the term $\frac{L}{S}$ is not significant.

$$\frac{T(S,L,W)}{S} = \frac{L}{S} + \frac{L}{f(W)}$$
$$f(W) = \frac{LS}{T} \tag{6.9}$$

The straight line in Figure 6.3 is the linear model approximation, i.e., $f(W) = \frac{W}{2}$, compared to GTNetS estimations (crosses). The crosses are obtained using the $\frac{LS}{T}$ as justified by Equation (6.8) and (6.9). The abrupt fall of points when W = 65553 B is because TCP maximum window are 16 bit limited in GTNetS so values above 65553 overflow. Analyzing the maximum error ϵ_{max} which is less than 5% of accuracy loss we can conclude that $2L$ is indeed a good approximation of round trip time RTT.

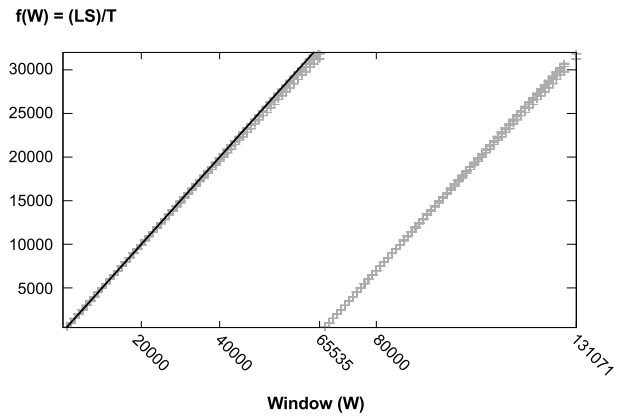


Figure 6.3: Transmission throughput as function of TCP window for GTNetS values (crosses) in comparison to SimGrid model (continuous line). This results reinforces that connection throughput is often limited by the round trip time. Moreover, this figure points out that two times the latency ($2L$) is a good approximation for RTT.

We are also interested to verify if the fluid model transmission rate bound is nominal bandwidth when the round trip time constraint is not reached. To gather enough data we massively varied flow size S, latency L and bandwidth B. The TCP maximum window parameter was always fixed to the maximum supported by GTNetS, i.e., 65535 Bytes, unless specified otherwise. To speedup experiments we run many simulations in parallel using the Bordereau cluster. Bordereau is a cluster composed by 93 IBM System x3455 nodes located at Bordeaux Grid5000 site. The node configuration overview can be checked in Table 6.1. To exploit effectively this system we relied in two utilities: OAR (the grid5000 scheduler) and TakTuk (a remote executions helper).

A comparison of the model (surface) versus GTNetS estimated values (points) for a fixed bandwidth of 56 kbit/s appears in Figure 6.4. In this experiment message size was linearly sampled with 167 points $S \in$ [1 KB, 600 MB] and latency was linearly sampled with 19 points $L \in$ [0.00001 s, 0.5 s]. A fixed bandwidth of 56 kbit/s (7000 B/s) guarantees that the transmission rate is not limited by $\frac{W}{2L}$ because it would be necessary a latency greater than 4 s in this case to be limited by the round trip time constraint.

Based on results of Figure 6.4 SimGrid is rather optimistic, i.e., linear model estimations are a lower bound of GTNetS communication cost. For this set of results the maximum error ϵ_{max} is 1.575271

Table 6.1: Configuration of an IBM System x3455 node.

Processor	Opteron(tm) 2218
Manufactor	AMD
Number of processors	4
Number of cores per processor	2
Clock frequency range	800 Mhz $\sim$ 2.6 Ghz
Cache L1 per core	1 MiB

which represents significant 380% of precision loss. If slow start is responsible for the growth of ϵ_{max} then short connections are directly affected since they do not reach steady-state. To confirm this the maximum error is less than 10% when size $S >$ 10 MB. So, the error of one link model for bandwidth bounded connections is acceptable if there is enough data to be transmitted. Our result show that with messages $>$ 10 MB connection will eventually reach steady-state.

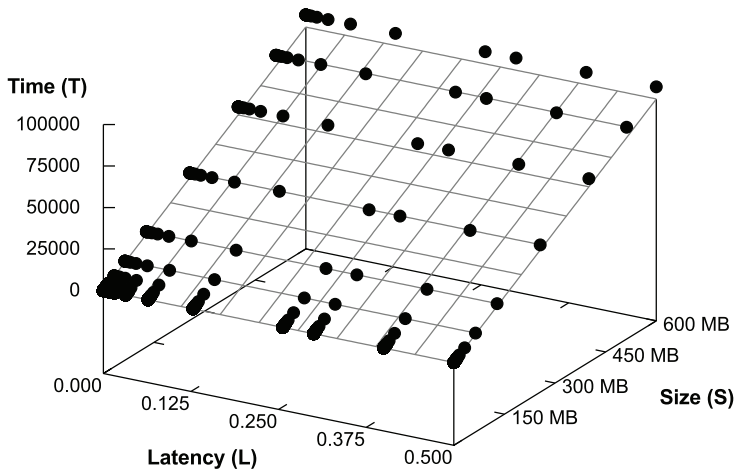


Figure 6.4: SimGrid model transmission time (surface) in comparison with GTNetS (points) in a 56 Kbit/s link. Time can be well estimated as a linear function of size, bandwidth and latency when limited by link bandwidth.

Figure 6.3 provides evidence that $\frac{W}{2L}$ is an important limiting factor of transmission rate while Figure 6.4 shows that bandwidth is a good approximation of transmission rate for large flows (when slow-start can be neglected). The open question now is to verify if the minimum among those two values is justified in the model when varying the middle link bandwidth.

To verify so, in Figure 6.5, the minimum between these two factors is plotted (surface) with the GTNetS results (points). Observing Figure 6.5 we can state that using the minimum of transmission rate and round trip time is an accurate model. For this results a fixed sized flow S of 10 MB was used while bandwidth was varied with 45 points $B \in$ [56 Kbit/s ; 10 Mbit/s].

One requirement of LSDC network models is to adapt to different usage cases. The model presented so far work well for long flows, however, our results point a pronounced for small messages. To increase model versatility, we next propose an accuracy improvement for these cases.

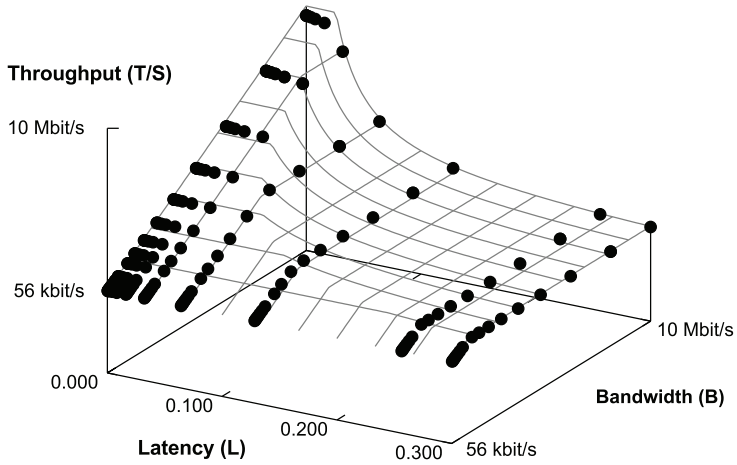


Figure 6.5: Transmission throughput of SimGrid model (surface) in comparison with GTNetS' throughput (points). When bandwidth is high flows become eventually limited by latency.

Improvements

The experiments presented until this point were already verified by Fujiwara *et al.* [34]. Here again, we reproduce her results aiming at reiterate her conclusions to point out flaws and effectively propose improvements when possible. Improvements for this simple case are obtained by best fitting the linear model with GTNetS results. Like this we hope to infer coefficients for latency and bandwidth that can overcome the two problems detected in this previous work: (i) small messages are willing to be inaccurate and (ii) transmission rate is slightly lower than nominal bandwidth. These coefficients are justified by two important TCP characteristics. First, the slow start algorithm which delays the time needed to reach steady-state is a source of error for short flows. Second, protocol layers use a parcel of bandwidth to provide routing support, enable congestion avoidance, detect errors and so on. Overhead data consumes significant throughput reflecting on precision loss for bandwidth bounded flows.

To acquire coefficient we do a best fit of the model in Equation (**??**) introducing one linear coefficient for bandwidth and another for latency. The idea here is to mitigate the effect of slow-start and overhead over model accuracy. Hereafter we propose in Equation (6.10) adjusting L and B with coefficients α and β to minimize absolute error, i.e., $min(|\log(T(L,B,S)) - \log(T_{GTNetS})|)$. The absolute error follows our error convention stated since the beginning of experiments.

$$T(L,B,S) = \alpha L + \frac{S}{\min(\beta B, \frac{W}{2L})} \tag{6.10}$$

Where

- α aims at mitigating the effect of slow-start
- β aims at reducing overhead traffic from available bandwidth

The slow-start algorithm prevents a flow of reaching too fast the maximum available transmission rate. This situation is undesirable because of the additive increase multiplicative decrease (AIMD) nature of TCP. Meaning that a new created flow achieves its transmission rate faster without contention. If a contention is detected the transmission rate is divided by two and will increase linearly for each acknowledge received over time. In Figure 6.6, it is illustrated how slow start behavior is captured by

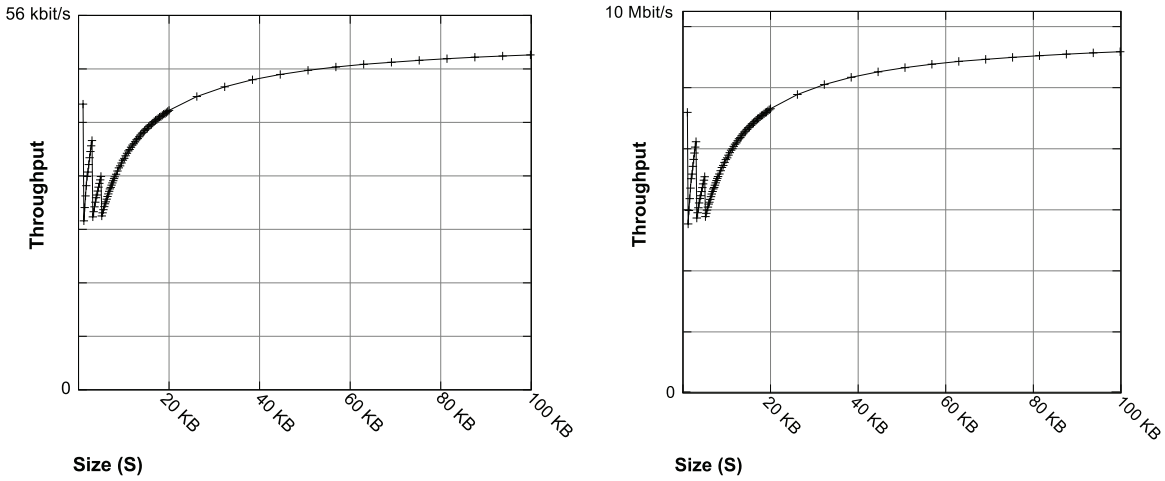


Figure 6.6: Throughput of short flows as a function of size for two network links, 56 kbit/s (left) and 10 Mbit/s (right). The slow start phase affects throughput of small messages but stabilizes for messages over 100 KB.

GTNetS. Using two different bandwidth parameters and a low latency (L = 0.00001) we see that indeed the achieved throughput ($\frac{S}{T}$) depends on message size due to the slow-start algorithm.

By the previous experiments we conclude that SimGrid is accurate for flows under 10 MB. This aspect of loosening accuracy for short flows is a characteristic of the fluid model that is hardly avoidable. Using linear regression we hope to mitigate maximum error.

The linear model is justified when flows reach steady-state, i.e., when the throughput achieved is near the available transmission rate. As illustrated before, the slow start algorithm will impose a gradual increase of transmission rate delaying the time needed to reach steady-state. This will impact on short flows accuracy. To increase the accuracy for short flows we propose a best fit to minimize the maximum error ($|\epsilon_{max}|$). Therefore, we need to respect a tradeoff between being accurate for short flows or accurate for long flows.

Table 6.2: Coefficients that best fit GTNetS values with respective error when varying the minimum message size. For messages over 100 KB the maximum error is acceptable.

Minimum message size		Coefficients α	β	Error (ϵ) ϵ_{min}	ϵ_{mean}	ϵ_{max}
$S \geq$	1 KB	7.05	0.53	0.00	0.33	0.57
$S \geq$	10 KB	10.55	0.80	0.00	0.12	0.25
$S \geq$	**100 KB**	13.01	0.97	0.00	**0.03**	**0.08**
$S \geq$	1 MB	15.06	0.95	0.00	0.01	0.02
$S \geq$	10 MB	22.34	0.94	0.00	0.00	0.02
$S \geq$	100 MB	18.42	0.92	0.01	0.02	0.02

Table 6.2 shows that best fitting selecting different messages sizes leads to different coefficients. Moreover, the table shows that the variation obtained is non-neglectable for small messages. Looking at the errors obtained with those coefficients, Table 6.2, it is conclusive that the best fit of GTNetS values for small messages directly increases the maximum error. Nevertheless, one can see that when best fitting for messages over 100 KB the error is less than 10%. This value offers a good tradeoff between

maximum error an message size. Hence we use the coefficients obtained with linear regression with flows above 100 KB.

Improved Results

After fitted using S >100 KB obtained coefficients, in Equation (6.11), we plot the enhanced model aside with the best fitted improved model in Figure 6.7. Bandwidth, latency, and size parameters were varied identically as the previous experiments.

$$T(L, B, S) = 13.01\ L + \frac{S}{\min\left(0.97\ B,\ \frac{W}{2L}\right)} \tag{6.11}$$

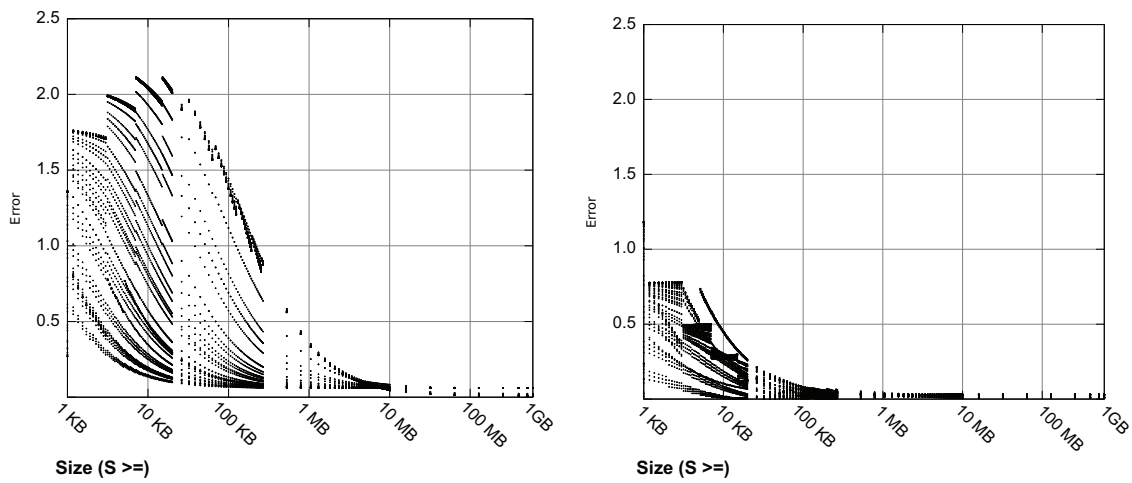


Figure 6.7: Error ϵ of original model (left) and error after fitting GTNetS results (right). Errors are significantly reduced after improvements.

Both graphs depict error as a function of message size. Analyzing the graph we can state that the error obtained when best fitting results indeed decreases significantly. The error for messages over 100 KB estimated as more than 100% with the original mode is reduced to less than 10% using the best fitted model of Equation (6.11). So, the validity range of the simulator is improved for messages under 100 KB.

6.3.2 Evaluation of the Max-Min Bandwidth Sharing Model

The model presented before showed to be accurate when one single flow, even if limited by the round trip time, pass through a single link. Nevertheless, in distributed computing environments network speed is often the bottleneck of performance. Contention scenarios might impact in the overall accuracy. Therefore, we evaluate the bandwidth sharing model pointing limitations and proposing improvements to accurate model networks.

Model

As pointed out by results the single link single flow model can be used to provide an accurate estimation of communication time whenever the available transmission rate is known. The problem now is to determine transmission rate out of nominal bandwidth when flows are concurrently sharing a network link.

In other words, given a set of links and a set of flows $\mathcal{F}$ we want to determine the amount of nominal bandwidth ρ_i that will be assigned to each flow $\mathcal{F}_i$ respecting that each link usage cannot exceed its nominal bandwidth capacity C_k, i.e., $\forall_k, \sum_{i|\mathcal{F}_i \text{ uses link } k} \rho_i \leq B_k$. And that is impossible for a given flow, F_i, to have a transmission rate greater than its TCP window per round trip time constraint, i.e., $\forall \mathcal{F}_i, \rho_i \leq \frac{W}{RTT_i}$. It is common sense that the TCP flow control objective to provide a minimum of bandwidth share for each flow. Hence, an intuitive models, is to use a Max-Min fairness. This sharing policy guarantees that the flow with lowest bandwidth achieve the maximum possible. So, the final contention aware network model appears in Equation (6.12), as explained in Chapter 4.

$$\begin{aligned}
&\text{MAXIMIZE } \min_i (w_i\ \rho_i)\,, \\
&\text{UNDER CONSTRAINTS} \\
&\begin{cases} \text{(6.12a)} \quad \forall_k, \displaystyle\sum_{i|\mathcal{F}_i \text{ uses } k} \rho_i \leq B_k \\ \text{(6.12b)} \quad \forall \mathcal{F}_i, \rho_i \leq \dfrac{W}{RTT_i} \end{cases}
\end{aligned} \tag{6.12}$$

In Equation (6.12) there is a weight w_i. This weight can optionally be used to model the effect of latency when contention is detected. A flow with lower latency would achieve slightly more bandwidth than a flow with higher latency due to the multiplicative decrease additive increase characteristic of TCP protocols. So at first we model the weight as being the sum of latency for links that the flow pass through, or $\forall \mathcal{F}_i, w_i = \sum_{k|\mathcal{F}_i \text{ uses } k} L_k$.

Quantitative Analysis

To analyze the behavior of TCP sharing algorithm we configure GTNetS to use the TCP Reno algorithm and RED for queuing policy on routers. The topology used is a simple dumbbell network. This topology is designed to provide a simple contention scenario with two concurrent flows, flow A from client1 to server 1 and flow B from client2 to server2 as illustrated by Figure 6.8. Contention will limit flows rate since the link in the middle has a variable bandwidth lower than the other links. The middle link is isolated by two routers to guarantee that the sharing is uninfluenced upon race conditions in protocol stack access.

In this platform all links have latency fixed to 0.01 s with the exception of the bottom-right link that has variable latency as it appears Figure 6.8. In our experiments the bandwidth of the middle link is sampled linearly with 45 points in [56 Kbit/s, 16 Gbit/s] and latency is linearly sampled with 25 points in [0.00001 s, 0.2 s]. The message size was fixed to 100 MB to assure that flows reach steady-state.

Figure 6.9 the sharing result is presented for four fixed bandwidth parameters, from left to right and top to bottom: 10 kB/s, 100 kB/s, 500 kB/s, 1 MB/s. In Figure 6.9, the throughput share achieved by flow A, presented as the bottom bars, is presented below the achieved throughput by flow B on top. In spite of being intuitive this model fail to correct model the sharing of TCP when the throughput is limited

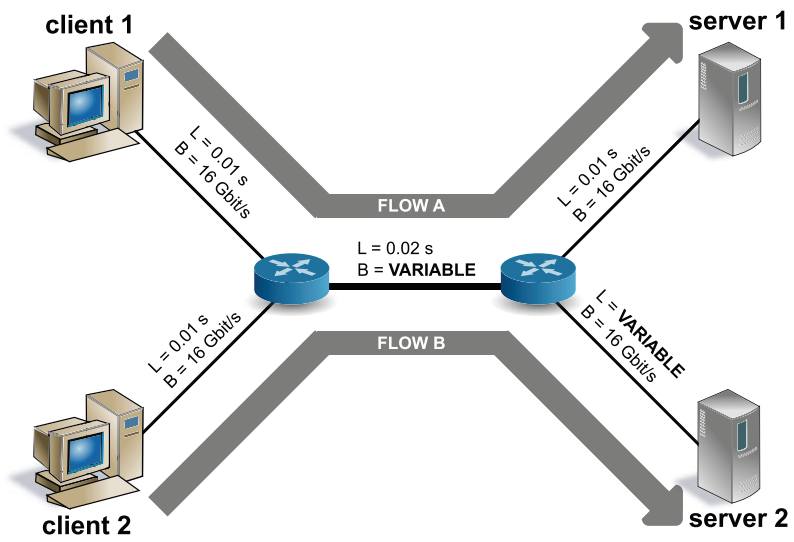


Figure 6.8: Dumbbell topology used to obtain a simple contention scenario.

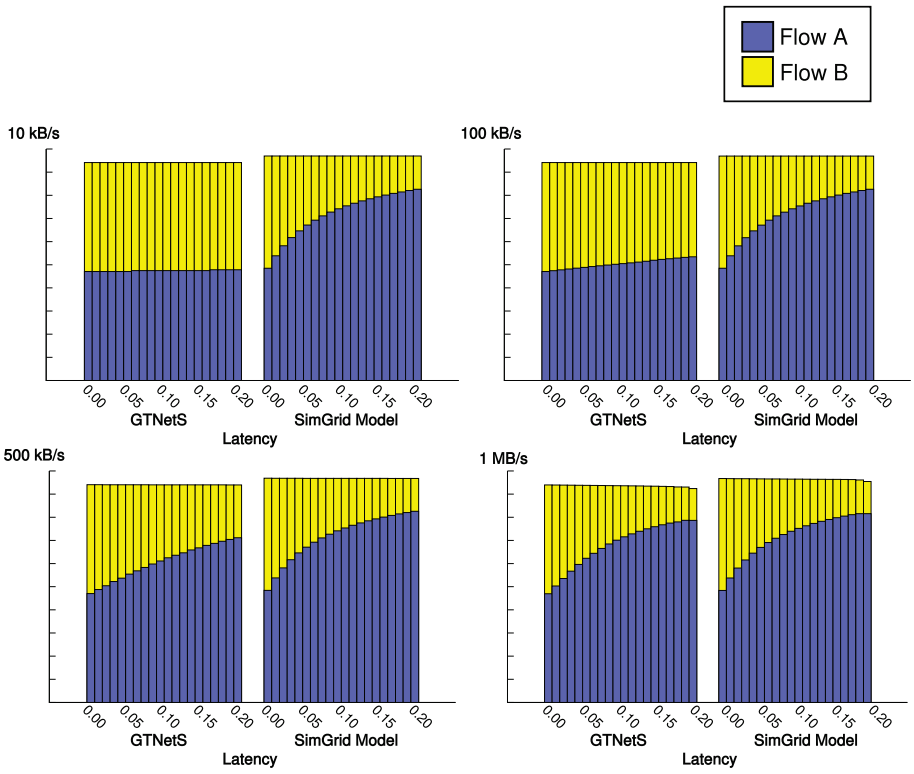


Figure 6.9: GTNetS throughput sharing result in comparison with SimGrid sharing model. SimGrid models is inaccurate when bandwidth for small bandwidth values.

by bandwidth. However, when flows are limited by the RTT constraint the model is indeed accurate, Figure 6.9.

Observing results, we can conclude that the model is inaccurate and also that the bandwidth capacity of middle link influences the sharing results. Following, we investigate the origin of such a significant discrepancy.

Improvements

Results of Figure 6.9 suggest that the sharing model is bad instantiated when flows are limited by bandwidth. After further investigation we concluded that GTNetS sharing is sensible to both, bandwidth and latency, disagreeing with the model sharing which is only influenced by latency. Translating to the model this leads that the weight w_i associated to each flow i modeled as the sum of latency of links (or half of round trip time) can be adapted to improve accuracy. This weight was chosen by design because it is logical that a flow with higher latency will take more time to be stabilized and hence, when it reaches steady-state, the flow will achieve less throughput than its rival.

$$\rho_A w_A = \rho_B w_B \Leftrightarrow \frac{\rho_A}{\rho_B} = \frac{w_B}{w_A} \tag{6.13}$$

To improve the model accuracy, we conducted many experiments to understand the behavior of the weight as expected by GTNetS. Basically, we know that when the weights are well instantiated the ratio of flows throughput will be equal to the inverse of the weight ratio. In Equation (6.13) an algebraic expression that formalize this idea is given. Following, we conduct a series of experiments trying to isolate GTNetS expected behavior.

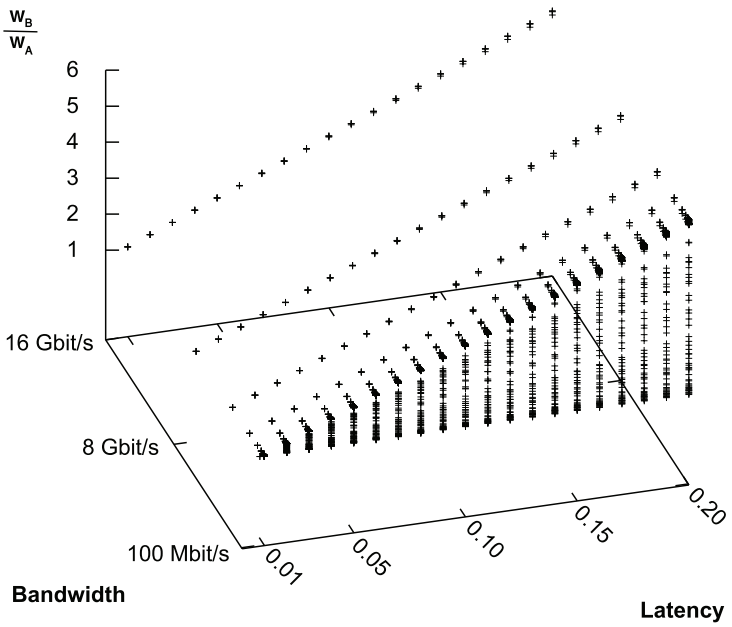


Figure 6.10: GTNetS throughput of flow A over flow B $\left(\frac{\rho_A}{\rho_B} = \frac{w_B}{w_A}\right)$. Observing GTNetS we conclude that the problem with the model is bad instantiation.

In Figure 6.10, the obtained throughput fraction $\frac{\rho_A}{\rho_B}$ (or $\frac{w_B}{w_A}$) are presented for every experiment. The behavior of the sharing is indeed dependent of the bandwidth of the middle link, specially for small

bandwidth values. So, the bandwidth needs to be considered in the weight computation in these cases. At this point we have enough evidence that the weight approximation would be inversely dependent on bandwidth (i.e., as bandwidth values increase they tend to influence less on the weight) and directly dependent on latency (i.e., as latency values they tend to influence more on the weight). Hence to improve the sharing model we propose a new weight approximation coupling with those two characteristics, as presented in Equation (6.14). Two important remarks. First, the sum of latencies is enough when bandwidth is neglected, this is intuitive but can be confirmed by results of Figure 6.10 when the bandwidth parameter grows the ratio of weight is linear and only depends on latency. Second, to determine the point where bandwidth must be neglected we introduced a new parameter γ.

$$\forall \mathcal{F}_i, w_i = \sum_{k|\mathcal{F}_i \text{ uses } k} \left(L_k + \frac{\gamma}{B_k} \right) \tag{6.14}$$

The factor γ must be fitted to reproduce satisfactorily the behavior of GTNetS, i.e., best fitting points of Figure 6.10. The fit criteria is to minimize the maximum error between GTNetS ratio and SimGrid ratio. Hence, with a dichotomy approach we can find γ such as the sum of maximum error is minimum. Again, we focus on maximum error to provide a guarantee of simulation accuracy in a worst case perspective. Figure 6.11 detail the result after fixing gamma.

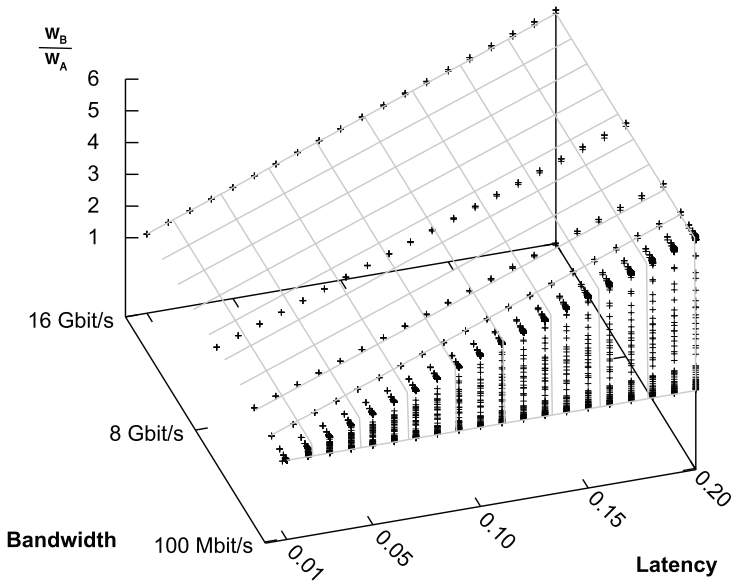


Figure 6.11: GTNetS sharing $\frac{\rho_A}{\rho_B}$ (crosses) and the approximated Max-Min weight (surface) when γ fits GTNetS behavior, i.e., $\gamma = 20537.14$.

The expected behavior with the dumbbell topology of the model can be verified in Algorithm 6.2. First the bandwidth share is computed using Equation (6.10) $share_A$ and $share_B$ respecting Equation (6.13). The simplest case is when both flows are limited by bandwidth share as illustrated by Algorithm 6.2. Otherwise, the Max-Min model privileges the flow that receives less transmission rate fixing its value to the minimum available bandwidth. After that, one flow is fixed and the remaining flow will receive the rest of available bandwidth unless it is bounded by the round trip time constraint.

Algorithm 6.2 Calculate $\frac{w_B}{w_A}$

$B \Leftarrow 0.969833.B$

$share_A \Leftarrow \frac{w_B}{(w_A+w_B)} \times B$
$share_B \Leftarrow \frac{w_A}{(w_A+w_B)} \times B$

$sat_A \Leftarrow \frac{65535}{2\times 0.04}$
$sat_B \Leftarrow \frac{65535}{2\times(0.03+L)}$

if $share_A \leq sat_A$ and $share_B \leq sat_B$ **then**
  return $\frac{share_A}{share_B}$
else
  if $sat_A \leq sat_B$ and $sat_A \leq share_A$ **then**
    return $\frac{sat_A}{\min(sat_B, B-sat_A)}$
  else
    return $\frac{\min(sat_A, B-sat_B)}{sat_B}$
  end if
end if

Improved Results

The absolute maximum error before adjusting the best fitted γ was approximately 1.71 which is approximately 224.93% of maximum error. Using γ = 20537.14, the best fitted value, the maximum error decreases to approximately 0.18, i.e., about 20% of maximum error. Using the values obtained from the dichotomy we have results presented in Figure 6.12.

6.3.3 Evaluation of Bidirectional TCP Connections

Continuing the model evaluation campaign the next tests aim at evaluating throughput sharing when flows have opposite directions. In theory the need to distinct between ongoing and incoming traffic arises from full-duplex network devices. In those cases down/up traffic are independent and both are supposed to achieve maximum transmission rate. Indeed assuming that ongoing traffic is independent of incoming traffic is true for network layer protocols. However this is false for high level transmission protocols such as, TCP/IP. For those protocols ongoing traffic generates incoming traffic to provide congestion control. Packets and acks can eventually collide creating a dependency between incoming and ongoing transmission rate, even if theoretically both can travel at full speed. In such cases, the SimGrid model presented before is unrealistic. In this section, we present experiments that point out those flaws presenting a workaround to improve network model accuracy in such cases.

Quantitative Analysis

Here we are particularly interested in the bandwidth share when limited by transmission rate. To guarantee this we will use a fixed bandwidth of B = 56 kbit/s and a fixed latency of L = 0.00001s in a single link platform as depicted by Figure 6.13. In those tests, to avoid the slow-start phase, the size of messages will be fixed to 100 MB. Finally, to evaluate the interactions of incoming and ongoing traffic we varied the number of simultaneous upload flows (outgoing traffic) and download flows (incoming traffic) as follows:

- $Df \in$ {0, 1, 2, 5, 8, 10, 12, 20}

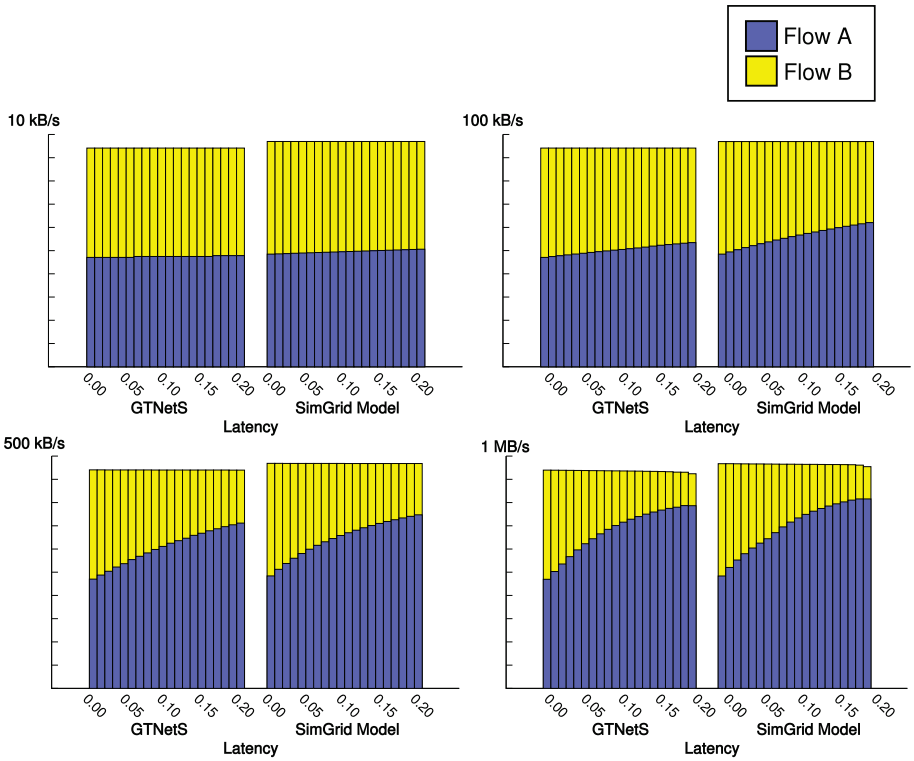


Figure 6.12: GTNetS bandwidth share in comparison with SimGrid model after fitting GTNetS. Resulting share is improved when flows are limited by bandwidth.

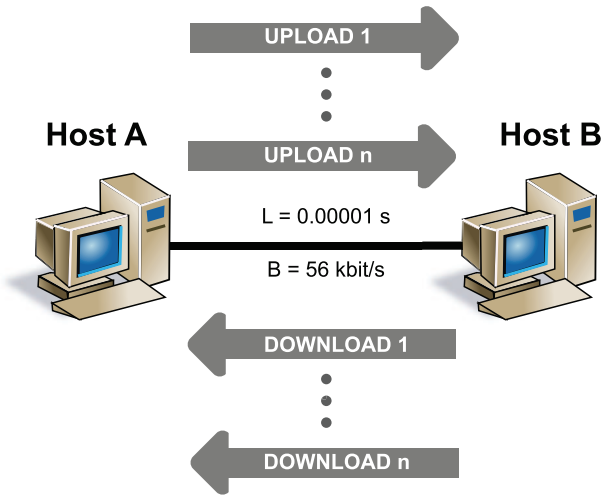


Figure 6.13: Bidirectional test topology to verify the influence of simultaneous uploads and downloads in a full-duplex network.

- $Uf \in$ {0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 30, 40, 50, 100}

This choice of parameters was made to stress contention, so that we have, in the worst contention case, 100 downloading flows plus 20 flows simultaneously transmitting. The sense of flows depend on the transmission perspective, uploads can be seen as downloads taken as perspective host B instead of host A. So, the results obtained are symmetric.

The first test aims at verifying if with zero uploads Uf= 0 the system behavior follows the SimGrid model. In that condition, each download flow Df_i should receive an equal share of the total available link bandwidth capacity. The result of this test is depicted by Figure 6.14 top graph. We plot the achieved mean throughput per flow when varying the number of download flows Df. We used a logarithmic scale to highlight that the bandwidth achieved per flow should be proportional to the amount of download flows (Df). In other words, for one flow we exStudentpected it to achieve 100% of throughput. In the case of two concurrent flows that each one achieve 50%, and for 3 flows that each achieve 33.33% and so on. This expected behavior is plotted in the graph as a continuous line. Since we are doing the mean achieved throughput per flow we plot the confidence interval (following a t-distribution with α = 0.05) to assure that the means are comparable. In Figure 6.14 at the bottom graph, we present the sum of achieved throughput by each flow as a percentage of link usage in linear scale. Looking at the bottom graph one can see that a degradation in link usage appears when using more than 10 up to 100 simultaneous flows.

Agreeing with our expectations the top graph of Figure 6.14 shows that the mean throughput achieved per flow is very close to the SimGrid model, (continuous line). Meaning that each download flow receives an equal share of the available bandwidth without concurrent uploads. However, there is a slightly degradation when the number of concurrent downloads increases as depicted by the bottom graph of Figure 6.14. This was expected due to the perturbation introduced when many flows share low level resources such as buffer and network device that are only perceived by GTNetS.

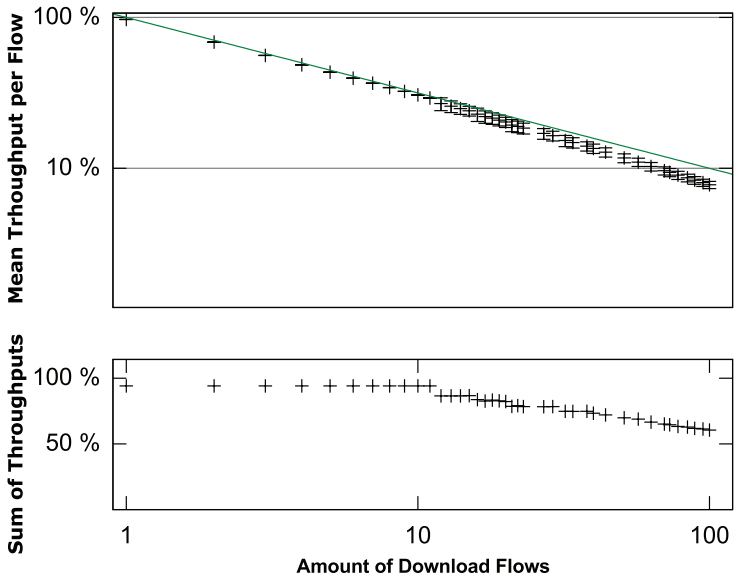


Figure 6.14: Throughput share when varying the amount of download flows. The previous model is enough to model such cases.

Following, to observe if upload flows change the behavior of concurrent downloads, we plot, in Figure 6.15, the mean throughput achieved per download flow with 5 concurrent uploads (left graph) or with

10 concurrent uploads (right graph). At the bottom of each graph the sum of achieved throughput is presented. This time the sum goes up to 200% because ongoing traffic is theoretically independent of incoming traffic. Like that, it is expected that the transmission rate available for both direction is twice the available bandwidth for each direction. In this case we expect that upload and download flows will be independent. So, we expect that the mean throughput of download flows will follow a linear behavior.

When the amount of download flows is less than the number of upload flows, we can observe that each flow will receive a fixed mean throughput, for instance, one fifth (or 20%), as illustrated in Figure 6.15 (left). Curiously, this value is exactly what each flows would receive if there were only 5 flows sharing the link. This unexpected behavior stops when there are the same number of download and upload flows. In Figure 6.15 (right) we can observe the same strange behavior. Hence the assumption that uploads and downloads are independent falls down and the SimGrid model, continuous lines, is inaccurate.

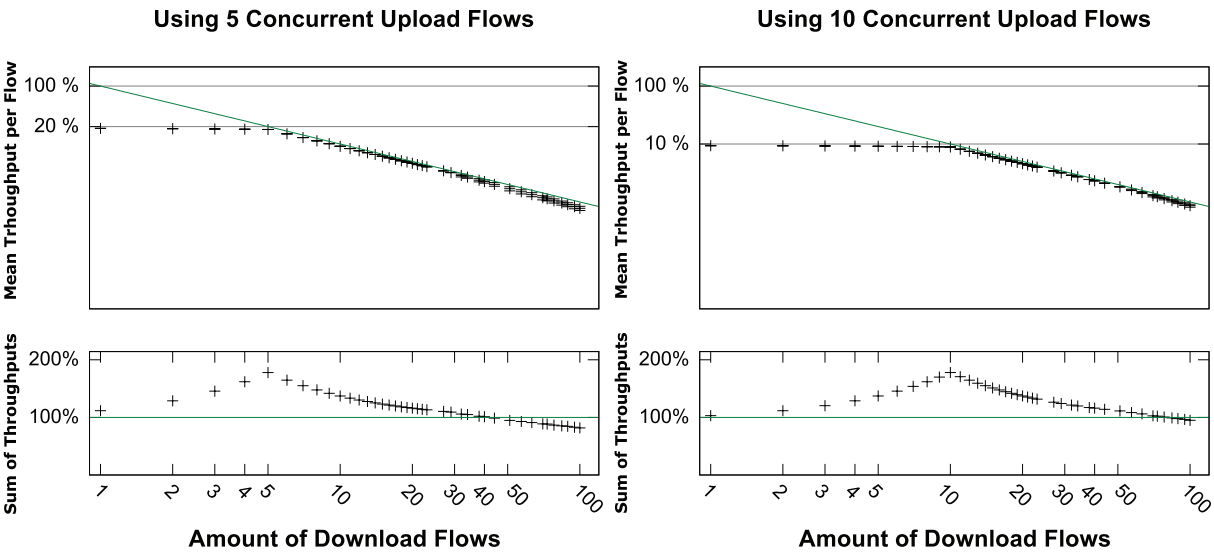


Figure 6.15: Throughput with concurrent download and upload flows.

So, the SimGrid model fails to model concurrent ongoing and incoming traffic. First, the model cannot distinguish between ongoing and incoming traffic. Therefore, the model is unable to correctly see the capacity of the link in both sides. Second, the continuous line of achieved throughput will be sensible by the amount of concurrent flows in both direction. So, to improve the model accuracy this phenomenon must be studied and incorporated in the model.

Ack Compression

Indeed, the phenomenon detected is known as ack compression. This phenomenon was documented in the early 1990's by Zhang *et al.* [99] where the authors defined it as "fluctuations in queue length due to interaction of ack and data packages." In their work, Zhang *et al.*, show how to reproduce such a phenomenon using a single link platform with two hosts both working as sender and receiver (bidirectional traffic). Their conclusion point out that ack packets may be stuck in the sender queue waiting for a data packets to be transmitted. When this kind of situation arrives, the connection will eventually be limited by the time spent delivering/receiving acks bounding network traffic.

To illustrate a delayed ack example let us imagine the following situation: Two hosts, one server and one client, are connected through a single full-duplex Ethernet link. In this example the time to send a data packet is many times slower than the time to send an ack packet. Continuing, the client is uploading two files on the server. Simultaneously, with the two uploads, the client is downloading one file. In this example one ack packet is generated for each received data packet[2].

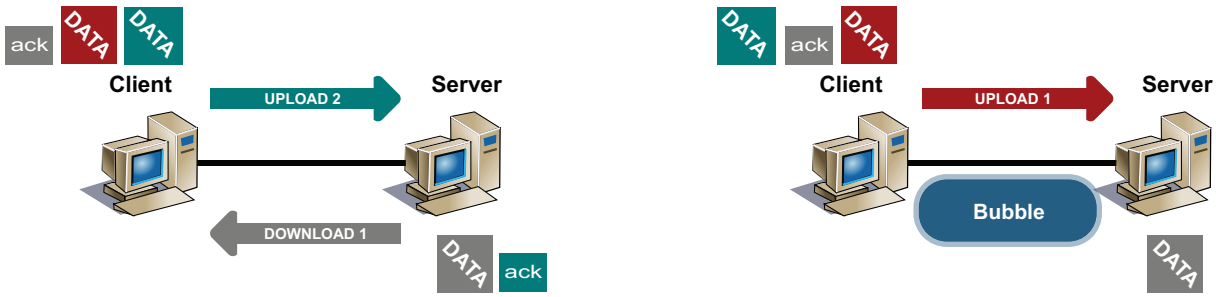


Figure 6.16: In TCP ack compression is a common phenomenon. When this phenomenon happens ack packets get stuck (or compressed) in the client sending queue forcing the download flow to use only 50% of available bandwidth.

If we consider that the link is full-duplex the expected behavior is that each upload receive half of the available bandwidth while the download receives the entire bandwidth. In practice, as depicted by Figure 6.16 left side, when the transmissions start one of the uploads is sending simultaneously with the download. As expected, in this first communication step, the link usage is optimal. However, eventually the second upload will transmit a data packet, depicted by Figure 6.16 right side, at this point the download is unable to proceed until the ack packet that is behind the upload data packet arrives. By consequence, the ack packet get stucked or compressed in the client sending queue generating a bubble. Like that, the fact that the ack packet is waiting for a data packet to arrive bound the download throughput, i.e., the link cannot be used for simultaneous download/upload 100% of the time.

Today even modern versions of TCP/IP that claim to be enhanced for maximum throughput as cubic, in Linux, lack of a mechanisms to improve bidirectional traffic. The reason is that many authors blame the use of asymmetrical devices, such as ADSL modems, to the only origin of ack compression. However, as presented here, ack compression is a phenomenon that is more likely to happen than expected because of the congestion control mechanism of TCP. Unfortunately, providing a solution for this problem is an open research problem and is out of the scope of this thesis. Here we are interested to provide reliable models to imitate this behavior. Targeting this goal, we present, in the next section a few modifications in our network model so that the simulator can reflect this behavior.

Improvements

As illustrated in Figures 6.13 and 6.14, the current model is incapable of modeling bidirectional traffic. To propose a full-duplex SimGrid model we modify the platform file doubling each link. With that we provide the possibility to distinguish among ongoing and incoming data transfers. Adding this mechanism we will introduce the idea of traffic control. That means, for each created flow, one opposite flow is created to simulate the handshake of the congestion avoidance protocol. Resulting that each transmission will

[2]In practice things are slightly more complicated but this simple situation is already enough to illustrate the phenomenon

be followed by the transmission of a smaller flow of acknowledges packets in the opposite direction. We define the size of acknowledges traffic empirically as 5% of data traffic.

The resulting of this improvement inside the Max-Min model are two. First all constraints are doubled with an opposite direction constraint. Second, each time a variable is created another variable is created consuming 5% of the connection bandwidth on the opposite direction, to represent the traffic of ack packets. This behavior is illustrated by Figure 6.17. In the figure each constraint has a friend constraint C_i^{in} and C_i^{out}. When one flow ρ_i is created another flow consuming 5% in the opposite direction ρ_i' must be also created.

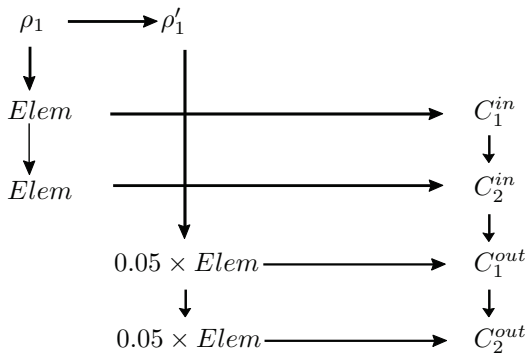


Figure 6.17: Sparse Max-Min representation when using the full-duplex model. One smaller data flow ρ_1' is created in opposite direction of the main flow ρ_1. The constraints are also doubled to reflect the possibility of using full power in both directions.

To illustrate why the full-duplex model changes the result of Max-Min algorithm we recall to the example of Figure 6.16. In this example we have two upload flows and one download. As we saw, ack packets get stuck in the client sending queue. This makes that the download flows uses only 50% of the available download bandwidth.

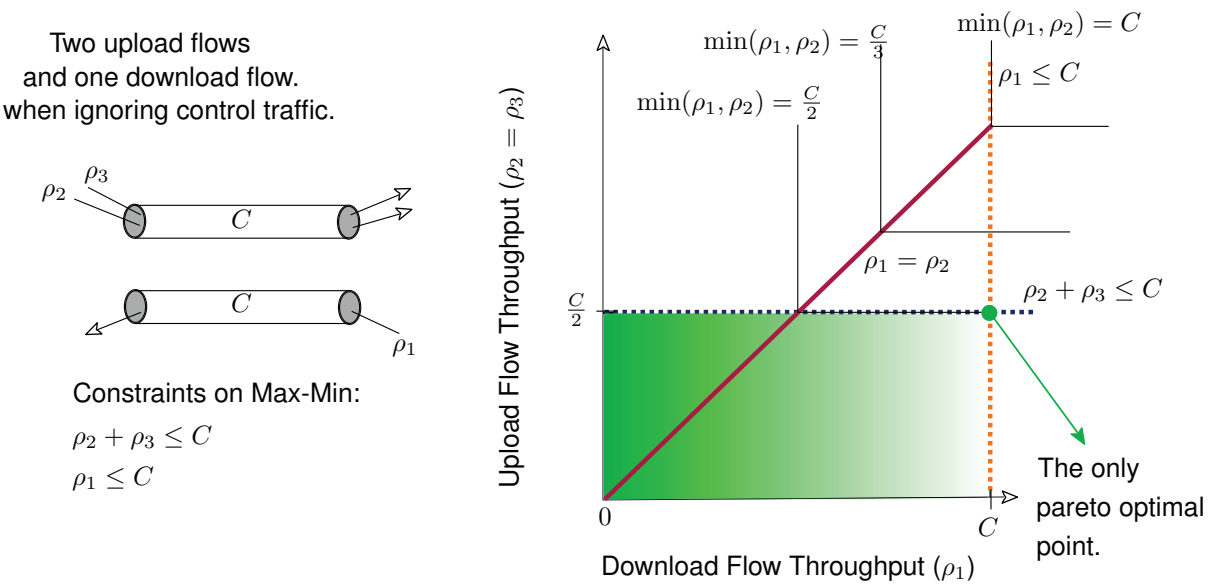


Figure 6.18: Ignoring control traffic the Max-Min algorithm finds the only available Pareto optimal point.

The Max-Min algorithm, detailed in Chapter 5, finds a Pareto optimal point as defined in Equation (6.15). For short, to consider a point Pareto optimal any other point compared to it that allocates

more bandwidth to a given variable will decrease another variable. As so, we consider any Pareto optimal point as a possible solution for the maximization problem.

$$\forall u \in U, \exists, v_i > u_i \Longrightarrow \exists j, v_j < u_j \tag{6.15}$$

Therefore, we present in Figure 6.18, a simplified version of the full-duplex model ignoring control traffic. In this simplified version the top link is used exclusively for upload passing two files. Similarly, the bottom link dedicated to download. In this full-duplex scenario there are two constraints inside Max-Min model. The figure shows aside, in a graph, a graphical 2D representation of achieved throughput of download flow (ρ_1) compared to achieved throughput of upload flows (ρ_2), since we can freely assume that $\rho_2 = \rho_3$. In the graph, the marked area represents the zone which respects the availability constraints. In this example, the only Pareto optimal point is $(C, \frac{C}{2})$. So, the solution is $\rho_2 = \rho_3 = \frac{C}{2}$ and $\rho_1 = C$. As we saw, this result is incoherent with TCP behavior that should give $\rho_1 \cong \rho_2 \cong \rho_3 \cong \frac{C}{2}$ due to ack compression. So, this simple model fails to respect TCP behavior.

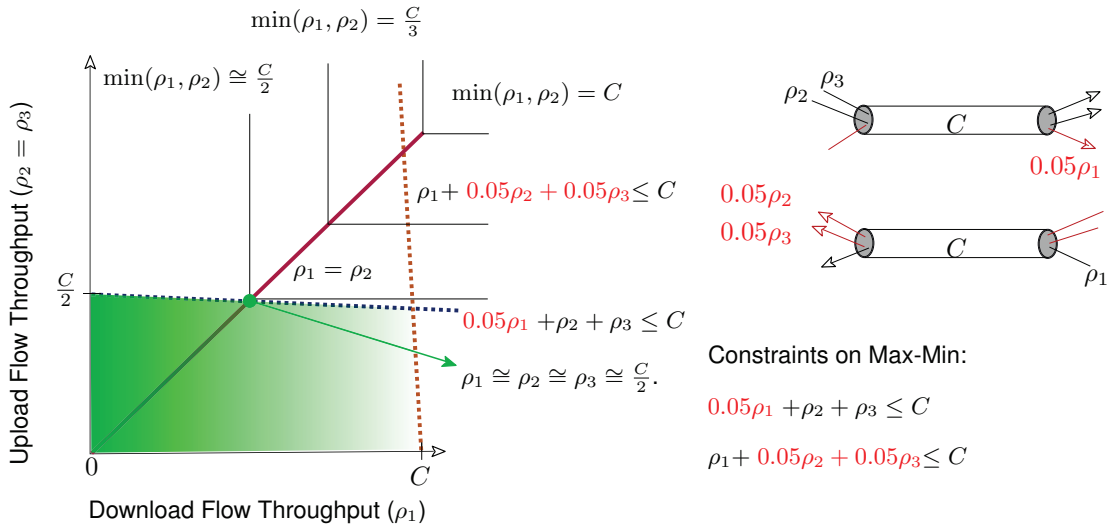


Figure 6.19: Adding control traffic in opposite direction Max-Min is able to improve TCP modeling. When adding control traffic each point on the edge of the validity zone is a Pareto optimal point, i.e., a possible solution. So, with the full-duplex model Max-Min is capable of correct modeling ack compression, i.e., $\rho_1 \cong \rho_2 \cong \rho_3 \cong \frac{C}{2}$.

If we extend the example of Figure 6.18 to match the full-duplex model proposed here we can find the correct answer. In Figure 6.19 we illustrate the same example but now the constraints consider opposite traffic. Observing the figure we see that, now, every point in the edge of the green zone is Pareto optimal, i.e., a feasible solution. Deforming the zone that respects the availability constraints Max-Min is able to find an improved solution, i.e., $\rho_1 \cong \rho_2 \cong \rho_3 \cong \frac{C}{2}$. Consequently, adding one flow to represent the cross-traffic we can significantly improve the model. We next evaluate the results of this improvements.

Improved Results

To test if the improvements indeed solved the problem of bidirectional traffic, we present here results using the same parameters as before. In other words, we varied the number of upload and downloads in the following intervals:

- $Df \in$ {0, 1, 2, 5, 8, 10, 12, 20}

- $Uf \in$ {0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 30, 40, 50, 100}

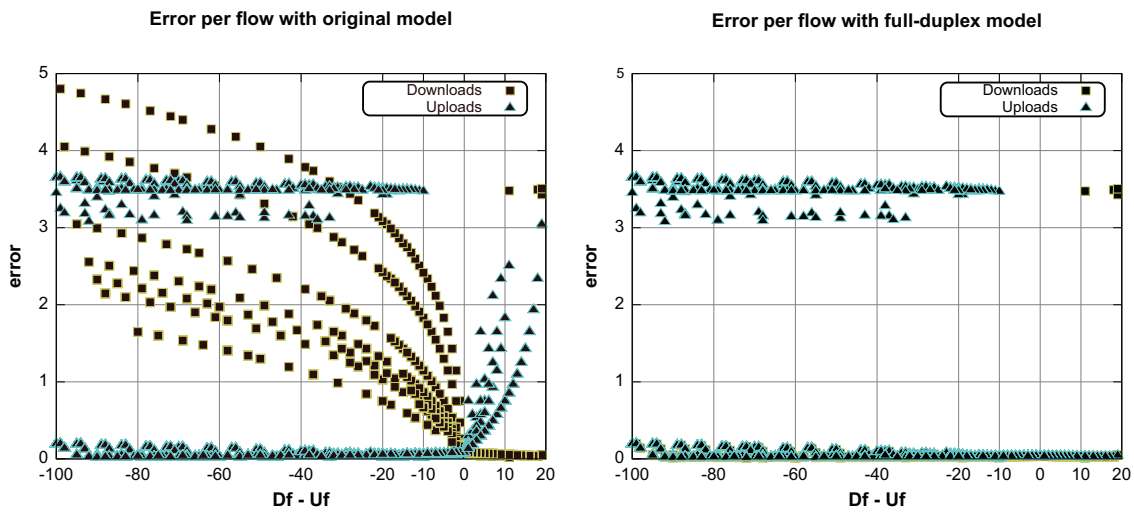


Figure 6.20: Error of each flow as function of the contention $D_f - D_u$ (difference between download flows and upload flows). We see results without the full-duplex model (left graph) and with full-duplex model (right graph). One can see that in spite of more organized results there are still flows hard to predict.

In Figure 6.20(left), we show the error when using the previous model, without full-duplex support. In Figure 6.20(right) we plot the error obtained with the full-duplex model. In spite of more organized, some flows are hard to predict even after the full-duplex improvement. We believe that this happens due to synchronization issues where only one or two flows, among various, are penalized with an insignificant bandwidth. However, one can see in Figure 6.20 (right) that when the number of flows in both directions is balanced this phenomenon is less likely to happen.

Based on the results we can infer that the difference of flows on opposite directions is directly related to bad estimations. This assumption is based in the blank gap between -10 and 10 in Figure 6.20. This blank gap indicates that the difference of flows going on opposite directions is one of the factor that determine the validity range of the fluid model. As so, we define a contention index K_l per link l, based on this measure, i.e., the absolute values of the difference among ongoing and incoming flows $K_l = |D_f - U_f|$. The proposed contention index has the advantage of being independent of perspective (symmetrical) and can detail a per link usage. Using this index we can warn users when their simulations are more likely to be inaccurate.

Until this point, all proposed improvements were tested under the same settings they were detected. Such an approach reinforces that the improvements were relevant and fixed the accuracy issue they were meant to fix. However, testing the proposed improvements under general settings is needed to state that the gain is effective under general settings. As such, we next evaluate network models using varying several parameters.

6.3.4 Evaluation with Random Network Topologies

The ideas presented here improved the network model in the same conditions where inaccurate results were detected. To reinforce our conclusions we need to do further investigation with more complex scenarios. Throughout this section we present an intensive campaign of experiments using a series of random generated platforms. Our goal is to verify if the improvements present the same accuracy gains in different scenarios. As so, to start we use two different “Random Topology” generators:

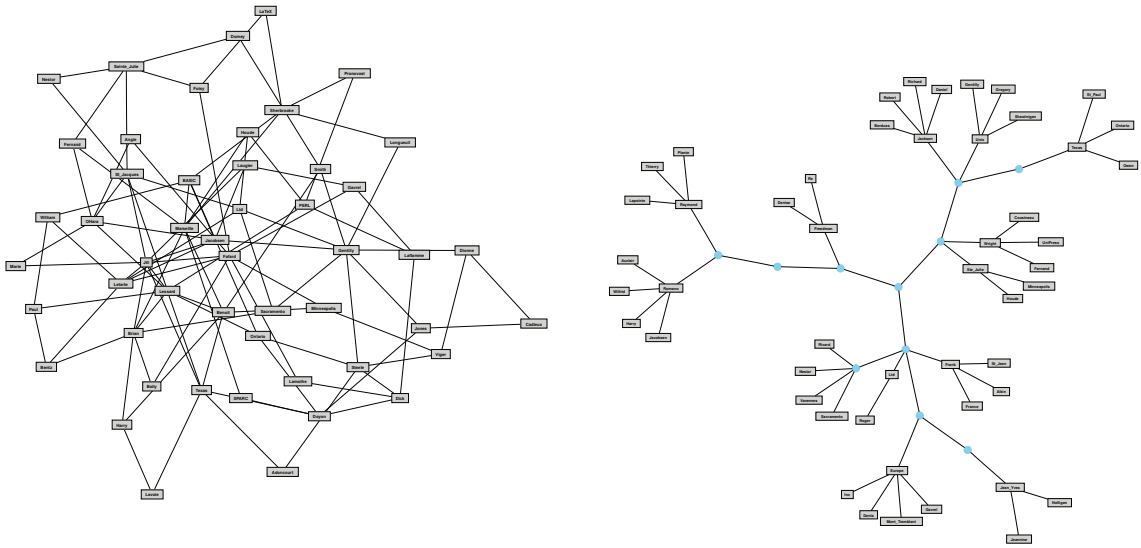


Figure 6.21: Comparison of a 50-node random platforms generated with **brite** (left) and **tiers** (right). The platforms differ significantly in topology.

- The BRITE generator [59] implements the Waxman model [93] proposed in 1988. Using BRITE nodes are placed uniformly in a 2D plane. The probability of two nodes being connected is proportional by their Euclidean distance. As an example, a Waxman platform with 50 nodes appears in Figure 6.21 (right);
- The Tiers generator [58] claims to be more realistic since today networks have many hierarchical domains. To reflect this, the Tiers standard sees the network in three levels.
  - Local Area Networks (LANs): also called edge nodes.
  - Metropolitan Area Networks (MANs): representing router, and switches.
  - Wide Area Networks (WANs): to model high bandwidth backbones.

  So, bandwidth and latency parameters are different for each domain: LANs have less bandwidth and lower latencies when compared to MAN or WAN. Nevertheless, in a topological perspective, LANs are more interconnected than MAN or WAN resulting in a visible difference from Waxman topologies as shown in Figure 6.21.

Random topology generation is a more complex subject than it seems. Several generators exist today: GT-ITM [18], Inet [47] and PLRG [3] to cite a few. However, it is out of the scope of this thesis to enter in the detail on the subject of random topology generation approaches.

Table 6.3: Platform characteristics of **test set**. Those platforms are generated randomly using Waxman and Tiers models. Even when the characteristics are similar the platforms differ in topology. Tiers format is more heterogeneous in bandwidth and latency.

	Topology Characteristics			Latency Range (ms)			Bandwidth Range (MB/s)		
Platform Model	Hosts	Routers	Links	Min	Median	Max	Min	Median	Max
Waxman	50	0	200	0.14	1.80	3.70	10.53	63.32	126.64
				0.10	1.90	3.70	16.67	62.62	125.23
				0.04	1.40	2.90	10.03	63.88	127.75
				0.18	1.70	3.50	10.44	63.80	127.59
	200	0	800	0.07	1.80	3.70	10.47	63.82	127.65
				0.04	1.80	3.60	10.17	63.54	127.07
				0.03	1.70	3.40	10.18	63.81	127.63
				0.02	1.90	3.80	10.06	63.76	127.53
Tiers	11	13	46	0.11	4.40	8.80	0.27	66.49	132.98
	32	28	118	$<$ 0.001	4.40	8.80	0.11	127.61	255.23
	50	35	168	$<$ 0.001	2.04	4.09	10.98	44,465	88,931
	100	60	318	$<$ 0.001	2.36	4.73	10.02	43,762	87,525

The testing sets comprised 12 platforms that are either small (11 and 50 nodes) or large (100 and 200 nodes), homogeneous platforms (B follows a uniform distribution in [10 MB/s, 128 MB/s]), or heterogeneous platforms (B follows a non-uniform distribution in [100 kB/s, 90 GB/s]). A more detailed description of platforms is available in Table 6.3. For these tests, we fixed the size of messages to 100 MB, avoiding with that the noise of the TCP initialization phase.

To test those platforms, 100 flows were generated between random pairs of end-points. Next, we vary 10 different deployments for each platform, totalling 120 experiments. During experiments, the TCP maximum window was fixed to 65535 B and the policy on routers was set to RED. To evaluate the accuracy, we use the same approach and error metric as before. So, all flows are stopped as soon as the first flow ends and their instantaneous throughput is computed using the amount of data transfered so far.

For the sake of clarity we plot four graphs in the same figure aiming at comparing the four performance improvements presented here. Like this, we put in evidence that the improvements proposed incrementally helped reducing the error. Hereafter, we compare worst cases (maximum error) trying to understand the simulation limitation. Also, we are particularly interested in mean error to observe accuracy values that are likely to happen.

In Figure 6.22 and Figure 6.23 we present the maximum error and mean error evolution. We start showing in the top-left corner the original model, adding incrementally the three improvements proposed in this chapter: α and β coefficients, γ coefficient, and finally, full-duplex. For the sake of clarity we use this ▣ to reference each graph. We keep in each figure the same convention described next.

- ▣ original SimGrid model, as presented in Chapter 5.

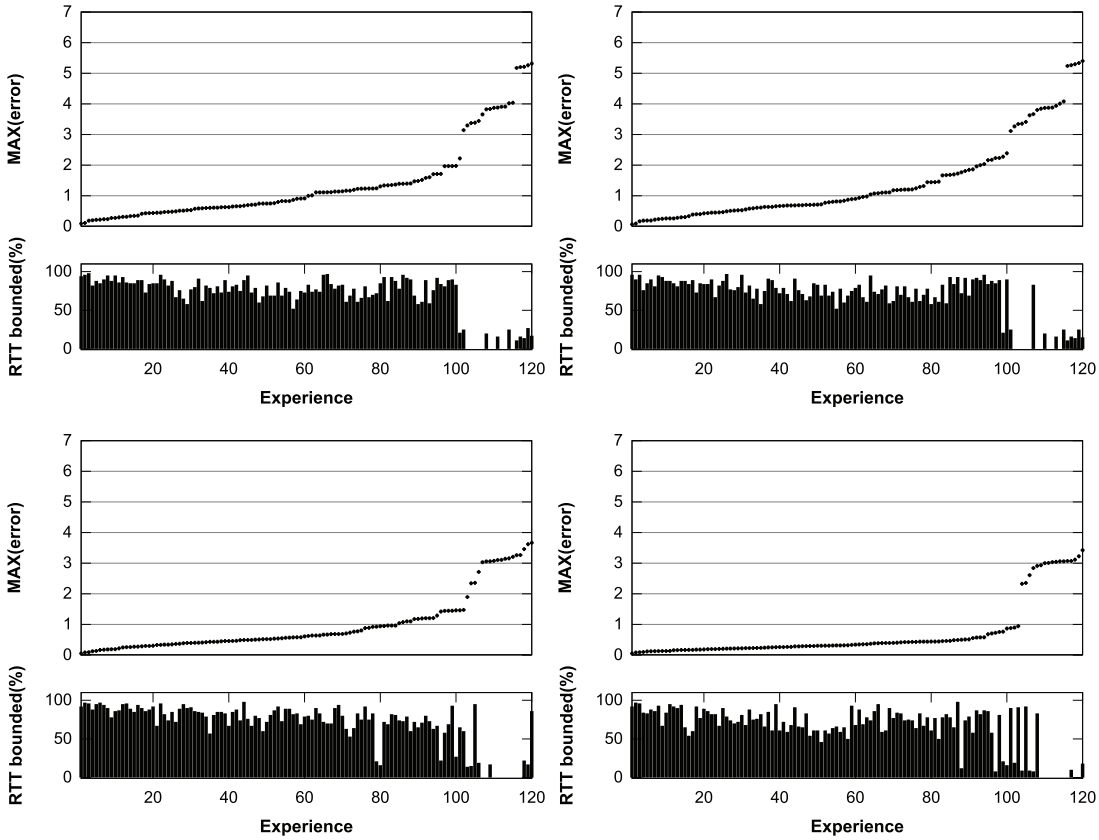


Figure 6.22: The maximum error after several improvements using random topologies. The model after improvements, bottom-right, has improved accuracy.

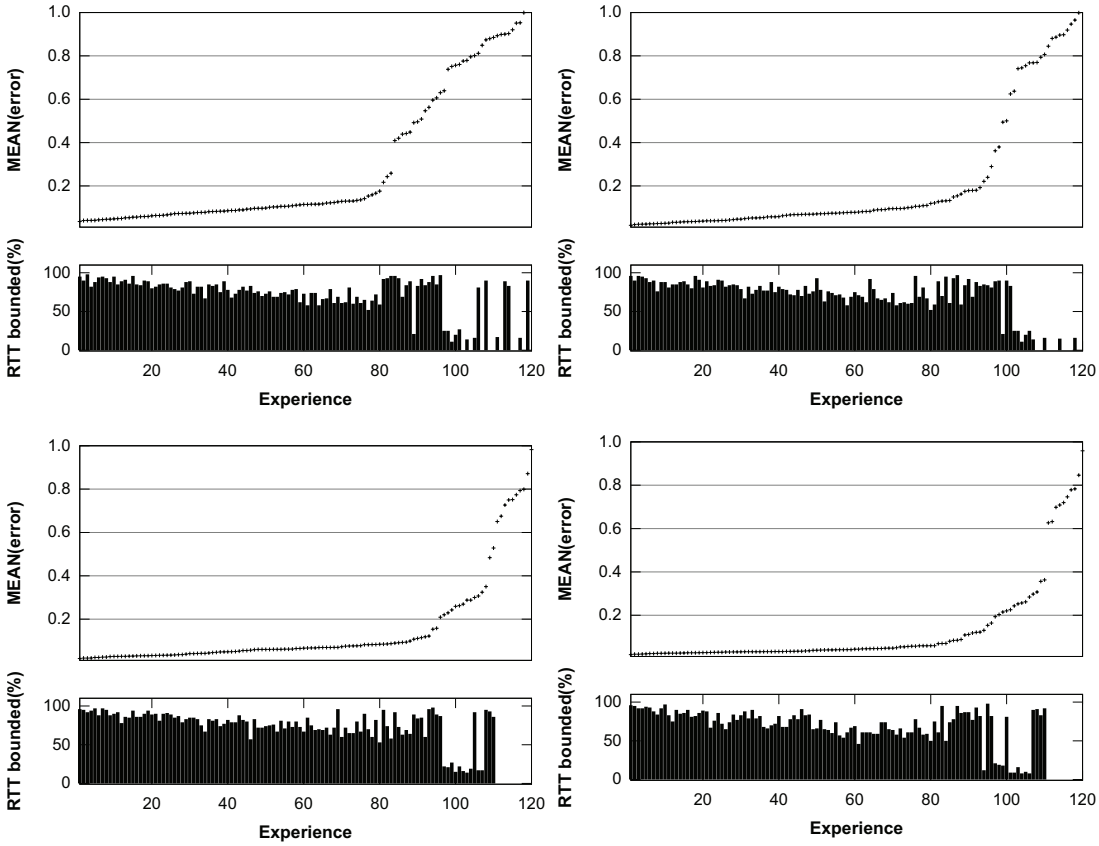


Figure 6.23: Mean error using after several improvements using random topologies. Although the mean error is improved, the mean error improvements are less significant.

- model with α and β.
- adds the γ parameters to best fit GTNetS sharing.
- the model that includes all improvements presented here plus full-duplex support.

In Figure 6.22, results of maximum error for each experiment are presented for the original model and the three different proposed improvements. All graphs are sorted by maximum error and the percentage of flows which were limited by latency in the model are presented on the bottom of each graph. The first improvement is the use of α and β to adjust the basic transmission model so it best fit GTNetS values . Based on the figure we can discard that any improvement is achieved in this scenario by adjusting α and β parameters.

Nevertheless, the additional of the γ parameter reduces the maximum error significantly. This last points out that correct instantiation was necessary to achieve acceptable accuracy using Max-Min model. Following, when using the full-duplex model the maximum error is also significantly reduced. Nevertheless, in spite of good results, we can also observe based on Figure 6.22 that the maximum error with some experiments is still hard to improve.

The presented improvements were effective to reduce maximum error. Nevertheless, we would like to guarantee also that these improvements happens frequently during an experiment. To verify if this hypothesis is true we plot on Figure 6.23 the resulting mean error using the same experiments. Observing the figure, in all cases, the maximum error was under 1. Moreover, the success of γ and full-duplex is also verified. This results reinforces that a few flows are mismatched while most flows achieve good accuracy. An open question observing Figures 6.22 and 6.23 is the corelation between bad estimations and the number of flows limited by round trip time.

To verify the corelation of bad estimations and latency limited flows we tuned the platforms presented in Table 6.3 to assure that every flow is limited by bandwidth. To do so, we reduced each link bandwidth and latency dividing it by 10 guaranteeing that all flows are bandwidth bounded.

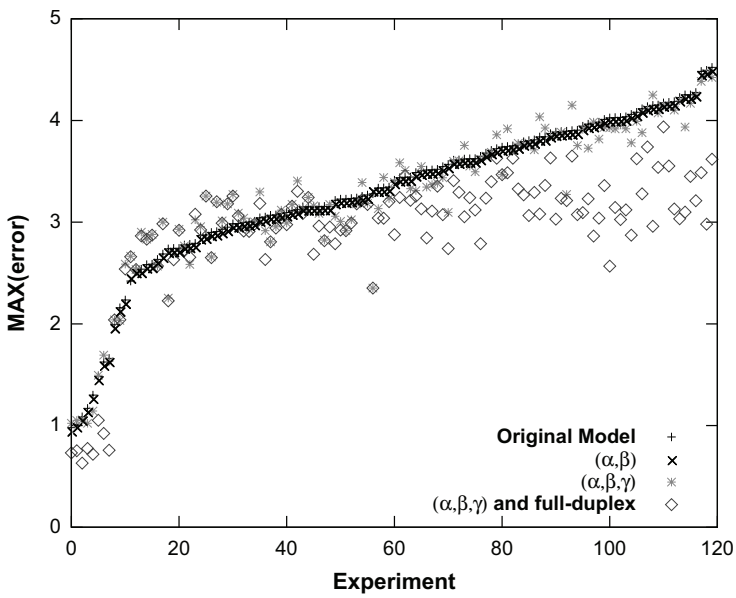


Figure 6.24: Maximum error when avoiding latency limited flows.

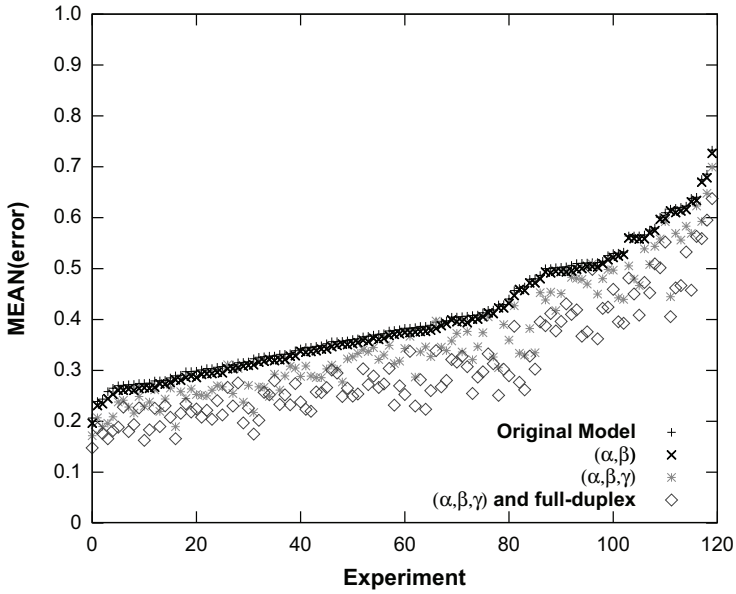


Figure 6.25: Mean error when avoiding latency limited flows.

These results, with all flows limited by the bandwidth is depicted by Figures 6.24 and 6.25. Figure 6.24, present maximum error while Figure 6.25 depict the mean error. At this time the original model and the three improvements are presented as four point styles. In this last experiment, we omitted the RTT bounded (%) bottom graph. This graph is unnecessary here once we certified that there are zero flows limited by latency. In these figures, we sorted the original model points by maximum error. The other models are presented with a different point shape respecting this order. As suspected, the graphs of Figures 6.24 and 6.25 confirm that in such cases it is harder to guarantee low maximum error.

In reality, LSDC applications deal with heterogeneity, interference and volatility. For this reason a more common scenario must involve several flows limited by latency and bandwidth. To explore a more realistic scenario we next run several experiments where more or less half of flows are limited by the round trip time whether the other half is limited by bandwidth.

The maximum and mean errors obtained with half of flows limited by bandwidth and the other half limited by latency appears in Figures 6.26 (maximum) and 6.27 (mean). Once again, we modified the original test set platform. Observing the figures we can notice that these results are similar to the previous tests where all flows were limited by bandwidth capacity. Consequently, we think that the amount of flows limited by latency is not the only factor that influence high maximum error values.

To investigate the causes of maximum error, we detail in Figure 6.28 a timeline of one experiment. This time we avoid finishing the simulation as soon as the first flow completes. Instead, we let the simulation run until the end. We present on top the SimGrid timeline with the GTNetS timeline (expected behavior) on the bottom. To compare both approaches, we draw a mark in the moment each flow finishes connecting them with a line. This test uses one experiment of the original platform configuration set.

As illustrated by Figure 6.28, in spite of a significant maximum error ($\epsilon_{max} = 2.91$) this error is neglectable in the long run. The plausible justification for that is our scale free error metric. Using the log based error, we put in evidence strong differences between flows, even if their duration is insignificant in the long run. This reinforces our choice of metric proving that it is rigorous.

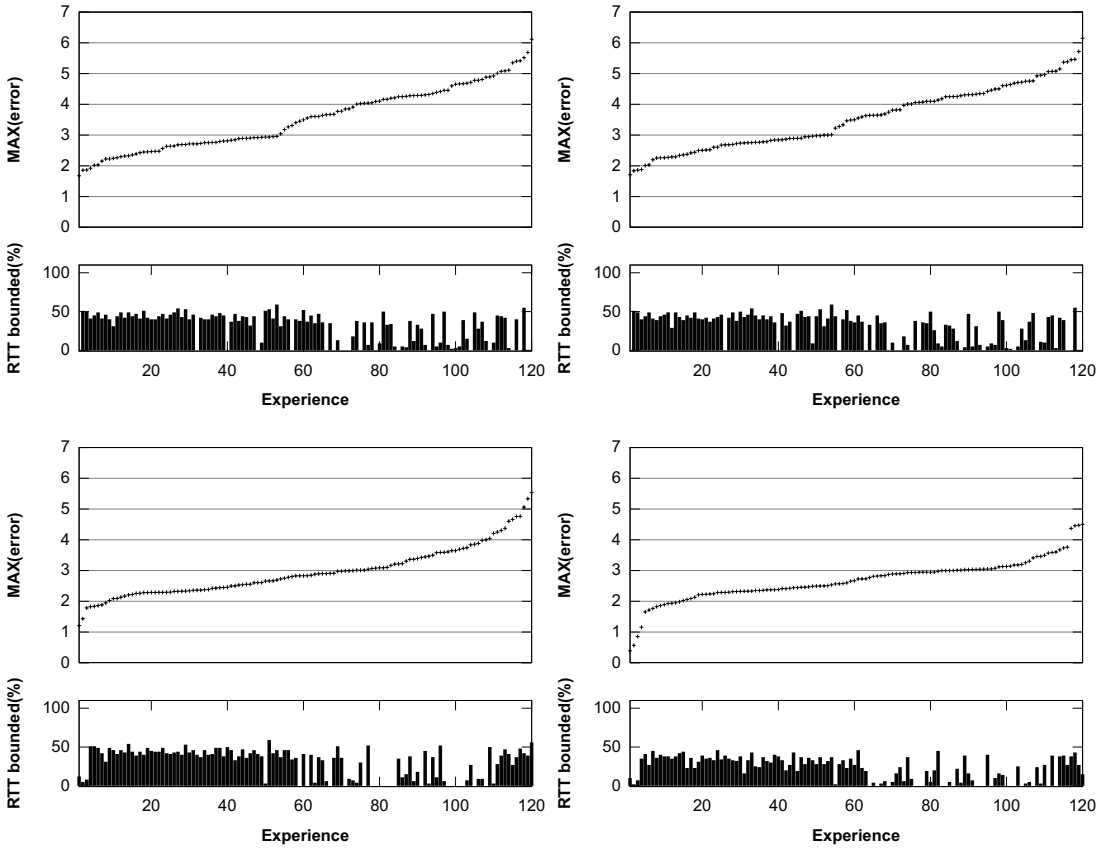


Figure 6.26: Max error when ≅ 50% of flows are limited by latency.

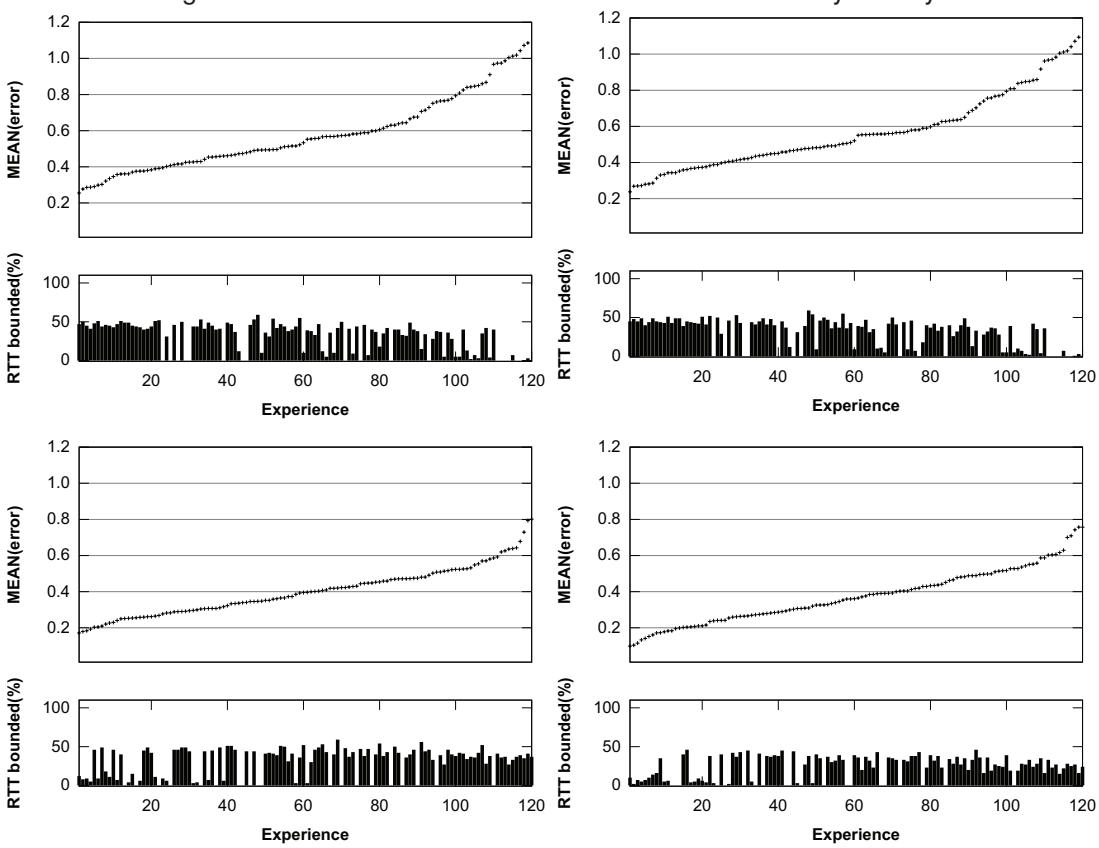


Figure 6.27: Mean error when ≅ 50% of flows are limited by latency.

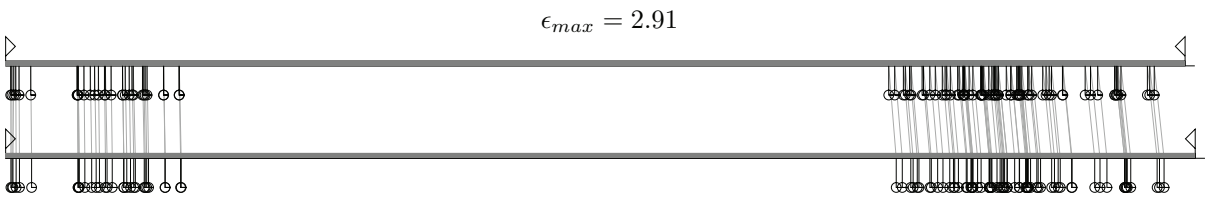


Figure 6.28: Simulation timeline indicating the moment when each flow finishes. The top timeline was obtained with GTNetS while the bottom one with SimGrid improved model. We draw a line corresponding flows in the model and GTNetS. In this case the maximum error is seen at the very beginning due to small finish time of the first finishing flow.

Nevertheless, we cannot see the same behavior of Figure 6.28 in all the experiments that have bad accuracy. In Figure 6.29 we detail experiment that has an error of 3.42. This time, the duration of the simulation is also close to the expected behavior. Nevertheless, we can see significant differences between the finish time of individual flows. In spite of our efforts, we cannot explain the possible causes for that. A possible approach to investigate further in this direction is to observe graphical representation of the platform and the error associated to each flow. This investigation is to be continued.

At this point, it is once more tempting to blame the wrong estimations of the Max-Min sharing model. Indeed, in spite of intuitive the Max-Min model was not proposed for modeling the TCP sharing behavior. A possible explanations for such bad results could be that Max-Min is not suitable to accurately model TCP and that other more appropriate models of TCP sharing, like those presented in Section 5.3, would lead to better results. We investigate this possibility in the next section.

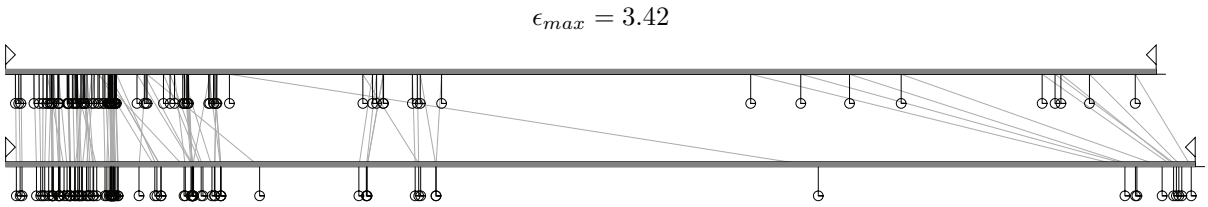


Figure 6.29: Flow termination (marks) timeline obtained with the GTNetS (top) and SimGrid improved model (bottom). Yet, in this case the maximum error is representative as measure.

We next evaluate models that propose a different sharing policy. With these models we hope to improve even more the accuracy when flows are limited by bandwidth.

6.4 Accuracy Evaluation of Duality Models for TCP/Ethernet

Low [55] proposed several models to reproduce the sharing mechanism of TCP. Those models were created based on a microscopic analysis of the TCP implementation and were explained in detail in Chapter 5. Since those models are made specifically to cope with the TCP sharing behavior, we should expect that using such models, we can improve the accuracy issues detected in the previous section.

6.4.1 Evaluation with Random Network Topologies

To present results we plot the version of improved model with the Low's Reno model. For both we present maximum errors and mean errors aside to give evidence of which one is the most accurate. To start in

Figure 6.30 one can see the maximum and mean errors for the 120 experiments comparing Low's Reno and the improved version of Max-Min. It is visible that the maximum of Max-Min improved model is very often a lower bound of Low's Reno. However, due to the frequency of occurrence of maximum error, the final improvement on the mean error is not as much significant.

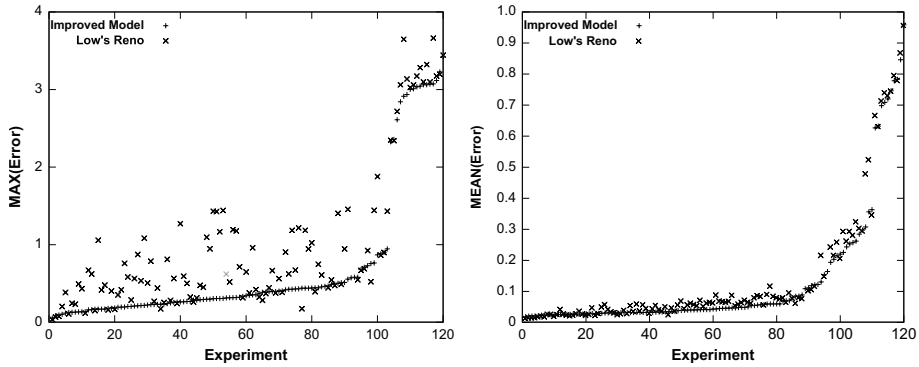


Figure 6.30: Max error (left) and mean error (right) compared to Low's Reno model with non-modified test set.

For the sake of completeness, we present in Figures 6.31 and 6.32, the results of maximum and mean errors when all flows step on the maximum bandwidth capacity and when $\cong$ 50% of flows are limited by latency. Indeed, the evidence points that the maximum error is significantly improved with the full-duplex feature and that the linear model shape itself is unrelated to bad estimations.

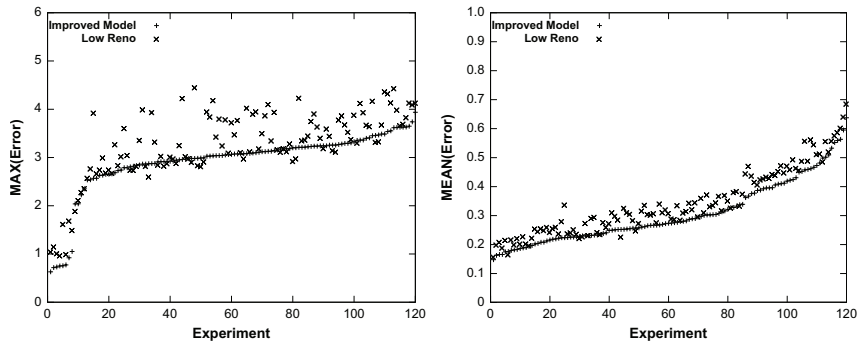


Figure 6.31: Max error (left) and mean error (right) compared to Low's Reno model when bounded by bandwidth.

To improve even more the sharing model available in SimGrid we tried some models that were specifically conceived to model the TCP sharing mechanism. Nevertheless, after comparing those models with the improved Max-Min, we realized they were unable to provide better estimations. This results from the inability of such models to account for the important phenomenon of TCP protocol highlighted in Section 6.3.3: ack compression. Indeed, except for this one, all improvements presented earlier can be used on both models. Given that most part of improvements are independent of the sharing mechanism the introduction of α and β, γ are straightforward into both models. Yet, as we are going to explain, adding control traffic in opposite direction is an improvement that only the Max-Min model can benefit.

In Figure 6.33 we plot a utility graph for the example of Section 6.3.3 (Figure 6.13) where two uploads and one download are concurrently transmitting through a full-duplex link. This simple example is enough to show that neglecting ack compression leads to bad estimation. In this particular case, the

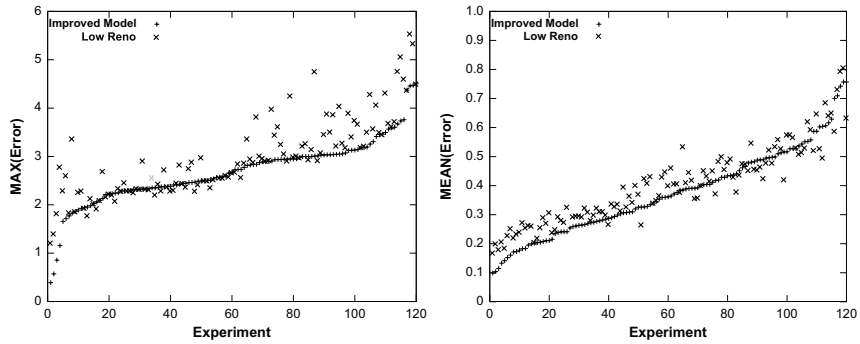


Figure 6.32: Max error (left) and mean error (right) compared to Low's Reno model when ≅ 50% of flows are limited by round trip time.

result of the sharing should be $\rho_1 \cong \rho_2 \cong \rho_3 \cong \frac{C}{2}$ (this example is detailed in Figure 6.19). To illustrate why Max-Min is always better than the Reno model we plot, in Figure 6.33, several possible values of the objective function. We can safely assume that $\rho_2 = \rho_3$, hence the corresponding objective function is $f(\rho_1, \rho_2) = \log(\rho_1) + 2\log(\rho_2)$. Due to the shape of the objective function instead of obtained the expected answer ($\rho_1 \cong \rho_2 \cong \rho_3 \cong \frac{C}{2}$) we obtain almost the same value as when ignoring control traffic, i.e., $\rho_1 \cong C$, $\rho_2 \cong \rho_3 \cong \frac{C}{2}$. This explains why, such models, although specifically designed from the microscopic behavior of TCP, are actually unable to truely account for the macroscopic behavior of TCP. The fundamental reason behind this is that such modeling assumes that acknowledgements are not really part of the trafic and are not delayed by cross-trafic, which is not true in practice. Surprisingly, the Max-Min model, after some adjustments, reveals a rather good alternative.

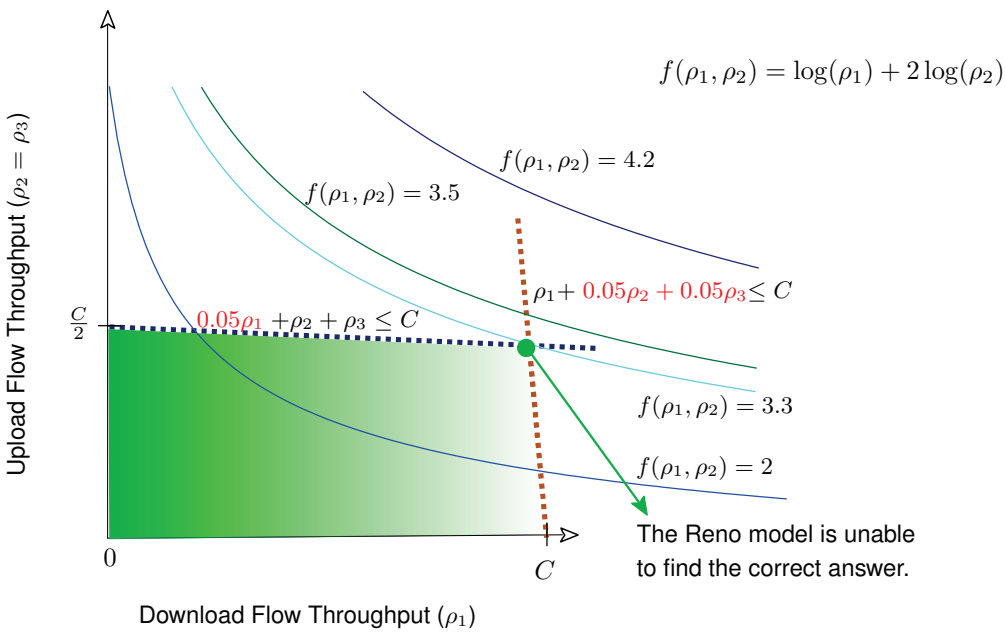


Figure 6.33: Representation of Reno model for a simple full-duplex scenario of one download and two uploads. Adding control traffic is helpless to Reno model since it still choose the Pareto optimal point in the corner.

6.5 Conclusion

This develops an exhaustive investigation study of models accuracy to reinforce the use of simulation in LSDC research. To do so we presented a series of accuracy results with improvements when we were able to identify the cause for bad results. Afterwards, we used random platforms to verify if improvements were still significant in complicated test scenarios. The results point two main important conclusions: (i) maximum errors can be significantly reduced if the models are well instantiated and (ii) fluid models are still inaccurate in some cases. We feel that some synchronization phenomena due to TCP synchronization mechanisms cannot be accounted by such simple fluid models. Indeed, the network resources are not shared in a fluid manner and mutual exclusion mechanisms are used to guarantee consistent access to resources. Unfortunately, we are unable to predict this kind of synchronization phenomena and sometimes they may significantly impact on the quality of the fluid prediction.

Nevertheless, fluid approaches present acceptable accuracy when the goal is evaluating and estimating performance of LSDC. Indeed, fine grain solution are unsuitable in our research community because they are slow. Moreover, fine grain approaches, such as packet-level simulators, do not scale well enough to satisfy LSDC research requirements. Consequently, we think that our proposal is a reasonnable tradeof in our context between speed/scalability and accuracy. Besides, this work also present an innovative framework for the accuracy evaluation of CPU and network models adapted to Large Scale Distributed Computing. This framework, is one of the main contributions of this thesis representing one step forward to overcome the lack of exhaustive accuracy studies in this domain.

Doing accuracy evaluation was a delicate and challenging subject for a series of reasons. First, it was innovative since that many accuracy studies are too simplistic it was not possible to rely on a well known methodology. Second, the order in which problems were detected and improvements were proposed does not follow the same logical order as presented here. To illustrate, the full-duplex feature was the last improvement made after we realizing that Low's model were unable to improve accuracy. To provide a better instantiation of models we proceeded with the models much like a programmer does when debugging source code but with the difference that we had to develop many of our debugging tools (R scripts, triva [76]). Third, there were many experiments with a wide variety of parameters which took a very large amount of time, even with the help of Grid5000 clusters. Last, we want to provide results in a repeatable manner, so the scripts and data corresponding to every experiment presented here are available for download [71] along with the SimGrid and GTNetS versions used during these tests.

Chapter 7

Simulation Speed Improvements and Results

An aspect of Large-Scale Distributed Computing (LSDC) simulations is the possibility to extrapolate reality. For instance, one can use more machines, one can increase processing power and even imagine an hypothetical scenario (like a network link to the moon). Particularly, in LSDC systems speed and scalability are a limiting factor when conducting simulation experiments. In LSDC specialists are interested in simulation because debugging and developing in such environments is hard and time consuming.

In the last chapter, we presented that network and CPU models implemented within SimGrid are accurate. However, the speed and scalability of simulations are an important factor that cannot be neglected on LSDC. SimGrid models are indeed very accurate, however, to prove the usability of SimGrid we still need to check for scalability and speed. In this chapter, we evaluate the scalability and speed of SimGrid's internal models presented in Chapter 5.

7.1 Performance Evaluation Methodology

In this chapter we are interested to measure the execution time of simulations. Measuring time it is possible to understand the trends of simulation to give an idea of how much is earned (in speed and scalability) and how much is spent (in accuracy). Hereafter, to assure stability of presented results, we conduct several measures always being careful when presenting average results. To guarantee that differences are statistically significant we plot the confidence interval enclosing each average point. Unfortunately, many times the confidence interval is too small to be perceived.

As presented before, we have several models implemented in SimGrid. However, as we saw in the previous chapter, the best accuracy is obtained with the Max-Min model when well instantiated. Hereafter, we use this best model explained in Chapter 6. We use the full-duplex model for network and the Max-Min for CPU. As before, we point out performance flaws proposing improvements of current model when possible.

An important point before starting to present results is to clarify the methodology approach used. The next items present scalability, workload, performance metrics and their specific meaning throughout this study.

7.1.1 Scalability Parameters

We are particularly interested to scale in a way that several hosts and links are simulated simultaneously. SimGrid usage may vary from using a few computing nodes in a full connected network (cluster) to thousands of computing nodes in a slightly coupled network (P2P). In both cases users would eventually look to stress the simulator exploiting at maximum the amount of resources available.

Two parameters determine the scalability of SimGrid simulations: computing hosts and network links. Hosts are single processing computing units connected through the network; Links are communication pipes connecting two or more hosts. In LSDC we look forward to have as many links and hosts as possible.

7.1.2 Workload Parameters

Typically, in a SimGrid usage we stress the number of hosts and communication. Those parameters are the bottleneck of the system since the complexity of Max-Min is direct affected by the number of actions inside the simulator. So, increasing the number of simulation actions, we increase the time to compute network events and simulation execution time. Hence, to stress the simulation parameters we focus on two parameters:

- **tasks** : amount of computational actions simultaneously active in the Max-Min system.
- **flows** : amount of communication actions simultaneously active in the Max-Min system.

Both parameters are important because they stress the usage of the platform. By consequence, this is also a limiting factor of the number of hosts that can be simulated at time. So, during this chapter our focus is to vary those two parameters measuring the simulation response time.

7.1.3 Speed Measures

To evaluate performance we introduce the concept of **simulation speed**. In LSDC systems, simulation is attractive because performance evaluation is slow. Generally, those applications are time consuming and to be able to delivery results in due time it is often needed to use a simulator. At this point, we distinguish two time measures:

- **runtime**: this is the real time spend during simulation. So, if a simulation takes 3 seconds to end, the runtime is 3 seconds. In LSDC, a long runtime has to be avoided.
- **simtime**: is the time inside the simulator. Representing, the models estimated time.

In LSDC, one is interested to have simtime many times grater that runtime. Somehow, this idea is translated to the distance between runtime and simtime where the longer the distance the faster the simulator will be. To measure this distance, as before, we need to respect the metric space constraints. In this chapter, we define the simulation speedup as the distance shown in Equation (7.1). Details of why using this metric instead of others were already discussed previously.

$$\log(T_{sim}) - \log(T_{ref}) \tag{7.1}$$

In the next section we present a series of simulation speed and scalability metrics. Our goal here is to promote fast simulations proposing a series of speed improvements when the simulation present some flaws.

7.2 Speed Evaluation of CPU Models

In SimGrid, the Max-Min solver is the model used for network and CPU. Consequently, the CPU model performance is direct related to the Max-Min system (LMM) solver. We next evaluate the performance of CPU models.

7.2.1 Naïve Implementation

In practice the solution is implemented inside the simulation kernel using the same LMM system that the network model uses. However, there is an important difference between CPU and network model. Computational resources are shared by tasks but a task cannot use multiple CPU resources simultaneously. By definition, a task is a single work unit, and as so when executed it produces one action that is attached to **one** CPU resource. So, this is a key difference of network resources and CPU resources. In network, one action normally uses multiple links. This implies that the Max-Min model can be used to solve both resources sharing. However, it is certain that in the CPU context resources are always independent between them. For network resources this is not always the case.

In the CPU model, tasks are translated to actions when host resources request to execute that task. This is defined by the *remaining amount* of work (in flops) to do. This amount of work to do is initialized upon creation with the task computational size. Following, at each simulation event (each time the simulation clock advances) this value is updated. To update this value we multiply the CPU working rate by the variation of time. In the simulation kernel each CPU action (or task) is translated to a variable in the LMM system and hence it is connected to a particular host work rate constraint. Resource consumption varies over time and depends also on the number of tasks sharing the same host. In a naïve implementation the resulting of the share will be the division of the available processing power by the number of actions simultaneously executing as it appears in Equation (7.2).

$$T(P_{rate}, P_n, W) = \frac{W}{\frac{P_{rate}}{P_n}} \tag{7.2}$$

To compute the next task completion, we compute a new solution of the LMM system. The complexity of this solution is at least the number of running actions on the simulator, i.e., $\Theta(|\text{actions}|)$. Following, it is needed to update all remaining work, before advance the simulated time. To do so, we have once again to reiterate all CPU actions, i.e., $\Theta(|\text{actions}|)$.

This first described implementation is naïve because even when actions are completely independent the fact of finishing a task will imply in a complete invalidation of the system. So, no result of the previous solution is reused in a new simulation iteration. At this point, a first optimization proposed is to avoid computing again the solution for the entire system if no changes were detected (no actions were created or finished). In this last case the complexity will be only to update the remaining of tasks using the result obtained before. Considering this last optimization, the complexity of LMM system at each simulation iteration is as follows:

- Create tasks, constant algorithmic complexity, $\Theta(1)$;
- Estimate time to finish next action, $\Theta(|actions|)$ (lmm_solve);
- Update the remaining work to do of active actions, $\Theta(|actions|)$ (lmm_update_actions_state);
- Remove finished tasks from the system, $\Theta(1)$;

As one can see the idea of recomputing the whole system each time even when there is only a local change is inefficient for the CPU model. In next section we propose a best way of computing only parts of the system that have been modified to improve performance.

7.2.2 Reducing Cost per Iteration

Updating only modified portions of the system seems to be appropriated because algorithmic complexity of estimate and update actions is significant. Most part of the time the simulation events do not affect the entire system. Like this, we can reuse previous solutions reducing the complexity of the computational intensive parts. However, this involves flagging portions of code and the complexity of removing tasks that before was constant must now be taken into account.

Partial Invalidation

Therefore, each time an action finishes we need to invalidate only actions that are affected by one resource shared by t hat action, i.e., actions that share the same resource. Due to that the complexity to remove an action of the LMM system is increased from $\Theta(1)$ to $O\|actions\|$. In the specific case of the CPU model this happens only if all action run on the same host. Though, in a typical distributed computing simulation, we are interested when several hosts are simultaneously processing. In spite of assuming that actions are not highly coupled is a particular characteristics of CPU models this optimization can also lead to better performance for network models.

Lazy Management

With partial invalidation the portions of the system that have not changed do not need to be recomputed anymore. Like that, only parts that have changed are recomputed reducing significantly each iteration step. Nevertheless, every time at least one action has changed, we need to compare them all to find the next action to end.

At each simulation step, besides calculating resource sharing SURF needs to search for the next action to finish. Now, searching for the next action to complete is done linearly comparing each action completion time. This intuitive algorithm has complexity $\Theta(actions)$ which represents an important slice of the overall simulation time.

As one can notice, this algorithm does not profit from previous results and even when only 1 task over 500 changes we need to compare all tasks to discover the next action to end. One possible solution, to improve performance here, is to sort actions by completion time. Therefore, keeping track of the comparisons made in previous simulation steps reducing the algorithmic complexity to find the next action to finish.

Here we introduce lazy action management which is a way of profit from previous actions' comparison. To do so we use a heap to sort actions. A heap is a binomial three where a father value is always

greater than its sons. Like that the greatest element will be kept in the root node of the three. With the help of a heap we can obtain direct the next action to end. So, instead of comparing every element in the array one just need to pop out the root element. Hence, the algorithmic complexity of finding the new action to finish reduces from $\Theta(|actions|)$ to $\Theta(1)$.

Although, lazy action management has the drawback of increasing the complexity to add and remove actions. When a task in inserted or removed it is necessary to rearrange the heap, what was not needed before, so the action remove routine increases in complexity from $\Theta(1)$ to $\Theta(\log(|actions|))$. However, if the system does not change one may reuse results and search for the next action to finish. This reduce when computing the next action to finish from $\Theta(|actions|)$ to $\Theta(1)$.

Using lazy action management we lost performance in task deletion and addition increasing the algorithmic complexity from $\Theta(1)$ to $\Theta(\log(|actions|))$. However, since we reduced the complexity of finding the next finish action from $\Theta(|action|)$ to $\Theta(1)$ we reduced the complexity of the main simulation loop considerably, from $\Theta(|actions|)$ to $\Theta(\log(|actions|))$. Moreover, for CPU resources, that are completely independent, this optimization is mostly certain to be useful since there is no need to rearrange the heap at each iteration. Therefore, this approach aims to improve the CPU model performance, although, it is expected to speed also network model.

Integrating Availability Traces

In SimGrid, one has the option to use traces to shape the availability of resources during simulation. This type of traces are useful to model resources that have variable availability over time. To enable the use of availability traces in SimGrid, one attaches to each resource a text file describing the percentage of available resource over time. We illustrated one availability trace file to shape the CPU power of a host in Figure 7.1[1].

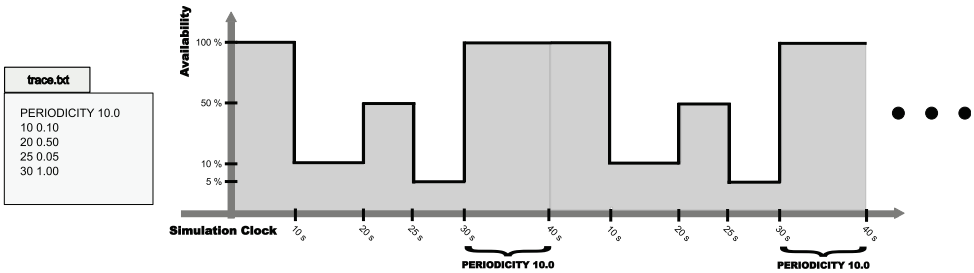


Figure 7.1: SimGrid's availability trace example revisited.

As we presented before, in Chapter 4, the simulation kernel implements availability traces introducing a dummy action in SURF each time a trace event is found. With the difference that trace actions do not imply in resource usage, but instead, those actions will change the constraints associated to each resource. Due to that fact, when a resource is affected by a trace event the constraints associated to the resource change and by consequence all application actions attached to that resource must be recalculated. In other words, each trace event will be translated to a simulation iteration. However, an iteration that is caused by a trace event does not represent an application state change. Like this, such an iteration is not perceived by the user as useful computation. In a worst case scenario, when running an application that stress the usage of traces, this way of implementing this feature impacts severely in the simulation speed.

[1] This example was already presented in Chapter 4, Figure 4.4, to avoid readability issues we use the same example here.

Using this approach impacts in the simulation speed mainly because each time the simulator has a new trace event to compute the simulator kernel will need to stop and recompute the Max-Min system. As we presented before, computing the Max-Min system is computational intensive. Therefore, this improvement has a significant effect in the simulator performance.

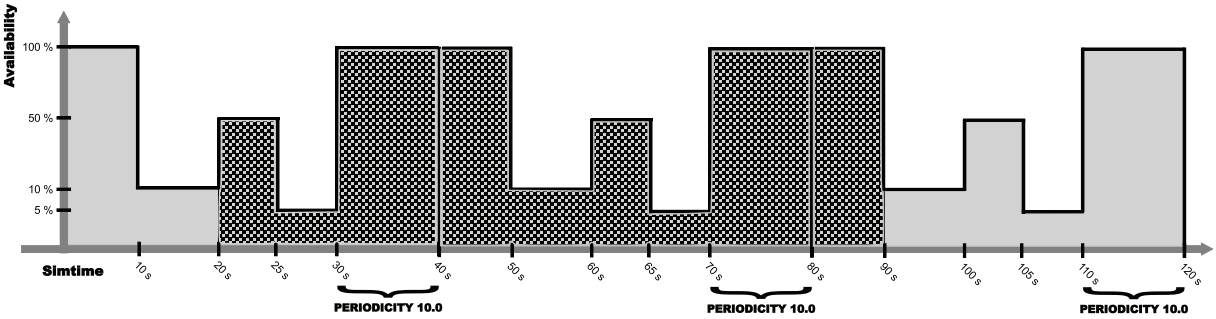


Figure 7.2: Integrating availability trace histograms one is able to advance the simulation to useful actions.

In Figure 7.1 the histogram is cyclic when it ends the same histogram is reiterated in a loop after the periodicity time. In this histogram we can observe that the surface of a given time interval is equivalent to the amount of processing power produced in that period by the host in question. So, an alternate approach to minimize the number of simulation kernel iterations is to determine when an action is about to finish based on integrating the availability histogram. Since we know that CPU actions share the resource uniformly this approach can be used even when the CPU is shared by a set of actions.

To exemplify this idea of integrating trace histograms let us imagine that the host in question have 20 MFlops of processing power and that its processing power availability is modeled by **trace.txt**, the trace example of Figure 7.1. Now, if we had one action to consume 930 MFlops and that this action starts at 20 seconds (simulation time) we can predict the finishing time of that action finding where the area on the histogram is 930 MFlops. The answer is illustrated by Figure 7.2 where the area of the histogram is computed to foresee that the action will end at time 90s.

A more efficient method for integrating trace histograms may be achieved relying on this cyclic characteristic. To do so, we find the processing power delivered by the host in question from the action starting point (20 seconds) until the end of the first histogram cycle. In our example, the area from 20 seconds up to 40 seconds (the end of the first histogram cycle) is 255. In other words, i.e., the power delivered during interval 20 up to 25 seconds plus the power delivered during interval 25 up to 30 seconds plus the power delivered from 30 up to 40 seconds ($(20 \times 0.5 \times 5) + (20 \times 0.05 \times 5) + (20 \times 1 \times 10)$). Like that, we can state that we need two more histogram cycles at least to complete 930 MFlops since the total area of an entire cycle is 475. Yet, adding the 255 of power achieved in the first cycle plus the power delivered during one entire cycle we obtain $475 + 255 = 730$ which is not enough to finish computing the action. However, we can also observe at this point that the less than one cycle is needed to finish computing the action, since $730 + 475 = 1205$ and that 1205 is greater than 930. Finally, following this approach one is able integrate the histogram.

This algorithm to foresee finish time of trace events has its complexity associated to the number of events, $O(|trace\ events|)$. This substantially reduces the number of simulation iterations since trace events are not modeled as actions the simulator can focus on the actions that are meaningful. Like this, the simulation kernel number of interactions were reduced from $O(|actions| + |trace\ events|)$ to

$O(|actions|)$. Since the number of simulation iterations is a computing intensive part this trace integration will significantly improve simulation speed. However, the complexity of computing the time to finish one action increased from $O(1)$ to $O(|trace\ events|)$ with trace events.

One important aspect of this trace integration approach is that it is not suitable to improve the performance of the simulator for network resources. This feature is possible based on the assumption that CPU resources are uniformly shared. In CPU each action receives an equal slice of the available processing power and like this it is even possible to use this approach when several actions are running on the same processor. We can find beforehand which action finishes first. In this case, the first action to finish (the one that has less work to) will finish first and we can do the same procedure dividing the processing power of the machine by the work to do.

Throughout this section we presented a series of improvements available in the SimGrid kernel. Those improvements are mostly made to provide fast simulations while they do not touch to the system precision. As we saw, sometimes the improvements are expected to improve performance, however in specific usage cases, some of the improvements can decrease the simulation performance. To evaluate when the proposed improvements successfully enhance performance, we next compare them with the original CPU model.

7.2.3 Analysis

The timings presented here were obtained using an AMD Opteron 248 Dual Core (2.2 GHz) with 1MB of L2 cache and 2 GB of RAM. Experiments were performed at least 10 times each and the 95% confidence interval based on Student's distribution is plotted on all graphs (even though they are generally so small that they can hardly be seen).

The example we used is part of SimBOINC [29] simulator. SimBOINC was created to model the BOINC system. BOINC [5] is a versatile volunteer computing system providing basic midleware to users that want to provide volunteer computing projects. In the context, the SETIHome [6] project gained importance over the years and is the goal of our experiments. To evaluate this projects we simulate the system running computational tasks of size linearly sampled between 0 and 8.10^{12} MFlops. This simulator uses a constant model for network to stress the usage of computational resources. Moreover, the BOINC project has a set of failure traces available thanks to the Failure Trace Archive (FTA) [49] which comes at hand to test the trace integration mechanism.

At first, to evaluate if the trace integration slows down the simulations we compare the improvements without using traces in Figure 7.3. In the figure we present the simulation time as a function of the number of hosts using logarithmic scales. As we can observe the time needed to simulate 2560 hosts without improvements is more than 10000 seconds (more than 2 hours). Doubling the number of hosts this simulation would take more than one day. For this reason simulations with more than 5120 hosts are intractable without improvements and is not presented in the graph. Similar, with partial invalidation the simulation time grows too fast to be tractable with more than 2560 hosts. However, we can clearly see that the lazy action management improves the simulator performance significantly making it possible to simulate more than 10000 in less than 2 minutes. Moreover, we see that the trace integration feature does not affect the simulator performance as expected since this example do not use trace files.

To evaluate the effectiveness with trace integration we depict in Figure 7.4 the results using one trace file per host. So, the improvements were effective because with trace integration we significantly reduce simulation response time.

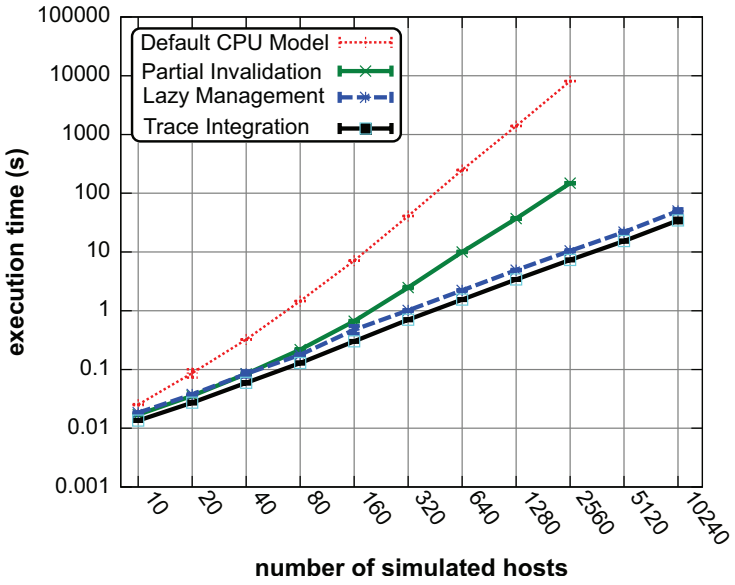


Figure 7.3: Simulation time as a function of number of hosts in logarithmic scale **without using trace files**. Lazy action management improves the simulator speed significantly and trace integration does not affect the system performance without availability traces.

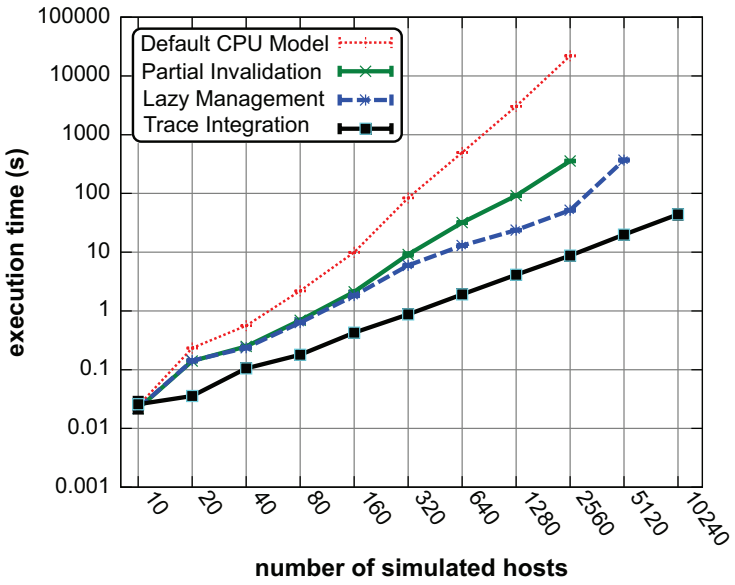


Figure 7.4: Simulation time as a function of number of hosts in logarithmic scale **using trace files**. Trace integration can significantly improve simulation speed in this case.

7.3 Speed Evaluation of Network Models

Until now, we presented a series of results that point out that SimGrid is very close to GTNetS. One of the main reasons to recall to simulations is justified by the simulations speed. So, to complete our study we verify the speed gain obtained when using simulation models. Moreover, it is needed to certify that the accuracy optimizations presented before do not hinder simulation performance. In the previous section we were particular interested in verifying the speed of the simulator in the CPU context. Now we are particular interested in the speed results of network model.

To assure that the optimizations presented before do not affect the system performance we present here a series tests comparing SimGrid available network models. With these tests we are interested in the simulator execution time, i.e., the time taken to finish a simulation. This measure is particularly interesting from an user's perspective because it gives the time needed to receive a response of the simulator.

The timings presented here, as before, 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. However, very often the confidence intervals are too small to be perceived with naked eye.

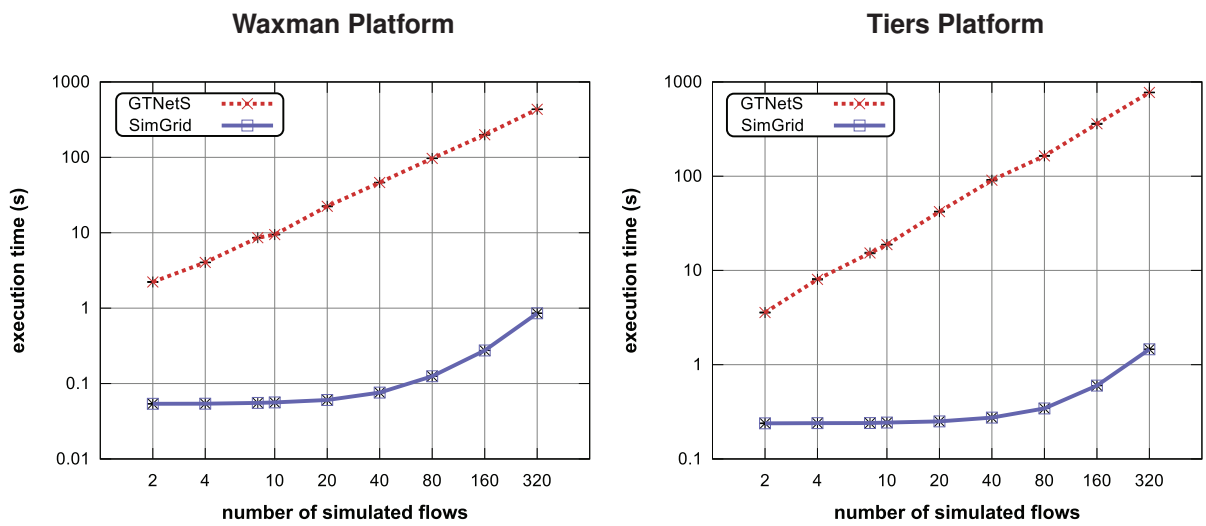


Figure 7.5: SimGrid versus GTNetS response time when varying the number of simulated flows for two 50-node platforms Waxman (left) and Tiers (Right). Notice that GTNetS is much slower than SimGrid in spite of results being very close as shown in Table 7.1.

7.3.1 SimGrid vs. GTNetS

To start, we compare here the SimGrid model with GTNetS. Since GTNetS uses fine grain simulation the GTNetS model is expected to be much slower than SimGrid's analytic models. To evaluate if this is indeed true. Even after the accuracy improvements proposed in the previous chapter. We compare the SimGrid full-duplex model against GTNetS using a simple application. This application creates several

flows distributed over the platform and wait for all of them to finish timing simulated time and execution time at termination.

To start evaluating SimGrid versus GTNetS we vary the number of concurrent flows using two platforms as depicted by Figure 7.5. When varying the number of flows simultaneously transmitting we directly increase the complexity inside the simulator, increasing the amount of variables inside the Max-Min system. In this figure we have SimGrid versus GTNetS execution times as a function of the number of simultaneous flows. In these tests, we used two platforms, on the right side of Figure 7.5, we present the results obtained with a 50-node platform that follows the **waxman** model. On the left side of Figure 7.5 we present results with a **Tiers** platform. As we can observe, SimGrid is indeed many times faster than GTNetS.

To verify that SimGrid simulation completion time are reliable we present, in Table 7.1, the simulation duration obtained with Tiers and waxman platforms in left and right tables. Looking at these data one is able to precisely measure the gain in speed obtained with the simulation helping the analyzes of Figure 7.5. In the table one can see that with the Tiers platform simulating 320 flows, GTNetS takes more than 7 minutes while SimGrid takes less than 1 second under the same settings. This result indicates that SimGrid network model is many times faster than GTNetS. Moreover, looking at Table 7.1 we can verify that the simulated time obtained with SimGrid and GTNetS are very close unless when contention of asymmetric flows is present[2]. This results reiterate the conclusions stated before, when the contention index of bidirectional traffic is near 10. In these settings SimGrid's network macroscopic model is more willing to be inaccurate.

Table 7.1: The execution time and simulated time with GTNetS and SimGrid when varying the **number of flows** in a Waxman platform (left) and a Tiers platform (right).

	Execution Time (s)		Simulated Time (s)		Maximum Contention
Flows	GTNetS	SimGrid	GTNetS	SimGrid	$\max\lvert Df - Uf\rvert$
2	2.22	0.05	1.17	1.13	1
4	4.04	0.05	1.26	1.22	1
8	8.60	0.06	1.72	1.70	2
10	9.45	0.06	1.86	1.84	2
20	22.44	0.06	1.88	1.83	4
40	46.28	0.08	3.64	3.57	4
80	96.77	0.13	4.36	4.30	6
160	199.18	0.27	8.83	8.72	6
320	433.87	0.85	38.80	10.70	9

	Execution Time (s)		Simulated Time (s)		Maximum Contention
Flows	GTNetS	SimGrid	GTNetS	SimGrid	$\max\lvert Df - Uf\rvert$
2	3.58	0.24	9.76	9.49	1
4	8.05	0.24	9.67	9.38	2
8	15.33	0.24	8.90	8.65	2
10	18.82	0.24	6.86	6.17	5
20	42.05	0.25	8.67	8.59	2
40	90.69	0.27	23.97	23.31	6
80	164.36	0.34	33.42	24.32	11
160	359.69	0.60	88.89	38.05	16
320	773.18	1.46	164.45	98.84	24

As discussed later, one main advantage of the macroscopic model in SimGrid is to be independent of message size. An unanswered question after the presented results is if indeed SimGrid's performance is independent of the size of messages. To test the performance of SimGrid versus GTNetS when varying the number of flows we plot the performance of GTNetS and SimGrid as a function of message size in Figure 7.7. These results are numerically detailed in Table 7.2. Looking at these results, we confirm that SimGrid simulations are independent of message size. Therefore, SimGrid is much faster than GTNetS. Being independent of message size and faster than microscopic simulations the scalability of SimGrid's macroscopic model indicate that this approach is the more adaptable to LSDC performance evaluation.

After these tests we could reiterate accuracy conclusion, at the same time, we presented performance results of the macroscopic network model implemented in SimGrid. Nevertheless, those last tests aimed in presenting the performance gain while using a macroscopic model instead of a packet-

[2]Contention is presented here as before, the maximum of the difference of download flows and upload flows, $\max\lvert Uf - Df\rvert$.

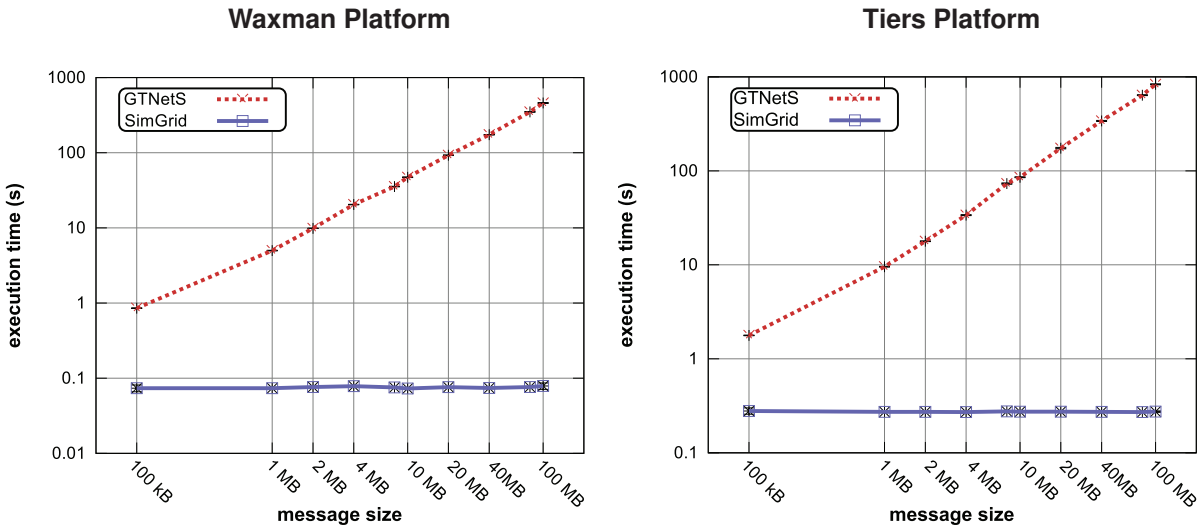


Figure 7.6: SimGrid versus GTNetS response time when varying the size of messages for two 50-node platforms Waxman (left) and Tiers (Right). Notice that GTNetS performance depends on the size of message while SimGrid performance, as expected, is independent of message sizes.

Table 7.2: The execution time and simulated time with GTNetS and SimGrid when varying **message size** in a Waxman paltform (left) and a Tiers platform (right).

Message Size	Execution Time (s) GTNetS	Execution Time (s) SimGrid	Simulated Time (s) GTNetS	Simulated Time (s) SimGrid	Maximum Contention $max\|Df - Uf\|$
100 kB	0.85	0.07	0.09	0.09	3
1 MB	4.99	0.07	0.30	0.28	3
2 MB	9.87	0.08	0.76	0.73	3
4 MB	20.43	0.08	0.89	0.81	3
8 MB	35.47	0.08	1.84	1.83	4
10 MB	47.13	0.07	3.33	2.95	4
20 MB	92.51	0.08	6.11	6.04	3
40 MB	173.63	0.07	9.01	8.91	3
80 MB	349.48	0.08	18.47	18.40	3
100 MB	458.75	0.08	26.29	25.91	4

Message Size	Execution Time (s) GTNetS	Execution Time (s) SimGrid	Simulated Time (s) GTNetS	Simulated Time (s) SimGrid	Maximum Contention $\max\|Df - Uf\|$
100 kB	1.78	0.28	0.19	0.20	5
1 MB	9.55	0.27	1.97	1.94	4
2 MB	17.82	0.27	4.18	4.25	5
4 MB	33.82	0.27	7.11	6.93	5
8 MB	73.26	0.28	10.78	10.72	6
10 MB	85.51	0.27	18.17	17.54	9
20 MB	174.50	0.27	27.15	25.45	4
40 MB	339.57	0.27	133.00	76.96	7
80 MB	638.75	0.27	147.18	142.89	6
100 MB	832.85	0.27	295.61	286.22	5

level approach. In last chapter, we presented several accuracy improvements to the network model. Until know we tested the usability of SimGrid showing that it is many times faster than other fine grain approaches with an acceptable and known accuracy limitations. However, it is still needed to verify the success of the accuracy improvements. So, next, we evaluate the performance of SimGrid models when varying the proposed accuracy improvements.

7.3.2 Comparing SimGrid's Models Performance

As we saw in the last chapter several improvements were added to the SimGrid network model to improve its accuracy. Therefore, accuracy gains were already stated and verified throughout the last chapter. Nevertheless, a question that remains open is if the models previously presented significant affect the simulator accuracy. Although that the accuracy of Reno low's models are worse than the improved macroscopic models, it is still interesting to check whether Low's models performance is comparable with the Max-Min model.

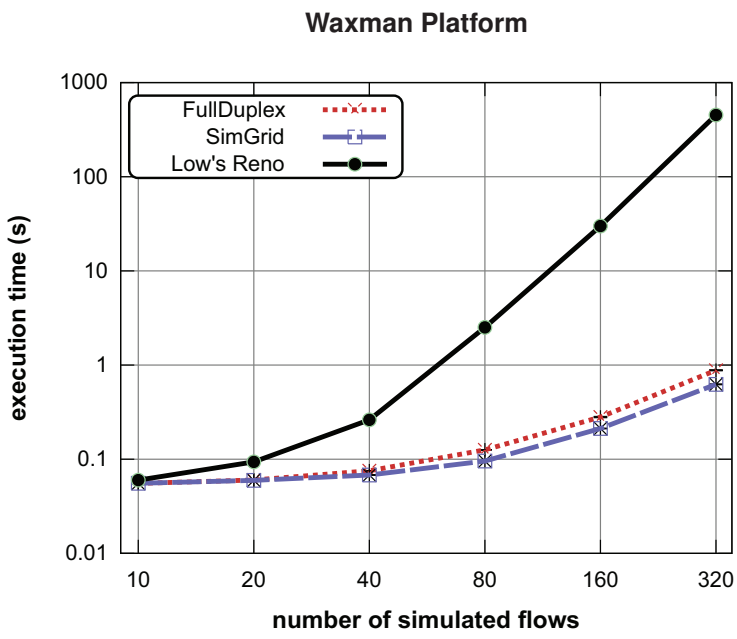


Figure 7.7: Performance of SimGrid original model versus full-duplex model and Low's Reno model performances for a **50-node's waxman platform**. The performance with full-duplex model is very close to the original model while Reno's performance is the worst one.

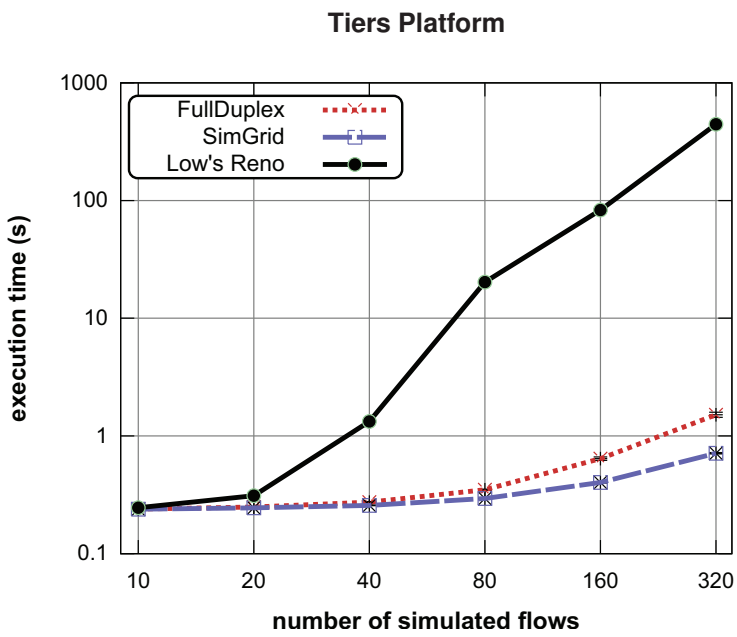


Figure 7.8: Performance of SimGrid original model versus full-duplex model and Low's Reno model performances for a **50-node's Tiers platform**. The performance with full-duplex model is very close to the original model while Reno's performance is the worst one.

In Figures 7.7 and 7.8, we present results of original SimGrid model, SimGrid improved model with full-duplex support and the Low's Reno model. We respective present results using the a waxman platform in Figure 7.7 and results with a Tiers platform in Figure 7.8. In these tests, we compared only with Reno's Low model. The reason for that is that this model is compatible in settings with the SimGrid and GTNetS. Nevertheless, the performance results obtained with Reno follow the same solving algorithm and hence lead to the close performance results. Analyzing the graph of Figure 7.8 we can state that the difference of execution time, from the original SimGrid model and the improved full-duplex one is not much significant. However, a significant difference between the two models is verified as the number of flows grows. This difference is expected since the full-duplex model doubles the amount of links and flows in the system. In Figures 7.7 and 7.8 we see that the performance of SimGrid is better than with the Reno model.

The results presented until know reinforce that the SimGrid accuracy improvements proposed in the last chapter improved significantly the simulator accuracy without hindering speed. Another point that is highlighted by these experiments is that more complex sharing models, such as those proposed by Low lead to the same accuracy issues as SimGrid original model without being faster, as so, those models are inappropriate for LSDC simulations.

In the previous chapter we proposed a series of speed improvements in the CPU model presented in SimGrid. In the next section we will explore possible solution to make the SimGrid current models speed even better.

7.3.3 Speeding SimGrid's Network Models

To improve the performance of the CPU model we proposed three optimizations:

- Partial Invalidation: consisting of invalidating partially the Max-Min system to reduce the effort at each simulator iteration;
- Lazy Management: introducing a heap so that actions are sorted by their completion time hence reducing the cost of finding the next action to finish at each iteration;
- Trace Integration: instead of inserting each trace event as a simulation event, the resource availability is integrated to reduce the number of iterations in the simulation kernel.

In practice, partial invalidation is a performance improvement that we implemented directly inside the linear Max-Min solver. For that reason, this feature can be used as it come to improve execution time of the network model. Nevertheless, lazy management, needs that the network model is changed so that it copes with the Max-Min solver and update the heap. Therefore, this feature was implemented.

In the CPU model, contrary to the network model, actions cannot use multiple resources at time. In CPU, one action uses one and only one CPU at time. As so the sharing resource part for CPU is simpler. Also, due to that characteristic the trace integration is an advantage because the sharing in each CPU is independent. However, in the network context using trace integration is more complicated since many flows may share many links. As so, when integrating traces the finish time of other flows must be still inserted as an event in the simulator. For this reason, the use of trace integration for network model is inappropriate.

To verify if the two proposed performance improvements for the network model are effective we present in Figures 7.10 and 7.10 the results obtained with a Waxman and a Tiers platform. In these

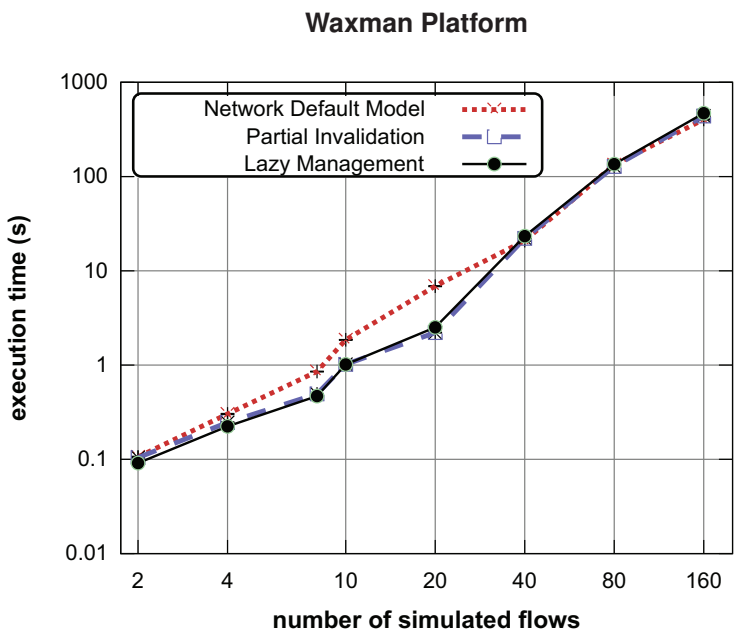


Figure 7.9: Performance of SimGrid default network model versus partial invalidation and lazy management with a **50-node's Waxman platform**. With lazy management and partial invalidation the execution time is improved for small contention scenarios.

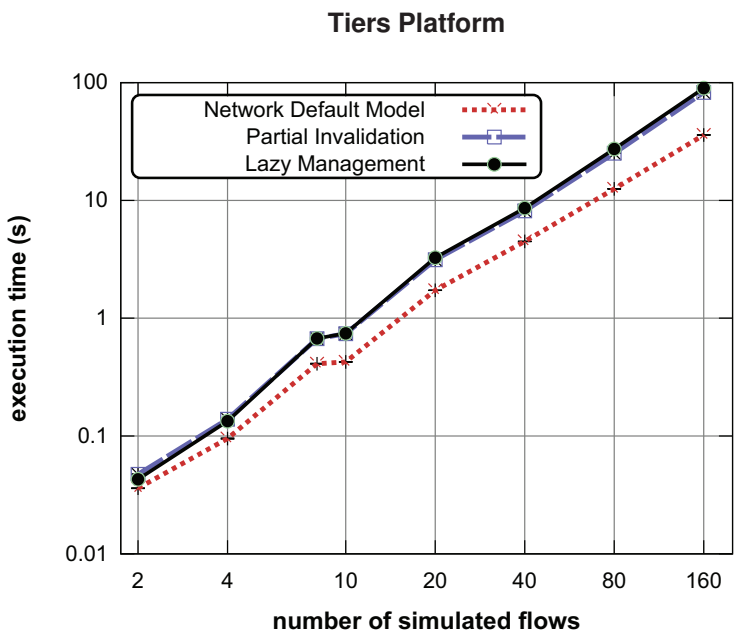


Figure 7.10: Performance of SimGrid default network model versus partial invalidation and lazy management with a **50-node's Tiers platform**. With lazy management and partial invalidation the execution time follows the same complexity .

graphs we plot the original SimGrid full-duplex model against the result when partial invalidation is active and when lazy management is active. Observing the graph with the Waxman platform in Figures 7.9 we can conclude that both optimizations are more valuable under low contention scenarios. As so, we can also observe that when the contention is too important all models follow the same complexity behavior. Nevertheless, under high contention scenarios, the case in Figure 7.10 with a Tiers platform, the non-optimized version is faster but the complexity of the algorithm remains the same.

7.4 Conclusion

In this chapter, we end the performance analysis of the models available in SimGrid. Until this point we presented in details the used models presenting evidence that: first, SimGrid models are accurate; second, SimGrid speed is many times faster than GTNetS. Even though, we presented some algorithmic enhancements for the CPU model and network models, we have verified, the optimization were much more effective in the CPU model.

Chapter 8

Conclusion

Nowadays, several distributed applications force the boundaries of computing power. To fulfill this demand, a common solution is to harness the processing power of commodity resources in a Large-Scale Distributed Computing (LSDC) system. It is particularly hard to conduct experiments in those systems due to several characteristics, such as heterogeneity, volatility and interference. Even so, specialists need to conduct performance evaluation experiments to validate alternatives comparing solutions. Therefore, experiments in such environments often rely on simulations.

Today, we find several simulators for LSDC research. In spite of claiming to address different goals, those simulators need to respect the same requirements of speed, scalability and repeatability, and model communication and computation to correctly estimate the behavior of LSDC systems. To do so, respecting the design goals, simulators must resort to models that estimate system behavior. Roughly speaking, these models trade accuracy per speed. Consequently, models need to be verified to trust in simulation results. Today we lack of accuracy studies concerning LSDC models.

During this thesis, we presented a comprehensive accuracy study of simulation models in the context of LSDC systems. We started evaluating several models using SimGrid as case study and the methodology previously studied [34]. After, we implemented new models verifying their accuracy under several settings. During the developed work, we improved accuracy incorporating various phenomena that were neglected before. Finally, we evaluated simulation speed and scalability, assuring that the proposed improvements have minimum effect in performance.

8.1 A “Historical” Perspective

Throughout this thesis our discoveries and conclusions appear in a logical order that follows a learning path. Nevertheless, before presenting our conclusion we would like to explain the chronological order of events to show the challenges of the developed work.

When we started working on this subject a preliminary study was already available [34, 35]. Their goal was to evaluate the accuracy and scalability of SimGrid network models and to propose possible reasons for bad-accuracy. In their conclusions they pointed out two main flaws in SimGrid models:

(i) They justify that because of slow-start, the Max-Min network model is expected to be accurate only if transmission are above 10 MB.

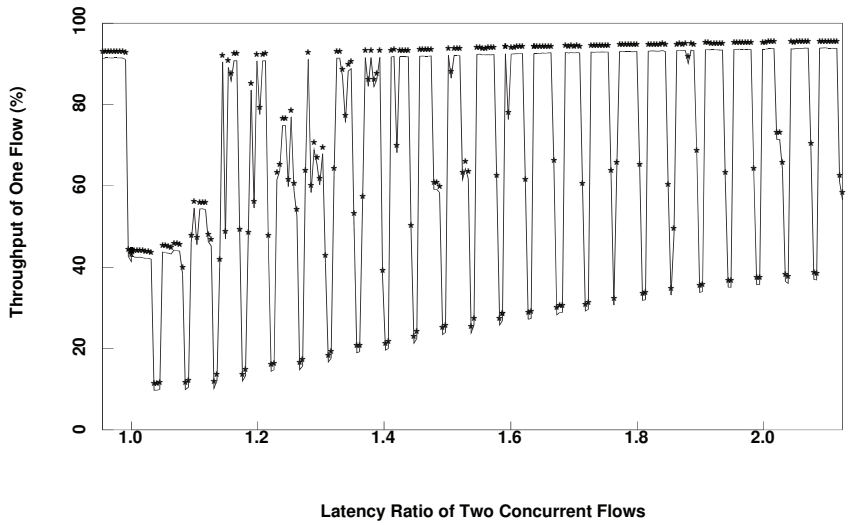


Figure 8.1: Phase effect example. The throughput of one flow, when two concurrent flows present slightly different latencies, may be significantly penalized. This happens because DropTail may privilege flows depending on the frequency that packets arrive.

(ii) The prediction of the model was show to be rather accurate in case of large latencies but had rather bad results whenever link bandwidth was a limiting factor

(iii) They explain, based on their results, that the sharing policy implemented by Max-Min was probably a bad choice to model TCP behavior.

The starting point of our research was to reproduce their results. Reproducing their experiments was challenging because, when we started, GTNetS support in SimGrid was broken due to several changes in both software. After that, we had several difficulties finding the same platforms and to assure that we were conducting similar experiments. Finally, we were able to mostly reiterate their experiments. Observing these results, we saw that slow-start indeed hinder the fluid simulation of small messages. For large messages, the sharing model indeed appeared to be as the only factor explaining strong errors.

Based on these conclusions, we next developed models that were specifically designed for TCP aiming at improving simulation accuracy. Our first step forward, in this direction, was to implement the models proposed by Low [55]. Then, while testing the accuracy of such models, we observed that changing the new sharing policy was helpless to improve accuracy. Thus, we tried to understand the reasons of such results. Since we were lost at this point, we restarted to build experiments from scratch, using simple scenarios. First, we tried a single link network. Doing so, we were able to see that the slow-start effect could be mitigated by the additional of linear coefficients in the fluid model. With that last improvements, we decreased the accepted threshold for small flows from 10 MB to above 100 KB. Next, we verified the dumbbell topology. During these tests, we found that we could improve even more the model by properly instantiating a weight. These improvements were also incorporated in the basic linear model, i.e., independently of the sharing policy.

Then, we proceeded to an evaluation of our improvements with more complex random generated platforms. Yet, the results were still very disappointing and identifying the cause was particularly hard. We focused on the most problematic situations and tried to isolate the flows with largest errors so as to simplify as much as possible problematic scenarios. This lead us to very simple situations where our

models had abnormally large errors without any sound explanations. Therefore we decided to slightly change the parameters of such situations and discovered that such situations were particularly unstable. An extremely small latency variation could result in a completely different sharing (see Figure 8.1). At this point, we discovered that the default configuration of queuing discipline of GTNetS, DropTail, was responsible for this instability. Using DropTail flows may be privileged depending on the arrival frequency of packets [32]. Due to this phenomenon, known as *phase effect*, our previous experiments performed with GTNetS had resulted in very noisy unstable behavior that rarely happen in practice and are recognized to be more a simulation artifact. Furthermore, the RED discipline, which solves this issue, is more commonly found and deployed in real networks. By reconfiguring GTNetS, we managed thus to get much more sound results. Despite its apparent simplicity, this last problem invalidates the accuracy analysis of the study presented by Fujiwara *et al.* [34, 35]. This issue had not been identified earlier because of too superficial analysis.

After solving this issue we were able to proceed again with the model verification but comparing with the RED-enabled version of GTNetS. Although results were much better, the error remained very large in some settings and we had to perform the same kind of exploration as earlier: focusing on the worst-case scenarios and simplifying them so as to try to identify the causes of such errors. In both situations, searching the reasons of maximum errors at this point was hard due to the absence of debugging tools. To understand the error origins we manually draw flows, using several colors to distinguish the error associated. This was very time consuming and difficult on big platforms. Using the debugging capabilities of SimGrid combined with recent developments of Triva [76] we found out that error was correlated to the amount of ongoing and incoming traffic. Therefore, our conclusions pointed that the sharing policy was unrelated to the accuracy issues. This lead us to realize that the unexpected behaviors were due to another neglected phenomenon: ack compression, which is explained in Section 6.3.3.

While testing Low's models, we understood that changing our sharing model for more TCP-based models would not significantly improve accuracy. At first sight these models seem appropriated. Although, our experiments point out that they cannot improve neither accuracy nor speed because they neglected some important phenomena. This modeling approach cannot correctly model TCP because it ignores ack compression and the cross traffic on ack transmission, which can hardly be incorporated in their framework. Therefore, our conclusion shows that conception and validation of models for network protocols need to be done iteratively, analyzing its accuracy while proposing the models. Although the models proposed by Low had somehow been validated, the corresponding experiments and tested scenarios were rather limited compared to our more systematic (although far from perfect or comprehensive) exploration, which probably explains why this issue had not been identified.

8.2 Contributions

Based on the results, the model that presented better accuracy, speed and scalability is still Max-Min. This is surprising since this model is intuitive and not particularly meant to correctly model TCP behavior. Through our studies, Max-Min proved to be adapted to modeling complex network phenomena. Due to its simplicity we were able to improve the model adjusting certain parameters and adding certain features necessary to improve simulation accuracy. This process reinforces a better modeling approach is to incrementally conceive models proposing, evaluating and improving them several times to reach a mature state.

The most accurate, fast, scalable models proposed in this thesis have been integrated to the SimGrid framework. The presented results point out that it is feasible to have an unified LSDC simulator framework. As presented, simulation is based on models, the quality of their results, accuracy, speed and scalability, highly depend on the models used. In essence, several LSDC have to solve the same modeling issues independently of their research objectives. For these reasons, we think it is reasonable to have an unified simulation framework such as SimGrid offering several models that are adapted to the design goals of LSDC.

When we started the present work, we were initially interested to verify the simulation validity range. Our goal was to determine a range of parameters and situations in which the simulation result is within a given bound of the correct result. Integrating such a knowledge to the simulator would have allowed to automatically raise warnings when a user was clearly using the simulator for invalid situations. As this work matured, we perceived that this goal was rather *utopic*. Even if we can state a validity range for a set of actions, the resulting state may be slightly different from reality and such approximations may propagate. The final behavior may thus be far from what reality: a small error, in a single action, may have a huge effect on the overall estimated behavior. Yet, such systems would then be somehow oversensitive (not to say chaotic) and studying them even with real measurements would be extremely problematic as well.

Anyway, even though our initial goal was certainly too ambitious to be achieved, aiming at such ambitious goal enabled us to improve our knowledge about the studied problem and forced us to never put details aside. Throughout this process, we presented several contributions:

- Highlighting that simulation for LSDC is a complex subject that is often overlooked. Thus, conducting accuracy studies of models is of extreme importance.
- Proposing for the first time a tentatively comprehensive accuracy study of CPU and network models adapted for LSDC, investigating the accuracy flaws, proposing and verifying improvements.
- Evaluating models scalability and speed, therefore guaranteeing that proposed improvements have minimal effect in performance.
- Proposing and using a model evaluation framework.
- Showing that the conception of models is a complicated process that start by observation, conception, experiments. However, evaluate the accuracy is important and cannot be neglected.
- Improving models for LSDC simulation that are already available and distributed in SimGrid.

8.3 Perspectives

The CPU model available in SimGrid remains very simple. In spite of the accuracy of such intuitive model for simplistic situations, it lacks mechanisms to correctly account for memory hierarchy affinity and sharing, which is very common today on multi-core machines. Moreover, with the appeal of Exascale computing, the use of graphical processing units is becoming popular. Another interesting resource to be incorporated in the presented models could be the support of GPU resources. Moreover, due to high cost of power supply it would certainly be interesting to have models that can be used to simultaneously estimate processing power and power consumption.

We feel that we reached a satisfying level of validity for network models that seems hard to improve any further while continuing to use fluid models. As we explained in Chapter 6, state-of-the-art sharing models are helpless to improve the kind of phenomena that impacts the most in network accuracy. In this thesis, we particularly focused on TCP over Ethernet, which are common devices used in distributed computing. Nevertheless, a possible future work will be to investigate models dealing with other network standards and hardware also commonly found in LSDC systems.

As one can see, there are still several open research points to use simulation as an effective generic solution for LSDC performance evaluation studies. Nevertheless, during this thesis we addressed a common issue to any LSDC simulator: the accuracy evaluation of network and CPU models. Our results show that the difficulty of conducting accuracy experiments lies on taking for granted allegedly obvious conclusions and trying to prove the validity of models rather than trying to disprove them.

Bibliography

[1] A Fast and Cycle-Accurate Simulator. http://facsim.snu.ac.kr/, last visited Mars 2011.

[2] A Tool for Network Experimentation and Measurement. http://www.ittc.ku.edu/netspec/, last visited Mars 2011.

[3] William Aiello, Fan Chung, and Linyuan Lu. A random graph model for massive graphs. In *32nd Annual Symposium in Theory of Computing*, pages 171–180, 2000.

[4] Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman. Loggp: Incorporating long messages into the logp model. In *7th Annual ACM Symposium on Parallel Algorithms and Architectures*, July 1995.

[5] D. P. Anderson. BOINC: a system for public-resource computing and storage. In *Proc. of the 5th IEEE/ACM Intl. Workshop on Grid Computing*, 2004.

[6] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: An Experiment in Public-Resource Computing. *Communication of ACM*, 45(11), 2002.

[7] Nazareno Andrade, Lauro Costa, Guilherme Germóglio, and Walfredo Cirne. Peer-to-peer grid computing with the ourgrid community. In *23rd Brazilian Symposium on Computer Networks (SBRC 2005) - 4th Special Tools Session*, 2005.

[8] Lars Arge, Michael T. Goodrich, Michael Nelson, and Nodari Sitchinava. Fundamental parallel algorithms for private-cache chip multiprocessors. In *12th Annual Symposium on Parallelism in Algorithms and Architectures*, June 2008.

[9] Andy Bavier, Mic Bowman, Brent Chun, David Culler, Scott Karlin, Steve Muir, Larry Peterson, Timothy Roscoe, Tammo Spalink, and Mike Wawrzoniak. Operating System Support for Planetary-Scale Services. In *First Symposium on Network Systems Design and Implementation (NSDI)*, March 2004.

[10] Olivier Beaumont, Larry Carter, Jeanne Ferrante, Arnaud Legrand, Loris Marchal, and Yves Robert. Centralized Versus Distributed Schedulers for Multiple Bag-of-Tasks Applications. *IEEE Trans. Parallel Distributed Systems*, 19(5):698–709, May 2008.

[11] 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.

[12] Dimitri P. Bertsekas. Dynamic behavior of shortest path routing algorithms for communication networks. *In IEEE Transactions on Automatic Control*, AC-27(1):60–74, 1982.

[13] Christian Bienia and Kai Li. Parsec 2.0: A new benchmark suite for chip-multiprocessors. In *5th Annual Workshop on Modeling, Benchmarking and Simulation*, June 2009.

[14] BLAS Frequently Asked Questions. http://www.netlib.org/blas/faq.html, last visited Mars 2011.

[15] BOINC Client Emulator. http://protopeer.epfl.ch/blog/, last visited Mars 2011.

[16] Rajkumar Buyya and Manzur Murshed. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, 14(13), November 2002.

[17] J. P. Buzen and U. O. Gagliardi. The evolution of virtual machine architecture. In *June 4-8, 1973, National Computer Conference and Exposition*, AFIPS '73, pages 291–299, New York, NY, USA, 1973. ACM.

[18] Kenneth Calvert, Matthew B. Doar, Ascom Nexion, Ellen W. Zegura, Georgia Tech, and Georgia Tech. Modeling internet topology. *IEEE Communications Magazine*, 35:160–163, 1997.

[19] Junwei Cao, D.J. Kerbyson, E. Papaefstathiou, and G.R. Nudd. Performance modeling of parallel and distributed computing using PACE. In *Performance, Computing, and Communications Conference. IPCCC '00*, Phoenix, AZ, USA, February 2000.

[20] Franck Cappello, Gilles Fedak, Derrick Kondo, Paul Malécot, and Ala Rezmerita. *Handbook of Research on Scalable Computing Technologies*, chapter Desktop Grids: From Volunteer Distributed Computing to High-Throughput Computing Production Platforms. IGI Global, 2009.

[21] M. Carbone and L. Rizzo. Dummynet revisited. In *SIGCOMM 2010*, New Delhi, India, August 2010.

[22] L.C. Carrington, R.L. Campbell, and L.P. Davis. How Well Can Simple Metrics Represent the Performance of HPC Applications? In *the ACM/IEEE SC 2005 Conference*, November 2005.

[23] 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.

[24] Henri Casanova, Arnaud Legrand, and Martin Quinson. SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In *10th IEEE International Conference on Computer Modeling and Simulation*, Mars 2008.

[25] Henri Casanova and Loris Marchal. A Network Model for Simulation of Grid Applications. Technical report, École Normale Supérieure de Lyon, October 2002.

[26] 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 *25th IEEE International Parallel and Distributed Processing Symposium (IPDPS'11)*, Anchorage (Alaska) USA, May 16-20 2011.

[27] 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. *ACM SIGPLAN Notices*, 28 (7):1–12, July.

[28] DIMEMAS. http://www.erdc.hpc.mil/hardSoft/Software/dimemas, last visited Mars 2011.

[29] Bruno Donassolo, Henri Casanova, Arnaud Legrand, and Pedro Velho. Fast and Scalable Simulation of Volunteer Computing Systems Using SimGrid. In *Workshop in Large Scale Systems and Application Performance, LSAP 2010, in conjuction with HPDC*, Chicago, June 2010.

[30] Trilce Estrada, David A. Flores, Michela Taufer, Patricia J. Teller, Andre Kerstens, and David P. Anderson. The Effectiveness of Threshold-Based Scheduling Policies in BOINC Projects. In *Proc. of the Second IEEE Intl. Conf. on e-Science and Grid Computing (e-Science)*, 2006.

[31] Trilce Estrada, Michela Taufer, Kevin Reed, and David P. Anderson. EmBOINC: An emulator for performance analysis of BOINC projects. In *Proc. of the 3rd Workshop on Desktop Grids and Volunteer Computing Systems (PCGrid)*, 2009.

[32] Sally Floyd and Van Jacobson. Traffic Phase Effects in Packet-Switched Gateways. *SIGCOMM Comput. Commun. Rev.*, 21:26–42, April 1991.

[33] Ian Foster, Carl Kesselman, and Steven Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15:1–??, 2001.

[34] Kayo Fujiwara. Cost and Accuracy of Packet-Level vs. Analytical Network Simulations: An Empirical Study. Master's thesis, Dept. of Information and Computer Sciences, University of Hawai'i at Manoa, April 2007.

[35] Kayo Fujiwara and Henri Casanova. Speed and Accuracy of Network Simulation in the SimGrid Framework. In *Second International Conference on Performance Evaluation Methodologies and Tools, VALUETOOLS*, Nantes, France, October 2007.

[36] Wojciech Galuba, Karl Aberer, Zoran Despotovic, and Wolfgang Kellerer. ProtoPeer: a P2P toolkit bridging the gap between simulation and live deployement. In *SIMUTools'09 2nd International Conference on Simulation Tools and Techniques*, Rome, Italy, 2009.

[37] Grand Challenge Problems. http://en.wikipedia.org/wiki/Grand_Challenge_problem, last visited Mars 2011.

[38] Roger W. Hockney. The communication challenge for mpp: Intel paragon and meiko cs-2. *Parallel Computing*, 20(3):389–398, 1994.

[39] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model. In *Workshop in Large Scale Systems and Application Performance, LSAP 2010, in conjuction with HPDC*, Chicago, June 2010.

[40] https://www.grid5000.fr. Grid5000, last visited Mars 2011.

[41] Fumihiko Ino, Noriyuki Fujimoto, and Kenichi Hagihara. Loggps: A parallel computational model for synchronization analysis. *ACM SIGPLAN Notices*, 36 (7):133–142, 2001.

[42] Teerawat Issariyakul and Ekram Hossain. *Introduction to Network Simulator NS2*. Springer Link, July 2008.

[43] V. Jacobson, R. Braden, and D. Borman. Tcp extensions for high performance, 1992.

[44] Manish Jain and Constantinos Dovrolis. Ten fallacies and pitfalls in end-to-end available bandwidth estimation. In *ACM Internet Measurements Conference (IMC)*, October 2004.

[45] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley- Interscience, New York, NY, April 1991.

[46] Sam Jansen and Anthony McGregor. Validation of simulated real world tcp stacks. In *Winter Simulation Conference*, 2007.

[47] Jared Winick and Cheng Jin and Qian Chen and Sugih Jamin. Inet: Internet Topology Generator. Technical Report CSE-TR-433-00, University of Michigan at Ann Arbor, 2000.

[48] Charles Killian, James W. Anderson, Ryan Braud, Ranjit Jhala, and Amin Vahdat. Mace: Language Support for Building Distributed Systems. In *Programming Language Design and Implementation*, June 2007.

[49] D. Kondo, B. Javadi, A. Iosup, , and D. Epema. The Failure Trace Archive: Enabling Comparative Analysis of Failures in Diverse Distributed Systems. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, 2010.

[50] Stein Kristiansen and Thomas Plagemann. NS-2 Distributed Clients Emulation: Accuracy and Scalability. In *SIMUTools'09 2nd International Conference on Simulation Tools and Techniques*, Rome, Italy, 2009.

[51] Arnaud Legrand, Loris Marchal, and Henri Casanova. Third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03). In *10th IEEE International Conference on Computer Modeling and Simulation*, Tokyo, Japan, May 2003.

[52] Lorenzo Leonini, Étienne Rivière, , and Pascal Felber. Splay: Distributed systems evaluation made simple (or how to turn ideas into live systems in a breeze). In *USENIX Symposium on Networked Systems Design and Implementation (NSDI'09)*, April 2009.

[53] Xin Liu and Andew A. Chien. Realistic Large-Scale Online Network Simulation. In *SC '04 Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, San Diego, Californie, USA, 2004.

[54] Xin Liu, Huaxia Xia, and Andrew A. Chien. Validating and Scaling the MicroGrid: A Scientific Instrument for Grid Dynamics. *In the Journal of Grid Computing*, II(2), 2004.

[55] Steven H. Low. A Duality Model of TCP and Queue Management Algorithms. *IEEE/ACM Transactions on Networking*, 11(4), 2003.

[56] Mace Web Site. http://www.macesystems.org/, last visited Mars 2011.

[57] L. Massoulié and J. Roberts. Bandwidth sharing: objectives and algorithms. *IEEE/ACM Transactions on Networks*, 10(3):320–328, 2002.

[58] Matthew B. Doar. A Better Model for Generating Test Networks. In *IEEE GLOBECOM*, pages 86–93, November 1996.

[59] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: An Approach to Universal Topology Generation. In *International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems - MASCOTS'01*, Cincinnati, Ohio, August 2001.

[60] G. Myers. *The Art of Software Testing*. John Wiley & Sons, New York, 1979.

[61] NAS - NASA Parallel Benchmark. http://www.nas.nasa.gov/Resources/Software/npb.html, last visited Mars 2011.

[62] NetPerf. http://www.netperf.org/netperf/NetperfPage.html, last visited Mars 2011.

[63] Network Emulation with NetEm. http://www.linuxfoundation.org/collaborate/workgroups/networking/netem, last visited Mars 2011.

[64] Thomas Neumann, Matthias Bender, Sebastian Michel, and Gerhard Weikum. A reproducible benchmark for p2p retrieval. In *First International Workshop on Performance and Evaluation of Data Management Systems, ExpDB 2006, in cooperation with ACM SIGMOD*, June 2006.

[65] P. Oliver, G. N. Baird, M. M. Cook, L. A. Johnson, and P. M. Hoyt. An experiment in the use of synthetic programs for system benchmarking. *Managing Requirements Knowledge, International Workshop on*, 0:431, 1974.

[66] OurGrid Community Web Site. http://www.ourgrid.org/, last visited Mars 2011.

[67] Firoiu Padhye and Krusoe Towsley. Modeling tcp reno performance: A simple model and its empirical validation. *IEEE/ACM Transactions on Networking*, 8 (2), 2000.

[68] Pathchar. http://www.caida.org/tools/utilities/others/pathchar/, last visited Mars 2011.

[69] Pathload. http://www.cc.gatech.edu/fac/Constantinos.Dovrolis/bw-est/pathload.html, last visited Mars 2011.

[70] Pathrate. http://www.cc.gatech.edu/fac/Constantinos.Dovrolis/bw-est/pathrate.html, last visited Mars 2011.

[71] Pedro Velho, Ph.D. candidate. http://mescal.imag.fr/membres/pedro.velho/publications.html, last visited Mars 2011.

[72] ProtoPeer. http://www.macesystems.org/, last visited Mars 2011.

[73] Martin Quinson. GRAS: A Research & Development Framework for Grid and P2P Infrastructures. In *18th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'06)*, Dallas, Texas, November 2006.

[74] G. 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.

[75] rl Staelin and Hewlett packard Laboratories. lmbench: Portable Tools for Performance Analysis. In *USENIX Annual Technical Conference*, pages 279–294, 1996.

[76] Lucas Mello Schnorr, Guillaume Huard, and Philippe Olivier Alexandre Navaux. Triva: Interactive 3D Visualization for Performance Analysis of Parallel Applications, March 2010.

[77] SimBoinc. http://simboinc.gforge.inria.fr/, last visited Mars 2011.

[78] SimBOINC: A Simulator for Desktop Grids and Volunteer Computing Systems. http://simboinc.gforge.inria.fr/, last visited Mars 2011.

[79] SimGrid. http://simgrid.gforge.inria.fr/, last visited Mars 2011.

[80] USS SimGrid. http://uss-simgrid.gforge.inria.fr/, last visited Mars 2011.

[81] TOP500 Supercomputing Sites. http://www.top500.org/, last visited Mars 2011.

[82] D.B. Skillicorn, Jonathan Hill, and W. F. McColl. Questions and answers about bsp. *Scientific Programming*, 6 (3):249–274, 1997.

[83] Allan Snavely, M. Tikir, L. Carrington, and E. Strohmaier. A Genetic Algorithms Approach to Modeling the Performance of Memory-bound Computations. November 2007.

[84] STREAM: Memory Bandwidh Benchmark. http://www.cs.virginia.edu/stream/, last visited Mars 2011.

[85] SPEC CPU Subcommittee. Spec cpu2006 benchmark descriptions. *ACM SIGARCH newsletter, Computer Architecture News*, 34 (4), September 2006.

[86] Susanne E. Hambrusch. Models for Parallel Computation. In *International Conference on Parallel Processing Workshop*, August 1996.

[87] 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 *Proc. of the 21st Intl. Workshop on Principles of Advanced and Distributed Simulation (PADS)*, 2007.

[88] D. Terpstra, H. Jagode, H. You, and J. Dongarra. Collecting Performance Data with PAPI-C. In *3rd Parallel Tools Workshop*, Dresden, Germany, 2009.

[89] The Network Simulator 2, NS2. http://www.isi.edu/nsnam/ns/, last visited Mars 2011.

[90] The Network Simulator 3, NS3. http://www.nsnam.org/index.html, last visited Mars 2011.

[91] The SunFlower Tool Suite. http://sflr.org/, last visited Mars 2011.

[92] George Tsouloupas and Marios D. Dikaiakos. Gridbench: A tool for the interactive performance exploration of grid infrastructures. *Journal of Parallel and Distributed Computing*, 67:1029–1045, 2007.

[93] Bernard M. Waxman. Routing of Multipoint Connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, December 1988.

[94] Beowulf Project webpage. http://www.beowulf.org/overview/history.html, October 2005.

[95] FightAIDS@Home webpage. http://fightaidsathome.scripps.edu/, last visited Mars 2011.

[96] SETI@Home webpage. http://setiathome.ssl.berkeley.edu/, last visited Mars 2011.

[97] WWW Computer Architecture Page. http://pages.cs.wisc.edu/~arch/www/tools.html, last visited Mars 2011.

[98] Xen hypervisor. http://www.xen.org/, last visited Mars 2011.

[99] Lixia Zhang, Scott Shenker, and Daivd D. Clark. Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic. *ACM SIGCOMM Computer Communication*, pages 133–147, September 1991.

[100] Gengbin Zheng, Gunavardhan Kakulapati, and Laxmikant V. Kale. BigSim: A Parallel Simulator for Performance Prediction of Extremely Large Parallel Machines. In *IPDPS*, 2004.