Description, deployment and optimization of medical image analysis workflows on production grids

Tristan Glatard

Université de Nice Sophia-Antipolis

ECOLE DOCTORALE STIC

S ’

THESE

pour obtenir le titre de

Docteur en Sciences

de l’Université de Nice Sophia-Antipolis
mention

I

présentée et soutenue par

Tristan GLATARD

le 20 novembre 2007

Description, deployment and optimization of medical image analysis workflows on production grids

Rapporteurs :

M Péter KACSUK
Mme Isabelle MAGNIN

Jury :

M	Christian BARILLOT	
M	Denis CAROMEL	Président
M	Frédéric DESPREZ	
Mme	Cécile GERMAIN-RENAUD	
Mme	Isabelle MAGNIN	
M	Johan MONTAGNAT	Directeur
M	Xavier PENNEC	Co-directeur
M	Michel RIVEILL	

A mes parents et grand-parents.

Remerciements

Cette thèse n'aurait jamais vu le jour sans le concours de multiples personnes que je tiens ici à remercier chaleureusement. Je tiens en tout premier lieu à remercier mes directeurs de thèse Johan Montagnat et Xavier Pennec. Ils ont été à l'origine de la définition d'un sujet de thèse particulièrement enrichissant de par sa pluridisciplinarité.

Johan, pour son encadrement au jour le jour, sa disponibilité exceptionnelle, quasi-quotidienne, sa capacité à suggérer des pistes de recherche, orienter mes tâtonnements et répondre sans délai à mes questions parfois naïves quel que soit son emploi du temps. Au delà de son encadrement scientifique et technique sans faille, sa capacité à faire partager son enthousiasme a constitué une source de motivation particulière, indispensable pour fournir un travail parfois exigeant. Un grand merci aussi pour m'avoir, en compagnie de Diane, changé régulièrement les idées, dans les eaux fraîches de la Maglia ou, les palmes aux pieds, à la rencontre des mérous.

Xavier, pour sa vision éclairée de l'interface grille / imagerie médicale, sa capacité à prendre du recul pour identifier les problèmes importants, s'illustrant dans des commentaires souvent exigeants mais toujours justifiés sur la rédaction, la présentation des résultats et l'importance accordée à l'adoption d'une démarche scientifique rigoureuse.

Je tiens aussi à remercier Michel Riveill et Nicholas Ayache pour m'avoir accueilli pendant ces 3 ans dans leur équipe de recherche et pour avoir su créer les conditions matérielles et humaines indispensables à la réalisation de cette thèse.

Thanks a lot to Professor Péter Kacsuk for having accepted the time-consuming task to review this manuscript and for the interesting collaboration that we had during those 3 years. Je tiens également à remercier Isabelle Magnin pour avoir accepté la charge de rapporteur en formulant des remarques fructueuses ainsi que pour son encadrement de mes débuts en recherche à Creatis.

Je souhaite aussi remercier les membres de mon jury, Christian Barillot, Denis Caromel, Frédéric Desprez, Cécile Germain-Renaud, Isabelle Magnin, Johan Montagnat, Xavier Pennec et Michel Riveill pour leur disponibilité malgré les aléas des conditions de trafic aérien le jour de la soutenance.

Au delà des conditions matérielles, le projet AGIR a grandement contribué à élargir le spectre de cette thèse. Un grand merci à tous ses membres et en particulier à Cécile Germain-Renaud qui le coordonne.

Merci aussi aux enseignants qui m'ont confié la charge de certains de leurs TD, Diane Lingrand pour le sérieux de ses supports, Pierre Mathieu pour son souci pédagogique contagieux et Jean-Yves Tigli pour sa confiance, sa bonne humeur et son dynamisme rare.

La plupart des développements logiciels présentés dans ce manuscrit doivent aussi beaucoup à l'assistance compétente des stagiaires de l'EPU, Yann Biancheri, Lydie Blanchet, Christophe Bonnet, Vincent Cave, Fabien Cordier, Fabien Gaujous, Amir Hnain, Patrick

Hoangtrong, Vincent Léon, Damien Mandrioli, Romain Raugi, Pascal Rolin et Thomas Rollinger.

Enfin, remercier ici les gens qui m'accompagnent au quotidien peut paraître dérisoire, tant leur apport dépasse le cadre de ce manuscrit, dans la durée comme dans la nature de leur soutien. Papa, Maman, Anaïs et Céline, ce travail vous appartient aussi. Caroline, merci pour ces lumineuses années, malgré tout.

Cette thèse a été financée par le projet ACI AGIR.

Abstract

Grids are interesting platforms for supporting the development of medical image analysis applications: they enable data and algorithms sharing and provide huge amounts of computing power and data storage. In this thesis, we investigate a medical image analysis problem that turns out to be a typical dimensioning application for grids, thus leading to develop new workflow description, implementation and optimization methods and tools. The basic application problem is the evaluation of medical image registration algorithms in absence of ground truth. Results obtained with a statistical method applied to a registration problem dealing with the follow-up of brain tumors in radiotherapy are presented. Those results allow to detect subtle flaws among the data. We extend this validation scheme in order to quantify the impact of lossy image compression on registration algorithms.

This application is representative of typical grid problems so that we study its deployment and execution on such infrastructures. We adopt a generic workflow model to ease the application parallelization on a grid infrastructure. A novel taxonomy of workflow approaches is presented. Based on it, we select a suitable workflow language and we design and implement MOTEUR, an enactor exploiting all the parallelism levels of workflow applications. A new data composition operator is also defined, easing the description of medical image analysis applications on grids. Benchmarks on the EGEE production grid compared to controlled conditions on Grid'5000 reveal that the grid latency and its variability lead to strong performance drops. Therefore, we propose a probabilistic model of the execution time of a grid workflow. This model is user-centric: the whole grid is considered as a black-box introducing a random latency on the execution time of a job.

Based on this model, we propose three optimization strategies aiming at reducing the impact of the grid latency and of its variability: (1) grouping sequentially linked jobs reduces the mean latency faced by a workflow, (2) optimizing the timeout value of jobs reduces the impact of outliers and (3) optimizing the jobs granularity reduces the risk to face high latencies. Significant speed-up are yielded by those strategies.

Résumé

En permettant le partage à grande échelle de données et d'algorithmes et en fournissant une quantité importante de puissance de calcul et de stockage, les grilles de calcul sont des plateformes intéressantes pour les applications d'analyse d'images médicales. Dans cette thèse, nous étudions un problème d'analyse d'images médicales qui s'avère être une application dimensionnante pour les grilles, conduisant au développement de nouvelles méthodes et outils pour la description, l'implémentation et l'optimisation de flots de traitements. Le problème applicatif étudié est l'évaluation de la précision d'algorithmes de recalage d'images médicales en l'absence de vérité terrain. Nous faisons passer à l'échelle une méthode statistique d'évaluation de ces algorithmes et nous montrons des résultats de précision sur une base de données concernant le suivi de la radiothérapie du cerveau. Ces résultats permettent notamment de détecter des défauts très légers au sein des données. Nous étendons ce schéma pour quantifier l'impact de la compression des images sur la qualité du recalage.

Cette application étant représentative de problèmes typiques survenant sur les grilles, nous nous attachons à son déploiement et à son éxecution sur ce type d'infrastructures. Pour faciliter une parallélisation transparente, nous adoptons un modèle générique de flots de traitements, dont nous proposons une nouvelle taxonomie. Pour répondre aux limitations de performance des moteurs d'exécution de flots existants, nous présentons MOTEUR, qui permet d'exploiter les différents types de parallélisme inhérents à ces applications. La définition d'un nouvel opérateur de composition de données facilite la description des applications d'analyse d'images médicales sur les grilles. Par une comparaison entre la grille de production EGEE et des grappes dédiées de Grid'5000, nous mettons en évidence l'importance de la variabilité de la latence sur une grille de production. En conséquence, nous proposons un modèle probabiliste du temps d'exécution d'un flot de traitement sur une grille. Ce modèle est centré sur l'utilisateur : il considère la grille toute entière comme une boîte noire introduisant une latence aléatoire sur le temps d'exécution d'une tâche.

A partir de ce modèle, nous proposons trois stratégies d'optimisation visant à réduire l'impact de la latence et de sa variabilité : (1) dans un flot de traitement, grouper les tâches séquentiellement liées permet de réduire la latence moyenne rencontrée, (2) optimiser la valeur du délai d'expiration des tâches prémunit contre les valeurs extrêmes de la latence et (3) optimiser la granularité des tâches permet de réduire le risque de rencontrer de fortes latences. Des accélérations significatives sont ainsi obtenues.

Contents

Notations

$\otimes$	all-to-all data composition operator
$\oplus$	one-to-one data composition operator
n_W	number of services on the critical path of a workflow
n_D	number of data items to be processed by a workflow
R	grid latency
$R_{i,j}$	grid latency faced by service i processing data item j
Σ	makespan of a workflow
J	total execution time of a job (including resubmissions)
w	CPU time of an application
p	number of jobs submitted by an application
f_V/F_V	pdf/cdf of the random variable V
E_V	expectation of the random variable V
σ_V	standard-deviation of the random variable V
t_∞	timeout value
q	probability for a job to timeout
ρ	outliers ratio
ϕ/Φ	pdf/cdf of the standard normal distribution

Introduction

Sharing data and algorithms with a community of users has produced fascinating applications of the Internet, such as the World Wide Web and, to some extent, open-source software projects such as the Linux kernel. Similarly, the development of digital devices has made available tremendous amounts of storage and computing resources. Those resources are distributed all over the world but still accessible: the *grid* [Foster and Kesselman, 1997] denotes the aggregation of heterogeneous resources transparently accessible by the end-user. In the medical image analysis domain, sharing data, algorithms and computing power suggests awesome applications benefiting from such resources aggregations.

1 Grids for medical image analysis applications

Computerized medical image analysis is now a well established area that provides assistance for diagnosis, therapy, and pathologies follow-up. It may benefit from the adoption of grid technologies in several aspects.

First, by providing computing power, grids tend to allow a wider clinical adoption of medical image analysis procedures. Indeed, the exploitation of medical image analysis algorithms in clinical context imposes time deadline constraints that are hardly satisfied by time consuming applications but could be approached thanks to the use of grids. Besides, from the medical image analysis scientists point of view, setting up large scale realistic experiments (such as parameter sweep applications [Sermesant et al., 2006] or image acquisition simulation [Benoit-Cattin et al., 2005]) is often difficult due to the required computing time. This kind of compute-intensive methods greatly benefits from grid technologies that speed up the experiments, specially in case of multiple trials required for parameters tuning and the detection of experimental errors. Applications dedicated to the validation or evaluation of medical image analysis procedures may also be fostered by computing power. Such applications often require to process large databases of images in order to obtain significant results. For instance, the quality of the results provided by the bronze standard method motivating this thesis is improving with the number of processed images and evaluated algorithms. In this case, the availability of computing power is directly related to the quality of the obtained results.

Second, the gridification of medical image analysis applications is also motivated by the fact that image processing algorithms may be more efficiently shared on grids. Pushing standards, grids are easing the use of medical image analysis tools for a large community of end users, not necessarily aware of computer technologies. These standards enable access to procedures needed for building large scale health-related experiments. In particular, the validation of medical image analysis procedures would benefit from a transparent sharing of codes. Making algorithms interoperate on a given platform may rapidly become intractable as some serious engineering problems will arise. For instance, the bronze standard application involves several different medical image analysis algorithms that may have been developed by different researchers and institutes using various programming languages on heterogeneous systems and architectures. Particular methods have been set up by grid developers to ease code sharing. As detailed in this thesis, the state-of-the-art solution to deal with code sharing is to adopt a Service Oriented Architecture (SOA) to develop application services and to use *workflows* as a programming paradigm to build complete applications on top of them.

Besides, with the growing inspection capabilities of imagers and the increase in medical data production, the need for large amounts of data storage increases. Beyond that, being able to share data is also a crucial challenge of medical image analysis applications that would often remain useless without the ability to access image databases of a significant size. Applications such as content-based image indexing and retrieval [Montagnat et al., 2005], atlas construction [Mazziotta et al., 1995] or statistical validation of algorithms [Pennec and Thirion, 1997], would greatly benefit from a massive sharing of medical images. Grids, in their *data-grid* aspect provide a common storage and indexation space allowing the users of a virtual organization to share their data. Yet, many issues (such as data/metadata access, anonymization and encryption for different users communities) remain specific to the sharing of medical data and systems such as the Medical Data Manager [Montagnat et al., 2007] or the Globus Medicus [Erberich et al., 2007] should become mandatory components of any grid medical system put in production.

Finally, another well established benefit of grid computing is the federation of scientific communities in Virtual Organizations (VOs). In the context of medical image analysis, several specialized communities can be identified for which the grid fosters collaborative work. Indeed, many actors are likely to take part in a grid-enabled medical image analysis system, with diverse skills, needs and constraints. In particular:

1. The developers of the applications (*e.g* medical image analysis scientists) should be able to describe a complete application from existing heterogeneous codes that have been developed independently from each other.
2. The end-users (*e.g* clinicians) should be able (i) to specify the data on which the application will be run and (ii) to execute it on a grid without worrying about the technical underlying details.

3. The grid experts should be able to transparently deploy any non grid-specific code and to efficiently execute complete applications composed of several of them.

In this thesis, we will try to keep the balance between those three aspects. Workflows are an interesting approach to make them cooperate. Apart from providing a transparent way to design and deploy applications on grids, they constitute a particularly suitable application exchange format among them. They are thus studied in depth in this manuscript. Moreover, in order to study realistic scenarios, we will study the deployment of those workflows on *production* grids.

2 Doing research on production grids ?

Production grids are 24/7 operating platforms that provide stable enough systems to support science. They support applications aiming at yielding scientific results in various fields but computer science, such as high energy physics, aeronautics, geology, earth observations, bioinformatics and medical image analysis. Thus, among the existing solutions, they are ideal platforms to target for the deployment and scientific exploitation of medical image analysis applications. Conversely, research in computer science is traditionally made on experimental platforms [Cappello et al., 2005, Cappello and Bal, 2007] that provide controlled environments allowing the setup of reproducible experiments. Those instruments are envisioned as "grid telescopes" developed by computer scientists for computer scientists and required to analyze and develop models and methods setting the basis of the software that would be put in production in a next step.

The grid platforms targeted by this manuscript are production infrastructures rather than experimental ones. Consequently, the grid deployment of the application and the subsequently studied optimization strategies will be performed in a non-reproducible context, which may be debatable from a computer science point of view. Indeed, one could wonder whether production grids should only be seen as a deployment platform for scientific applications or if they may also be considered for computer science research.

Even if the experimental conditions are not so favorable, it still remains worth observing the real sky. Studying production systems seems to be a required step at least to identify effective problems that could be studied in reproducible contexts in a next step. Indeed, production grids are not very well known systems: the prediction of the performance of applications is almost impossible today and even its analysis may be problematic. In practice, current production solutions for debugging [Duan et al., 2006] and even for resource selection [Jacq et al., 2007] rely on huge historical information logs that are empirically interpreted. No model of those platforms are available or used in production. The understanding of production platforms requires the collection of realistic data (such as workloads [Feitelson, 2002, Medernach, 2005]) and their improvement expects the identification and expression of the new problems raised by the production exploitation of grid applications. That is why we study those production sys-

tems in this manuscript. Beyond their practical interest, our intent is to provide some material helping in their understanding in order to allow the development of new optimization methods specific to those platforms. Experimental platforms will only be used as a reference to quantify the performance of production grids.

3 Manuscript organization and contributions

The approach adopted in this manuscript is to start from a typical medical image analysis application and to consider it as a dimensioning use-case to study grids. One the one hand, the power of those infrastructures allows us to derive new original results in the medical image analysis field. On the other hand, this application-centric approach led us to design new optimization methods on grids to foster the performance of applications. This thesis is divided into three parts, each of them being composed of three chapters. Part I deals with the description of medical image analysis workflows. After a presentation of the new results obtained from the image registration application, it aims at identifying suitable workflow models for medical image analysis. Part II is devoted to the execution of workflows on production grids: its goal is to determine the level of performance that can be achieved on those infrastructures and to highlight the causes of performance drops. Based on those conclusions, part III proposes new strategies to optimize workflows execution on production grids.

Chapter 1. Chapter 1 presents the new results obtained with the bronze standard method. The idea of this method and preliminary results demonstrating the feasibility proof of the approach were presented in [Pennec, 2006b]. In this thesis, we enable its operational mode thanks to grid technologies which allow the method to reach its full power. Results are presented on a large database related to the follow-up of brain radiotherapy. On this registration problem, we demonstrate that a sub-voxelic precision (about 0.15° in rotation and 0.4 mm in translation) is achieved by the tested algorithms. The large size of this database and of the computation involved allows to compute statistically significant results, which permits subtle detections such as a 1.2 degree tilt of part of the images highlighted by our results [Glatard et al., 2006f]. We then propose a study of the influence of a lossy image compression method on the quality of the registration. Based on an experiment involving 3,000 registrations, we conclude that the influence of the tested compression method is almost unnoticeable until a compression ratio of 48. Coupled to similar results shown in [Raffy et al., 2006], it constitutes a set of indications tending to suggest that lossy compression could be considered in some applications. The bronze standard application studied in this chapter illustrates several typical grid problems. In particular, it benefits from a workflow design, as studied in the next chapters.

Chapter 2. In this chapter, we propose a new classification of workflow descriptions for medical image analysis applications. This taxonomy is based on the distinction between the roles of the clinician (specifying the data), the medical image scientist (defining the treatments to perform) and the grid expert (mapping the tasks to the resources). This taxonomy allows to select suitable workflow representations for a given applicative context. In particular, we underline that the use of the traditional workflow representation used on grids requires to mix the roles of the clinician and the image analyst, which is not suitable. The work presented in this chapter is original but not published yet.

Chapter 3. Based on this taxonomy, chapter 3 rationalizes the use of service workflows for medical image analysis applications and provides a description of the bronze standard application in this paradigm using the Scufl language. After the successful description of the bronze standard application in Scufl, we propose a study of the expressiveness of this language. We demonstrate that it is possible to implement a Turing machine in Scufl, thus guaranteeing that the language is expressive enough to describe a larger class of applications.

Chapter 4. Because existing solutions do not provide a satisfying parallelization of service workflows, the implementation of an optimized workflow engine [Glatard et al., 2006c] is presented in chapter 4. This development is based on an existing workflow language (Scufl). Yet, in a fully parallel execution, one of its data composition operators is not well-defined. Thus, we propose a new semantic for this operator and we detail the subsequent implementation [Montagnat et al., 2006, Glatard et al., 2008b]. This new operator aims at facilitating the development of medical image analysis workflows in a parallel context.

Chapter 5. The workflow implementation of the application enables a fair comparison between different grid systems. Chapter 5 details its interface with grid platforms [Glatard et al., 2005, Glatard et al., 2006d] and presents experimental results comparing production grids and dedicated clusters. Results demonstrate a speed-up of 44 of a typical bronze standard execution on a 60-nodes dedicated platform, which is close to the theoretical bound that could be achieved. Because of the latency of the system, the execution in similar conditions on the EGEE production grid only provides a speed-up of 10, which can be explained by the high latency of such production grids. The end of the chapter proposes a multi-grids model [Glatard et al., 2006b] that is used to quantify the gap between production and experimental platforms. Given a number of jobs, this model is able to determine the proportion of them to submit on a production grid rather than on a cluster to minimise the pay-off of the latency. For instance, we demonstrate that from a performance point of view, there is no need to use the production grid rather than a 20 nodes cluster under a threshold of 50 one minute long jobs. This threshold grows to 230 jobs when comparing the production grid to a

100 nodes cluster. The experiments and metrics introduced in this chapter provide new methods to benchmark the performance of production grids with respect to traditional clusters. The variability of the grid latency is also identified as particularly high (up to 5 minutes) on the EGEE production grid.

Chapter 6. This chapter proposes a new modeling of workflow on grids. This model deliberately considers the grid as a black box. Given the high variability of production platforms, we consider that it introduces a random latency on the execution time of a job. A probabilistic model for the performance analysis of workflows [Glatard et al., 2007c] is first presented. Given the topology of the workflow, it allows to determine the expectation and standard deviation of its execution time on a highly variable grid. Using this model, we demonstrate that the latency variability leads to a performance drop of a factor 2 on the execution time of the bronze standard application. This motivates the need for strategies to reduce the impact of this variability. Experimental results characterizing the distribution of the grid latency in production conditions are then shown [Glatard et al., 2007a]. Even if extreme precautions have to be taken when a latency model is assumed, we suggest that the probabilistic distribution of the latency is heavy-tailed. We believe that the probabilistic grid approach adopted in this chapter is likely to generate new methods for the optimization of applications on production grids, as initiated in chapters 8 and 9.

Chapter 7. In this chapter, we propose to group some services of the workflow in order to reduce the impact of the grid latency. A grouping rule ensuring that parallelism is preserved is proposed. Its application yields significant speed-ups, ranging from 1.5 to 3 on the bronze standard application. Grouping services is not straight-forward in a classical service-oriented architecture. Thus, to implement our strategy, we propose a generic application service wrapper which is able to perform the grouping, while still conforming to the services standards [Glatard et al., 2006a, Glatard et al., 2008a].

Chapter 8. Grids are operational systems prone to inevitable failures at multiple levels. In addition to the high variability of the latency, some jobs may get lost or at least not finish in a “finite” time. Those jobs are called outliers. Thus, submitting a grid job introduces a risk that has to be controlled. To do that, chapter 8 proposes a strategy to optimize the timeout value of jobs on production grids [Glatard et al., 2007b]. Relying on the approach introduced in chapter 6, we derive a probabilistic model of the execution time of a job including timeout and resubmissions. Based on it, we suggest that the weight of the tail of the distribution of the latency is a discriminatory parameter for setting a timeout value to the jobs. Optimizing this value is shown to be particularly important on production grids and latency reductions of a factor 1.4 are shown with this method on the EGEE grid.

Chapter 9. To further reduce the risk to face high latencies and outliers, one can control the granularity of a user job, *i.e* the number of grid jobs that will be submitted to compute it. Chapter 9 proposes a method to optimize this job granularity. It exploits the same probabilistic approach as previously, which seems to be relevant enough to optimize job submission parameters with a new angle, focusing on the grid variability seen from a user's perspective. This method goes one step further than the grouping presented in chapter 7: here, the data parallelism of the application is deliberately limited in order to reduce the risk to face high latencies. The method proposed in this chapter is shown to significantly speed-up applications while reducing the global load imposed to the grid [Glatard et al., 2006e].

Part I

W

Chapter 1

Performance evaluation of medical image registration using bronze standards

Contents

Image registration is an important procedure for medical image analysis applications. It aims at finding a geometrical transformation between two images so that they are best superimposed. Evaluating the performance of registration is not trivial because of the lack of ground truth in medical image analysis. The goal of the bronze standard application is to provide a framework for the evaluation of registration results in absence of ground-truth and gold standard [Pennec, 2006b]. The foundations of the method are first described. Then, an experimental use-case related to the follow-up of brain radiotherapy is presented. Finally, a study of the impact of lossy images compression on registration is developed and demonstrates the power of the method. The bronze standard application is the motivating use-case of the following chapters of the manuscript. It gathers grid challenges related to algorithms and data sharing as well as computing power needs that are addressed in the remaining of this thesis.

Le recalage d'images est une procédure importante pour les applications d'analyse d'images médicales. Son but est de trouver une transformation géométrique entre deux images pour qu'elles se superposent au mieux. Evaluer la performance du recalage n'est pas trivial à cause de l'absence de vérité terrain en analyse d'images médicales. Le but de l'application des étalons de bronze est de fournir un cadre pour l'évaluation des résultats du recalage en l'absence de vérité terrain et d'étalon-or [Pennec, 2006b]. Dans un premier temps, les fondements de la méthode sont décrits puis un cas d'utilisation concernant le suivi de la radiothérapie du cerveau est présenté. Enfin, une étude de l'impact de la compression d'images avec pertes sur le recalage est développée et démontre la puissance de la méthode. L'application des étalons de bronze motive les chapitres suivants de ce manuscrit. Elle concentre des enjeux en termes de puissance de calcul et de partage d'algorithmes et de données qui sont étudiés dans la suite de cette thèse.

The goal of this chapter is to present, from a medical image analysis point of view, the bronze standard application which will be our main use case for the study of grid workflows in the next chapters. This application aims at evaluating the performance of medical image registration algorithms with respect to a statistical truth which is determined as a mean of several independent measurements. Those results can be computed in parallel, thus exploiting a natural coarse grain parallelism.

1.1 Medical image registration problems

Medical image registration is a very common procedure in medical image analysis and it has been extensively studied during the last decades [Bankman, 2000, Maintz and Viergever, 1998, Hajnal et al., 2001, Makela et al., 2002, Gholipour et al., 2007]. Its goal is to estimate a transformation enabling the resampling of a *floating image* onto the geometry of a *target image*, so that both images are best superimposed. Many classes of registration problems exist, depending on the nature of the transformation searched (rigid, affine, deformation field), on the modalities of the images to register (Magnetic Resonance, Single Photon Emission Computed Tomography, Computed Tomography, Ultra-Sound, Positon Emission Tomography) and on their geometry (2D, 3D, 4D). In practice, a registration method aims at optimizing a similarity measure between 2 input images, considering a particular transformation space. As in every optimization procedure, performance problems may arise in particular because of the presence of local minima of the optimized criterion. The output transformation of a registration algorithm may be more or less close to the *correct* solution which is most of the time unknown (see figure 1.1). Intensity-based methods may optimize similarity measures such as the sum of square distances between the intensities of the images (SSD), the correlation ratio or coefficient (CR and CC [Roche et al., 1998]) or the mutual information (MI). Alternately, feature-based methods optimize a distance between features extracted from the images, such as crest-lines.

Rigid registration assumes that the target and floating images are two separate acquisitions of the same rigid object. In this case, the registration problem resumes to the finding of a rotation and a translation (6 parameters for 3D images) so that the floating image can be transformed in the frame of the target one. This type of registration problem may for instance correspond to successive acquisitions of a non-deformable organ of a patient (intra-patient registration). In addition to rotation and translation, *affine registration* also considers shear and scaling in all the directions, thus leading to the optimization of 12 parameters for 3D images.

Non-rigid transformations allow local deformations of the objects to register. An extension of the rigid and affine registration problems in this direction is to consider locally rigid or affine transformations, as proposed in [Little et al., 1997] and [Arsigny et al., 2005]. With a small number of intuitive parameters tuning the number of rigid or affine components as well as their definition domains, those transformations are able to address problems involving several rigid objects such as the registration of the head (skull and neck) [Commowick and Malandain, 2006].

However, problems concerning highly deformable organs are more properly addressed by non-rigid transformations. This is the case for the whole abdomen (deformation due to breathing [Sarrut et al., 2006]), pathological images (such as tumoral follow-up images), and of course inter-patient images (including atlas to patient). In this case, the output transformation of the registration is a deformation field, which corresponds, for 3D images, to 3 parameters per voxel.

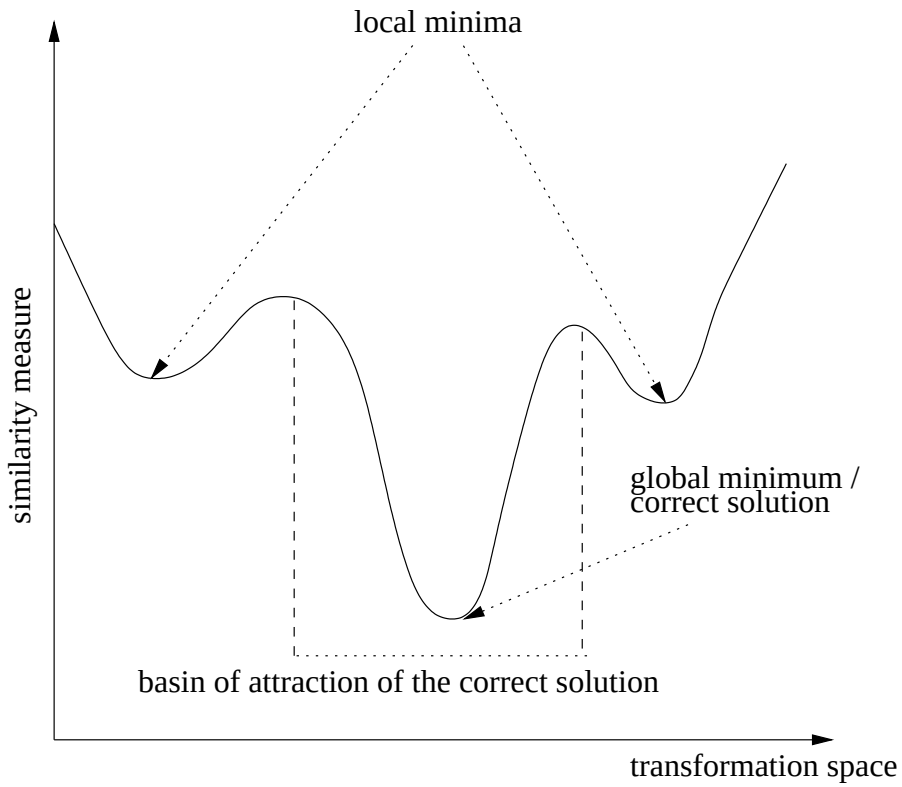

Figure 1.1: Illustration of the registration problem: the transformation space is browsed by the optimization strategy in order to minimize a similarity measure between the floating and the reference images. The basin of attraction of the global minimum characterizes the robustness of the method.

The registration problem is also characterized by the nature of the images to register: they can correspond to the same imaging modality (intra-modal registration) or to different ones (inter-modal registration [Hellier and Barillot, 2004, Arbel et al., 2004]). The geometry of the images can also vary among the registration problems. For instance, registering 2D to 3D images is required in some image guided therapy clinical applications [Nicolau et al., 2003], in particular in fluoroscopy [Micu et al., 2006, Heining et al., 2006] and registering cardiac sequences implies 4D images [Makela et al., 2003].

1.2 Performance evaluation: the bronze standard method

The performance of registration algorithms is critical for many image-based clinical procedures but quantifying it is difficult due to the lack of gold standard in most clinical applications. In most cases, there is no reference to which the result of a registration algorithm can be compared. To analyze registration algorithms from a technical point of view, one may consider them as black boxes that take images as input and that return a transformation. The performance evaluation problem is to estimate the quality of this transformation. However, no registration algo-

rithm will perform the same for all types of input data and it is important to keep in mind that a performance estimation of an algorithm is only valid for the particular registration problem considered in the evaluation study. For instance, registering CT images of the head of the same patient could be much more accurate than registering the abdomen of the same patient because some deformations occur in the second case due to breathing and heart beating. Likewise, one algorithm may perform very well for multimodal MR registration but poorly for SPECT/CT. This means that the evaluation data set has to be representative of the targeted typical clinical application problem: all sources of perturbation in the data should be represented, such as acquisition noise and artifacts, pathologies,... It cannot be concluded just from one experiment that one algorithm is better than the others for all applications.

1.2.1 Performance quantifiers

As far as the registration result is concerned, one can distinguish between gross errors (convergence to wrong local minima) and small errors around the exact transformation. Gross errors may impact the *robustness* which can be quantified by the size of the basin of attraction of the correct solution (see figure 1.1) or by the probability to find the correct transformation. Small errors may be sorted into *systematic biases*, *repeatability* and *accuracy* [Jannin et al., 2002]. Repeatability accounts for the errors due to internal parameters of the algorithm, mainly the initial transformation, and to the finite numerical accuracy of the optimization algorithm, while the external error accounts for the propagation of the data errors into the optimization result. It is important to notice that the accuracy measures the error with respect to the truth (which may be unknown), while the precision or repeatability only measures the deviation from the average value, *i.e.* it does not take into account systematic biases, which are often hidden. There are numerous reasons to have systematic biases that are usually forgotten. For instance, a calibration error in the acquisition system will consistently bias all the images acquired with that device. Unless another calibration is done or an external reference is used (e.g. another acquisition device), there is no way to detect such a bias. In terms of statistical modeling, this means that all the potential error sources have to vary among the measures in order to be considered as random and be included in the performance evaluation.

In a statistical setting, considering the true transformation as a random variable naturally leads to quantify the repeatability (resp. accuracy) of a registration method as the standard deviation of the related observed transformations (considered as realizations of the random variable) or as the expected RMS distance to their mean (resp. to the exact transformation), or more interestingly with their covariance matrix as the transformation uncertainty is usually non isotropic (*e.g.* radians and millimeters for rotation and translation part of a rigid transformation). Then, the variability of the transformation can be propagated to some target points of interest inside the images [Pennec and Thirion, 1997] in order to obtain local estimations of the accuracy of the transformation [Pennec et al., 1998]). We obtain the so-called Target Reg-

istration Error (TRE) [West et al., 1997]. This should not be confused with the Fiducial Local Error (FLE) which is related to the value of the similarity criterion at the minimum.

1.2.2 Performance evaluation

Several methods have been investigated in order to assess the performance of registration algorithms. The main problem is to determine a satisfying reference to perform the evaluation. It can then be used to evaluate the robustness, repeatability and accuracy of registration methods. Three different classes of approaches can be used to determine the reference. The bronze standard method studied in this thesis belongs to the third one, which can be used in absence of ground truth and gold standard.

Data simulation. One of the simplest evaluation schemes is to simulate noisy data, to apply a known transformation on it and to measure how far is the registration result from the true one (the ground truth is obviously known). Even if in some cases images may be faithfully simulated (e.g SPECT [Grova et al., 2001], MRI [Benoit-Cattin et al., 2005] or CT of the breathing abdomen [Sarrut et al., 2006]) with a very high computational cost due to the complexity of image acquisition physics, the main drawback of synthetic data is that it is very difficult to identify and model all the sources of variability, and especially unexpected events (pathologies, artifacts, etc). Forgetting one single source of error (e.g. bias due to chemical shift in features extracted from MRI [Pennec et al., 1998] or camera calibration errors in 2D-3D registration [Nicolau et al., 2003]) automatically leads to the underestimation of the final transformation variability.

Phantoms. The second evaluation level is to use real data in a controlled environment, for instance imaging a physical phantom in different positions/orientations. There is possibly a gold standard, if one can precisely measure the motion or deformation of the phantom with an external apparatus. However, it is difficult to test all the clinical conditions with such a phantom (e.g. many different pathologies or even different localization of the same pathology). Moreover, it is often argued that these phantoms are not representative of real in vivo biological systems. One level closer to the reality, experiments on cadavers correctly take into account the anatomy, but fail to exhibit all the errors due to the physiology, thus producing images that may be very different from the in-vivo ones.

Performance evaluation without gold standard. The last level of evaluation methods is the one addressed in this thesis. It relies on a database of in-vivo real images representative of the clinical application. Such a database can be large enough to span all sources of variability, but there is usually no gold standard registration to compare with. One method is to perform a cross comparison of the criteria optimized by different algorithms [Hellier et al., 2003]. However,

this does not give any insight about the transformation itself. A more interesting method for registration evaluation is the use of consistency loops [Holden et al., 2000, Roche et al., 2001]. The principle is to compose transformations that form a closed circuit and to measure the difference of the composition from the identity. This criterion does not require any ground truth, but it only measures the repeatability as any bias will get unnoticed. A last type of methods is to see the ground truth as a hidden variable, and to estimate concurrently the ground truth and the quality as the distance of the computed results to this reference (EM like algorithms), as it was exemplified for the validation of segmentation by the STAPLE algorithm [Warfield et al., 2004]. The bronze standard method belongs to this class of methods. Specific validation methods can also be envisaged for non rigid registration in absence of ground truth. For instance, [Schestowitz et al., 2006] assesses the registration in terms of the quality of a model constructed from the registered images: their idea is that a correct registration produces anatomically meaningful images.

1.2.3 The bronze standard method

The principle of the bronze standard method introduced by Pennec in several papers (*e.g* [Nicolau et al., 2003]) and synthetized in [Pennec, 2006b] is similar to STAPLE but concerns registration. From a set of measurements (*i.e.* registrations between pairs of images obtained with several methods), the exact transformations and the variability of the registration results with respect to these measures have to be estimated. Let us assume that we have n images of the same organ of the patient and m methods to register them, i.e. $m \times n \times (n-1)$ transformations $T^k_{i,j}$ (we denote here by k the index of the method and by i and j the indexes of the reference and target images). The goal here is to estimate the $n-1$ free transformations $\bar{T}_{i,i+1}$ that relate successive images and that best explain the measurements $T^k_{i,j}$ (see figure 1.2).

The bronze standard transformation between images i and j is obtained by composition of the free parameters: $\bar{T}_{i,j} = \bar{T}_{i,i+1} \circ \bar{T}_{i+1,i+2} \circ \ldots \circ \bar{T}_{j-1,j}$ if $i < j$ (or the inverse of both terms if $j > i$). The bronze standard method considers the exact transformations as hidden variables of an overestimated system: $n-1$ transformations have to be estimated whereas $m \times n \times (n-1)$ observations are available. The exact transformations are estimated as the ones that minimize the prediction error of the observations:

$$\left\{\bar{T}_{i,i+1}\right\} = \arg\min_{T_{i,i+1}} \sum_{i,j\in[1,n],k\in[1,m]} d\left(T^k_{i,j}, \bar{T}_{i,j}\right)^2 \tag{1.1}$$

where d is a distance function between transformations.

In the particular case of rigid transformations, the distance function can be chosen as a robust variant of the left invariant distance on rigid transformations developed in [Pennec et al., 1998]:

$$d(T_1, T_2) = \min\left(\mu^2(T_1^{(-1)} \circ T_2)\,,\, \chi^2\right) \quad \text{with} \quad \mu^2(R(\theta, n), t) = \theta^2/\sigma_r^2 + \|t\|^2/\sigma_t^2$$

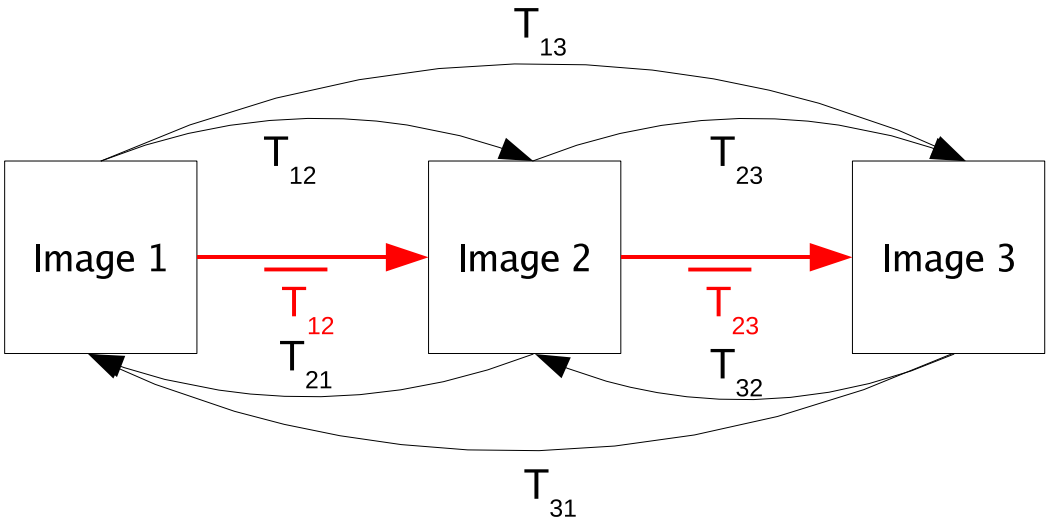

Figure 1.2: The basic principle of the bronze standard method is to exploit redundancy among the measurements. The transformations to estimate (red arrows) are obtained as a means of measurement compositions (black arrows).

where θ is the angle of rotation R and n is the unitary vector defining its axis. t is the translation vector of the transformation. Details on the general methods for doing statistics on Riemannian manifolds and Lie groups are given in [Pennec, 2006a]. The Mahanalobis norm μ is normalized by the variances σ_θ^2 and σ_t^2 of the observations that have to be properly estimated.

The larger the number of registered images, the more accurate the estimated bronze standard. It is also important to use several algorithms based on different methods and developed by various people to prevent the results from being systematically biased by a specific registration technique or implementation usage. Results over several patients are averaged to obtain more significant estimations.

In this process, we do not only estimate the optimal transformations, but also the rotational and translational variance of the "transformation measurements", which are propagated through the criterion to give an estimate of the variance of the optimal transformations. Of course, these variances should be considered as a fixed effect (i.e. these parameters are common to all patients for a given image registration problem, contrarily to the transformations) so that they can be computed more faithfully by multiplying the number of patients.

The estimation $\bar{T}_{i,i+1}$ is called *bronze standard* because the result converges toward the perfect registration as the number of methods m and the number of images n increases. Indeed, considering a given registration method, the variability due to the noise in the data decreases as the number of images n increases, and the registration computed converges toward the perfect registration up to the intrinsic bias introduced by the method. Now, using different registration procedures based on different methods, the intrinsic bias of each method also becomes a random variable, which is hopefully centered around zero and averaged out during the minimiza-

tion procedure. The different bias of the methods are now integrated into the transformation variability. To fully reach this goal, it is important to use as many independent registration methods as possible. As a consequence, computing significant bronze standard estimations requires heavy computations.

Criterion 1.1 is in fact the log-likelihood of the observations $T_{i,j}^k$ assuming Gaussian errors around the bronze standard registrations with a variance σ_r^2 on the rotation and σ_t^2 on the translation. An important variant is to relax the assumption of the same variances for all algorithms, and to unbias their estimation. This can be realized by using only $m-1$ out of the m methods to determine the bronze standard registration, and by using the obtained reference to determine the accuracy of the last method (a kind of leave-one-method-out test). This uncertainty is then propagated into the final bronze standard registration (including all methods) to estimate its accuracy.

1.3 Follow-up of brain tumors evolution in radiotherapy

1.3.1 Data and registration problem

The targeted clinical application is the planning and follow-up of the radiotherapy of brain tumors. In such an application, several registrations are needed, with very different characteristics. Firstly, a monomodal but highly deformable atlas to patient registration that takes into account pathologies (tumors) is often performed to segment target volumes and organs at risk in the current image, in order to optimize the dose planning [Commowick et al., 2005]. To be more accurate, multimodal images are often taken (e.g. MR T1 and T2), sometimes with a contrast agent to enhance the tumor (gadolinium injected T1). A multi-modal rigid registration is needed to relate all these images in the same coordinate system. Last but not least, assessing the evolution of the tumor in follow-up images is important to evaluate the result of the treatment. This is the monomodal rigid registration problem that is considered for those experiments. Quantifying its accuracy is important to ensure the precision of the tumor evolution estimation in the assessment of the efficiency of clinical treatments. Precisely registered longitudinal studies may be used to validate the quality (reproducibility and accuracy) of segmentation algorithms used for radiotherapy planning. An accurate registration of longitudinal brain images is also needed in many other image-based studies of neuro-degeneratives diseases like multiple sclerosis or Alzheimer's disease [Dugas-Phocion, 2006].

To evaluate all these registration / segmentation problems, a database of 110 patients with 1 to 6 times points and MR T2, T1 and gadolinium injected T1 modalities was acquired at a local cancer treatment center (courtesy of Dr Pierre-Yves Bondiau from the "Centre Antoine Lacassagne", Nice, France) on a Genesis Signa MR scanner. Among them, 29 have more than one time point and were suitable to inclusion in the rigid registration evaluation study. Only

the injected T1 images were selected. These images are more demanding for registration than other MRI sequences as the gadolinium uptake is likely to vary at different time points, leading to local intensity outliers. All T1i images are 256×256×60 ×16 bits thus leading to a 7.8 MB size per image (approximately 2.3 MB when lossless compressed).

Four different registration algorithms were considered. Two of them are intensity-based: `Baladin` [Ourselin et al., 2000] has a block matching strategy optimizing the coefficient of correlation and a robust least-trimmed-squares transformation estimation; `Yasmina` uses the Powell algorithm to optimize the SSD or a robust variant of the correlation ratio [Roche et al., 2001]. The two others are feature-based and match specific crest lines (extracted using the third derivatives of the images) with different strategies [Pennec et al., 2000]: `CrestMatch` is a prediction-verification method and `PFRegister` is an ICP algorithm extended to features more complex than points. In the computation of the bronze standard registration, `CrestMatch` is used to initialize all the other algorithms close to the right registration. This ensures that all algorithms converge toward the same (hopefully global) minimum. A visual inspection is performed a posteriori on the bronze standard registration to ensure that this "optimal" transformation is indeed correct. As we are focusing on accuracy and not on the robustness, this initialization does not bias the evaluation.

1.3.2 Accuracy results

The evaluation procedure was run on the 29 selected patients with $\sigma_r = 0.15$ degrees, $\sigma_t = 0.42$ mm and a χ^2 value of 30. A high number of registration results were rejected in the robust estimation of the bronze standard transformations. A visual inspection revealed that there was a scaling problem along the z axis and a and shear problem in the yz plane for one of the images involved in each of these rejected registrations. A detailed analysis of the DICOM headers showed that the normal to the slices (xy plane), given by the cross product of the `Image Orientation` vectors, was not perfectly parallel to the slice trajectory during the acquisition (axis obtained from the `Image Position` field, i.e. the coordinates of the first voxel of each slice.). This tilt was found to be +1.19 degree in most of the images and -1.22 degree in 13 images. It seems that nothing in the DICOM standard ensures that 3D images are acquired on an orthogonal grid: it would be interesting to better specify the acquisition protocols on the MR workstation. There, we can see that the bronze standard method is able to detect subtle flaws in the images that ever the radiologists had not noticed.

Thus, images are not in an orthogonal coordinate system and should be either registered with an affine transformation (which adds 6 additional parameters among which only one -the tilt- has a physical justification) or the tilt should be taken into account within the rigid registration algorithm, but this solution was not implemented for the considered algorithms. As the tilt was small, we chose not to resample the images (in order to keep the original image quality), but rather to perform an uncorrected rigid registration within the group of images with

Number of time points:	2	3	4	6
Registration per patient (and per algorithm):	2	6	12	30
Patients (including/without tilted images):	15/**15**	6/**7**	7/**2**	1/**1**
Total number of registrations:	120/**120**	144/**168**	336/**96**	120/**120**

Table 1.1: Summary statistics about the image database used.

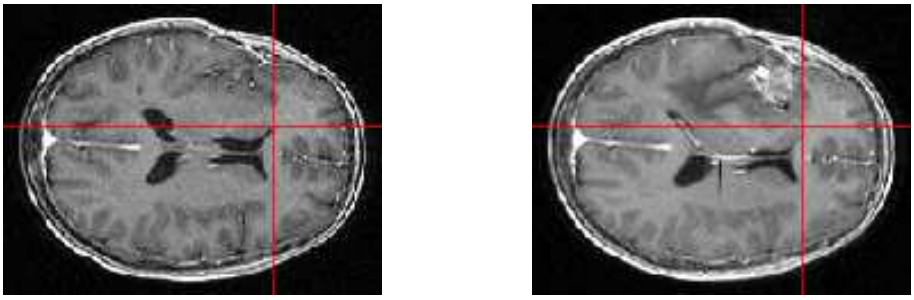

Figure 1.3: Example of a slice of two registered 3D images with a high deformation. One can clearly see that the tumor growth has pushed the right hemisphere and several different rigid transformations may locally account for the differences between the two brains.

a positive tilt only. This led us to remove 13 images among the 82, and 4 patients for which only one image was remaining (the statistics on the remaining number of patients, images and registrations are given in table 1.1).

The bronze standard application was run again with the same parameters on this reduced database of 25 patients. This time, only 20 registrations were rejected, among which 15 were concerning two patients with a very high deformation in the tumor area, leading to some global deformations of the brain (Figure 1.3). In that situation the rigid motion assumption does not hold any more and several ”optimal” rigid registrations may be valid depending on the area of the brain. The last 5 rejected transformations involve two acquisitions with phase-encoded motion artifacts which impacted differently feature-based and intensity-based registration algorithms, leading to two non-compatible sets of transformations. However, it was not possible to visually decide which result was the “right” one.

Excluding these 20 transformations which correspond to special conditions where the rigid assumption does not really hold, we obtained mean errors of 0.130 degree on the rotations and 0.345 mm on the translations. The propagation of this error on the estimated bronze standard leads to an accuracy of 0.05 degree and 0.148 mm. We then determined the unbiased accuracy of each of the 4 algorithms by comparing its results to the bronze standard computed from the 3 others methods. Results are presented in table 1.2 and show slightly higher but equivalent values for all algorithms.

This experiment demonstrates that the bronze standard method can be precise enough to detect very small deviations from the rigidity assumption (tilts of 2 degrees) in images, and that the 4 used rigid registration algorithms actually reach a subvoxel accuracy of 0.15 degree

Algorithm	$\sigma_r(deg)$	$\sigma_t(mm)$
CrestMatch	0.150	0.424
PFRegister	0.180	0.416
Baladin	0.139	0.395
Yasmina	0.137	0.445

Table 1.2: Accuracy results

in rotation and 0.4 mm in translation for the registration of longitudinal T1 injected 1x1x2mm images of the brain. In the next section, we propose to estimate the impact of a lossy compression algorithm on the performance of the registration. This is another example that will demonstrate the power of the bronze standard method.

1.4 Impact of lossy compression on registration

With the generalization of digital image acquisition and manipulation devices, an increasing number of medical images are archived in digital warehouses. Manufacturers provide DICOM compliant devices interfaced to local storage facilities and PACS. The emergence of multi-sites PACS and technologies such as data grids eases the integration and archiving of medical data at a large scale. Furthermore, recent regulations show a trend for long term archiving of patient data. Given the tremendous amount of radiology data acquired daily in clinical centers (tens of TBytes per year) and the will for long term archiving, optimizing storage space is increasingly needed [Germain et al., 2005].

Image compression can lead to drastic data size reduction and compression algorithms, such as JPEG, have been included in the DICOM standard. Lossless compression ensures a perfect reconstruction of the compressed data but leads to the lowest compresion ratio: in the range of 3.3 to 3.9 for the brain MRIs with a large black background described in section 1.3.1 (see figure 1.3). Compression with loss can achieve much better compression ratios but at the cost of approximative reconstruction. In the medical area, the use of lossy compression should be considered with care given the sensitivity of image-based diagnosis and knowing that it will be impossible to recover the original data. Most often, in the current practice, only lossless JPEG is considered to compress DICOM data, if any compression is applied at all.

A trade-off has to be found between efficient image archiving and the quality of archived data. In the literature, a growing interest for multi-dimensional medical data compression recently appeared [Menegaz and Thiran, 2002, Unser et al., 2003, Kassim et al., 2005]. The authors often let to the user the choice of the compression factor and therefore the image quality. An important question is the impact of lossy compression on automated medical image analysis procedures. Some recent studies show that a reasonable level of lossy compression may remain acceptable in this case. For instance, Raffy *et al* [Raffy et al., 2006] made a quantitative

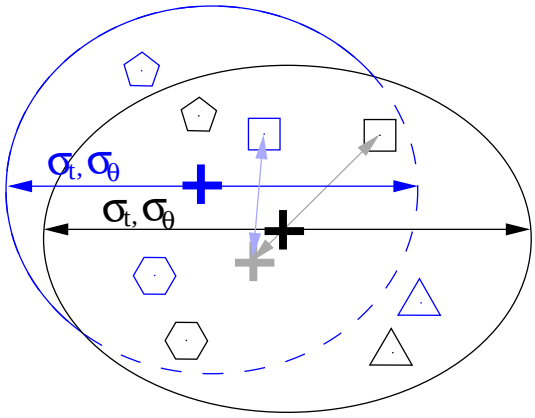

Figure 1.4: Illustration of the bronze standard method applied to uncompressed and compressed images (representation of the transformations in the 2D plane). Each registration algorithm (identified by a given shape) produces transformations in the compressed (blue) and uncompressed (black) cases. Bronze standards are depicted by crosses. Ellipses represent covariances. The accuracy of a particular algorithm is determined with respect to the bronze standard obtained *without* this algorithm (see gray cross and arrows for the "square algorithm").

evaluation of the impact of an increasing compression factor on the computed-aided diagnosis to detect pulmonary nodules. The study shows that the detection performance of solid lung nodules did not suffer until a compression ratio of 48. In this section, we consider the impact of lossy compression on rigid registration algorithms thanks to the bronze standard method. An experimental framework for estimating the impact of compression on accuracy, repeatability, and robustness is first described and then applied to the clinical problem of the follow-up of brain radiotherapy, as it was done in section 1.3.

1.4.1 A framework for evaluating the impact of compression

The founding hypothesis of this evaluation framework is to consider the transformations obtained from the uncompressed images as the reference for the evaluation. Then, the goal is to estimate to what extent the compression makes the registration results deviate from their original locus in terms of robustness, repeatability and accuracy.

1.4.1.1 Building the reference registration with the bronze standard method

On uncompressed images, the reference registration is built using the statistical bronze standard method. Figure 1.4 diagrams the bronze standard notations for the compression study. Each algorithm is represented by a shape and produces transformations in the compressed (blue) and uncompressed (black) cases. Bronze standards are depicted by crosses. The reference for the evaluation is built exclusively from the uncompressed images (black items). Outliers are first removed thanks to the χ^2 threshold of equation 1.1 and visualization checking as explained

above. The uncompressed bronze standard (black cross) is then computed and the standard-deviations of the transformations (σ_θ and σ_t) are measured. Those variances are re-injected in the minimization procedure of equation 1.1 which is iterated until they converge towards a stable estimation. They characterize the repeatability without compression. The accuracy of each algorithm will be obtained from the average distance between its measured transformations and the standard built from the remaining methods (gray cross and arrow on figure 1.4).

1.4.1.2 Evaluating the robustness

The number of outlier transformations gives an estimation of the robustness of the algorithms with respect to the compression. Outliers should be detected by an exhaustive visual checking, as for the uncompressed case. To avoid this tedious manual operation, one can rather rely on an automatic comparison *with the uncompressed reference* which has already been validated. This comparison is made thanks to the χ^2 test included in the mean computation of equation 1.1. Among the transformations rejected by the χ^2, a visual inspection has to be performed to determine whether they effectively correspond to wrong local minima (when it is obvious that a manual registration can lead to a better result). In this case, the whole patient is removed, for each algorithm: the absence of a specific algorithm for a given patient could bias the quantification of the accuracy of the remaining ones. Moreover, to allow a fair comparison between the compression ratios, patients leading to a wrong local minimum in *any* of the compression ratios are excluded for the repeatability and accuracy studies. Otherwise, it would be likely that high compression ratios would have been evaluated on less patients than lower ones, thus leading to potential artificial standard-deviation reduction.

1.4.1.3 Evaluating the repeatability

For each compression ratio, the repeatability is measured by the variances σ_θ and σ_t of the transformations obtained *from the compressed images only*, after having removed the patients leading to a wrong local minimum in one of the compression ratio by comparison with the uncompressed reference. Repeatability is pictured by ellipses on figure 1.4. It is determined without performing any χ^2 test in the distance of equation (1.1). Indeed, due to potential biases on compressed images, one transformation may be considered as an outlier with respect to compressed images while it is an inlier for uncompressed images (and vice versa). This is the case for instance of the blue triangle in figure 1.4.

1.4.1.4 Evaluating the accuracy

The transformations obtained from the uncompressed images are considered as the reference for the evaluation. The accuracy of each algorithm is computed by measuring the mean distance of compressed transformations to the uncompressed reference. To avoid biases, the evaluated

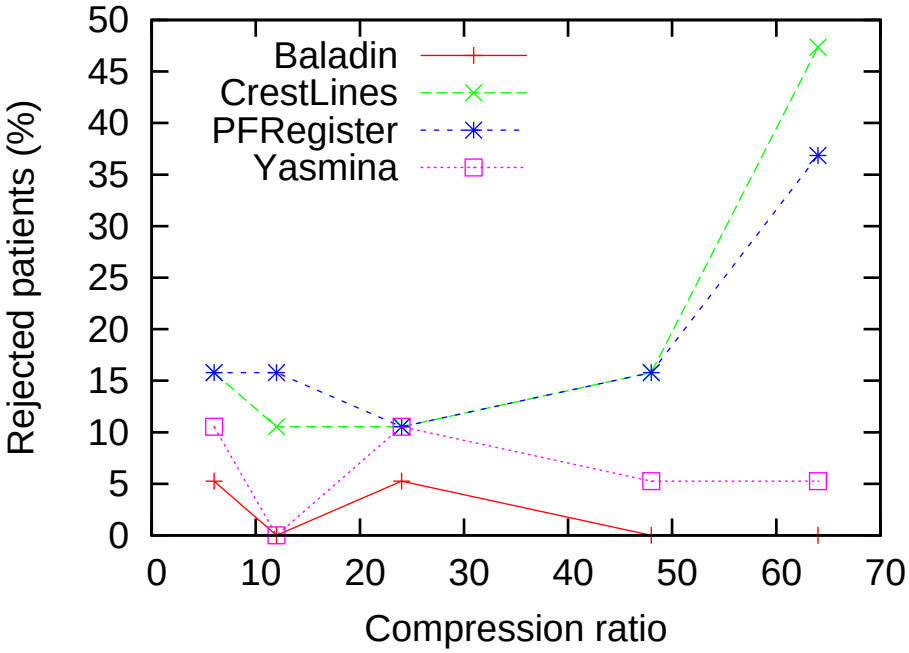

Figure 1.5: Ratio of outlier patients with respect to the compression ratio

registration algorithm is excluded from the algorithms used to build the uncompressed reference. It should be noticed that taking into account uncompressed images to build the reference does not imply that the accuracy is always worse for compressed images. It is for instance the case of the transformation of the algorithm depicted with a square on figure 1.4: compression has brought it closer to the bronze standard obtained without compression. This could for example be the consequence of a smoothing effect resulting from the compression.

1.4.2 Experiments

Experiments were made with the same setup as in section 1.3. The related database has been compressed at compression ratios 6, 12, 24, 48 and 64, with the 3D-SPIHT algorithm [Kim and Pearlman, 1997]. SPIHT is a zero-tree-based compression algorithm that is known to have produced some of the best results in 2D images coding. It has been extended to 3D and adapted to medical images [Xiong et al., 2003]. Figure 1.7 shows an image and the effect of 3D-SPIHT compression on it for a compression ratio of 64.

1.4.2.1 Impact of compression on the performance of algorithms

Robustness. The ratio of outlier patients is plotted on figure 1.5 for each algorithm. `Baladin` is the most robust method (at most 1 patient is rejected by the χ^2 test). `Yasmina` is also very robust, with 1 or 2 rejected patients. For those two algorithms, the behavior does not seem to be monotonic with respect to the compression ratio: some patients are rejected for low ratios but are again accepted for higher ones and vice versa. The good robustness of those methods

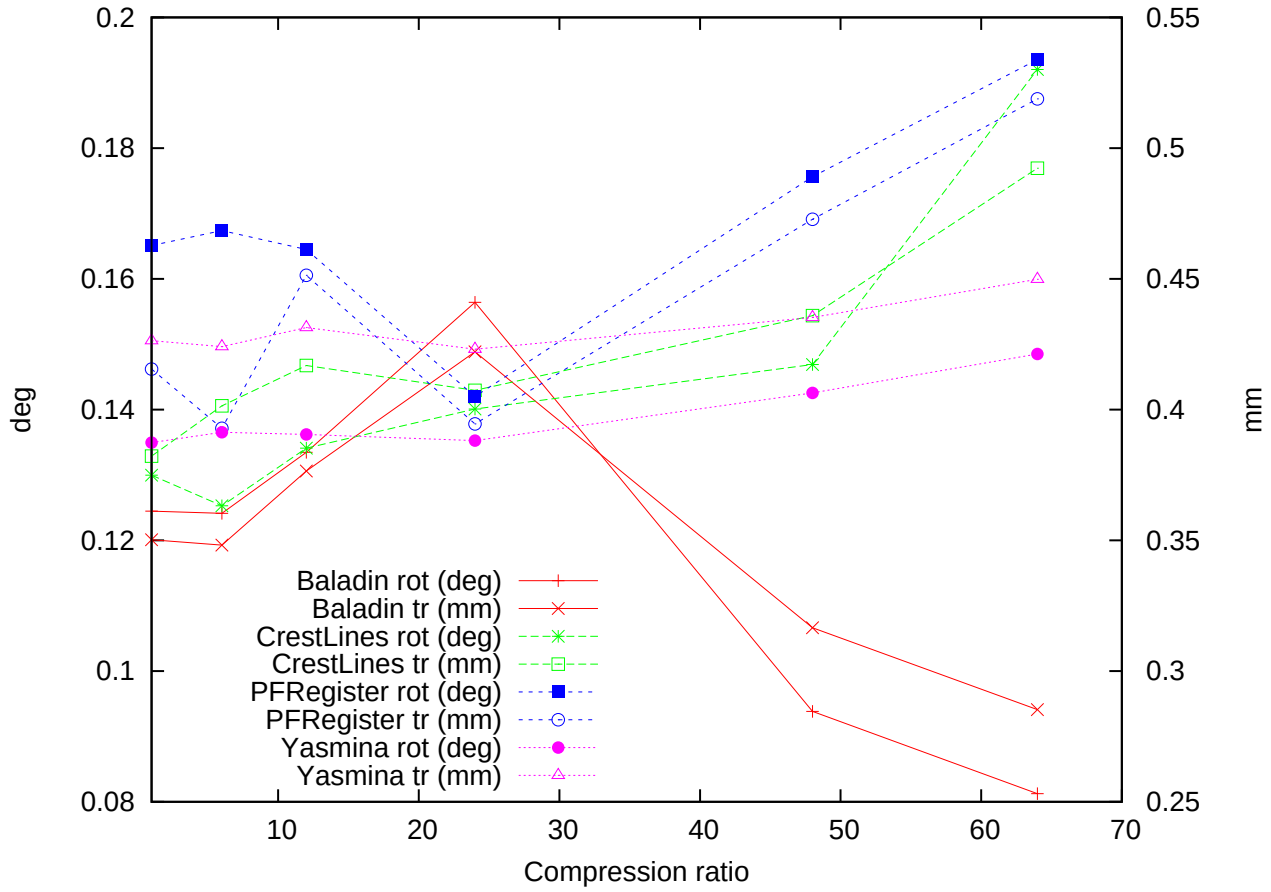

Figure 1.6: Accuracy of the algorithms.

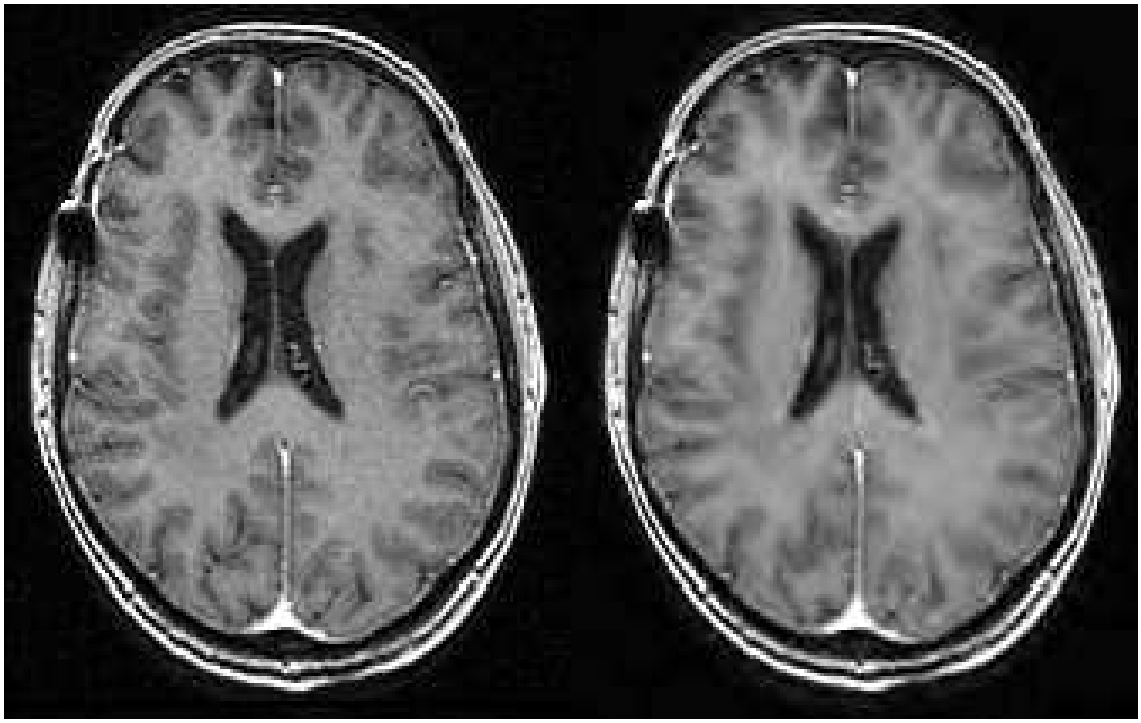

Figure 1.7: Image slice without (left) and with (right) compression (compression ratio = 64). The main structures are still well identified but a blurring effect is clearly visible.

may be a consequence of their multi-scale strategy: they both use a pyramid of under-sampled images and initialize the input transformation of a given sampling level with the result of the upper one. The robustness of the crest-lines methods is lower, which may be explained by the extraction of the crest-lines at a single scale. The number of rejected patients is almost constant until a compression ratio of 48, with 2 or 3 patients rejected. For a compresion ratio of 64, it highly increases up to almost 50% of rejected patients for `CrestLines`. At this compression ratio, `PFRegister` performs a little bit better, with only 37% of rejected patients, which could be explained by a more robust matching of the crest-lines. The fact that feature-based methods are less robust to compression may come from the use of first to third order derivatives of the image to extract crest-lines, which are very likely to be impacted by the compression procedure. To illustrate it, we represented on figure 1.8 the longest detected crest lines with and without compression. The compression significantly disturbs the detection of those lines.

It also has to be noticed that the study of the robustness led us to rapidly identify experimental mistakes in the data base. For instance, switches among the image names were easily detected by the statistical procedure, whereas a visual check of all the images would have required a significant time and is clearly not scalable.

Repeatability. Among the patients rejected for at least one method, 4 were corresponding to wrong local minima for at least one compression ratio. They were removed and the repeatability and the accuracy were evaluated on the remaining 296 transformations for each algorithm.

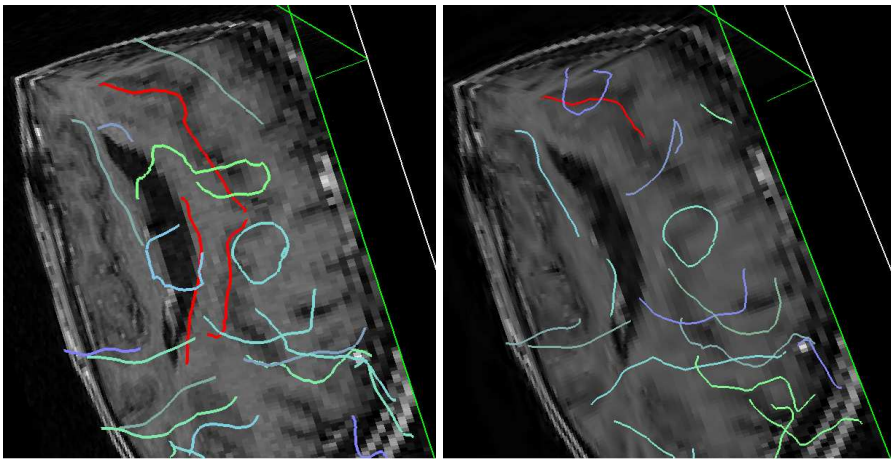

Figure 1.8: 3D views of the longest crest lines detected without (left) and with compression (right). The red line riding through to the upper part of the visible ventricle is not detected on the compressed image. Two other red lines on the left are going through brain sulci. Only a small part of one of these is visible on the right side.

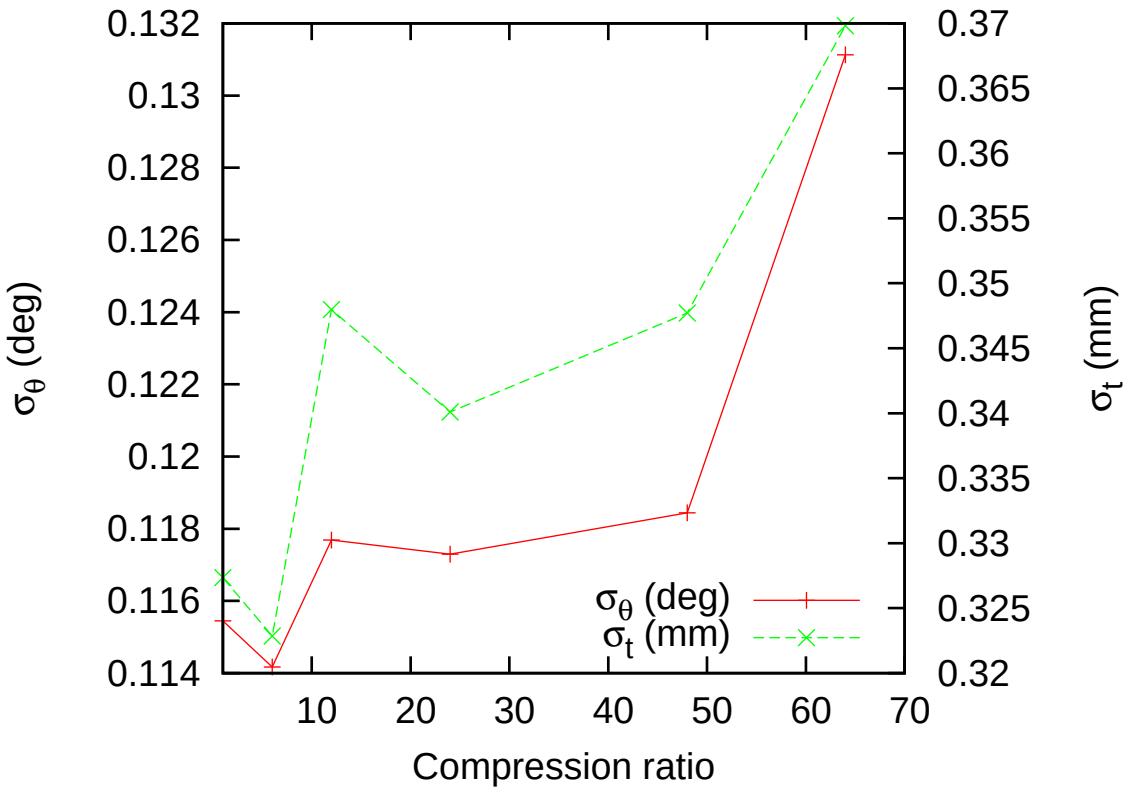

Figure 1.9: Mean variances of the transformations with respect to the compression ratio.

Figure 1.9 plots the evolution of the mean variances of the transformations. Despite a subtle improvement of 1% at the compression ratio 6, the main behavior is an impairment of 4 to 6% before a strong decline of 13% for a compression ratio of 64.

Accuracy. Figure 1.6 displays the accuracy of the algorithms with respect to the compression ratio. The accuracy of feature-based methods is highly reduced at a compression ratio of 64. At this compression level, the mean error of `CrestLines` has increased by 48% for the rotation and 29% for the translation whereas the one of `PFRegister` has increased by 17% for the rotation and by 25% for the translation. `Yasmina` is quite insensitive to the compression: its mean error only increases by 10% for the rotation and by 5.5% for the translation. More surprisingly, after a brief rise until a compresion ratio of 24, the accuracy of `Baladin` is improving: for a ratio of 64, it is 34% better than without compression for the rotation and 18.5% better for the translation. The fine behavior of `Baladin` and `Yasmina` can be explained by the fact that both algorithms include a multi-scale handling that may compensate the effects of potential noise introduced in the images. Moreover, in `Baladin`, only the most significant blocks (the ones with the largest standard deviations) are considered for the block-matching.

1.5 Conclusions and motivations for the following

1.5.1 Medical image analysis results

In this chapter, the application of the bronze standard evaluation framework was studied on a clinical use-case related to the follow-up of brain radiotherapy. Rigid registration algorithms were evaluated on a database of brain follow-up MRIs. Experiments demonstrate that the bronze standard method can be precise enough to detect very small deviations from the rigidity assumption (tilts of 2 degrees) in images, and that the 4 rigid registration algorithms used actually reach a subvoxel accuracy of 0.15 degree in rotation and 0.4 mm in translation for the registration of longitudinal T1 injected 1x1x2mm images of the brain.

An evaluation of the impact of the 3D-SPIHT compression algorithm on the registration shows that the robustness, repeatability and accuracy are little impacted below a significant compression ratio (48), in particular if the registration algorithm has a good multi-scale handling. Beyond this threshold, the tested methods based on crest-lines are highly penalized: half of the patients can be considered as outliers and their accuracy is lowered by 50%. Surprisingly, compression improves the registration accuracy (up to 30% for `Baladin` on our setup) probably because the registration algorithm focuses on informative subsets of the image. Thus, lossy compression does not seem to be problematic for the registration until a given compression ratio (48 in our study), which looks similar to the results found in [Raffy et al., 2006] on another clinical problem. Evaluating the impact of other compression algorithms on different

registration methods should yet be done to allow more general conclusions.

In conclusion, the bronze standard method is able to estimate the performances of rigid registration algorithms in the absence of gold standard and to evaluate the influence of parameters such as the compression ratio of the images. Moreover, it is highly scalable and makes outliers easily detectable whereas a visual check of a large amount of transformations could not be done in a reasonable amount of time.

1.5.2 The need for grid workflows

From a computer science point of view, the bronze standard is nicely described as a *workflow* (a graph of connected processings) as it requires many mostly independent registrations with several algorithms on different data sets. Figure 1.10 pictures the precedence constraints between the algorithms implied in the experiments presented in this chapter. In the reminder of this thesis, we particularly focus on this application as it is representative of a large class of medical image analysis applications. Indeed, it is common to build image analysis procedures from basic image processing algorithms. The representation and the execution of such procedures as workflows enable a generic processing of many similar image analysis tasks. In addition, there are many medical image analysis procedures involving large data sets for different needs (statistical studies over populations, performances evaluation such as the bronze standard, epidemiology, ...). They require heavy computations, dominated by this data-parallel nature. The workflow-based approach eases the deployment of such computations over remote parallel grid resources. It decouples the application from the execution infrastructure, thus releasing the application developers from the most complex computational problems, especially parallelization. As explained in chapters 2, 3 and 4, the workflow-based design of this application will allow to transparently exploit this parallelism on a grid.

The bronze standard runs presented in this chapter are typically 30 hours long on state-of-the-art PCs (see chapter 5 for detailed benchmarks of the algorithms), which would make the whole compression experiment of section 1.4 7.5 days long (regular run + 5 compression ratios). The first benefit expected from a grid execution of this application is a reduction of the total execution time of the application. As demonstrated in chapter 5, the time of this week-long experiment could be reduced to 4 hours on a dedicated cluster of 60 nodes and to 18 hours on a shared production grid in similar conditions. Based on the analysis performed in chapter 6, further optimizations presented in chapters 7, 8 and 9 aim at bridging the gap between those two kinds of grid infrastructures. The benefit expected for the bronze standard application from the availability of computing power is twofold. First, the reduction of its execution time brings the application closer to a clinical exploitation: even if a reasonable execution time is certainly not the only parameter allowing a clinical usage, daylong runs remain prohibitive in such conditions. Second, from a medical image analysis point of view, the computing power offered by a grid allows wider experiments producing more relevant results. The quality of the

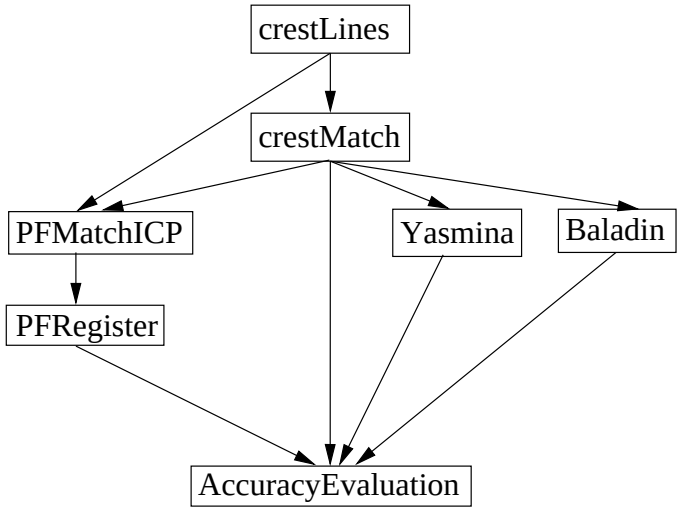

Figure 1.10: Dependencies between the algorithms implied in the bronze standard experiments presented in this chapter. Arrows denote precedence constraints. Each part of a registration algorithm (`crestLines`, `crestMatch`, `Yasmina`, `Baladin`, `PFMatchICP` and `PFRegister`) is iterated on the whole image database, thus triggering a natural data parallelism. The CrestMatch feature-based method tested in this chapter is made of two independent part: `crestLines` extracts salient lines from the images and `crestMatch` finds a transformation between them. As explained in this chapter, the 3 other registration algorithms are initialized with the result of CrestMatch. Those 3 methods are completely independent, thus benefiting from workflow parallelism. The PFRegister method (composed of `PFMatchICP` and `PFRegister`) is a robust variant of CrestMatch and also relies on the `crestLines` results. The accuracy evaluation runs once all the registrations have been computed.

accuracy results provided by the bronze standard application is increasing with the amount of registrations performed and the number of algorithms used. Results computed from a few pairs of images registered with a single algorithm would measure the *repeatability* of the method, *i.e* its variability around a mean potentially far from the truth. As the amount of computed transformations and algorithms used increases, those results are converging towards *accuracy*: more and more sources of variability are taken into account and biases are averaged out by the bronze standard estimation. Thus, the available computing power is directly related to the quality of the medical image analysis results obtained.

Another motivation for the workflow design of the bronze standard application presented in the next chapter is its need for algorithms sharing. Integrating registration algorithms developed by several different research teams in a bronze standard evaluation procedure widens the spectrum of covered biases and further increases the relevance of the method. Making medical image analysis *services* available and composable in applicative workflows then becomes a crucial need.

Finally, even if it is not extensively studied in this thesis, the gridification of the bronze standard application provides by itself a way to share data by benefiting from the data management facilities offered by contemporary grid middlewares. Similarly to algorithms sharing, sharing data enhances the significance of the bronze standard method because it makes possible the building of large scale image databases: again, the relevance of the accuracy results yielded by the method is increased by the sweeping of a large number of variability sources, which is only possible through the consideration of large image databases.

Chapter 2

A taxonomy of workflow approaches for medical image analysis applications

Contents

This chapter deals with the description of application workflows on grid infrastructures. We propose a classification of workflow description approaches with respect to the specification of functions, data and resources in the language. This classification distinguishes five workflow classes, each of them corresponding to a particular user profile. Based on this taxonomy, the main existing workflow languages are then reviewed.

Ce chapitre traite de la description d'applications construites par chaînes de traitements sur des infrastructures de grille. Nous proposons une classification des approches de description selon la présence des fonctions, des données et des ressources dans le langage. Cette classification distingue cinq classes de chaînes de traitements, chacune d'entre elles correspondant à un profil particulier d'utilisateur. A partir de cette taxonomie, nous passons en revue les principaux langages de description de chaînes de traitements existants.

The study made in this chapter is motivated by the need of applications to share algorithms across institutes and administrative domains. In particular, as detailed in the previous chapter, the bronze standard application greatly benefits from it as it allows wider registration algorithms comparisons and produces more relevant results. The state-of-the-art solution for addressing such a problem is to wrap codes into *services* and to *compose* them into a *workflow*. Many workflow approaches have been envisaged, answering needs issuing from various users profile ("domain" scientists with scripting skills, computer scientists testing parallelization methods, end-users without any programming background, ...). After a historical overview of code reusability methods leading to the emergence of services, workflow approaches are reviewed and classified in section 2.2. Our point is to determine a workflow approach that could allow an easy workflow representation and usage for an end-user (*e.g* a clinician), a familiar programming model for the application developer (*e.g* the medical image analysis scientist) and an efficient grid deployment for the computer scientist. The typical envisaged scenario would be to have clinicians understanding and using workflows composed by medical image analysis scientists from existing services and execute them efficiently on a grid platform. The clinician could thus focus on his/her work without having to deal with problems concerning the execution and distribution of the processings.

2.1 Sharing algorithms: from assembly to services

Sharing algorithms is the modern vision for code reusability which has been considered for a while in computer science: the emergence of Service-Oriented Architectures (SOA) is the result of a long process to foster code reusability in software engineering. Assemblies, the earliest programming languages for micro-processors, were architecture and system specific. Writing an application using such languages requires a deep knowledge of the target architecture (such as the number and size of registers) and the resulting code is definitely not portable. Yet, the use of procedures and libraries revealed an early concern about the necessity to reuse proven code as much as possible. Then, compiled languages such as C became independent from the architecture and allowed software projects to be easily portable on various kind of platforms. Object-oriented languages such as C++ went further in code reusability by providing the ability to define classes that were supposed to be reusable in a number of different applications. For

instance, in the medical image analysis domain, the Insight Tool Kit (ITK)[1] is a currently widespread library intensively used for sharing state-of-the-art algorithms among scientists. Yet, C and C++ languages remain highly dependent on the operating systems: some APIs are different in nature from an OS to another one and porting a code may involve a significant burden (see for instance the differences between (win)sockets APIs on Linux and Windows, the availability of process forking, ...). Later on, the Java language abstracted from the operating system thanks to the use of (system-dependent) virtual machines that are able to interpret pre-compiled code and execute it on-the-fly. Still, as noticed by Gannon in [Gannon, 2007]:

> " Object-oriented programming was thought to be the solution to reusability but it only got us part of the way. Object-oriented concepts are powerfull but they do not guarantee that a class built for one application can be easily reused in another. To build truly reusable software, one must design the software as part of a component architecture that defines rules and contracts for deployment and reuse. "

The concept of *component* programming cited here has been introduced as early as in 1968 by McIlroy who suggested an analogy between industrial techniques and software production. In [McIlroy, 1968], he underlines the ideas of sub-assemblies and interchangeable parts that could both be applied to industrial products and software. He also claims that software components have to be considered as black boxes, offering families of parametrizable *on-the-shelf* components for a given job. Last but not least, McIlroy identifies the need for being able to compile and use the components on various architectures without performance loss. Nowadays, a common definition of a component is the one of Szyperski [Szyperski, 2002]:

> " A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties. "

A contract describes the function implemented by the component independently from its implementation and a definition of the context dependencies could for instance be found in [Gannon, 2007]:

> " By context dependencies, we refer to the conditions that must be satisfied by the host environment in order to operate properly. For example, does the component require a specific version of the JVM or libraries ? "

For instance, the WComp application development environment dedicated to the adaptation of component assemblies on heterogeneous and dynamic resources considers the context as the presence or absence of software components, of resources (software subsystems) and specific devices [Cheung-Foo-Wo et al., 2006].

[1] http://www.itk.org

Hot debates have been conducted in order to determine whether objects could be considered as components or not. For instance, in [Pfister and Szyperski, 1996], the authors explain why object oriented programming does not fulfill the needs required for building component-based architectures. In particular, they explain that:

> " A component is defined as a collection of cooperating objects, with a clearly defined boundary to other objects or components. Objects inside a component typically are intertwined tightly, while interaction across the component boundary is relatively weak. "

This distinction is not purely semantic but can also have severe consequences on the performance of the software systems. Indeed, since method calls are not always local procedure calls anymore but can be remote, which is several orders of magnitude slower, it is important to be aware of the boundary of a set of locally interacting objects, which justifies this definition of components. A complete zoo of component models and implementations have been proposed, among which Microsoft COM/DCOM [Box, 1997] and OLE, Java Beans, Sun's Enterprise Java Beans [Monson-Haefel, 2001] and related simplified models such as Spring[2] or Pico[3], Apache Avalon[4], and the Corba Component Model (CCM) which inspired the Corba Component Architecture [Armstrong et al., 1999] which is for instance implemented in SciRunII [Zhang et al., 2004] or XCAT3 [Krishnan and Gannon, 2004]. Based on the Fractal[5] [Bruneton et al., 2004] component model, Proactive components are formed of one or several so-called active objects and offer grid computing facilities such as distribution, asynchronism, mobility or security [Baduel et al., 2006].

Services are defined as an exposed piece of functionality with three properties:

1. The interface contract to the service is platform independent.
2. The service can be dynamically located and invoked.
3. A service does not call another service (loose coupling).

The two firsts points are motivated by the development of distributed applications across the Internet and the third one ensures the independence of a given service with respect to other ones. This property strongly distinguishes services from objects or components that may require dependencies (such as the presence of a given library, or the connection with another component) to be fulfilled in order to run properly. Services live in Service-Oriented Architectures (SOA) that are basically composed of three actors: the service *provider* runs the service on a particular endpoint (*i.e* a port and Internet address) and publishes its interface in a service *broker*, which allows the *consumer* to discover the service and to invoke it.

[2] http://www.springframework.org
[3] http://www.picocontainer.org
[4] http://avalon.apache.org
[5] http://fractal.objectweb.org/

Web-Services are the most common implementation of services and have been standardized by the W3C[6]. A Web-Service is an endpoint whose interface is specified by a Web-Service Description Language (WSDL) document and that is accessible through the Simple Object Access Protocol (SOAP). As many web technologies, Web-Services specifications rely on XML to ensure platform and language independence. Web-Services can be dynamically discovered from repositories that may for instance conform to the Universal Description Discovery and Integration (UDDI) standard. However, UDDI has recently been highlighted to be severely limited [Atkinson et al., 2007] and Web-Services discovery is still an extremely active research domain. Major criticisms of Web-Services coming from the grid computing community are the fact that they are stateless and bound to a particular resource. OGSA and WSRF are evolutions of the Web-Services initial specification that, among other evolutions, allow a service (i) to be statefull and (ii) to be dynamically deployed on a resources thanks to the use of a service factory [Wagstrom et al., 2002].

2.1.1 Composition models

Components and services may be composed through different approaches to build an application. Workflows are one of the software composition paradigms that emerged jointly with the concept of software components. A brief overview of component composition is thus needed prior to the introduction of workflows. The concept of component composition appeared in the 1970s, after McIlroy introduced the first reference to *software components* in 1968 [McIlroy, 1968]. A component may have input and output *ports*, corresponding to the input and output arguments of the underlying function and that are used by the composition systems. In [Gannon, 2007], Gannon sketches a state of the art of software composition systems. The *direct composition* mechanism consists in connecting input ports to output ones, thus building an application as a data flow graph. Yet, Gannon underlines that applications involving components that have functional or method interfaces or components that have interfaces based on sending and receiving one-way messages may not be easily described by such graphical notations. Gannon also notices that components must be able to maintain input queues or to block upstream output ports in case of multiple invocations. Strategies also have to be defined when the component receives a different number of inputs on its ports. We study those strategies in a parallel environment in section 4.3.1 of chapter 4. Gannon also identifies a problem related to Web-Services composition: they return a response to the caller rather than to another Web-Service. To avoid that, a proxy can be implemented that redirects the output to the right component, as it is done in Kepler, Triana and Taverna that are workflow composition systems able to integrate Web-Services. Yet, it requires the workflow manager to centralize the whole data exchanged in the workflow, which can raise performance issues.

[6] http://www.w3.org

Gannon also cites the *bus-based* composition mechanism, as a metaphor from the hardware design. In this scheme, the component framework provides a message bus, on which components are plugged with a unique identifier. Each component listens on the bus and captures messages that are sent to it. This model is for instance adopted by JXTA [Brookshier et al., 2002]. As in hardware systems, the bus-based composition system reduces the amount of connections required to build the application. Thus, adding and removing components is easier than in a direct composition paradigm. The bus is also a central control entity that could facilitate the management of the application such as for instance the integration of transversal concerns like security. The publish-subscribe model is a particular kind of bus-based composition. In this model, a component may subscribe to some events once it is connected to the bus. Then, it only receives messages corresponding to the subscribed events.

Defining *interactions* between instances of components is another approach to composition [Blay-Fornarino et al., 2004]. Their main features are (i) to be independent from the component model and language and (ii) to provide dynamic adaptation facilities to applications. Interactions allow direct communications between the component in order to prevent the application from relying on a centralized manager which is potentially a bottleneck. They may connect components from different frameworks and can be dynamically created and destroyed during the execution of the application. Interactions can be defined using a dedicated language, the Interaction Specification Language (ISL), which allows to define consistent interaction merging algorithms.

The goal of Aspect Oriented Programming (AOP) [Kiczales et al., 1997] is to be able to easily add or remove a transverse concern (such as security or logging) in a software architecture. Such a problem is studied for instance, in [Barais et al., 2006], where the authors specify a set of rules to automatize the integration of new concerns in a software architecture. Examples of implementations of AOP systems are AspectJ[7] and AspectC++[8]. AOP differs from classical component composition because it focuses on non-functional properties (*i.e* on properties that are not required for the global functioning of the application) rather than on the description of the application itself. AOP is used on top of a composition system.

A similar approach is meta-level programming that is also used on top of an object composition system. This concept allows to define meta-classes (*i.e* the class of a class) and thus to redefine the method call mechanism, the object creation process,... It is then possible to add code before and after methods invocations. Example of such an approach is the Common Lisp Object System (CLOS) [Bobrow et al., 1988].

The term *workflow* is used to denote the representation of an application in the services community as well as in grid computing. In both cases, it corresponds to the description of the logic of an application independently from the implementation of its components and from

[7] http://www.aspectj.org
[8] http://www.aspectc.org/

the target infrastructure. Workflows are a particular kind of software component composition model. They are studied in the remaining of this chapter.

2.1.2 Workflow definitions

The workflow management coalition[9] proposes the following definition of workflow management:

> " Workflow management is the automation of business procedures or "workflows" during which documents, information or tasks are passed from one participant to another in a way that is governed by rules and procedures."

This broad definition reflects the diversity of the application domains where workflows are used. Indeed, before being studied for the description of distributed applications, workflows have been used to describe the organization of production processes in companies as well as the interaction between several business entities. This definition focuses on the data transmission among interacting participants which can be denoted as *tasks*, *services*, *processes*, *transitions*, *activities*, *functions* or *components* depending on the workflow approach. The enactment of a participant can be called *invocation* (mostly for services), *execution* (for tasks), *firing* (for transitions and activities) or simply *call* (for a function or component). In the reminder of this thesis, those terms will be used indifferently, depending on the workflow context.

Workflow is a particular type of software composition system, where the participants of the workflow are the components to be assembled. In [Gannon, 2007], Gannon notices that workflows act at a different scale than software composition systems: they deal with human-scale processes that are scheduled over time. Similarly, in the field of distributed applications, workflows deal with coarse rather than fine grain parallelism which is better described with traditional parallel programming approaches such as MPI or OpenMP. Gannon also underlines that workflows refer to a centralized execution in which a single engine is responsible for the control of the process. In a grid context, this property of workflows has pros and cons. On the one hand, it is true that a centralized perspective simplifies a lot the control of the execution of the application in order to be able to provide a representation of the status of the application to the user. But on the other hand, centralization may lead to dramatic performance limitations, in particular when dealing with applications that involve large numbers (hundreds of thousands) of participants: the scalability of the application may then be highly disturbed by the centralized workflow approach.

In [Mayer et al., 2004], the authors propose a definition highlighting the platform-independence of the workflow definition:

[9] http://www.wfmc.org

> " We consider a workflow to be the organization of a structured application in an abstract fashion, such that the implementation of the atomic tasks being organized is independent from the organization itself. "

This aspect of workflow programming is crucial in the grid computing area, where applications are typically composed from heterogeneous codes and software, each of them having its own architecture or system requirements. It is also motivated by the emergence of service and component-based programming models that promote code reusability and platform-independence, as detailed in section 2.1. While traditional scripts (that are often considered as the ancestors of workflows because their are used as the glue among several executables, in particular in scientific applications that make an extensive usage of Perl scripts) are tightly coupled to the platform, workflows provide a representation of the logic of the application independently from the implementation. It is particularly important for grid applications, where the heterogeneity of the resources and middlewares is critical. Built on top of service-oriented architectures, workflows foster code reusability, thus reducing applications development time. As a consequence, workflows are increasingly cited as a transparent way to deploy applications on grids and a large amount of applications rely on them for a successful grid-ification. Examples from various domains are extensively described in [Taylor et al., 2007] whereas [Montagnat, 2006] focuses on medical image analysis applications.

Works related to service composition propose alternate definitions where the workflow itself is viewed as a service [Wagstrom et al., 2002, von Laszewski et al., 2004]:

> " We define the term workflow as a set of rules that define the interactions between a set of services in order to be composed into a meta-service. "

This definition allows a hierarchical composition of applications, starting with basic workflows of services, then exposing them as services themselves and finally composing those *composite* services to produce another application. Yet, some fundamental problems arise in such development processes, in particular when workflows sharing one or more services are composed [Nemo et al., 2007b, Nemo et al., 2007a]. In this case, redundant services invocations could occur, leading to performance or even semantic problems in the application. Actually, considering workflows as composite services breaks the loose coupling hypothesis of SOAs: the composite service is tightly coupled to the basic services that compose it.

Workflows may also be characterized by the use of a simple graphical language for end-users, thus easing code understanding and application development. Grids are expected to provide new methods for scientists, not restricted to computer science and the availability of simple programming environments is necessary. Workflows offer a unified and simple view of complex experiments that may gather heterogeneous codes from various developers and institutes. Barga and Gannon indeed noticed in [Barga and Gannon, 2007] that:

> “ The result is a workflow in which each step is explicit, no longer buried in Java or C code. Since the workflow is described in a unified manner, it is much easier to comprehend, providing the opportunity to verify or modify an experiment. ”

2.1.3 Workflows classifications

Several workflow classifications have been proposed in the literature. In [Gil, 2007], Gil distinguishes *templates*, *instances* and *executable* workflows. Templates capture the structure of the workflow independently from the data. It may for instance be defined by a medical image analysis scientist to describe the logic of its application independently from the data. A workflow instance specifies the data to be processed: it could be defined by the clinician to execute a workflow on a particular data set. The executable workflow is determined and optimized by computer scientists: it defines the data location and includes steps for data transfers. This kind of classification does make sense in grid computing, where an important effort is made in order to make a heterogeneous distributed computing infrastructure transparently accessible to the application developers and users. The workflow system is viewed here as a part of the middleware linking the abstract (template) application representation provided by the user to the (concrete) executable one required for a grid execution.

Yu and co-author proposed another taxonomy of workflow management systems for grid computing [Yu and Buyya, 2005a, Yu and Buyya, 2005b]. This classification includes some intrinsic workflow properties which are important from a computer-science point of view. In particular, their classification of workflow design approaches concerns (1) the workflow structure, (2) the workflow model and specification, (3) the workflow Quality of Service (QoS) constraints and (4) the workflow composition system:

1. The workflow structure consists in separating Direct Acyclic Graphs (DAG) from non-DAG workflows. As we will see, this distinction helps to determine the applicability of grid scheduling algorithms. Indeed, avoiding cycles in workflow graphs is restrictive but leads to predictable sets of tasks.
2. The workflow model is close to the one presented in [Gil, 2007] and by the literature related to the Pegasus workflow manager [Deelman et al., 2003]: it distinguishes concrete workflows (where resources are defined) from abstract ones.
3. The workflow QoS constraints (*e.g* time limit constraints) may be specified at the task level or at the workflow level.
4. Finally, the workflow composition system separates user-directed composition from automatic composition. The automatic composition seems to correspond to the mapping done by Pegasus between the metadata description of the required data products and a workflow containing information for data derivation of application components. In the user-directed composition, Yu distinguishes language-based and graph-based modeling.

Similarly, the Petri-Net approach adopted in [Hoheisel and Alt, 2007] leads the authors to group workflow description languages into two classes: script-like (programming language, complex semantics) and graph-based (a few basic graph elements). In the use-cases envisaged in this thesis, graphical programming is not crucial as the targeted application developers are medical image analysis scientists that have strong programming skills. Yet, a graphical workflow representation remains interesting as it allows a direct understanding of the application by the end-user (*e.g* the clinician), which was not the case with traditional scripting languages (shell, Perl, Python, ...). As noticed in the introduction of [Taylor et al., 2007], the shift away from the earliest script workflow representations to graphical workflows came from the need for using distributed resources. Indeed, the graph representation of a workflow application also provides a natural parallelization. Thus, graphical workflow representations are suitable exchange formats between the application developer, the end-user and the grid expert. However, in [Gannon, 2007], Gannon identifies some limitations of the expressiveness of graphical languages. In particular, he claims that in general, such graphical languages are not Turing complete, leading to some hard programming limitations.

Control and *data flows* are traditionally distinguished [Shields, 2007]. Data flow refers to approaches where the enactment of a participant is only conditioned by the availability of data items in its ports. A pure data flow is also called a *pipeline* [Rex et al., 2003]. On the contrary, control flow refers to the classical approach of imperative software programming, where the execution of instructions is conditioned by control structures such as `for`, `if`, `while` or `switch`. The term workflow is used to denote both the control and the data flow of an application [Hoheisel and Alt, 2007]. The approach adopted in this thesis tends to integrate the algorithmic logic in the components of the workflow as much as possible: the workflow is rather a mean to share existing algorithms and to benefit from coarse grain parallelism than a panoptic programming language which would lead to a more complex representation, hardly interpretable by end-users. Still, expressiveness limitations are expected with pipelines and simple control flow constructs may be required.

Business and *scientific* workflows are also distinguished, the former being said to focus on the control flow whereas the latter concentrates on data flow. Scientific applications are indeed sometimes described by pure pipelines. However, this distinction seems to be more and more inadequate: languages and engines from the business community are spreading into scientific communities [Slominski, 2007] and formal models such as Petri-Nets and π-calculus tend to unify both approaches. Barga *et al* study in [Barga and Gannon, 2007] the common points and differences between them. In [McGough et al., 2007], the authors also define scientific workflows as conceptual representations, where only the interactions between the tasks required to perform an experiment are described. The actual set of tasks required to produce the application and their interactions are then denoted as *middleware* workflow. Those definitions must be related to the distinction between abstract and concrete workflows identified in [Gil, 2007] and

exploited by the Pegasus workflow manager [Deelman et al., 2003]: the mapping from abstract to concrete workflow corresponds to the scheduling problem, which justifies this classification.

Finally, the expressiveness of the workflow language may be another way to sort workflow approaches. When dealing with scientific applications, workflow languages should remain quite simple: the logic of the application is embedded into the components which may be implemented using classical programming languages and the resulting representation of the application should be easily interpretable. Yet, as already stated, expressiveness problems could occur and it may be quite difficult to describe useful applications with too circumscribed languages. Methods to study the expressiveness of a workflow language include workflow patterns [van der Aalst and ter Hofstede, 2002, van der Aalst et al., 2003, Kiepuszewski, 2003] (the ability of the language to describe a set of pre-defined patterns is studied), schema relations [Mendling and Müller, 2003] (the XML schema of the languages are semantically compared) and the study of the Turing completeness, which is a formal way to prove that a language is able to implement any computable function [Lewis and Papadimitriou, 1981].

2.2 From formal workflow models to their execution

In this section, we propose a classification of workflow descriptions that aims at easing the choice of a workflow approach for a given category of users. This classification is based on the presence or absence of *functions*, *data* and *resources* in the workflow representation. It extends Gil's one [Gil, 2007] and it is based on the amount of information concerning the process execution that is provided inside the workflow description. Our classification is summarized on figure 2.1: existing languages will be studied from completely formal models to concrete schedules of tasks-graphs.

Such a classification is particularly suitable for the scenario that we envisage in this thesis. Indeed, it allows to precisely separate the concerns of the main actors of a medical image analysis grid workflow, namely:

- the medical image analysis scientist who develops the workflow and its components (the functions),
- the clinician end-user who instantiates the workflow on the data,
- the grid expert who performs the grid deployment and in particular the scheduling of the workflow on the resources.

Five main workflow classes can be distinguished:

1. Formal workflow models correspond to languages where no information is given about the nature of the implied activities, the amount and type of data processed and the used resources. Those models are suitable for workflow analysis because they offer an abstract

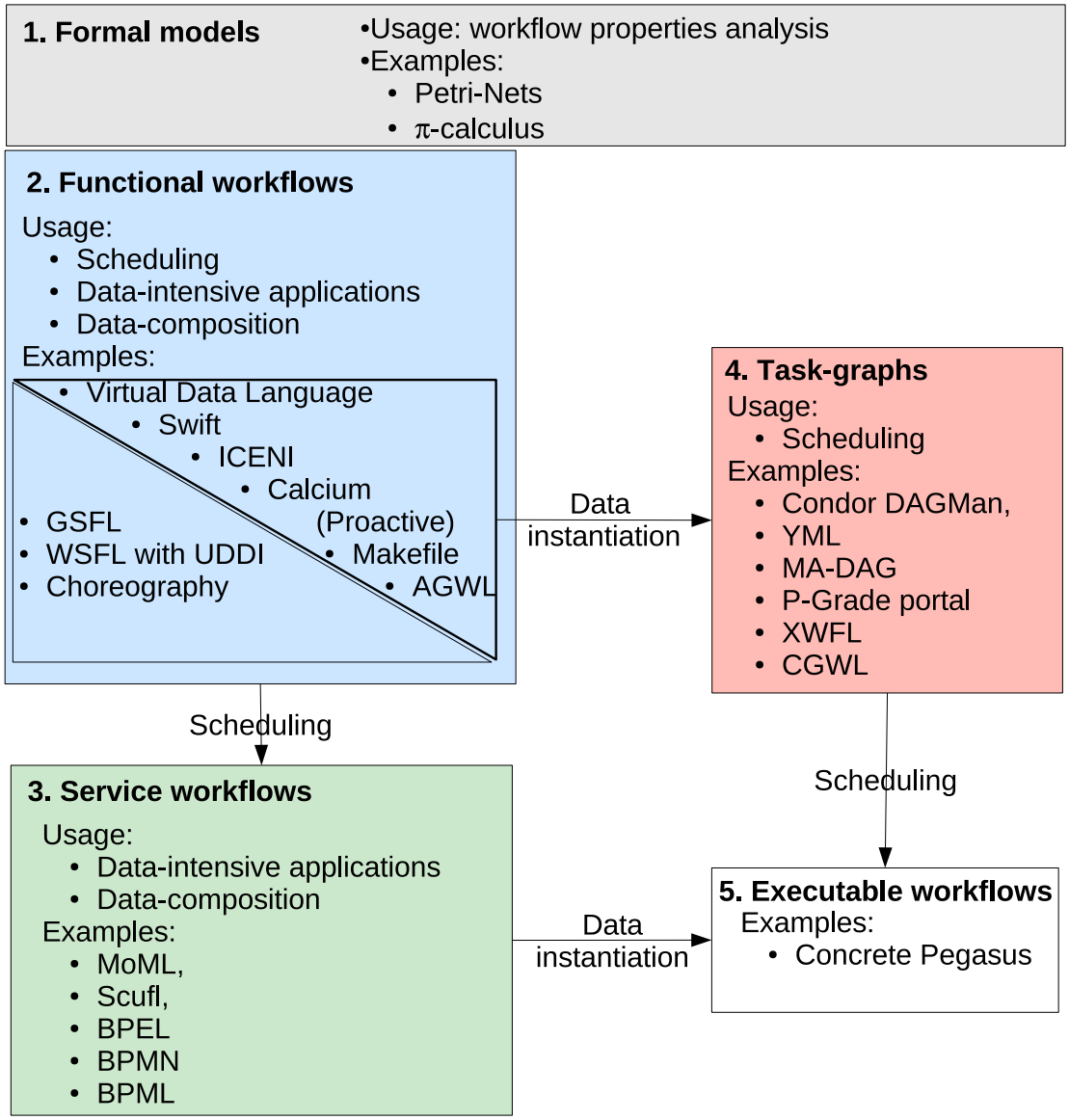

Figure 2.1: Classification of workflow languages. Formal models are the most abstract workflow representation, where neither functions, nor data nor resources are defined. In functional workflows, only the functions are defined. Functional workflows become task-graphs when they are instantiated on data. Similarly, service workflow can be derived by specifying resources in functional workflows. Executable workflows correspond to the most concrete representations, where functions, data and resources are defined. Workflow managers help to move from one class to another one.

representation of the application. For instance, properties such as liveness (the absence of deadlocks) and boundedness (of the amount of generated data for instance) can be inferred from such models. Formal models may be used by computer-scientists to perform a theoretical analysis of the application.

2. Functional workflows are the class of workflows for which only the participants and their dependencies are defined. To become executable, such workflows have to be instantiated on the data which is provided at runtime. The workflow can then be iterated on the data according to *data composition* operators, which will be studied in section 4.2 of chapter 4. This workflow class is particularly suitable for applications handling a lot of data items. Indeed, it prevents the developer from an exhaustive description of the whole task set required by its application: the developer only has to describe the functional template of the application which is instantiated on the data by the workflow manager at runtime. Consequently, in a functional workflow, the size of the handled data sets is not represented in the workflow language and will only be known at runtime. Thus, *it is not possible to determine the number of tasks generated by functional workflows before their execution*. This property can be used to determine whether a workflow model belongs to this class or not. In particular, this class contains traditional script languages and workflow languages that have elaborated control constructs allowing to define for instance dynamic loops, *i.e* loops for which the number of iterations cannot be known before runtime. Resources are not defined in this workflow class. Thus, studying the scheduling of those workflows is possible to some extent, considering the fact that the total number of executed tasks is unknown prior to the execution. Such kind of workflows allows to separate the concerns of the medical image analysis scientist and the clinician. A functional workflow is defined by the medical image analyst alone and is used by the clinician who defines the input data at runtime. It is then passed to the grid expert that performs the scheduling.

3. In service workflows, both functions and resources are specified. As in functional workflows, the data is not defined and is specified at runtime by the user, which provides several interesting properties. Actually, those workflows include resources in their description through their reference to Web-Services. A WSDL document indeed specifies the endpoint of the service, so that the workflow manager cannot perform any scheduling. Yet, optimizing job submission parameters is still possible downstream, for instance at the level of a particular submission service, such as the one proposed in section 7.2.2 of chapter 7 of this thesis. As in functional workflows, a service workflows should be defined by the medical image analyst and instantiated on the data by the clinician. No scheduling is required yet.

4. In tasks-graphs, both functions and data are defined and mixed. A task is defined as the association of a treatment (*i.e* a function) with data items (*i.e* the parameters of the

function). In this class of workflows, the tasks to be executed are completely defined: the workflow representation specifies their number as well as their nature. Tasks-graphs can be characterized by the fact that the number of tasks in the workflow is known prior to the execution: it is a static workflow representation. This class of workflows is intensively used in the development of parallel applications. They are the most suitable representation for scheduling. Indeed, the only missing information to have the workflow completely defined is the mapping onto resources, which is the goal of scheduling. Because the number of tasks has to be predictable, conditional operators are not allowed in task-graphs. Yet, the case of exceptions, compensation handlers, retries allowed in case of failure and other fault-tolerance mechanisms has to be distinguished from conditional operators. Indeed, even if those constructs lead to the generation of a potentially unpredictable amount of tasks by the workflow, they only concern particular execution conditions. A tasks-graph could only be produced by *both* the clinician and the medical image analysis scientist because it mixes the data with the functions. It is used by the grid expert to perform the scheduling.

5. Executable workflows correspond to the mapping of a tasks-graph onto resources. They are the output of any tasks-graph scheduling algorithm and can be directly executed. However, some operations on the workflow representation may still be performed by the workflow manager on an executable workflow. For instance, in [Ramakrishnan et al., 2007], the authors propose a strategy to reduce the data footprint of the workflow during the execution. The executable workflow is analyzed in order to determine the instant when a temporary file produced by the workflow can be deleted, thus leading to a reduction of workflow failures due to full disks.

Yu's workflow structure (see section 2.1.3) may vary among the five workflow classes described in this section. Because of their static nature, tasks-graphs and executable workflows have to be directed acyclic graphs (DAGs) or at least to contain only static loops (*i.e* loops for which the number of iterations is known before runtime). On the contrary, functional and service workflow may contain dynamic loops. QoS constraints considered in Yu's classification may be defined in each class of ours. Similarly, automatic and user-directed composition may be envisaged in each of the five classes.

The distinction between graphical and script-based approaches is fundamentally orthogonal to the classes presented here. However, due to the external context, some classes may favor one paradigm. For instance, service workflows are tightly coupled to the Web technologies which make an intensive use of XML languages that are naturally represented as a graph. On the other hand, most of the tasks-graphs presented on figure 2.1 (except the P-GRADE portal that is clearly dedicated to end-users and provides a user-friendly GUI) have been developed and adopted by computer-scientist and thus rely on a script-based approach.

Control and data flow approaches may be present in every class of this taxonomy. In a

tasks-graph or executable workflow, the dependencies between tasks may be defined either by precedence constraints (*i.e* control links) or by the availability of a file or any kind of data item (*i.e* data links). Similarly, a functional workflow may launch a particular execution because of the control flow (*e.g* in script-like workflow languages) or the availability of data (*e.g* in GSFL [Wagstrom et al., 2002]). The service workflow class gathers data-flow languages such as Scufl and control-flow oriented ones such as BPEL. Besides, this taxonomy focuses neither on the classical business versus scientific dichotomy. Those aspects are assumed to be transverse to the proposed workflow classes.

The expressiveness of the underlying workflow language is not clearly related to the classes presented in this section. Beyond intrinsic limitations (such as the static nature of tasks-graphs and executable workflows that prevent them to contain dynamic loops), languages of a given class can exhibit very different capabilities in terms of control flow description and operators. In the following of this section, we review some existing examples belonging to those five classes.

Finally, it has to be noticed that this classification categorizes workflow *descriptions* rather than workflow management *systems*. Indeed, a given workflow system can use several different workflow descriptions during the workflow life cycle. Actually, one of the goals of a workflow system is to make a workflow description successively move from one class to another one in order to finally reach the *executable workflows* class, where every usable workflow system could appear. Yet, on figure 2.1, we chose to put a given workflow system in the highest possible class (the order being defined by the arrows on the figure) of the various workflow descriptions that it handles. For instance, the potential belonging of P-GRADE to the *executable workflows* class is implicit.

2.2.1 Formal workflow models

Two broad classes of formal models have been proposed: *Petri nets* and *π-calculus*. There has been hot debates about the superiority of one model above the other one [Smith and Fingar, 2003, van der Aalst, 2004] and they both lead to the development of systems or standards relying on them. For instance, π-calculus are said to have inspired the development of choreographies (presented in section 2.2.2) whereas various workflow engines are based on Petri nets [van der Aalst and ter Hofstede, 2005].

2.2.1.1 Petri Nets

Petri nets have been introduced in the thesis of C.A Petri, in 1962 [Petri, 1962]. It is a graphical modeling tool applicable to many systems and particularly suitable for parallel systems as they extend the notion of state machine with concurrency.

A Petri Net is a particular kind of directed bipartite graph, associated with a set of tokens. It

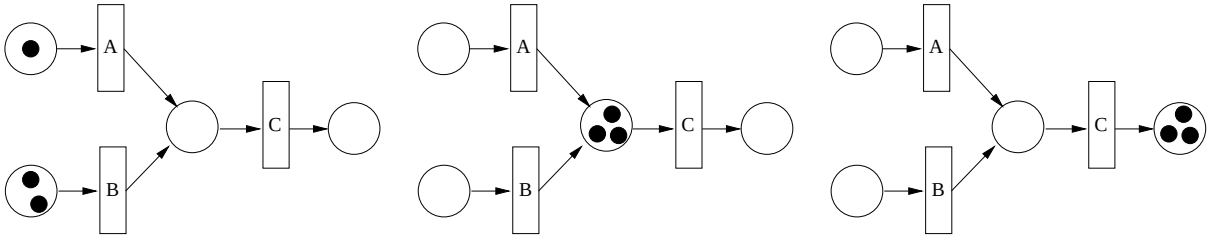

Figure 2.2: Evolution of the multi-merge workflow pattern implemented with Petri-Nets for a particular initial marking.

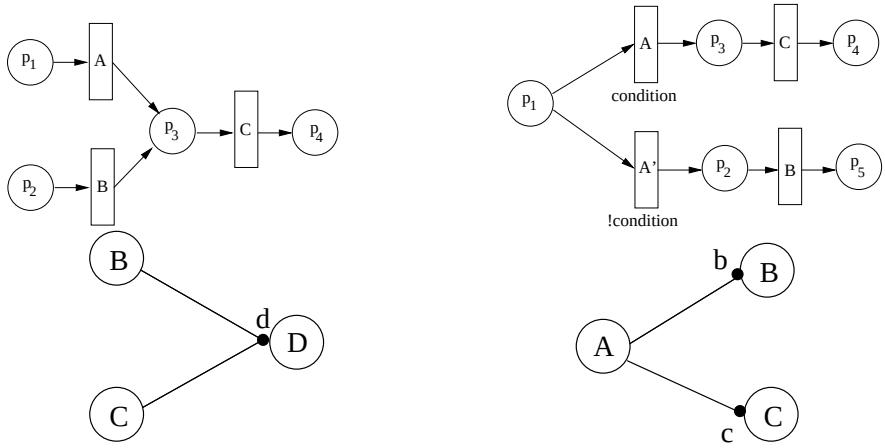

Figure 2.3: Implementation of workflow patterns. Left: Multi-merge ; Right: exclusive choice. Top: with Petri-Nets ; Bottom: with π-processes.

is made of two kinds of nodes called *places* and *transitions*. Edges of the graph are either from a transition to a place or from a place to a transition. State machines are a subclass of Petri nets: in a state machine, each transition has exactly one input place and one output place. Tokens are located in places. Multiple edges linking the same transition to the same place or the same place to the same transition can be represented as a weighted edge whose label denotes the number of corresponding unary edges. A transition is enabled when all of its input places contain at least the number of tokens of the corresponding edge label. It is then ready to *fire*. After a transition has fired, it produces for each output places the number of tokens of the corresponding edge label [Murata, 1989]. An illustration of the transition firing rule is given on figure 2.2. The top line of figure 2.3 (adapted from [van der Aalst et al., 2003]) displays an implementation of two classical workflow patterns using Petri-Nets: the multi-merge and the exclusive choice.

Several extensions of Petri Nets have been proposed and used for various applications. For instance, timed nets [Magott, 1984] introduce delays associated with transitions and/or places and stochastic Petri Nets associate a random variable to the time delays [Ajmone Marsan et al., 1984]. Inhibitor edges have also been introduced in extended Petri Nets: they disable the transition to which they are connected when their input place has a token [Agerwala, 1974]

Colored Petri Nets (CPN) were introduced to ease the manipulation of data values in Petri

Nets [Kristensen et al., 1998]. They are particularly used for workflow modeling and are the basis of the YAWL workflow system [van der Aalst and ter Hofstede, 2005]. CPN are a subclass of High Level Petri Nets, which also include Hierarchical Petri Nets. In a CPN, tokens are distinguishable: each of them is associated to a *color* which represents a data value. Places have an associated *color set* which represents the data type to which belong the colors of all their tokens. Edges are annotated with expressions that determine the exact data values removed and added by the firing of a transition.

2.2.1.2 π-calculus

In [Smith and Fingar, 2003], the authors claim that some of the procedures used in business cannot be modeled using workflow engines that do not rely on π-calculus. They suggest to adopt the term *process* to denote workflows relying on the π-calculus formalism. A singular characteristic of π-calculus is that it is able to exchange information among participants whose relationships evolve as a result. This feature is called *mobility*. Mobility is required to model processes where the exchange of information fosters the link between participants. The example of email exchanges is often cited to illustrate such a behavior: by receiving emails sent to multiple recipients, a participant becomes aware of addresses of other people, thus developing her communicating ability. Partisans of π-calculus advocate that static representation systems such as Petri nets cannot properly represent mobility [Puhlmann, 2006].

Pi-calculus is an extension of process algebra aiming at handling concurrency. It has been proposed by Milner [Milner, 1999]. Pi-calculus is described in terms of *processes*, *channels* and *names*. Channels are used by processors to exchange messages. Both messages and processes are called names and thus cannot be distinguished. The sending of a message u over a channel x is written $\overline{x}(u)$, whereas receiving the message u over the channel x is denoted by $x(u)$. Channels themselves can also be sent and received, which make possible the description of mobile processes such as the email use-case described in the previous paragraph. The sending and receiving of a message u over any channel can be abbreviated respectively by $\overline{u}$ and u. Processes can be composed sequentially by the operator $''.''$ or in parallel, with the notation $''|''$. The choice operator $''+''$ is also available as well as the $''!''$ unary operator which is used to specify that a process can be iterated as many times as required. A condition about a particular name can be expressed by the $[x = y]$ notation. There are two particular processes: 0, which does not do anything and stops the process execution and τ, which corresponds to a hidden activity, that does not take part into the global process [Woodman et al., 2007]. A τ process is an activity that corresponds to an effective participant of the workflow as defined in section 2.1.2: for instance, it may model the computation of a service operation on some data, which is seen as a black box from the workflow point of view.

It is clear that π-calculus is able to model both control and data flows. Van der Aalst's workflow patterns [van der Aalst et al., 2003] are expressed using π-calculus

in [Puhlmann and Puhlmann, 2005]. Examples from this work are recalled here, to illustrate the π-calculus formalism. The bottom line of figure 2.3 presents two workflow patterns: the left of the figure presents shows the *multi-merge* whereas the right of the figure displays the *exclusive choice*. The π-calculus representation of the multi-merge is the following:

$$\begin{aligned} B &= \tau_B.\bar{d}.0 \\ C &= \tau_C.\bar{d}.0 \\ D &= !d.\tau_D.0 \end{aligned}$$

Each line of this equation models a particular process of the workflow. After their execution, processes B and C both send the same name d which is required by process D. The presence of a $''!''$ operator in front of process D indicates that it will be replicated as many times as needed. In this case, two copies of D will be done. The *exclusive choice* is represented by the following π-processes:

$$\begin{aligned} A &= \tau_A.(\bar{b}.0 + \bar{c}.0) \\ B &= b.\tau_B.0 \\ C &= c.\tau_C.0 \end{aligned}$$

For this process, the choice operator + is needed to distinguish the 3 invocation cases.

The π-calculus formalism has been extended to the case of Web-Services orchestrations in [Mazzara and Govoni, 2005]. In this work, the authors add a transaction operator which is able to cope with faults and to trigger a recovery process `if` the fault message is received. Based on the same `web`π_∞ extension of the π-calculus, another application to orchestrations is proposed in [Lucchi and Mazzara, 2007], where the authors detail a π-calculus based semantics for WS-BPEL. It is highlighted that the three different error handling mechanisms of WS-BPEL are not necessary and a novel orchestration language based on the idea of event notification as the unique error handling mechanism is proposed. In the context of choreographies, a formal model of WSCI (see section 2.2.2) using a process algebra approach (CCS) is proposed in [Antonio et al., 2004] and applied to web service compatibility, replaceability and the automatic generation of adapters.

2.2.2 Functional workflows

Virtual Data Language. The Virtual Data Language (VDL) [Zhao et al., 2007b] is a functional workflow language that derives from a former VDL [Foster et al., 2002]. It is executed by the Swift [Zhao et al., 2007a, Stef-Praun et al., 2007] workflow engine which derived from the Virtual Data System (VDS). It has control flow constructs such as `for each`, `if`, `switch` and `while`. It is based on the declaration of procedures written in a C-like syntax. Procedures can be atomic or made by other procedures. VDL does not make any assumption about the size

of the input data sets. However, the underlying workflow manager expands the VDL definitions into a tasks-graph (see section 2.2.4) and executes them. This is made possible by the fact that `foreach` nodes are expanded at runtime thus enabling data sets to have a dynamically determined size. We guess that a similar late expansion system is used for the other control flow constructs that lead to the execution of tasks whose number is not known before runtime. The data types representation is extensively described in VDL. It relies on an XML Data Set Typing and Mapping (XDTM) that allows the types of data sets and procedures to be defined abstractly in terms of XML schema. Separate mapping descriptors then define how such abstract data structures translate to physical representations. For instance, XDTM provides mappings from file names to their absolute path in a file system. Yet, data is not instantiated inside the VDL representation. This is made at the engine level. Both those arguments lead us to put this approach in the functional workflow class, even if it is tightly interfaced with tasks-graphs: what is called a "high-level" workflow representation in Fig.17.8 of [Zhao et al., 2007b] is a functional workflow because data segments are not defined on this representation. The described implementation of the VDL prototype converts this workflow definition into a tasks-graph by expanding DAG nodes (Fig. 17.9 of [Zhao et al., 2007b]).

GSFL. The Grid Service Flow Language (GSFL) has been designed as an adaptation of the WSFL to grid services, which have different needs from standard Web-Services [Wagstrom et al., 2002]. In particular, the authors underline the fact that the workflow specification needs to be able to allow communication between the services to avoid the workflow manager to become a bottleneck centralizing the data transfers. As already noticed in section 2.1.1, avoiding centralized enactment is not straightforward with Web-Services, whereas OGSA introduced facilities for that. In particular, GSFL provides a mechanism to connect notification sources and sinks defined in the OGSA. GSFL is also able to handle OGSA registries and factories for creating grid services. A GSFL document defines services providers, the activity model, the composition model and the life-cycle model. Service providers are the list of services involved in the workflow. They can be located statically, by a hard specification of an endpoint or invoked using factories. In the latter case, resources are not defined in the workflow document, which leaves room for further scheduling. The so-called activity model identifies the particular operations of the services involved in the workflow. The composition model describes the data and control flow between the activities and the life-cycle model contains a list of precedence links describing the order in which the services execute.

ICENI / ICENI-II. ICENI's authors identify two different workflow representations: the spatial and the temporal ones [Mayer et al., 2004, McGough et al., 2004]. A workflow is denoted *spatial* when none of the relations between its participants are precedence constraints: in this case, links between services could for instance be determined by event notifications or data

links. On the contrary, a workflow is *temporal* when all its relations are precedence constraints. The authors notice that even if the temporal representation is the most suitable to determine a planning of task allocations, the spatial representation is the most user-friendly. Thus, in their ICENI system, the user builds a spatial representation which is then mapped to a temporal one for scheduling purposes. The user specifies the workflow in a spatial expression, which, in our terminology, corresponds to a functional representation. This user-defined workflow is also called an execution plan. At this stage, components are described in terms of meaning and behavior. ICENI then converts it to a temporal description, *i.e* a tasks-graph. As underlined by the authors, problems appear when the functional description is not acyclic, as discussed in the next sections. As in GSFL, the components themselves talk to their partners, without any execution centralization. ICENI II is described in [McGough et al., 2007, McGough et al., 2006]. Three steps are identified in the workflow generation: specification, realization and execution. Specification produces an abstract workflow whereas realization aims at validating the workflow and then map its elements to concrete resources. Execution deals with the monitoring of the application and functionalities to allow component migration.

Calcium Calcium [Caromel and Leyton, 2007, Caromel et al., 2008] is a framework based on skeletons [Cole, 1991] that are a workflow programming model aiming at hiding the complexity of parallel and distributed applications. It is built upon the Proactive middleware [Caromel et al., 2006]. Calcium has a set of control constructs: *farm* (task replication), *piped* (staged computation), *seq* (wrapping of execution functions), *if*, *while*, *for*, *map* (single instruction, multiple data), *fork* (multiple instruction, multiple data), *d&c* (divide and conquer). As stated by the authors in [Caromel and Leyton, 2007]:

> New tasks can be dynamically produced by the interpreters when data parallelism is encountered.

Consequently, we classify this approach in the functional workflows.

AGWL. The Abstract Grid Workflow Language (AGWL) is the workflow language used by the ASKALON workflow manager [Fahringer et al., 2007] which offers two interfaces for generating large-scale scientific workflows in a compact and intuitive representation: graphical modeling using the UML standard and a programmatic XML based language. AGWL workflows can be either generated from a graphical UML description or directly written by the end-user. AGWL workflow descriptions are definitely independent from the execution resources. A dedicated scheduler is responsible for resource allocation and a resource manager handles reservation. AGWL workflows include both control-flow and data-flow. Control-flow constructs include `sequences`, `dags`, `for`, `forEach`, `while` and `do-while` loops, `if` and `switch` constructs and more advanced constructs such as `parallel activities`, `parallelFor` and `parallelForEach` loops and collection iterators. The user can also specify properties and

constraints (such as memory requirements) for activities and data flow dependencies. An example from [Fahringer et al., 2007] underlines that dynamic loops (*i.e.* loops for which the number of iterations cannot be known before runtime) can be defined, which lead us to put this language in the functional workflows category. ASKALON uses another language, CGWL in order to have a tasks-graph representation of the workflows. Before the execution, the workflow manager performs a mapping from AGWL to CGWL.

Choreographies. The term *choreography* originates in a metaphor of a workflow which is viewed as an artistic work performed by actors, *i.e* the participants of the workflow. In that sense, choreography is opposed to *orchestration*: in a choreography, each actor is linked to other ones and the global process is obtained as a result of those local interactions. On the contrary, in an orchestration, actors are directed by a central conductor which manages the whole orchestra. Choreography and orchestration are terms that are tightly related to the Web-Services, as specified by the W3C. Choreography is thus often categorized as a decentralized approach whereas orchestration is centralized [Mayer et al., 2004]. However, even if the workflow *description* is not centralized in a choreography as it is in an orchestration, the practical implementation of a workflow manager that would permit such a decentralized execution is not specified. Extensions of Web-Services such as WSRF and OGSA seem to be mandatory in order to have such a decentralized execution.

The initial choreography specification was the Web-Services Choreography Interface[10] (WSCI). WSCI allows a Web-Service to define *interfaces* that describe processes from its operations. Operations can be composed in sequential or parallel executions and loops and conditions can be defined. WSCI interfaces describe choreographies between the operations of a Web-Service. WSCI defines *global models* on top of operations. Global models describe choreographies between interfaces of several services. It provides a set of connections (mappings) between pairs of individual operations of communicating participants. In [Antonio et al., 2004], authors formalize WSCI using π-calculus. WSCI set up the basis for the development of the Web-Services Choreography Description Language(WSCDL)[11]. In this language, *interactions* are defined among different *roles*. Roles can be played by different *behaviors* that may (optionally) be linked to particular WSDL interfaces. Indeed, the W3C candidate recommendation for WSCDL specifies that:

> " A behavior without an interface describes a roleType that is not required to support a specific Web Service interface. "

Thus, WSCDL choreographies are far from being executable: they only describe patterns for message exchanges among abstract participants. According to the W3C, a choreography language is not an executable business process description language or an implementation lan-

[10] http://www.w3.org/TR/wsci/
[11] http://www.w3.org/TR/ws-cdl-10/

guage. The role of specifying the execution logic of an application will be covered by these specifications.

YAWL. YAWL is built upon the Petri-Nets formalism. Its specification originates in an exhaustive study of workflow managers with respect to a set of workflow patterns [van der Aalst and ter Hofstede, 2005]. Thus, the goal of YAWL is to overcome the expressiveness limitations of the contemporary workflow management systems. It is based on high-level Petri nets, to which extended constructs such as advanced synchronization, multiple instances and cancellation patterns are added, thus defining the extended workflow nets (EWF).

Makefile. Makefiles are a particular kind of functional workflows that completely relies on data flow. Participants of the workflow are defined by a command line that includes services (executable) and data (arguments of the command line). Tasks are linked by precedence constraints. When a task is ready to be executed, it is effectively fired if and only if one of its input files has been modified since the last invocation. Makefiles can include conditionals and loops. Thus, the number of tasks generated by the execution of a makefile may not be known prior the execution. Moreover, with the `-j` option of the workflow engine `make`, it is possible to define a number of processes that may run concurrently, potentially on different CPUs, so that the resources are not defined inside the Makefile. Consequently, it has to be categorized as a functional workflow.

2.2.3 Service workflows

Scufl (Taverna) workflows. Scufl is a data-flow oriented language that basically describes the pipeline of an application. Participants of Scufl workflows are called *processors*. Many of them can be specified: for instance, string constants fire only once and return a single string value. Web-Services can also be enacted by specifying a WSDL document and a particular operation as well as compiled Java code or Beanshells processors[12] that embed a piece of Java code. Sources and sinks correspond to the inputs and outputs of the workflow. Each of them may contain several data segments on which the workflow is iterated. Their content is not specified inside the Scufl document: it is independent from the workflow description and is only known at runtime. In that sense, Scufl is a typical example of functional workflow. However, Web-Services processors are bound to a particular resource, included in their WSDL description. A Scufl workflow instantiated on some input data could thus be considered as an executable workflow rather than a tasks-graph.

Processors have input and output *ports* that can contain several data items and are connected to other ones with *data links*. A data link is just a pipe between an output port of a processor

[12] http://www.beanshell.org/

and an input port of another one. An output port can be connected to several input ports. In this case, the data items are broadcasted to all the connected input ports. Similarly, several output ports can be linked to a single input port. In this case, data items are buffered into the input port according to their order of arrival. Data composition operators allow to define iteration strategies between the input ports of a processor. Iteration strategies are used to control how multiple data items inside the input ports are combined. They are described in section 3.1.1 of chapter 3.

Coordination constraints can be specified in Scufl and provide elementary control links. Such a link specifies that a processor has to wait for another one before starting its execution, even if there are no data dependency between them. This is the only kind of control link available in Scufl. No control operators such as *for* or *while* are available. Nevertheless, the `FailIfFalse` and `FailIfTrue` processors are defined to implement conditional branching in a workflow, although no control operator such as *if* is defined in Scufl. Those processors fail or succeed depending on their Boolean input value, thus discarding or enabling the processors depending on them in the workflow. Apart from that, the workflow is completely driven by the presence or absence of data in the input ports of a processor: a processor will fire if and only if all of its ports contain adequate data. It is not possible to define variables in Scufl. As a consequence, there is no expressions nor operators in the language.

MoML (Kepler) workflows. Participants of a MoML workflow are called *actors*. In MoML, each actor must define the type of each of its ports. Links (called *relations*) can only be defined between ports with compatible types. Ports participating in several relations have to be defined as *multi-ports*.

MoML defines no semantics for an interconnection of components. It instead provides a mechanism for attaching a "director" to a model. MoML knows nothing about directors except that they are instances of classes that can be loaded by the class loader [Lee and Neuendorffer, 2000]. Four directors are available in Kepler: Continuous Time (CT), Discrete-Event (DE), Synchronous Data Flow (SDF) and Process Networks (PN). The CT director is used to model physical systems: the workflow is then directed by a clock. In the DE director, the workflow is also directed by a clock: each actor communicates with the other ones by sending them timestamped signals. The director orders those signals and distributes them to their targets. In the PN director, each actor is executed in a dedicated thread. Relations between actors are waiting queues of finite capacity. Writing into a queue is never blocking whereas reading in an empty queue is blocking. The SDF director is used to simulate data flows.

Orchestrations: BPEL, BPML, BPMN, WSFL, XLANG. Orchestrations are workflows of Web-Services. This denomination originates in a metaphor of a workflow which is viewed

as a musical partition interpreted by the participants and directed by the workflow engine. Orchestrations differ from choreography by the point of view adopted by the developer. In an orchestration, a single workflow engine is responsible for the execution of the application. It centralizes the services invocations so that services do not communicate between each other [Mayer et al., 2004]. Orchestration is also referred to as a concrete workflow whereas choreography is abstract. Indeed, in a choreography, resources are not mandatorily defined whereas orchestration precisely defines services WSDL and consequently endpoints.

The *de facto* orchestration standard is BPEL[13] [Mc Ilraith and Mandell, 2002, Wohed et al., 2003, Khalaf et al., 2003, Emmerich et al., 2005, White, 2006, Slominski, 2007]. It was defined considering previous specifications: WSFL, XLANG, BPML and BPMN that did not survive the BPEL emergence. In [Wagstrom et al., 2002], the authors provide a technology survey of workflow languages for Web-Services. In particular, a detailed analysis of WSFL is provided. WSFL includes both control and data links. From our classification point of view, a remarkable feature of this language is the identification of the services participating in the workflow by using a *locator* element which allows a service to be described by a static (hard reference to a WSDL), a local, a UDDI (the service is looked up using the UDDI API) or a mobility (the service provider is referenced in a message generated by some activity of the workflow) binding, which would allow us to put this language in the functional workflow class.

In its current 2.0 version, BPEL includes several control constructs: `switch`, `pick`, `while`, `for each`, `repeat until`, `wait`, `sequence` and `flow`. Activities may include Web-Service invocations, receive and reply and variable assignation. It proposes a fault handling mechanism through the `exit`, `throw`, `rethrow` and `compensate` constructs. Because of those control constructs, it is not possible to convert a BPEL workflow definition to a DAG. In particular, it is not possible to determine the number of service invocations, which may be dependent on the nature of the input data.

2.2.4 Tasks-graphs

Condor DAGMan. Condor DAGMan [14] is one of the most used tools for tasks-graphs. It allows the user to define precedence constraints between Condor jobs that are submitted to a pool of resources. So-called “pre” and “post” scripts may be defined to be executed respectively prior or after the job itself. Fault-tolerance facilities are also available, such as the ability to define a number of retry attempts in case of failure during the execution. Such retry specifications have been used to define while loops with DAGMan, by making a job fail and retry until the stopping condition has been reached[15].

[13] www.ibm.com/developerworks/library/ws-bpel/
[14] http://www.cs.wisc.edu/condor/dagman/
[15] https://lists.cs.wisc.edu/archive/condor-users/2005-November/msg00000.shtml

P-GRADE portal. The P-GRADE portal is a tasks-graph workflow manager based on the Condor DAGMan [Kacsuk et al., 2003, Kacsuk and Sipos, 2005]. It is able to submit jobs simultaneously on various grid middlewares, including GT2, GT4, LCG and gLite with a secured access mechanism [Kacsuk et al., 2006b]. An interesting feature of the P-GRADE portal with respect to our workflow description classification is the possibility to define parametric tasks [Kacsuk et al., 2006a]. Parametric tasks allow the user to define tasks whose parameters vary in a given range. Parametric tasks bring tasks-graphs closer to the functional workflow approach as they are templates to generate several tasks. Yet, parametric tasks can still be expanded into a tasks-graph and they generate a predictable amount of tasks. Anyway, parametric tasks make a P-GRADE workflow description far more flexible than most of the task-graphs. P-GRADE is an interesting example of a trade-off between tasks-graphs and functional workflows.

DIET MA-DAG. DIET is a grid middleware providing scalable scheduling facilities for grid servers [Caron and Desprez, 2005]. MA-DAG, a workflow management system has been developed on top of it [Amar et al., 2006] and is based on a DAG model. This approach focuses on scheduling, by offering the ability to use different advanced algorithms. Multi-workflow scheduling is also under investigation.

XWFL. The Workflow Enactment Engine (WFEE) uses the xml-based Workflow Language (XWFL) [Yu and Buyya, 2004]. This language allows users to describe tasks and their dependencies. This language is made of three sections: parameter definitions, task definitions and data link definitions. This language supports both abstract and concrete workflows: resources can be specified so that we could also put this language in the executable class. Parameters can be used in order to define parametric tasks as described in the previous paragraph. Data links are then used to specify the tasks-graph.

Yvette ML. The YML framework defined YvetteML, a parallel programming language which is used to model workflows [Delannoy and Petiton, 2004, Delannoy et al., 2006]. YvetteML includes a component model and a graph description language. Components are defined as an encapsulation of task nodes of a directed acyclic graph representing a complex application. They represent a chunk of computation requiring no communication with the rest of the application. Components are made of a so-called abstract declaration, which specifies the type and mode (in, out or inout) of the parameters as well as a user-provided implementation that adds some decorations to a C/C++, Fortran or Java code in order to be able to compile it on different platforms. The YvetteML graph language is a control-flow language. Several control constructs dedicated to parallel applications are present such as *par do*, *seq do*, *wait* or *signal*. A typical example (extracted from [Delannoy and Petiton, 2004]) of the YvetteML graph language is:

```
const problemSize := 10000;
event evt[2];
var MatrixReal vRes[1];
par(i:=1; problemSize) do
   compute fillMatrixReal(vRes[i],problemSize,i);
   signal(evt[i,1]);
end par do
```

The YML Framework interacts with the user using a compiler which translates components into binary applications. The model of the YML workflow framework can contain loops, iterations and branching: the compiler completely expands graphs to make them ready for scheduling. Loops are unrolled, condition evaluated, unvisited branches spread out of the graph and constants are propagated. The compiler translates applications described using the YvetteML language to a set of components calls. Regarding our classification of workflow descriptions , the YvetteML compiler acts as a translator from a functional workflow instantiated on its input data to a tasks-graph. However, the dynamicity of the functional workflow approach cannot be handled by YML and the number of tasks generated by the application is foreseeable. That is why we put it in the tasks-graph class. Yet, the YvetteML workflow language remains very similar to the one of the Virtual Data Language [Zhao et al., 2007b].

2.3 Moving from a class to another one.

The goal of a workflow management system is to move from the workflow definition provided by the user to an executable workflow. A significant amount of work may be necessary to go from one category to another one.

From tasks-graphs to executable workflows. Moving from tasks-graphs to executable workflows is the operation done by the *scheduling* of task-graphs. This problem consists in finding a task execution and resource allocation planning in order to optimize one or more criterion (such as the makespan of the application, the fairness, ...). This problem is NP-complete as soon as the number of resources is bounded or the communication costs between resources are taken into account. This family of problems has been extensively studied [Legrand and Robert, 2003]. In particular, list heuristics have been proposed and adapted to different constraints such as the heterogeneity of the resources [Topcuoglu et al., 2002].

From functional to service workflows. Moving from functional to service workflows requires to be able to get the endpoint of a service from an “abstract” description. It can be done either by dynamic service instantiation (*e.g* by the use of an OGSA service factory) or by service look-up in a registry (*e.g* UDDI). Besides, semantic services discovery is a very

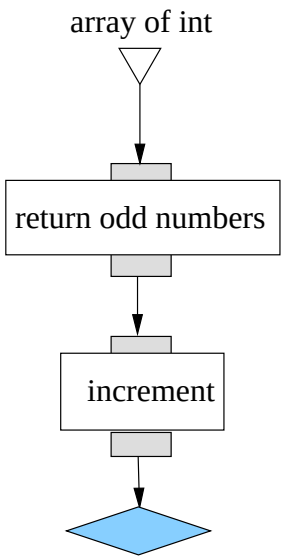

Figure 2.4: Example of a functional workflow that could not be converted into a tasks-graph: the `return odd numbers` participant extracts some elements of the input array of integers and pass them to the following `increment`. The number of invocations of the `increment` participant cannot be determined in advance.

hot topic and several approaches are still studied [Kovács et al., 2007, Atkinson et al., 2007, Spanoudakis et al., 2007, Song et al., 2007].

From service workflows to executable workflows. The shift from service workflows to executable workflows is typically done at runtime. Services invocations are dynamically determined during the execution, according to the availability of data items in the ports of the services, eventually after the application of some data composition operators (such as iteration strategies in Scufl) or because of some relations defined by control constructs (for instance in BPEL).

From functional workflows to tasks-graphs A description of the input data of a functional workflow is sufficient to convert it to a tasks-graph if the functional workflow is assumed:

- to be acyclic or expandable to a DAG before runtime
- not to contain any conditional control structures such as `if` or `switch` and
- not to generate data sets whose size is determined at runtime.

Indeed, the above hypotheses prevent the workflow from generating an unpredictable amount of tasks at runtime. For instance, if we consider the functional workflow of figure 2.4, it is obvious that it could not be converted to a tasks-graph: the `return odd numbers` participant returns a data set whose size depends on the nature of `array of int`. Thus, one could not determine the number of invocations of the `increment` participant before runtime. The conversion from

functional workflows to tasks-graphs could be done separately for sub-parts of the workflow, as it is done for instance by the engine supporting the Virtual Data Language, where sub-parts of the workflow are progressively converted to tasks-graphs as soon as the number of tasks to be generated is known (see section 17.7.2 of [Zhao et al., 2007b]). A typical construct preventing a functional workflow to be transparently mapped to a tasks-graph is the *foreach* that leads to dynamic loops where the number of iterations cannot be known before the instantiation of the workflow on the data. To get closer to the functional approach, an interesting extension of the tasks-graphs are parametric tasks descriptions where a generic task can be described for a whole parameter range, resulting in the execution of multiple jobs, as done for instance in the P-GRADE portal [Kacsuk and Sipos, 2005].

2.4 Conclusions

In this chapter, a review of the existing workflow description approaches has been provided. A classification distinguishing functional, services, executable workflows and tasks-graphs has been detailed, based on the presence or absence of functions, data and resources in the workflow specification. The suitability of each workflow class has been highlighted: even if tasks-graphs are clearly more suitable for workflow scheduling implementations, the data composition facilities and the dynamicity of service workflows allow a simpler representation of the applications. Moreover, this approach allows to better separate the concerns of the three main actors envisioned in this thesis: the clinician, the medical image analyst and the grid expert.

Existing workflow descriptions and their corresponding implementations offer a very diverse and complete set of tools providing to the user the required facilities to build his/her application. Depending on his/her profile, different approaches could be chosen. For instance, a Perl-addict scientist wanting to describe the workflow of her application could select a script-based functional approach such as Swift that offers all the constructs and data types facilities of a traditional scripting language. On the opposite, users that do not have any programming background may be more easily targeted by graphical composition systems such as the P-GRADE portal, Taverna, Triana or Kepler. Finally, parallel programming computer scientists may better chose a tasks-graph workflow language such as Condor DAG-Man or Yvette-ML as this approach is more suitable to implement smart workflow scheduling algorithms because of the static nature of the tasks-graphs.

The forthcoming research on this area may thus be more focused on the effective adoption of existing workflow managers by large users communities rather than on the development of yet another workflow system. The adoption of stable workflow platforms is a prerequisite for further investigations that have to be initiated by real users' need. Actually, this direction is the

one adopted by the leading workflow projects such as the P-GRADE portal[16], Taverna[17], Swift[18] or Gwendia[19]. In that sense, we study the implementation of the bronze standard application with the Scufl services language in the next chapter.

[16] http://portal.p-grade.hu/
[17] http://taverna.sourceforge.net
[18] http://www.ci.uchicago.edu/swift/
[19] http://gwendia.polytech.unice.fr/

Chapter 3

The bronze standard service workflow

Contents

Based on the taxonomy detailed in the previous chapter, we advocate here the adoption of service workflows for medical image analysis applications. The workflow of the bronze standard application is described with the Scufl language which is particularly interesting through its data composition strategies. An analysis of the expressiveness of this language is finally proposed through the implementation of a universal Turing machine.

Dans ce chapitre, nous motivons l'adoption de chaînes de traitements de services pour les applications d'analyse d'images médicales. La chaîne de traitement de l'application des étalons de bronze est décrite avec le langage Scufl qui fournit des opérateurs de composition de données particulièrement intéressants. Enfin, une analyse de l'expressivité de ce langage est proposée à travers l'implémentation d'une machine de Turing universelle.

The workflow classification presented in the previous chapter revealed a wide spectrum of different workflow approaches. Even if some works still focus on specific workflow patterns to further enhance the languages [van der Aalst and ter Hofstede, 2005], we will concentrate on the study of the adoption of an existing workflow language for our application. Among the presented workflow approaches, we advocate here the use of service workflows for medical image analysis applications. In particular, the bronze standard application will be described with the Scufl language.

3.1 The bronze standard workflow

3.1.1 Motivations for the use of service workflows

The Scufl workflow of the bronze standard application is depicted on figure 3.3. Apart from the algorithms sharing needs of this application, which motivates the adoption of a service-oriented architecture, several reasons rationalize the use of service workflows rather than task-graphs for this application.

Separation of clinical, medical image analysis and grid concerns. The adoption of service workflows enables a clear separation of concerns between the 3 actors of the typical scenario envisioned in this thesis. Indeed, in such a paradigm, the medical image analyst builds a workflow from existing services potentially connected to the grid with the help of the computer scientist. Such workflows are then exposed to the clinician that only specifies the input data on which to run them. The data instantiation is the last step before the execution of the workflow. Conversely, in task-graphs, those 3 roles are mixed. To specify the input data, the clinician would have to modify the workflow itself, by adding new tasks, which could not be automatable in case of complex patterns.

Compact description of large workflows. From a user point of view, the main difference between task-graphs and service/functional workflows appears when considering the re-execution of the same workflow over different input data segments, as it is done by the registration services of the bronze standard workflow that are iterated on a complete image database. In a

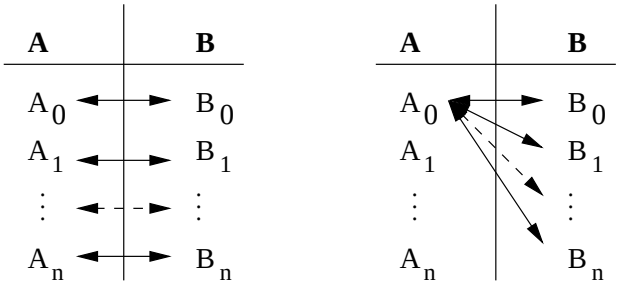

Figure 3.1: Data composition operators. Left: one-to-one. Right: all-to-all. **A** and **B** represent ports of a service. They may contain several data items (A_i, B_j) on which the service is going to be iterated.

task-graph, executing the same processing over two different data segments results in the description of two independent tasks. This approach enforces the replication of the execution graph for every input data to process, which becomes intractable when the workflow is made of hundreds to thousands of tasks. On the contrary, the description of a functional or service workflow is independent from the size of the input data set, which keeps the workflow description compact and more easily graphically representable.

Data composition. Thanks to the absence of data instantiation in the workflow description, operators acting on the data flow itself can be defined: it may simplify the description by avoiding the use of complex control patterns. In particular, *iteration strategies* over the input ports of a service are available in Scufl. When a service owns two inputs or more, an iteration strategy defines the composition rule for the data coming from all the input ports pairwise. Iteration strategies are composed of data composition operators. Considering two input sets $A = \{A_0, A_1, \ldots, A_n\}$ and $B = \{B_0, B_1, \ldots, B_m\}$ of a service, the *one-to-one* data composition operator consists in processing each data item of the first set with the matching data item of the second set in their order of definition. The other composition strategy available in Scufl is the *all-to-all* operator which consists in processing all the input data items from the first set with all the input data items from the second set, thus producing $m \times n$ results. The action of those operators is illustrated on figure 3.1. Using iteration strategies to design complex data interaction patterns is a very powerful tool for data-intensive application developers. For instance, the sweeping of a service over a whole parameter range can be described with a single all-to-all operator between the parameter to sweep and the other inputs of the service. In the following of this thesis, all-to-all operators will be denoted by $\otimes$ and one-to-one by $\oplus$.

Dynamic data sets. Task-graphs and functional/service workflows differ in depth in their handling of data. The dynamic nature of the data description in the functional and service approaches enable the definition and execution of a workflow although the whole input data is not known in advance. It will be dynamically fed in as new data is being produced by sources.

Indeed, it is common in scientific applications that data acquisition is a heavy process and that data segments are being progressively produced. Some workflows may even act on the data production source itself, stopping data production once computations have shown that sufficient inputs are available to produce meaningful results. This dynamicity is also required when the input data is the result of a data base query whose response size is not known in advance. A significant difference between the task-graph and functional/service workflow approaches coming from the ability of the latter to deal with dynamic data sets is that there may exist loops in a functional workflow, even in absence of specific control constructs. In a task-graph, loops have to be completely expanded in the workflow description: dependencies between tasks are precedence constraints and the workflow graph thus has to be acyclic. Consequently, the implementation of dynamic loops (*i.e* loops whose number of iterations is not known before runtime) is not possible in a task-graph whereas it is in a functional or service workflow. For instance, figure 3.2 corresponds to the Scufl implementation of the following `for` C++ loop:

```
for(i=i0;i<nMax;i++) cout<<i<<endl;
```

In this workflow, the `inferior` processor compares its two arguments. It is initialized with the `i0` string constant value. All the subsequent values will be compared to the same `nMax` value which is an input of the workflow that will be defined at runtime. This behavior is obtained by the use of an all-to-all data composition operator between the inputs of `inferior`. The Boolean value returned by `inferior` is piped to the `Fail_if_false` conditional processor. If it fails, then no more processor can be fired and the workflow halts. Otherwise, the coordination constraint allows `increment` to be fired. `increment` only increments its input, which is also initialized by the `i0` value. The output of this processor is looped back to its input. Self-looping allows the workflow to maintain a state variable (`i` on figure 3.2), whereas all the processors are stateless and the definition of variables is not possible. The value `j` resulting from `increment` is then passed to the `inferior` processor and a new iteration starts.

3.1.2 Description of the bronze standard workflow

The Scufl service workflow of the bronze standard application (see chapter 1 for scientific details about this application) is depicted on figure 3.3. In the upper part of the workflow, image pairs are registered with four different algorithms. Then, the computed transformations are converted to a single format and compared by the bronze standard statistical procedure to produce the accuracy estimations (one per algorithm). Every piece of data exchanged in this workflow is a string. Files are represented by references (grid file names). This workflow has been implemented in Scufl using the Taverna workbench [Oinn et al., 2004] which provides a very user-friendly and stable GUI to describe workflows.

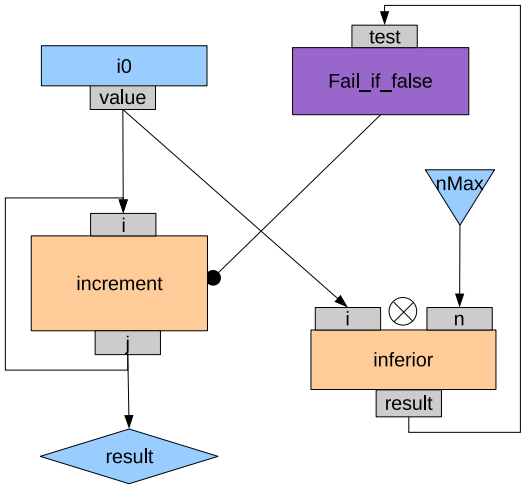

Figure 3.2: Example of a looping workflow in Scufl. Orange boxes represents Beanshells participants, purple ones are conditionals and blue rectangles are string constants. Data links are figured with arrows and coordination constraints with circle-terminated arrows. Blue triangles are input and blue diamonds are outputs.

Inputs and outputs. The inputs of this workflow are the lists of image pairs to register (`floatingImage` and `referenceImage`), the options of each registration algorithm (`sizeCrestLines`, `PFMOption`, `BaladinOption` and `YasminaOption`), the name of the file where to store the produced transformations (`FileName`) and the name of the methods to test with the bronze standard procedure (`methodToTest`).

Registration algorithms. The first registration algorithm is composed of the pair `crestLines`/`crestMatch`. `crestLines` extracts salient lines from the images (one per input images) and `crestMatch` finds a transformation between the produced crest-lines. The `crestMatch` service returns (i) a transformation which is passed to the 3 remaining registration algorithms and (ii) a comment string which will be appended to the transformation file by the `writeResult` service. The `PFMatchICP`/`PFRegister` algorithm is a robust variant of crest-match. First, the `PFMatchICP` service selects some relevant matching points from the lines produced by `crestLines` ; `PFRegister` then produces (i) a transformation and (ii) a comment. Similarly, `Baladin` and `Yasmina` are initialized with the transformation produced by `CrestMatch` and produce a transformation as well as a comment. Each transformation is first downloaded from the grid to a local storage space by the `getFromGrid` service, then converted to a suitable format by the `formatConversion` service and finally written inside the result file by the `writeResult` service. `writeResults` has 5 input parameters: the name of the two registered images, the transformation found by the registration algorithm, the corresponding comment and the name of the result file.

Accuracy estimation. Finally, the estimation of the accuracy is performed, for each algorithm, by the `bronze standard` service. This service takes as input the name of the result file (`fileName`) and the name of the registration algorithm to assess (`methodToTest`). The iteration strategy between its ports is an all-to-all: for each method to test, the file name has to be the same. This service is a synchronization barrier. Indeed, it has to wait for *all* the data items to be processed by all the registration algorithms to begin its execution. This synchronization barrier is expressed with 4 coordination constraints in Scufl. Actually, the synchronization here acts at two different levels. First, a synchronization has to be done between all the data items of a given registration service: this is done by a given coordination constraint. Then, the 4 registration algorithms have to be synchronized, which is done by the "diamond" pattern created by the 4 coordination constraints.

3.1.3 Semi-automatic workflow generation by merging

At this point, one could have noticed that the generation of the workflow of the bronze standard application may not be completely straight-forward. Even without considering implementation details such as the compatibility between data formats exchanged by the algorithms, including or removing a particular registration algorithm from the whole workflow involves a global understanding of the application which may not always be the case of an end-user. Indeed, in an ideal scenario, this application could be exposed (i) to clinicians that would specify the data to use to run an existing evaluation procedure and (ii) to medical image analysis scientists that may want to assess the accuracy of their own registration algorithm with respect to standard ones by including it into an existing bronze standard workflow. In this scenario, the whole bronze standard workflow is built from the basic ones corresponding to the registration algorithms. One could for instance merge two existing workflows as depicted on figure 3.4. In a Service-Oriented Architecture (SOA), this problem can be addressed by considering that the two basic workflows are themselves services (they are called *composite* services) that could be composed in order to build a new workflow. Apart from breaking the loose coupling hypothesis of SOAs, composing composite services is not suitable as it may lead to performance or even semantic problems in the application, in particular in case of overlapping services, as discussed in [Nemo et al., 2007a] and in [Nemo et al., 2007b] for the particular example of the bronze standard application. Considering for instance the example of figure 3.4, it is obvious that a trivial composition of the two basic workflows would lead to two different invocations of the `CL` service, which is clearly not efficient. Elaborating workflows merging strategies is thus needed two fulfill such scenarios.

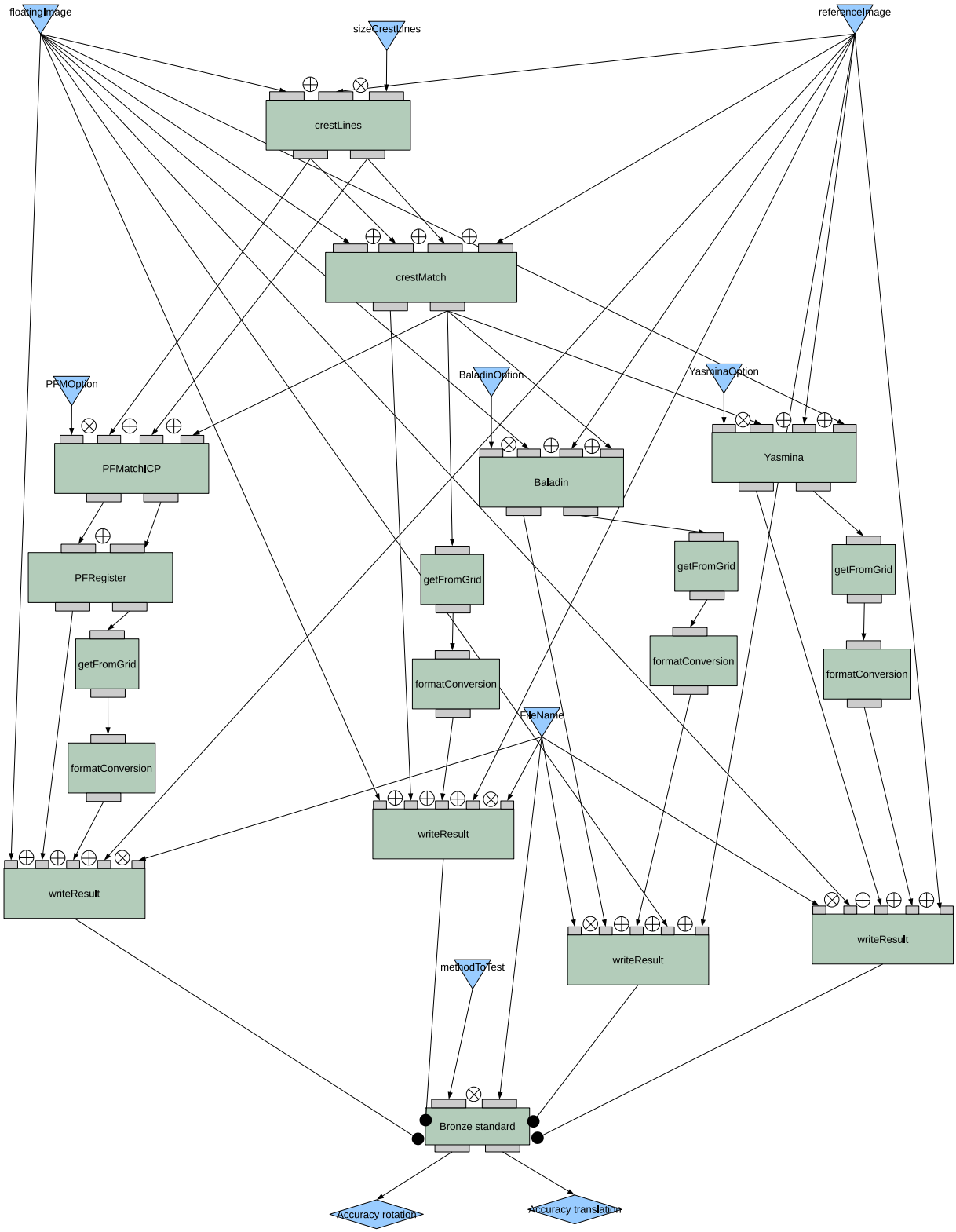

Figure 3.3: Workflow of the bronze standard application. Green boxes represent Web-Services and data dependencies are depicted with arrows. Coordination constraints are figured with dot-terminated arrows. Inputs are figured with blue triangles and outputs with blue diamonds

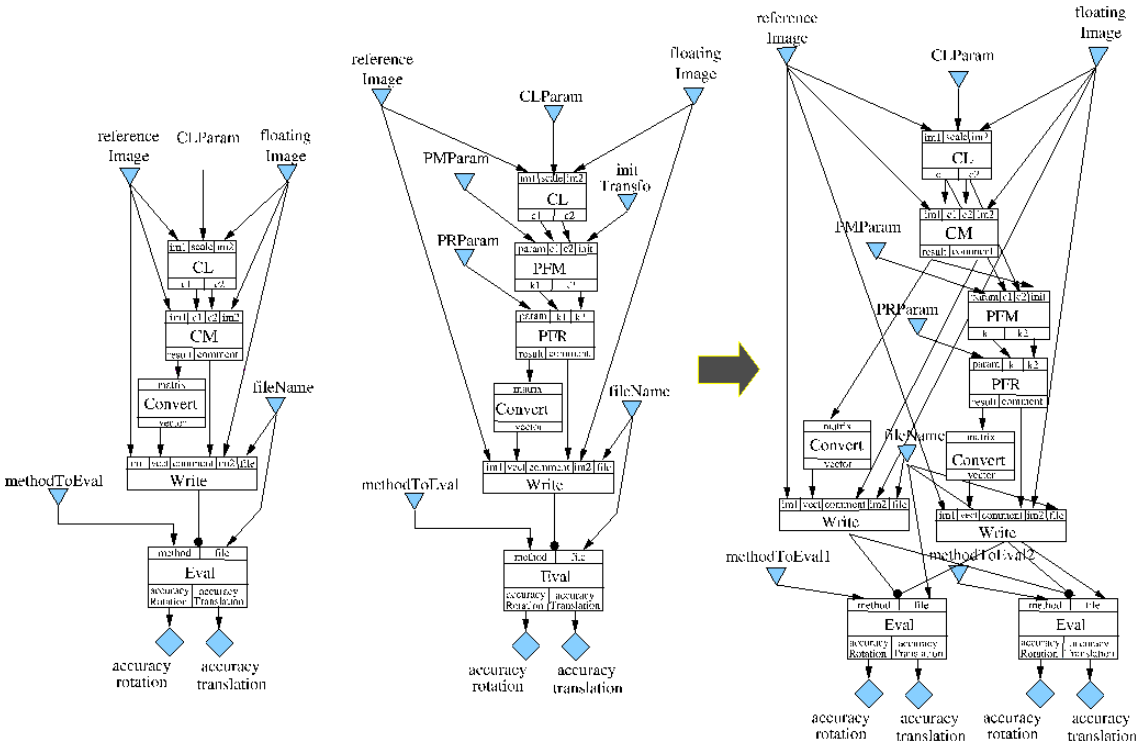

Figure 3.4: `crestMatch` registration workflow (left), `PFMatch` registration workflow (center) and merged bronze standard workflow (right).

3.2 Expressiveness of the selected workflow language

The reader may still wonder whether a simple service language such as Scufl is expressive enough to describe scientific applications. As detailed in section 2.2.3, this language remains very simple and does not include any control constructs (they have to be implemented by specific services). In this section, a study of the expressiveness of the Scufl language is provided. Among the available approaches to evaluate the expressiveness of a workflow language, an implementation of a Turing machine is described here. Beyond the theoretical result obtained at the end of the section, achievements and limitations of this implementation help to understand the capabilities of the Scufl language.

3.2.1 Description of the Turing machine

Turing machines are the most formal way to prove the expressiveness of a language. According to the Church-Turing hypothesis, every computable function can be computed by a Turing machine. Therefore, every language that is as expressive as a Turing machine is said to be Turing complete and would be able to implement any algorithm. A direct way to show that a language is Turing complete is to implement a Turing machine with it, as done for instance in [Veldhuizen, 2003] to show that C++ templates are Turing complete. Yet, little Turing completeness proofs can be found in the literature for workflow languages.

A Turing machine is made of a tape, a head, a state and a transition function. The tape contains cells where symbols belonging to a finite alphabet are printed. The set of states is finite too. Particular states of the machine are the initial one and the set of final ones. The head is positioned on a given cell of the tape. At each iteration:

1. The head reads the current symbol on the tape.
2. The transition function produces a new state, a new symbol and a head shift from the current state and symbol.
3. The head writes the new symbol on the tape and moves one cell left or right depending on the shift given by the transition function.
4. The state of the machine is updated with the new state. If the new state is final, then the machine halts. Otherwise, a new iteration starts.

This description is quite minimal and completely informal. A complete presentation of Turing machines is provided *e.g* in [Lewis and Papadimitriou, 1981].

We assume the realistic but restrictive hypothesis that the tape of the implemented Turing machine is finite, thus preventing the algorithm to use an unbounded amount of resources. Figure 3.5 presents the implementation of a Turing machine in Scufl and is detailed in the next paragraphs.

The three sources `Ribbon`, `initState` and `stopState` respectively contain the input tape of the Turing machine, the initial state of the machine and the final states. The `ReadInitSymbol` processor is used to initialize the machine with the first symbol to be interpreted. It has two parameters, the tape and the initial index i and simply extracts the i^{th} character of the string obtained from the tape. The following symbols will be read by the `readSymbol` processor whose enactment is conditioned by the failure of the halting test. The obtained symbol is piped to the `Transition` processor which combines it with the current state of the machine to produce a new state (`outState`), a new symbol (`outSymbol`) and a movement of the head.

The new state produced is looped back to the input of the `Transition` processor. Self-looping allows the `Transition` processor to maintain the state of the Turing machine (remember that the use of global variables is not possible in Scufl). At a given iteration of the machine, the current state is obtained by proper data composition on the inputs of the `Transition` processor. This processor is the core of the Turing machine, as it implements the transition rules. In this section, it is assumed to be implemented with a Beanshell processor, which captures the whole logic with a piece of Java code. A detailed Scufl implementation of this processor is presented in the next section.

The `movement` output of the `Transition` processor is passed to the `moveHead` processor. This processor only computes a new value of the head index from the shift passed by `Transition` and the current index value. Here again, the current value of the `index` variable

is maintained thanks to self-looping: the output of the `moveHead` processor is connected to its `index` input. The `zero` string constant is also connected to the `index` input of `moveHead` to initialize the index value to 0.

The new index *i* generated by `moveHead` is piped to the `write` processor as well. This processor also takes as input the `outSymbol` returned by `Transition` and the current tape of the machine. It replaces the i^{th} character of the current tape by `outSymbol` and returns the obtained new tape. The state of the current tape is kept thanks to a self-looping.

`Transition` also returns the new state of the machine (`outState` parameter) which is passed to the `testHalt` processor. `testHalt` compares it to the final states provided as input of the workflow and returns a Boolean string piped to the `Fail_if_true` conditional processor. Thus, if the current state of the machine corresponds to a final state, then the conditional processor fails and `readSymbol` does not fire, which makes the whole workflow stop because of the lack of symbol to consume. Else, `readSymbol` reads the next symbol and the machine iterates once again.

Finally, the `result` output of the workflow contains a history of the values of the tape, the last one being the result of the Turing machine.

The data composition operators of all the processors of the workflow except `testHalt` are one-to-one operators because the processors have to correctly match the current values of the tape, head index and/or symbol over the successive iterations. The `testHalt` processor has to test the value of the current state of the machine with *all* the final states, which justifies the presence of an all-to-all operator between its input ports.

3.2.2 Example on a string length computation

The above-described workflow was implemented inside the Taverna workbench and executed with MOTEUR, our home-made Scufl engine described in the next chapter. It was tested on examples described with the Turing Machine Markup Language (TMML[1]) which provides an easy-to-parse XML description of Turing machines.

In particular, a string length computation Turing machine was implemented. Its initial state is `start` and it has 2 final states, namely `string_is_null` and `stop`. 7 other states can be reached. At the end of the computation, the tape contains only an integer, which represents the length of the initial string.

The right of figure 3.6 displays this Turing machine executed with MOTEUR. The initial tape was the string `ab`. A total amount of 20 symbols have been read by the machine and passed to the `Transition` processor. The `testHalt` processor run 42 times. Indeed, including the initial state, 21 states have had to be tested by this processor and for each state to test, 2 invocations are required because there are 2 final states to compare with. The conditional processor

[1] `http://www.unidex.com/turing/index.htm`, (c) 2001 Unidex, Inc.

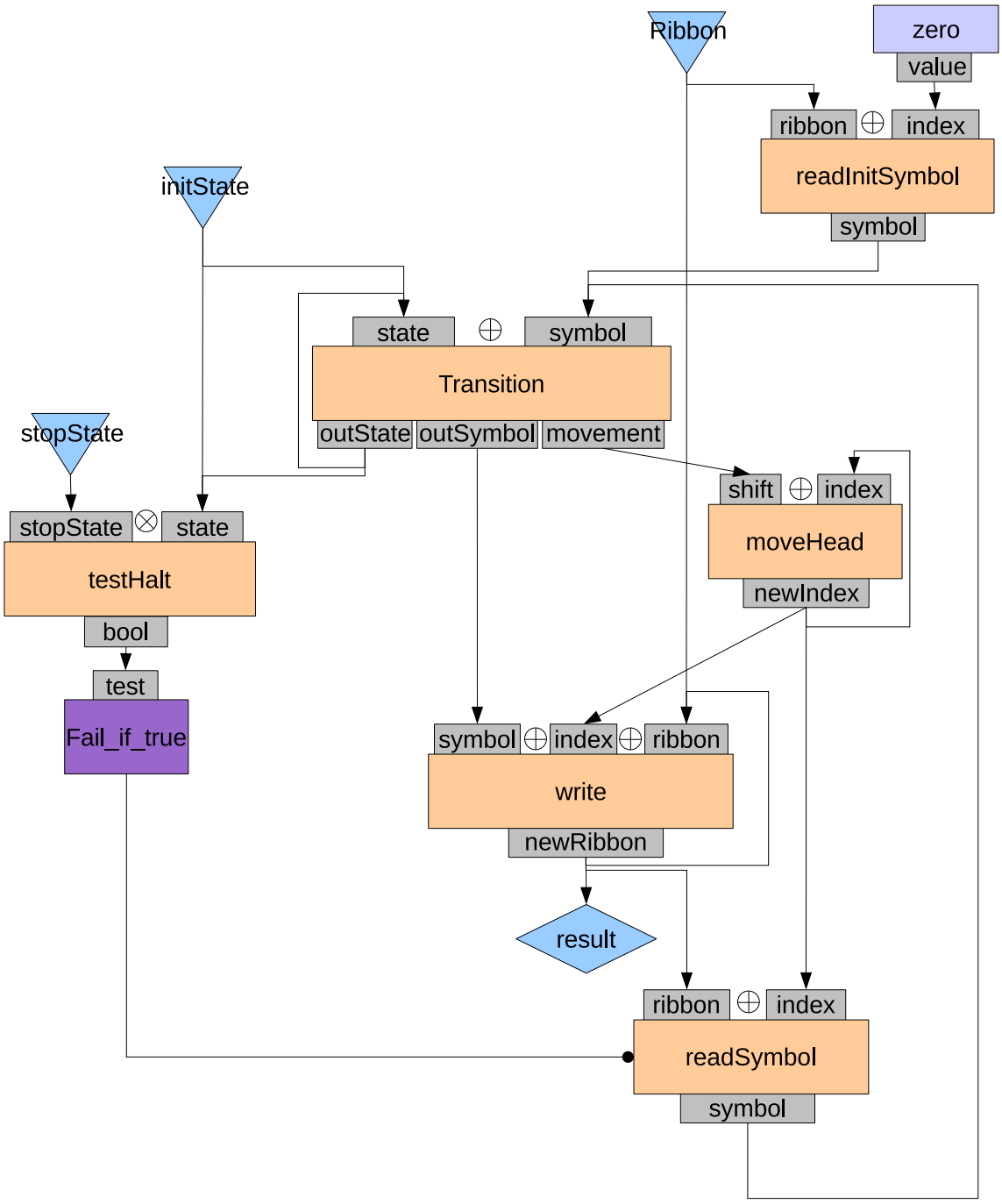

Figure 3.5: Implementation of a Turing machine in Scufl.Orange boxes represent Beanshell processors and the purple one is a conditional processor. Sources and sinks are pictured with blue triangles and diamonds and the blue rectangle is a string constant. Arrows denote data links and the dot-terminated line is a coordination constraint.

only failed once, the last time it was invoked. The left of figure 3.6 shows the corresponding tape obtained for each iteration. The last one effectively only contains the length of the initial tape.

3.2.3 Limitations of this implementation

Parallelism exploitation: Scufl is intrinsically a parallel language: it allows processors to be iterated on several data sets (through the `workers` attribute of the `processor` tag). Given a suitable engine, data parallelism and pipelining can be exploited from the Scufl representation to obtain an efficient execution. However, in this Turing machine implementation, enabling parallelism would completely puzzle the execution. For instance, if several different tapes are provided as input, the current state, index and tape of the machine could not be properly maintained. The use of a sub-workflow wrapping the Turing machine could help to cope with this problem.

Synchronization between conditional test and `readSymbol`: A more fundamental limitation of this Turing machine implementation is the synchronization between the conditional `Fail_if_true` processor and the `readSymbol` one. The firing of the `readSymbol` determines the firing of the `Transition` and subsequent processors. In Scufl, in absence of coordination constraint, the firing of a processor is determined by the availability of data items in its input ports. Because of the loops included in the workflow of this implementation of the Turing machine, data items are always available in the inputs ports of the `readSymbol` processor. Therefore, one should guarantee that the conditional processor is fired *before* each invocation of the `readSymbol` processor. Otherwise, the `readSymbol` processor could fire several times between two consecutive invocations of the `Fail_if_true` processor. This would certainly lead to some errors because of wrong halt detection. We solved this problem by firing the processors in a sequence order in our MOTEUR workflow engine, thus ensuring that the `Fail_if_true` processor is always fired before the `readSymbol` one. However, this kind of behavior is not specified in the Scufl document and is not handled by Taverna. A specification of the behavior of the engine should probably be included in the Scufl language, as it is done for instance in the MoML data-flow language through the definition of specific *directors* [Ludäscher et al., 2005].

Code wrapping: As every workflow language, Scufl is able to define invocations to services, whose implementation is external to the workflow specification. Thus, one should keep in mind that some logic of the Turing machine implementation is embedded into those processors whose code is written using a traditional programming language such as Java. Therefore, the expressiveness of Scufl is tightly coupled to the one of the languages used to implement the processors. To properly assess the expressiveness of Scufl, one should be aware of that and limit the amount of non-Scufl code included inside the processors. In particular, the `Transition`

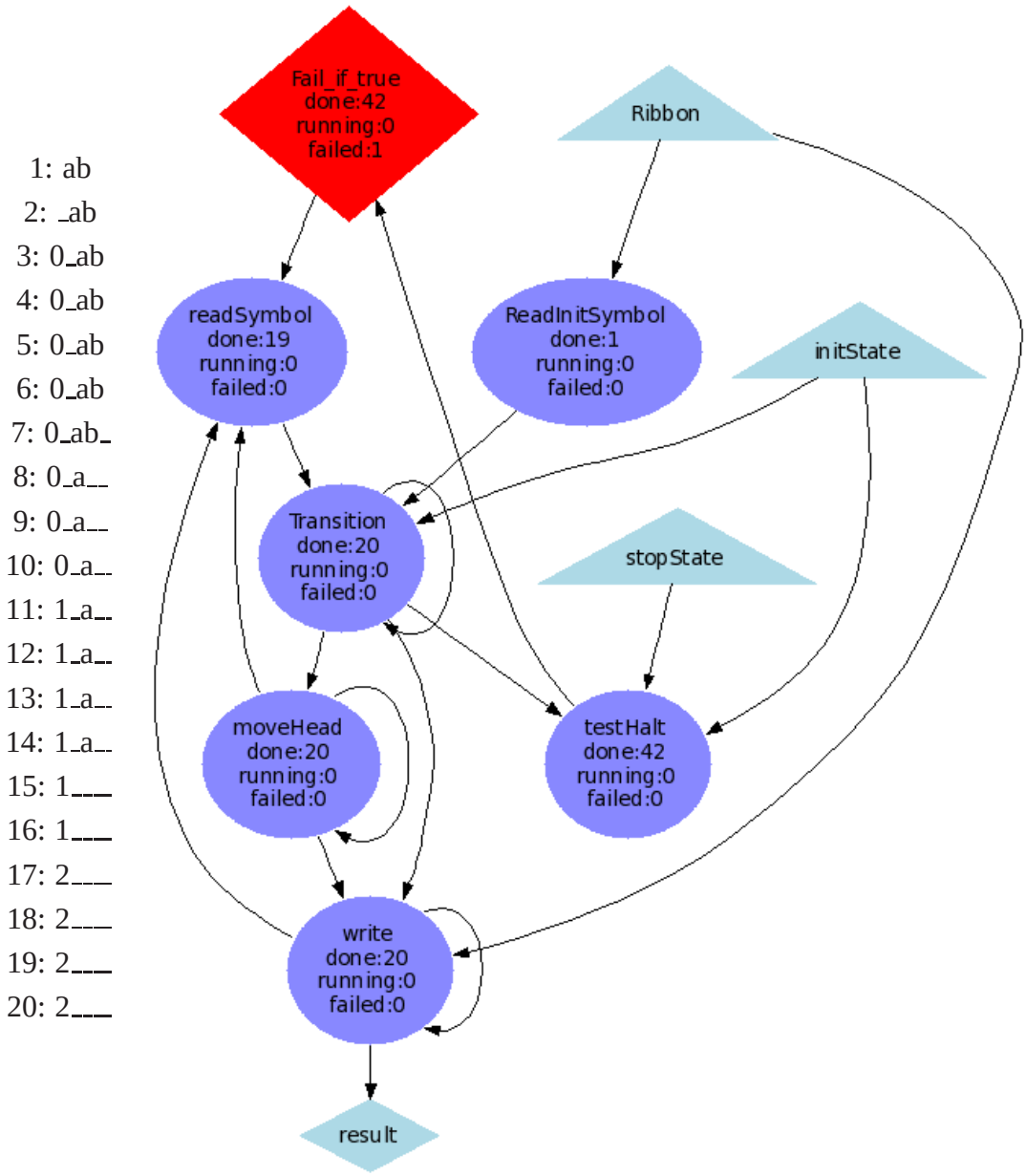

Figure 3.6: Right: Run of the Turing machine on a string length algorithm through the MOTEUR engine. Ellipses represent Beanshell processors, triangle are sources, rectangles are string constants, blue diamonds are sinks and the red one is a conditional processor. Iterations numbers of the processors are written inside. Failed processors are colored in red whereas successful ones are in dark blue. Port names are omitted to ease legibility. Left: corresponding states of the tape for each iteration. White spaces are figured by '_'.

processor includes a whole set of tests to implement the transition rules. Exaggeratedly, putting all the Turing machine logic inside a single processor would produce a correct implementation but would not prove anything about the expressiveness of the Scufl language. On the other hand, preventing the workflow from using any external processor is not relevant because no arithmetic operators are available by default in the Scufl language: under this hypothesis, the implementation of an incrementation or string concatenation would not be possible at all.

3.2.4 A universal Turing machine in Scufl

The above-described implementation of the Turing machine is not universal because the `Transition` processor has to be implemented for every set of rules. Moreover, as already suggested, it embeds a significant amount of code, which limits the evaluation of the expressiveness of the Scufl language. To cope with those limitations, the implementation of the `Transition` processor is detailed in Scufl in this sub-section.

The corresponding workflow of this processor is depicted on figure 3.7. It is made of a sub-workflow (`Nested_Workflow`) which tests the matching between a given transition rule and the current state and symbol of the machine. This sub-workflow has 7 different inputs:

- `currentState` and `currentSymbol` denote the current parameters of the machine. They must be compared to the conditions of the tested transition rule.
- `inState` and `inSymbol` are the conditions of the tested transition rule.
- `outSymbol`, `outState` and `movement` are the consequences of the transition rule. These are the value that must be returned by the sub-workflow if the tested transition rule matches the current parameters of the machine.

The sub-workflow first tests the equality of the current parameters of the machine with the conditions of the tested transition rule. This is done through the `is_equal` and `is_equal1` processors that just compare two strings and return a Boolean, which is tested by the conditional `Fail_if_false` and `Fail_if_false1` processors. If *both* of the conditions are true, then the outputs of the rule are piped to the outputs of the sub-workflow through `nop` processors. Otherwise, the sub-workflow fails and do not return anything.

The `Nested_Workflow` sub-workflow is embedded into a global workflow. This is required (i) to allow to define iteration strategies between the different inputs of the sub-workflow even if they are not all connected to the same processor and (ii) to allow the `nop` processors to return only the correct output parameters of the transition rule. Indeed, if the sub-workflow was alone iterated on the whole transition rules, the `nop` processor would produce the complete set of `outState` parameters as soon as the `Fail_if_false` processor would succeed.

The iteration strategy of the `Nested_Workflow` is depicted on figure 3.8. On the left side of the picture, the current state and symbol of the machine are composed with a one-to-one

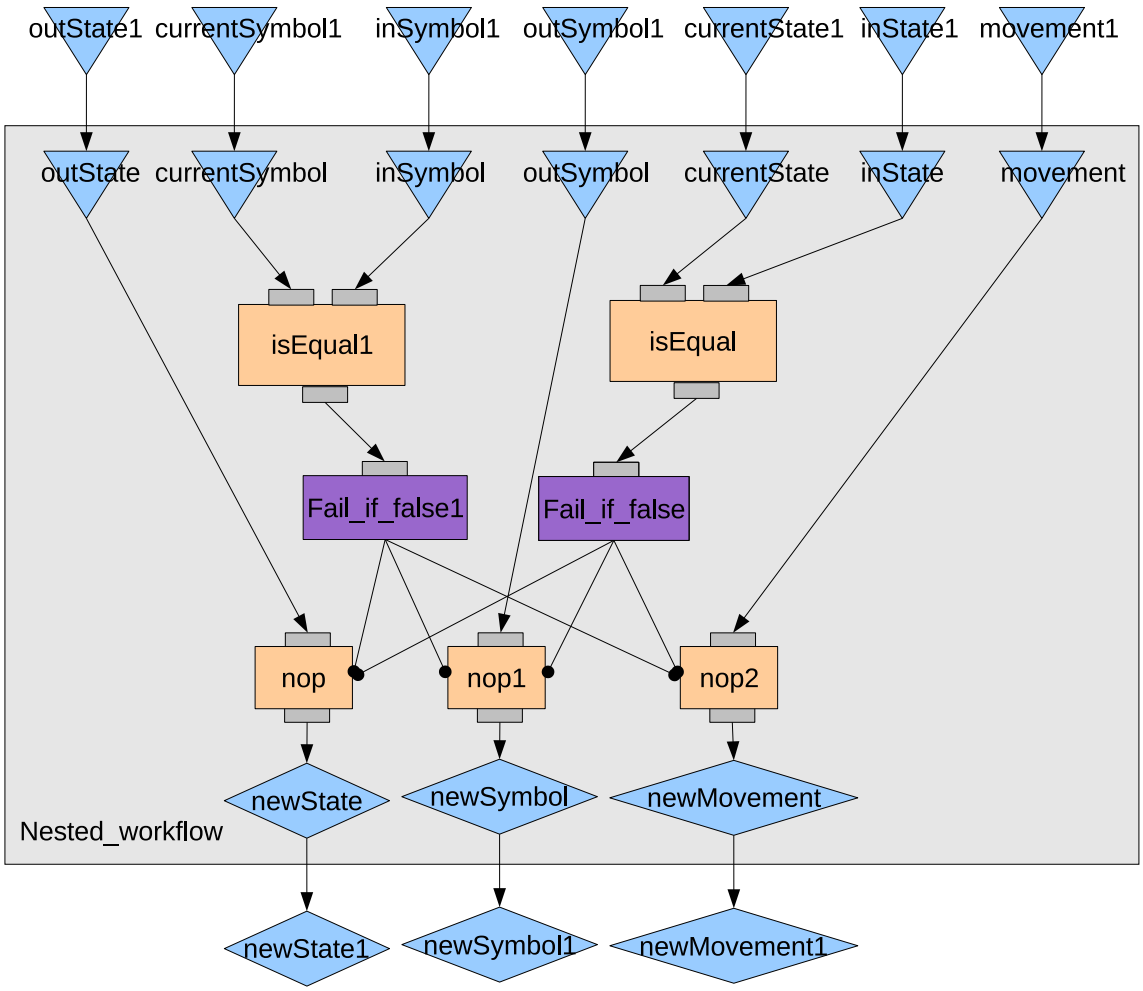

Figure 3.7: Implementation of the Transition processor in Scufl. The use of a sub-workflow allows to define iteration strategies over the sources of `Nested_Workflow` and to properly separate the input data set.

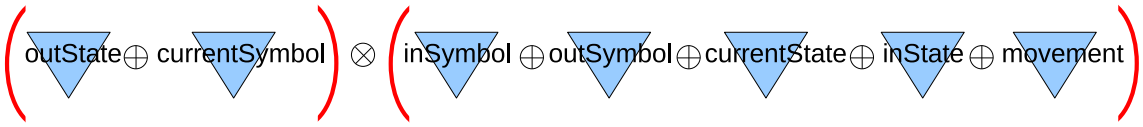

Figure 3.8: Iteration strategy of the `Nested_Workflow` sub-workflow of figure 3.7. The current parameters of the machine are compared with all the transition rules.

operator, to be able to associate only the right symbol with the right state. Similarly, on the right side of the picture, all the items of the transition rules are composed with one-to-one operators. The two terms in brackets are composed with an all-to-all operator, in order to test the current state and symbol with *all* the transition rules of the machine.

3.3 Conclusions

Service workflows were selected as a suitable paradigm to implement medical image analysis applications. The main rationales for this choice is their ability to separate clinical, medical image analysis and grid concerns, to handle dynamic data sets and to provide a compact description of large workflows. In particular, data composition operators of the Scufl language allow a simple way to describe complex applications. The workflow of the bronze standard application has been described with details and implemented in Scufl using the Taverna workflow manager[2]. Finally, the expressiveness of the Scufl language has been studied through the implementation of Turing machines. A universal Turing machine has been implemented using the Scufl data-flow language. Even if some restrictions remain (such as the embedding of some Java code in external processors), it is thus possible to conclude that Scufl is a Turing complete language. Therefore, it would theoretically be possible to implement any algorithm in Scufl. It highlights the facts that the expressiveness of such a data-flow oriented language is not as limited as one could think, even if the language remains very simple and easy to manipulate.

Yet, even if, as shown in this chapter, the workflow description language is quite satisfying, there is still room for an effective interface of workflow managers with grid infrastructures. Considering for instance the example of the EGEE European grid project[3], the majority of the applications is still using a low level grid-expert approach for workflow deployment, relying on scripts or task-graphs (Condor DAGMan tool). In this case, a specific knowledge of the middleware is necessary. Grid-enabled service workflow managers could leverage this issue, opening the door to a wider adoption of the grid in various communities. Keeping in mind this necessity for grid-interfaced high-level service workflows, an implementation of an efficient

[2] http://taverna.sourceforge.net
[3] http://public.eu-egee.org/

service workflow engine is detailed in the next chapter.

Part II

W

Chapter 4

The MOTEUR engine for service workflows

Contents

Services workflows have been shown to be a suitable way of describing medical image analysis workflows. In this chapter, an implementation of a Scufl workflow engine is proposed. This development is motivated by the implementation of a fully parallel service workflow engine. The handling of iteration strategies in a fully parallel workflow engine is not straightforward and a dedicated algorithm is thus proposed. Finally, MOTEUR, a hoMe-made OpTimizEd scUfl enactoR is described and its performance is analyzed.

Dans ce chapitre, l'implémentation d'un moteur de flots basé sur le langage Scufl est proposée. Ce développement est motivé par l'implémentation d'un moteur de flots de services. Ceux-ci constituent une approche adaptée à la description d'applications d'analyse d'images médicales. La gestion des stratégies d'itération dans un moteur de workflows pleinement parallèle n'est pas triviale et nous présentons un algorithme qui lui est dédié. Enfin, MOTEUR est décrit et ses performances sont évaluées.

Service workflows are a suitable way to describe medical image analysis applications. However, as discussed in chapter 2, optimizing their performance is not straightforward because of two main reasons. First, services are black boxes bound to an endpoint and they isolate the workflow manager from the underlying execution infrastructure, preventing it from controlling the resources allocation. Second, in service workflows, the number of computing tasks cannot be forecast before the actual execution of the application, which limits the applicability of scheduling heuristics. Yet, performance remains a major concern for the execution of scientific applications, in particular on the grid platforms that are targeted by this work. In this chapter, we perform a first step towards an efficient grid execution of service workflows by identifying the parallelism levels that could be achieved in such a workflow paradigm (section 4.1). In the subsequent fully parallel service workflows, handling the data composition operators described in chapter 3 is problematic. Indeed, the order of data items may be completely disturbed by the concurrent execution of different jobs and a dedicated algorithm has to be designed in order to keep track of their provenance during the execution. Such an algorithm is presented in section 4.2 and implemented into our parallel service workflow engine: MOTEUR.

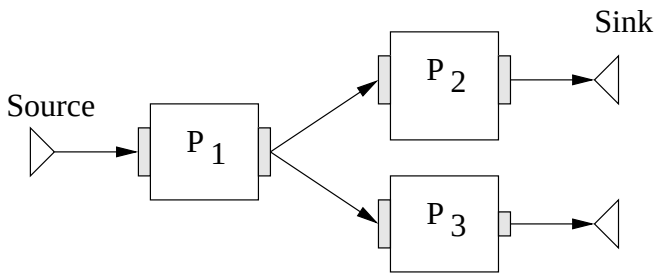

Figure 4.1: Example of a service workflow exploiting workflow, data and service parallelism.

4.1 Parallelism exploitation in service workflows

4.1.1 Asynchronous service calls

To enable parallelism during the workflow execution, multiple application services have to be invoked concurrently. The calls made from the workflow enactor to these services need to be non-blocking in order to exploit the potential parallelism. GridRPC services may be called asynchronously as defined in the standard [Nakada et al., 2005]. Web Services also theoretically enables asynchronous calls. However, the vast majority of existing web service implementations do not cover the whole standard and none of the major implementations [Van Engelen and Gallivan, 2002, Irani and Bashna, 2002] do provide any asynchronous service calls for now. Alternatively, asynchronous calls to Web-Services may be implemented at the workflow enactor level, by spawning independent system threads for each processor being executed.

4.1.2 Workflow parallelism

Given that asynchronous calls are possible, the first level of parallelism that can be exploited is the intrinsic workflow parallelism depending on the graph topology. For instance, if we consider the simple example shown in figure 4.1, processors P_2 and P_3 may be executed in parallel on any data item. This level of parallelism is implemented in all the existing services workflow managers.

4.1.3 Data parallelism

Several input data segments are likely to be processed using a given workflow. Services can be instantiated as several computing tasks running on different hardware resources and processing different input data segments in parallel. *Data parallelism* denotes that a service is able to process several data segments simultaneously with a minimal performance loss. Enabling data parallelism of course implies that the services are able to process many parallel connections.

P_3	X	D_0 D_1 D_2
P_2	X	D_0 D_1 D_2
P_1	D_0 D_1 D_2	X

Figure 4.2: Data parallel execution diagram of the workflow of figure 4.1

Consider the simple workflow made of 3 services and represented on figure 4.1 and suppose that we want to execute this workflow on 3 independent input data segments D_0, D_1 and D_2. The data parallel execution diagram of this workflow is represented on figure 4.2. On this kind of diagram, the abscissa axis represents time. When a data segment D_i appears on a row corresponding to a processor P_j, it means that D_i is being processed by P_j at the current time. To facilitate legibility, D_i denotes the piece of data resulting from the processing of the initial input data set D_i all along the workflow. For example, it is implicit that on the P_2 service row, D_0 actually denotes the data resulting from the processing of the input data set D_0 by P_1. Moreover, the processing time of each data segment by each service is assumed to be constant, thus leading to cells of equal widths. Data parallelism occurs when different data segments appear on a single square of the diagram whereas intrinsic workflow parallelism occurs when the same data segment appears many times on different cells of the same column. Crosses represent idle cycles.

Fully taking into account data parallelism is critical in service workflows, whereas it does not make any sense in tasks-graphs. Indeed, in this case it is covered by the workflow parallelism because each task is explicitly described in the workflow description.

4.1.4 Service parallelism and synchronization barriers

Input data segments are likely to be independent from each other as for instance in embarrassingly parallel applications. *Service parallelism* corresponds to the concurrent execution of two independent data segments by two different services that are sequentially linked. This pipelining model, very successfully exploited inside CPUs, can be adapted to sequential parts of service workflows. Consider again the simple workflow represented in figure 4.1, to be executed on the 3 independent input data segments D_0, D_1 and D_2. Figure 4.3 presents a service parallel execution diagram of this workflow. Service parallelism occurs when different data segments appear on different cells of the same column (data parallelism is disabled on this diagram). Similarly to data parallelism, this level of parallelism does not make any sense in

P_3	X	D_0	D_1	D_2
P_2	X	D_0	D_1	D_2
P_1	D_0	D_1	D_2	X

Figure 4.3: Service parallel execution diagram of the workflow of figure 4.1

tasks-graphs because it is covered by workflow parallelism.

Synchronization barriers are defined when a service needs to simultaneously process several data segments. It can for example correspond to the computing of a mean on a set of previously computed results, all of them being produced by a single service. In service workflows, synchronization barriers differ from the classical join pattern of tasks-graphs: whereas a join pattern synchronizes the results produced by *different* services, a synchronization barrier corresponds to the synchronization over *several* data segments produced by a *single* service. Consequently, the exploitation of service parallelism is proscribed at synchronization barriers.

In service workflows, exploiting those three types of coarse grain parallelism (workflow, data and services) does not lead to any burden for the user because they can be directly determined from the graph of services. Their exploitation is mandatory to obtain an efficient grid execution of the workflow. Yet, they could lead to definitions problems in the control flow of the application, which may imply operators assuming an order on the data items going through the data pipeline. In particular, Scufl iteration strategies are disturbed by such a fully parallel execution.

4.2 Data composition strategies in a parallel service workflow

A strong motivation for the adoption of service workflows as a relevant approach to describe medical image analysis applications is the availability of data composition strategies, which provide highly expressive operators for the handling of large data sets. In a service workflow, each service may receive several input data items on each of its input ports. Depending on the desired semantics, the user might envisage various input composition patterns between the different ports.

4.2.1 Basic data composition operators

As described in chapter 3, and illustrated on figure 3.1, there are two main data composition operators (one-to-one and all-to-all), very frequently encountered in scientific applications [Oinn et al., 2004]. Note that other composition patterns with different semantics could be defined (*e.g. all-to-all-but-one* composition). However, they are more specific and consequently more rarely encountered. Combining those two data composition operators enables very complex data composition patterns.

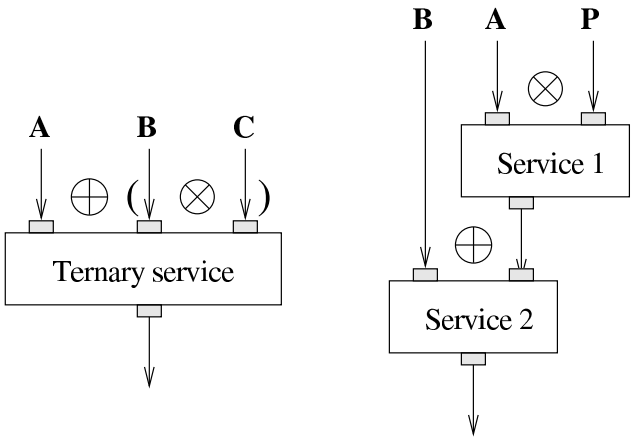

Figure 4.4: When the cardinality of the data set produced from an all-to-all (⊗) operator ($B \otimes C$ on the left example and $A \otimes P$ on the right) differs from the one of the other operand of the one-to-one (A on the left and B on the right), a semantic problem occurs in the definition of the one-to-one (⊕).

A common example of the *all-to-all* composition operator is the case where all pieces of data in the first input set are to be processed with all parameter configurations defined in the second input set (parameter sweep application). In this case, if A and B are two data sets of size n and m respectively, the cardinality of $\mathbf{A} \otimes \mathbf{B} = \{A_1 \otimes B_1, A_1 \otimes B_2 \ldots A_1 \otimes B_m, A_2 \otimes B_1 \ldots A_2 \otimes B_m \ldots\ldots A_n \otimes B_1 \ldots A_n \otimes B_m\}$ is $m \times n$ (For simplification, the result of processing the pair of input data (A_1, B_1) by a service will be denoted $A_1 \otimes B_1$).

The *one-to-one* strategy is the classical case where an algorithm needs to process every pair of input data segments independently. An example is a matrix addition operator: the sum of each pair of input matrices is computed and returned as a result. If two input data sets A and B are considered, we have: $\mathbf{A} \oplus \mathbf{B} = \{A_1 \oplus B_1, A_2 \oplus B_2, \ldots\}$. The implementation of the one-to-one operator in a fully parallel workflow engine is not straightforward. Indeed, this operator assumes that the input operands are ordered. If data parallelism is not activated, then each service can number its output data items on-the-fly, which makes sense as their order cannot be disturbed. If data parallelism is activated but service parallelism is disabled, then every service could keep track of the order by renumbering is output data items after the last one has been processed. But if both data and service parallelism are implemented, then the order of data items may be completely disturbed among different services of the workflow and maintaining a consistent order between them is not trivial. Moreover, if the two input data sets do not have the same size ($m \neq n$), then a precise semantics has to be defined for the one-to-one operator.

4.2.2 Semantics for the one-to-one operator

As illustrated at the left of figure 4.4, the pairwise one-to-one and all-to-all operators can be combined to compose data patterns for services with an arbitrary number of input ports. In this

case, the priority of these operators needs to be explicitly provided by the user (parentheses explicitly express priorities in the figures). In some cases, the semantics of the one-to-one is not well defined. For instance, if the input data sets are $\mathbf{A} = \{A_0, A_1\}$, $\mathbf{B} = \{B_0, B_1\}$, and $\mathbf{C} = \{C_0, C_1\}$, then the all-to-all operator between B and C produces 4 data items ($B_0 \otimes C_0$, $B_1 \otimes C_0$,$B_0 \otimes C_1$,$B_1 \otimes C_1$) whereas the cardinality of A (to be composed by a one-to-one) is only 2.

In the Taverna workbench (version 1.5 has been tested[1]), a truncation of the $B \otimes C$ data set is done, thus producing:

$$\mathbf{A} \oplus_{Taverna} (\mathbf{B} \otimes \mathbf{C}) = \{ \; A_0 \oplus (B_0 \otimes C_0), \quad A_1 \oplus (B_1 \otimes C_0) \; \}$$

This semantics is ambiguous: first, it is dependent on the order of computation of the elements of $B \otimes C$, which is completely arbitrary. For instance, it seems that there is no objective reason for composing A_1 with $B_1 \otimes C_0$ rather than with $B_0 \otimes C_1$. Second, this semantics prevent the all-to-all operator from being commutative. Indeed, if ports B and C of the `ternary service` of figure 4.4 are switched, then Taverna will produce the following data set:

$$\mathbf{A} \oplus_{Taverna} (\mathbf{B} \otimes \mathbf{C}) = \{ \; A_0 \oplus (C_0 \otimes B_0), \quad A_1 \oplus (C_1 \otimes B_0) \; \}$$

which differs from the previous one.

This semantic flaw is not restricted to this particular example. Actually, similar issues appear as soon as a one-to-one operator is applied subsequently to an all-to-all operator. For instance, the example given at the right of figure 4.4 corresponds to a classical situation where an input data set $\mathbf{A} = \{A_0, A_1\}$, is processed by a first algorithm (using different parameter configurations $\mathbf{P} = \{P_0, P_1, P_2\}$), before being delivered to a second service that compares the results with a matching number of data items $\mathbf{B} = \{B_0, B_1\}$. For example, in the medical image analysis context, `Service1` could correspond to a smoothing of the images included in A with the parameters put in P, performed prior to a pairwise registration with the images contained by the set B implemented by `Service2`. In this case, the user wants `Service2` to combine $A_i \otimes P_j$ with B_i. Yet, the output data set according to the semantics adopted by Taverna would be:

$$\mathbf{B} \oplus_{Taverna} (\mathbf{A} \otimes \mathbf{P}) = \{ \; B_0 \oplus (A_0 \otimes P_0), \quad B_1 \oplus (A_0 \otimes P_1), \; \} \tag{4.1}$$

In this case, the output data set is not only truncated but also mismatched: comparing $A_i \otimes P_j$ with B_j does not make any sense in our example. Thus, another definition of the one-to-one operator has to be proposed. The method proposed in the next section is to let the user define and control its own one-to-one semantics, by specifying the data sets that are semantically correlated.

[1] http://taverna.sourceforge.net

4.2.3 A new data composition algorithm

The definition of the one-to-one operator proposed here is based on the specification of correlation groups by the user. Indeed, given that two correlated input data sets **A** and **B** are provided, the user can expect that the data A_i will always (*i.e* for any service of the workflow) be analyzed with the correlated data B_i, regardless of the algorithm parameters P_j considered. For instance, the user may define a correlation group between the input data sets **A** and **B** on the workflow displayed on the right of figure 4.4, then producing the following data set, where A_i is always consistently combined with B_i:

$$\mathbf{B} \oplus (\mathbf{A} \otimes \mathbf{P}) = \left\{ \begin{array}{ll} B_0 \oplus (A_0 \otimes P_0), & B_1 \oplus (A_1 \otimes P_0), \\ B_0 \oplus (A_0 \otimes P_1), & B_1 \oplus (A_1 \otimes P_1), \\ B_0 \oplus (A_0 \otimes P_2), & B_1 \oplus (A_1 \otimes P_2) \end{array} \right\} \tag{4.2}$$

On the contrary, if this workflow is executed with a different semantics, then the user may define a correlation group between the input data sets **B** and **P** and in this case, the resulting output data set will be:

$$\mathbf{B} \oplus (\mathbf{A} \otimes \mathbf{P}) = \left\{ \begin{array}{ll} B_0 \oplus (A_0 \otimes P_0), & B_1 \oplus (A_0 \otimes P_1), \\ B_0 \oplus (A_1 \otimes P_0), & B_1 \oplus (A_1 \otimes P_1), \end{array} \right\} \tag{4.3}$$

The idea here is to let the user define its own semantics for the one-to-one operator by defining correlation groups. A group is a set of input data tuples that defines a relation between data items coming from different sets. For instance:

$$\mathrm{G} = \{(A_0, B_0, C_0), (A_1, B_1, C_1), (A_2, B_2, C_2)\}$$

is a group establishing a relation between 3 data sources **A**, **B** and **C**. Elements of a group are called group *instances*: in this example, $G_0 = (A_0, B_0, C_0)$ and $G_1 = (A_1, B_1, C_1)$ are two instances of the group G. Note that those data items can be unambiguously numbered as they belong to *input* data sets that are specified prior to the execution and their order may not be disturbed by parallelism. In this case, group instances establish a semantic correlation between the input data items A_i, B_i and C_i. Of course, more complex groups could be specified by the user, depending on the semantic of the application. For instance:

$$\mathrm{H} = \{(A_4, B_0), (A_1, B_2), (A_2, B_5), (A_6, B_6)\}$$

is a group containing 4 instances establishing a non trivial correlation between data items.

The one-to-one composition operator only makes sense for the processing of related data items. Therefore, only data items belonging to a same group instance should be considered for the processing of the one-to-one operator by any service. When considering a service directly connected to the *input* data sets of the workflow, determining relations between data is straightforward. However, when considering a complete application workflow such as the one of the bronze standard application illustrated in figure 3.3 in the previous chapter, other

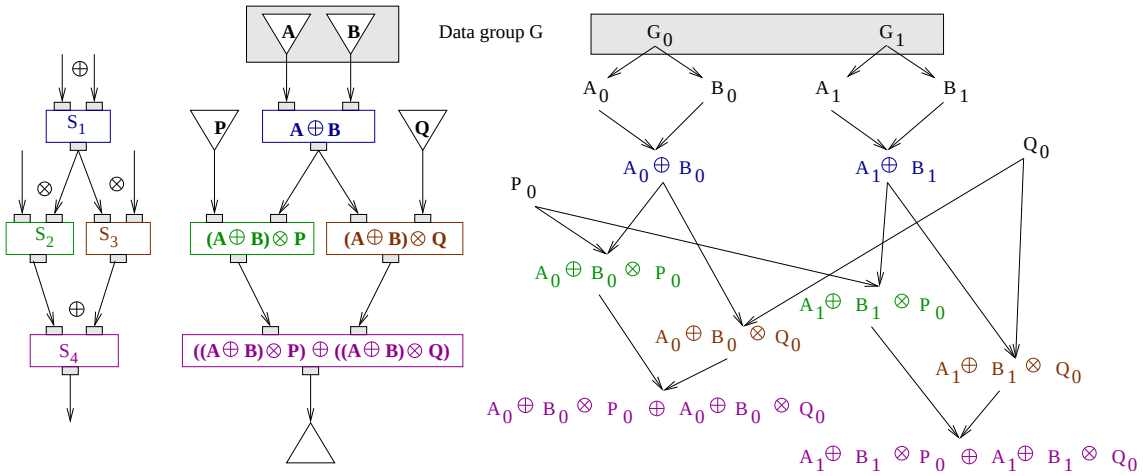

Figure 4.5: Workflow example (left), associated data sets directed graph (center), and the associated directed acyclic data graph.

services need to determine which one of their input data segments are correlated. The one-to-one composition operator does introduce the need for the algorithm described below.

Conversely, note that the all-to-all operator does not rely on any pre-determined relation between input data. Any number of inputs can be combined, with very different meanings (such as data to process and algorithm parameters). Each piece of data received as input yields to one or more invocations of the service for processing.

4.2.3.1 The one-to-one algorithm

The left part of figure 4.5 represents a sample workflow made of 4 services and combining the one-to-one and the all-to-all composition operators. In the center of the figure is represented the directed graph of the produced data sets. Given 4 input data sets, **A**, **B**, **P** and **Q**, the complete workflows produces

$$((\mathbf{A} \oplus \mathbf{B}) \otimes \mathbf{P}) \oplus ((\mathbf{A} \oplus \mathbf{B}) \otimes \mathbf{Q}).$$

as output of the S_4 service. Given the one-to-one operator semantics described above, the data set $\mathbf{A} \oplus \mathbf{B}$ produced by the first service will be non empty if and only if data items in **A** and **B** are related through a group G that is represented in gray at the top of the figure (two group instances are defined so that A_i, the ith element of **A**, is correlated with B_i, the ith element of **B**).

Considering the inputs of service S_4, two input data items $(A_i \oplus B_i) \otimes P_k$ and $(A_j \oplus B_j) \otimes Q_l$ should be combined if and only if $i = j$. Indeed, combining A_i with B_i, or a subsequent processing of these data items, does make sense given that the user established a relation between this input pair through the group instance G_i.

To formalize this approach we need to consider the data production Directed Acyclic Graph that is represented in right of figure 4.5. This graph shows how all data items are combined by the different processings. The *input* data items are parents of all the *produced* data. The formal relation between each data pair (A_i, B_i) is represented through a group instance G_i, parent of

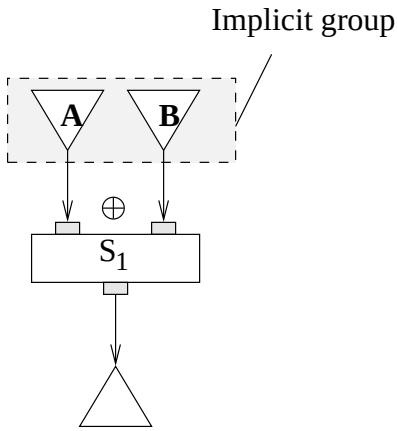

Figure 4.6: Implicit groups definition.

both A_i and B_i. Input data items that have no group parent such as P_0 and Q_0 will be named *orphan* data.

The directed data graph is constructed from the roots (workflow inputs) to the leafs (workflow outputs) by applying the two following simple rules implementing the semantics of the one-to-one and the all-to-all operators respectively:

1. Two data segments (graph nodes) are always combined in an all-to-all operation.
2. Two data segments are combined in a one-to-one operation **if and only if** there exists a common *group instance* ancestor to both data items in the data DAG.

4.2.4 Implicit combinations

The proposed algorithm aims at providing a strict semantics to the combination of data composition operators, while providing intuitive data manipulation for the users. Data groups have been introduced to clarify the semantics of the one-to-one operator. However, it is very common that users are writing workflows without explicitly specifying pairwise relations between the data. The order in which data segments are declared or sent to the workflow inputs are rather used as an implicit relation.

To ease the workflow generation by the user, groups can be implicitly generated when they are not explicitly specified by the user. Figure 4.6 illustrates such a case. The reason for generating an implicit group is straight forward: two input data sets are being processed through a one-to-one service. The systematic rule that can be applied is to create an implicit group for each *one-to-one* operator whose input data sets are orphans. For example, in the case illustrated in figure 4.6, the input data sets **A** and **B** are orphans and bound *one-to-one* by the S_1 service. An implicit group is therefore created between **A** and **B**.

The implicit groups are statically created by analyzing the workflow topology and the input data sets before starting the execution of the workflow.

4.3 Implementation of MOTEUR and overhead quantification

We implemented MOTEUR, a workflow engine taking into account (i) the three kinds of parallelism described in section 4.1 and (ii) the precise one-to-one semantics proposed in section 4.2. MOTEUR uses the Scufl language from Taverna for the workflow description (see sections 2.2.2 and 3.2 for an overview of this language). It supports workflows made with Web-Services, string constants, Beanshells[2] and some local Java classes (such as the `Fail_if` class used to implement conditional branching). More specific processors available in the Taverna workbench such as Biomoby or Talisman ones are not available yet. An interface to DIET GridRPC servers has also been developed. An XML dialect is used to describe input data sets and specify correlation groups between them. It simply describes each item of the different inputs of the workflow. Complex WSDL types are supported, which is not trivial: traditional Web-Services toolkits (Apache Axis[3], gSOAP[4], SOAP::Lite[5]) generate classes from XML type descriptions that are used by the client invocation methods compiled after the type parsing. In the case of a workflow engine, the type has to be handled dynamically: for each complex WSDL type, MOTEUR generates a particular data structure which is instantiated on the data items at runtime. A dedicated serializer enables the conversion from this data structure to the XML (SOAP) expression of the type and is interfaced with the Axis API thanks to design patterns[6].

Enactor implementation. A generic `Processor` class represents a service of the workflow and implements the basic functionality such as the handling of data composition operators between its ports. This class is derived for every kind of Scufl processor and a specific interface is implemented for each of them. A central enactor periodically queries each `Processor` object to determine whether it is ready to be enacted. The corresponding processor then computes the data sets resulting from the application of its data composition strategy on its ports containing the data segments coming from its predecessors. If a given input data set has not been already previously computed, a dedicated thread is then started for the computation. When the computation is finished, the execution thread pushes the results from its output ports to all of its connected input ports.

Synchronization barriers. Implementing synchronization barriers in a data and service parallel workflow requires an in-depth inspection of the service tree. For instance, considering the

[2] http://www.beanshell.org

[3] http://ws.apache.org/axis2/

[4] http://www.cs.fsu.edu/~engelen/soap.html

[5] http://www.soaplite.com/

[6] The handling of complex WSDL types inside MOTEUR owes a lot to the work of Patrick Hoangtrong and Pascal Rolin during their internship at the “Ecole Polytechnique Universitaire” of Nice Sophia-Antipolis in February 2007.

workflow depicted on figure 4.7, where service F synchronizes the data produced by both B and E, it must be guaranteed that B and E have produced *all* their data segments before F starts. A necessary and sufficient condition to ensure this is that *all the ancestors of F are inactive and have processed a non-null number of data sets*. We use this condition to implement the synchronization barriers in MOTEUR.

Data composition algorithm. To implement data composition operators, MOTEUR dynamically resolves the data combination problem by applying the following algorithm.

1. *Initialize the directed acyclic data graph:*
   (a) *Create root nodes for each group instance G_i and add a child node for each related data.*
   (b) *Create root nodes for each orphan data.*
2. *Start the execution of the workflow.*
3. *For each tuple of data to be processed:*
   (a) *Update the data graph by applying the two rules (as defined in section 4.2.3.1) corresponding to the one-to-one and the all-to-all operators.*
   (b) *Loop until there is no more data available for processing in the workflow graph.*

To implement this strategy, MOTEUR needs to keep representations of:

- the topology of the service workflow;
- the data graph;
- and the list of input data that have already been processed by each service.

The data graphs also ensures a full traceability of the data processed by the workflow manager: for each data node, the parents and children of the data can be determined. Besides, it provides a mean to unambiguously identify each data produced. This becomes mandatory when considering parallel execution of the workflow introduced in section 4.1.

4.3.1 Comparison with other service workflow engines

Data composition. The one-to-one and the all-to-all data composition operators were first introduced and implemented in the Taverna workflow manager. They are part of the underlying Scufl workflow description language. In this context, they are known as the *dot product* and *cross product iteration strategies* respectively. The strategy of Taverna for dealing with input sets of different sizes in a one-to-one composition is to produce the $\min(m, n)$ first results only. However, the semantics adopted by Taverna when dealing with a composition of operators

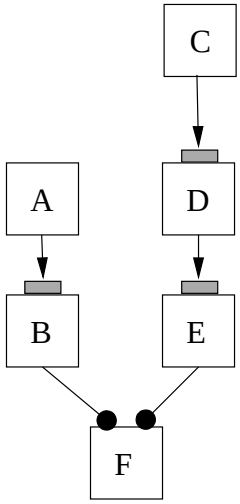

Figure 4.7: Implementation of the synchronization barrier: *F* starts when *A*, *B*, *C*, *D* and *E* are inactive and have run at least once

as illustrated in figure 4.4 is not fully satisfying as already discussed in section 4.2.3. The Kepler and Triana workflow managers only implement the one-to-one composition operator. This operator is implicit for all data composition inside the workflow and it cannot be explicitly specified by the user. We could implement an all-to-all data composition operator in Kepler by defining specific actors but this is far from being straightforward[7]. Kepler actors are blocking when reading on empty input ports. The case where two different input data sets have a different size (common in the all-to-all composition operator) is not really taken into account. Similar work can be achieved in Triana using the various *data stream* tools provided. However, in both cases, the all-to-all semantics is not handled at the level of the workflow engine. It needs to be implemented inside the application workflow.

Parallelism exploitation. Workflow parallelism is available in Taverna, Kepler and Triana. Data parallelism is available in Taverna but service parallelism is not available in this system yet, although it is planned for the coming Taverna II. Kepler implements the service parallelism within its PN director. In this execution framework, each processor (actor in the Kepler vocabulary) is executed on a dedicated thread. Strategies have been developed to cope with nested collections [McPhillips and Bowers, 2005] and to retrieve data provenance [Bowers et al., 2006] in service parallel workflows but data parallelism is not available.

In table 4.1, the characteristics of the main service workflow managers are compared to MOTEUR, considering the data composition operators and the levels of parallelism implemented. It provides a qualitative evaluation of our prototype. To our knowledge, MOTEUR is the only workflow manager that implements the two basic data composition operators and

[7] A study of the implementation of the all-to-all operator in Kepler was done by Lydie Blanchet and Fabien Cordier during their internship at the “Ecole Polytechnique Universitaire” of Nice Sophia-Antipolis in January 2006.

Workflow engines	Data composition		Parallelism		
	one-to-one	all-to-all	Workflow	Data	Service
Taverna	X	X	X	/	O
Kepler	X	O	X	O	X
Triana	X	O	X	O	X
MOTEUR	X	X	X	X	X

Table 4.1: Comparison of the main service-based workflow managers. X: present; /: limited; O: absent

the 3 levels of parallelism at the same time. In particular, it differs from the Taverna workflow manager by its implementation of service parallelism. On production grids, service parallelism is likely to provide a significant speed-up on applications. As detailed in section 6.1.4 of chapter 6, the activation of service parallelism leads to a 1.9 speed-up on the workflow of the bronze standard application running on the EGEE production grid (the speed-up of the application on EGEE with respect to a sequential execution is 13.2 with service parallelism (DSP case) and sinks to 7 without it (DP case)). The tools reported in table 4.1 are exclusively *service* workflow engines, as defined by the taxonomy presented in chapter 2. Engines belonging to other workflow classes may exhibit approaching features, expressed in different ways. For instance, it has already been stated that both data and service parallelism are inherent to task-graphs and do not require any specific handling. Similarly, the parametric tasks available in the P-Grade portal [Kacsuk et al., 2006a] (which is based on a task-graphs paradigm) allow to specify data composition operators that may emulate one-to-one and all-to-all.

4.3.2 Performance evaluation

To handle asynchronous service calls, MOTEUR creates a Java thread for each invocation, which allows parallel service invocations even if the corresponding APIs do not provide asynchronous methods. However, Java threads handling may raise scalability problems. In order to quantify its overhead, MOTEUR was benchmarked on a workflow made of a single service concurrently invoked on several data items. This service has a single string input and returns a single string result as well. This benchmark was performed on a Pentium IV, 1.8GHz, 512MB RAM running Linux 2.6.21. We used Sun's JVM 1.6.0 with a maximum heap size of 350MB and a thread stack size of 100kB. Figure 4.8 plots the overhead of MOTEUR with respect to the number of concurrent service invocations given that the invoked service was a remote server (DIET server or Web-Service), or a local method (Beanshell or Java class). The plotted overhead is the total one generated by the concurrent invocations. To avoid network latencies, the Web-Service and the DIET server were deployed on the same machine as MOTEUR.

For each kind of service, the curve starts with a linear phase before entering a saturation

phase (the end of the linear phase is determined as the abscissa from which the error of the linear approximation of the experimental data is greater than 1s). The slope of the linear phase quantifies the scalability of the interface before it saturates. As it could have been expected, local method calls (Java class and Beanshell) are the most scalable. For Java classes, the overhead (6ms/invocation) only comes from the Java thread creation whereas the invocation of a Beanshell also requires the loading of a Java interpreter (Beanshells dynamically interpret Java code), which makes the overhead grow to 16ms/invocation. Among remote method calls, DIET servers are the most scalable in the linear phase, with an overhead of 33ms/invocation whereas the one of Web-Services is 76ms/invocation. This is consistent because DIET servers are invoked through a lightweight binary protocol whereas Web-Services use an XML text-based protocol which requires costly (de-)serializations.

Yet, on these experiments, the saturation phase of the DIET interface begins at 230 concurrent invocations, which is quite poor compared to the Web-Services, which stay on the linear phase until 500 invocations. This could be explained by our naive implementation of the MOTEUR/DIET interface: in MOTEUR, as for every other kind of services, a dedicated object is responsible for the whole DIET request, from the initialization to the completion. However, before being able to perform any request, the DIET API requires an initialization procedure which is quite heavy in terms of memory (in this experiment, the initialization requires 1.3MB). In our implementation, the initialization is performed before *every* DIET request, which rapidly saturates the memory. We are investigating solutions to cope with this problem with the DIET team. A Java API for DIET, currently under testing, should help resolving the problem.

Comparatively, Beanshells enter their saturation phase for 1100 concurrent invocations and local Java classes for 1400. Yet, even for 1000 concurrent Web-Services invocations, the resulting total overhead is 2.5 minutes, which is twice lower than the job latency on a production grid infrastructure (about 5 min *per* job, see chapter 6). The bronze standard experiments detailed in section 1.3 of chapter 1 involves about 400 concurrent service invocations, which leads to an overhead inferior to 40 seconds according to this benchmark. This overhead is reasonably low compared to the total computed CPU time of this experiment which is 30 hours.

4.4 Conclusions

In this chapter, we presented the design and implementation of MOTEUR, a workflow engine based on the Scufl language. As concluded at the end of chapter 2, existing workflow languages seem to be sufficiently expressive to describe the majority of applications and we thus focused on the *execution* of the workflow rather than on its description. The development of MOTEUR was motivated by the need for an efficient service workflow enactor. It implements the 3 kinds of parallelism that could be achieved in a service workflow (*i.e* workflow, data and service parallelism). In particular, the presence of service parallelism yields a close to 2 speed-up factor

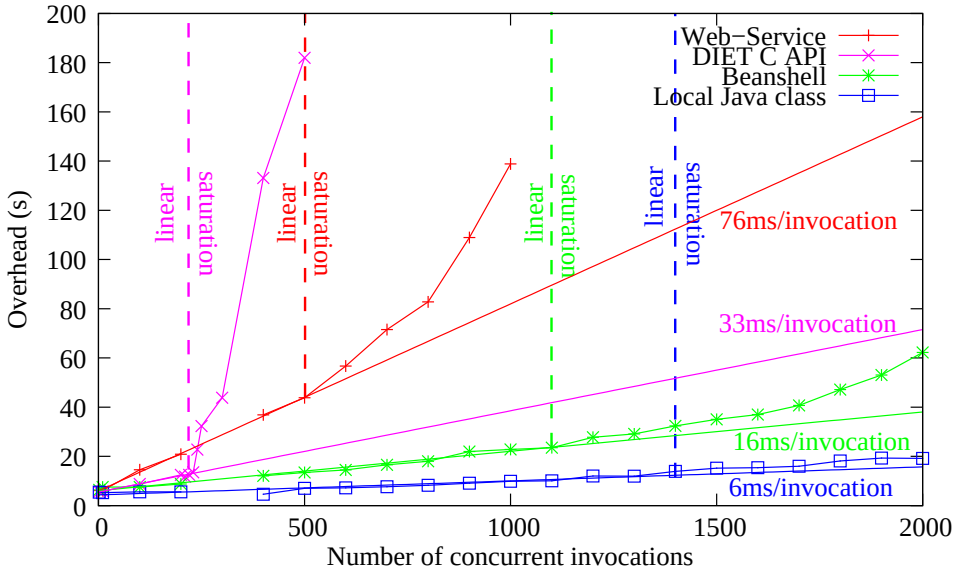

Figure 4.8: Overhead of MOTEUR with respect to the number of concurrent service invocations. For each kind of services, the curve starts with a linear phase before entering a saturation phase.

on the workflow of the bronze standard application running on the EGEE production grid. To make this implementation possible, we proposed a semantically consistent definition of the one-to-one data composition operator. Indeed, in its first implementation inside the Taverna workbench, this data composition operator assumes an order on the data items produced by the workflow, which leads to an unpredictable behavior in a fully parallel engine. In the algorithm that we proposed, the user is responsible for the semantic of this operator through the definition of correlation groups among the input data sets. This new operator is designed to facilitate the description of medical image analysis workflows in a parallel context.

Workflow systems provide a uniform view of an application running on heterogeneous systems and architectures. Therefore, they constitute an interesting tool to compare several grids, as a given application can be transparently executed on different platforms by a single workflow engine. In the next chapter, we exploit this feature of workflows and provide a comparison of various grid systems through the use of MOTEUR. Then, in the following of the manuscript, performance optimization methods are proposed and aim at reducing the impact of the latency on production grids, which is identified to be a major cause of performance drops in chapter 6.

Chapter 5

Production grids versus dedicated clusters

Contents

The goal of this chapter is to compare the performance of a production grid (EGEE) with dedicated clusters of Grid'5000. Even if the latter are obviously providing highly superior speed-ups, quantifying the difference with the former allows to build a reference and determine what could be expected from performance optimization strategies dedicated to production grids. A fair comparison is made possible by the use of a single workflow manager to execute the application on both systems. The overhead of MOTEUR is shown to be negligible on our application which typically involves dozens of hours of CPU time with jobs durations inferior to 10 minutes. Even if a significant speed-up is obtained in production, the grid latency and its variability are determined to be the main causes of performance drops on EGEE. Thus, the last sections of this chapter provide a comparative analysis of the latencies of EGEE and Grid'5000 clusters. Based on it, a model determining a job allocation strategy on those two platforms is finally presented and provides an additional metric for the comparison of those systems.

Le but de ce chapitre est de comparer les performances d'une grille de production (EGEE) à celles obtenues sur des grappes de calcul dédiées de Grid'5000. Même s'il est évident que ces dernières permettent une accélération de l'application supérieure à ce qui peut être obtenu en production, quantifier cette différence est intéressant car cela établit une référence permettant de déterminer ce qui peut être espéré de stratégies d'optimisation dédiées aux grilles de production. Une comparaison objective est possible grâce à l'utilisation d'un unique gestionnaire de flots sur les deux systèmes. Nous montrons que le surcoût de MOTEUR est négligeable sur notre application qui met en jeu des temps CPU de quelques dizaines d'heures et des tâches de durées inférieures à 10 minutes. Même si une accélération significative est obtenue en production, la latence de la grille et sa variabilité sont les principales causes de perte de performance sur EGEE. Les dernières sections de ce chapitre sont donc consacrées à une étude comparative des latences mesurées sur EGEE et sur Grid'5000. Enfin, à partir de ces résultats, un modèle permettant de déterminer une stratégie d'allocation de tâches entre ces deux plateformes est présenté et fournit une métrique supplémentaire pour la comparaison de ces systèmes.

5.1 Grid platforms and middlewares

The concept of grids emerged from the idea that resources (computing power, storage, network...) interconnected through a high performance network could be considered as a single and sustainable platform accessible and shared among multiple users. The intent of the grid is to provide scientific and business production services that foster information technologies use, similarly to information networks that have become a primal and seamless support. A grid can be defined as an aggregation of heterogeneous and autonomic resources administrated in a decentralized way and whose accessibility policies refer to Virtual Organizations (VO) of users [Foster and Kesselman, 1997]. The grid *middleware* aims at hiding the low levels details to the end-user, so that the grid can be seen as a unique computer from her perspective.

Grids are traditionally sorted into *computing* and *data grids*, the former being devoted to the aggregation of CPUs whereas the latter focuses on data storage. An early example of data grid is the World Wide Web itself, coupled with search engines that provide a unified view of a huge amount of files distributed on heterogeneous resources federated thanks to the HTTP and HTML standards. Later emerged, peer-to-peer data management systems and overlay networks provide a uniform data access by the ability of each participant (*i.e* peer) to get some indexing information by querying its neighbours in a logical network.

Computing grids can be classified according to the kind of computing resources concerned. *Desktop grids* consist of the aggregation of personal workstations over the Internet, focusing on the exploitation of idle cycles. They have been successfully demonstrated by the Seti@home project[1] for decrypting space signals [P. Anderson et al., 2002] or more recently by the Decrypthon project[2] that helped in the sequencing of the human genome. On the other hand, academic grids (*clusters of clusters*) correspond to the aggregation of traditional clusters provided by computing centers, each using a classical batch system to handle its local computing resources. Such a grid is merely a super-batch system capable of handling tremendous amounts of computations, and particularly efficient in processing independent and large grain parallel computations. Effective examples of such infrastructures are the EGEE European production grid [Laure et al., 2006] and the Grid'5000 French experimental one [Cappello et al., 2005] that are mostly devoted to scientific computations.

Classical middleware approaches to computing grids are the *meta-computing* and the *global-computing*. In a meta-computing approach, the user is able to perform remote procedure calls (*e.g.* service invocations) to a set of predefined services running on the resources. A scheduler is responsible for the service finding and the load balancing of requests between the potentially numerous instances of a given service. An example of such a middleware is the DIET platform [Caron and Desprez, 2005, Amar et al., 2006, Caron and Dail, 2005] which proposes a scalable hierarchical requests scheduling approach. Other examples of meta-computing plat-

[1] http://setiathome.ssl.berkeley.edu/
[2] http://www.decrypthon.org/

forms are Ninf-G [Tanaka et al., 2003], Netsolve [Casanova and Dongarra, 1997] and Globus Toolkit 4 [Foster, 2005] which is based on WSRF. The goal of global-computing middlewares is to provide a unified view of the resources so that they are used as a single computer. In this approach, applications are not associated to the resources in the sense that they do not need to be pre-installed as it was required in meta-computing: the user submits *jobs* that may correspond to the execution of any command-line and that are allocated to resources by the middleware. Examples of global-computing middlewares are Globus Toolkit 2 [Foster and Kesselman, 1997] and gLite [Laure et al., 2006].

Among existing computing grids, a distinction has to be made between *production* and *experimental* grids. Experimental grids such as Grid'5000 [Cappello et al., 2005] or DAS3 [Cappello and Bal, 2007] have been developed by computer scientists for computer scientists and are viewed as an observation instrument to study the computing platform and its middleware stack from low level network protocols to scheduling algorithms. Their goal is to provide a reconfigurable platform where experiments are reproducible and can be performed in a controlled environment. On the other hand, production grids aim at supporting applications by supplying a huge amount of computing and data storage resources, operated 24/7 and federated by a stable middleware whose development should be the result of the research made on experimental platforms. However, because of the scaling-up performed by production systems (spread all over the Internet), the complexity of their middlewares and their sharing among large users communities (typically thousands of users), the resulting behavior of those systems is not properly understood yet: some jobs fail or are even lost without proper reports of the problems, they face high and unpredictable latencies and consequently, the efficiency of applications is lower than expected.

Therefore, considering production grids is interesting not only for the exploitation of domain-specific applications (and in particular of medical image analysis ones) but also for the computer science research itself because characterizing the behavior of production systems is the first step towards their study in controlled conditions (*e.g* by simulation [Casanova, 2001, Buyya and Murshed, 2002, Casanova et al., 2003, Legrand et al., 2006] or realistic load injection into controlled systems). Experiments performed in controlled environments should be inspired by realistic applications deployment and production systems should in return adopt the solutions developed thanks to experimental platforms.

In this chapter, a production grid (EGEE) is compared to dedicated clusters of the experimental Grid'5000. In section 5.2, the workflow of the bronze standard application (see chapter 2) is run on both platforms in similar conditions thanks to the use of the same workflow engine: MOTEUR (chapter 4). The work presented in this section has been done in collaboration with Cécile Germain-Renaud and Emmanuel Jeannot inside the AGIR French national ACI project[3]. Based on the conclusions of this section, a comparison of the latencies of the

[3] http://www.aci-agir.org

systems is then performed (section 5.3) and finally, a multi-grids model provides additional quantitative metrics to compare them.

5.1.1 EGEE infrastructure and gLite middleware

EGEE is a production grid infrastructure that has been operating since April 2004. It combines computational and storage resources provided by several computing centers all over the world. Each participating site configures, runs, and maintains a local batch system managing its local computational resources. About 36,500 CPUs spread over 230 clusters covering 50 countries are available. Including disks and tape robots, the total storage capacity is 23PB (figures from http://goc.grid.sinica.edu.tw/gstat). This infrastructure is shared by 5,000 users, gathered into VOs and a wide range of applications have been ported on it, including CERN's high energy physics experiments (ALICE, ATLAS, LHCB)[4], bioinformatics in-silico drug discovery [Jacq et al., 2007, Jacq et al., 2004] and medical image analysis applications [Montagnat et al., 2004a, Glatard et al., 2005, Blanquer Espert et al., 2005].

Computing centers are federated by the gLite middleware. This middleware is based on various components coming from the European DataGrid middleware, Globus Toolkit, Condor, and other toolboxes. gLite integrates the sites' computing resources through its Workload Management System (WMS). Figure 5.1 pictures the global architecture of this system which can handle different kinds of jobs, all of them being described with the Job Description Language (JDL)[5]:

- Simple jobs can be simple batch jobs (*i.e* requiring a single CPU) or MPI-based ones and could eventually be interactive (*i.e* asking for some input from the user during their execution).
- Compound jobs can be Condor DAGs (see chapter 2), collections of jobs (group of jobs with no dependencies between them) or parametric jobs (jobs having variable attributes in their JDL).

The user submits jobs from a `User Interface` (a dedicated machine running the Scientific Linux distribution[6] that allows the installation of gLite) to the `WMProxy` that creates the job id, registers it to a `Logging and Bookkeeping` (`LB`) service and returns it to the user. In case of compound jobs, sub-jobs are registered to the `LB` and the corresponding files are unpacked. The jobs are then delivered to the `Workload Manager` which is responsible for the job submission. The `Workload Manager` queries a `Resource Broker` (`RB`) to find resources (whose characteristics are stored inside `Information Supermarkets`) matching the job requirements. If no resources are found, the job is queued in the `task queue` and periodically retried. Otherwise,

[4] http://public.web.cern.ch
[5] https://edms.cern.ch/file/555796/1/
[6] https://www.scientificlinux.org/

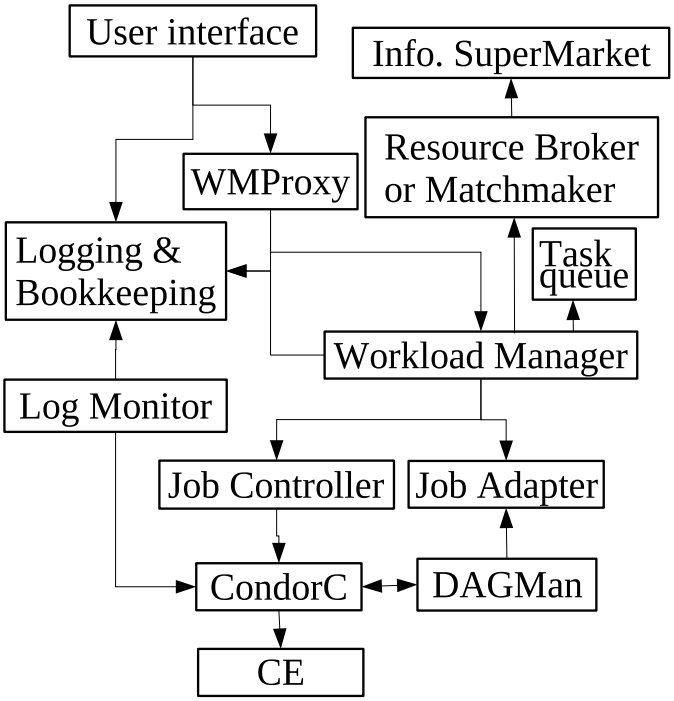

Figure 5.1: Overview of the gLite Workload Management System (WMS). The WMS is made of several components, each of them being distributed across the world. Coupled with a permanent load from the users, it introduces a significant latency (several minutes) on the jobs.

the job format is adapted to the submission entity by the `JobAdapter` and forwarded to the `JobController` that passes it to `CondorC`. `CondorC` finally enqueues the job in one of the `computing elements` (`CE`) which are queues of the computing centers. Each `CE` is managed by a local batch scheduler that schedules jobs to the available `Worker Nodes` (`WN`). The job is then monitored by the `LB` during its execution on the `WN`.

Several `RBs` are available on EGEE and users should use many of them to run applications involving the submission of a large number of jobs. The choice of `RBs` is let to the user. `RBs` may "see" different resources (`computing elements`) and accept submissions from various VOs depending on their configuration. Conversely, a given `computing element` may be seen by several `RBs` that do not communicate between each other, except through the load information advertised in the `Information Supermarket`. Moreover, each local batch scheduler is configured by a particular administrator. Consequently, the overall EGEE scheduling policy is not centrally defined, but results from the interactions of largely autonomous policies.

The components of the Workload Management System are distributed across the world. Coupled with a permanent load from the users, it introduces a significant delay (denoted *latency*) between the job submission to the beginning of its execution. Latencies in the order to 5 to 10 minutes are commonly encountered, with a very large variability.

To reduce the grid latency, users communities running Short Deadline Jobs (SDJ) [7] have defined and implemented the concept of a *Virtual Reservation* (VRes) [Germain et al., 2006]. Each of the p physical processors of a computing center is virtualized into k virtual processors, providing pk slots to the local batch scheduler. A part of these slots are dedicated to short jobs that could bypass traditional batch queues to be directly executed on those reserved slots (when a virtual slot is unused, the computing bandwidth is transparently returned to the other class of jobs sharing the same physical processor). However, it comes at the price of shared-time execution of the jobs on physical CPUs. Moreover, given that only a limited number of virtual slots is available, jobs submitted to the VRes queue are rejected when the system runs out of free slots and they need to be resubmitted. Thus, this solution remains circumscribed to a limited number of critical jobs with a high priority. It allows a two grains priority scheme (a job *is* short or *is not*) whereas most of the applications mix jobs with a continuum of durations. Besides, VRes queues may be victim of their success in the sense that if they saturate, a lot of resubmissions become mandatory to have a job executed and the latency starts to increase again. Finally, this strategy acts at the computing element level: it prevents a job from waiting in a *local* batch queue. Even if this queuing time is supposed to be the most important cause of the latency, the experiment that will be presented in chapter 6 will show that in average, it only represents 35% of the total latency (see table 6.2).

Inside EGEE, specific resources are dedicated to data storage. Data transfers between the *Storage Elements* (SEs) and the worker nodes are mainly done through the gridFTP protocol. Data files are identified by *Logical File Names* (LFNs). LFNs identify files which may be replicated in multiple physical instances for fault tolerance and optimization reasons. File catalogs (LFC, FireMan) give a uniform view of the distributed (and potentially replicated) storage system as in a traditional file system. Replication is handled by the users though.

Inside the EGEE grid, security relies on X509 certificates. A valid certificate is required to submit jobs and manage files. Certificates are issued by national certification authorities and registered by VO managers. Access to resources is controlled with respect to the VOs. Yet, grid entry points are not controlled: every authenticated user could set up a user interface and submit jobs from it.

5.1.2 Grid'5000 clusters and OAR batch scheduler

Grid'5000 is a national grid infrastructure composed of 13 clusters, distributed in 9 French cities and totalizing about 3000 CPUs. Sites are linked with 1 Gbits/s or 10 Gbits/s connections. Within each cluster, the nodes are located in the same geographic area and communicate through Gigabyte Ethernet links. Communications between clusters are made through the Renater French academic network.

[7] http://egee-na4.ct.infn.it/wiki/index.php/ShortJobs

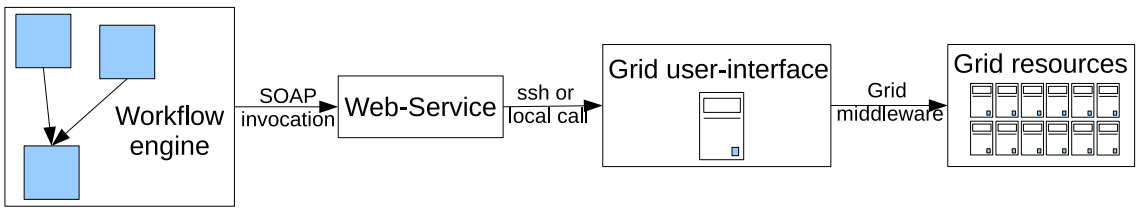

Figure 5.2: Interaction between MOTEUR and the gLite/OAR middlewares

Each user has a single (UNIX) account per site allowing access to all the clusters. Access is restricted to a set of front-ends (one per site) that are accessible through `ssh`. The platform is completely isolated from the Internet for security reasons. Nodes cannot even be seen from outside the platform (names and IPs are allocated on a private network).

The platform is completely reconfigurable. Users can deploy their own system and environments (in particular, they can get root access) on each node through the *kadeploy*[8] tool and reboot the machines through the network.

Nodes of a given cluster are accessible through the OAR resource allocation system [Capit et al., 2005] that provides all the basic mechanisms of classical batch schedulers such as advance reservation, batch and interactive jobs, matching of resources (job/node properties), hold and resume jobs, multi-queues with priority, best-effort queues (for exploiting idle resources), compute nodes checking before launching, dynamic insertion/deletion of compute node, backfiling, first-fit scheduler with matching resource and advance reservation. A cross-clusters super-batch system, OARGrid was still under development at the time of this study.

Home directories of users are mounted with NFS among the nodes and front-end of a given cluster. Each node also has a significant amount of (local) scratch disk space but no data management system is deployed by default.

5.1.3 Workflow deployment on grids with MOTEUR

As already stated (see chapter 2), deploying applications on grids using service workflows provides (i) a natural coarse-grain parallelization of the application with the workflow description and (ii) a transparent execution on several grid platforms thanks to the system-independent workflow language. Conversely, the workflow engine is isolated from the grid infrastructure by the services layer. Thus, the submission and handling of grid jobs is the responsibility of the services themselves. Indeed, in a pure SOA implementation, services are black boxes and the workflow engine has no information about their implementation.

To ensure the execution over a grid infrastructure, the services should either be relocatable or an intermediate layer is needed to control submission towards the desired resources. We are particularly considering the exploitation of the EGEE production grid which does not provide such service migration. The intermediate layer then becomes mandatory. Figure 5.2 presents

[8] http://www-id.imag.fr/Logiciels/kadeploy/index.html

the global picture of the interface of the workflows with grids. The workflow engine invokes (Web-) services that either connect to a User Interface (or grid front-end for Grid'5000) or just perform a system call to the middleware command-line interface for submitting and monitoring jobs. Such services only aim at wrapping an existing legacy code into a grid job and submitting it to the grid. Even if some technical application-specific details may complicate this task, it can be automated in most of the cases, thus leveraging the burden of the application developer. A Generic Application Service Wrapper is presented in chapter 7. It also allows workflow-level optimizations of the execution.

5.2 Comparison of systems on the bronze standard workflow

MOTEUR allows to compare the execution of workflows on various kinds of grids in similar conditions from the application point of view . In this section, a comparison of the EGEE production grid with dedicated clusters of Grid'5000 is presented. The goal is to quantify the performance gap between production conditions and dedicated clusters that will be considered as the reference in the remainder of this thesis. Results of this section will motivate the development of the optimization strategies presented in part III.

The bronze standard application is used here as a grid benchmark. The considered workflow is depicted on figure 5.3. It corresponds to a simplified version of the complete workflow presented on figure 3.3, at the end of chapter 3. Indeed, only the computationally intensive part of the workflow (*i.e* the services that lead to the submission of a grid job) is used here. The lightweight operations such as format conversions and the final statistical procedure are computed locally. This workflow is composed of 6 services, each of them being iterated over hundreds of data items in a typical run. The characteristics of the services are presented in table 5.1. A significant variability in the execution times is observed (standard-deviations range from 9% to 30% of the average values) even if runs have been performed on nodes with identical characteristics and input images all have the same size. It comes from the algorithms themselves that may converge more or less rapidly depending on the content of the images.

We made growing scale experiments by executing the bronze standard workflow on input data sets with variable size, from 2 to 126 pairs of images (according to figure 5.3, for each image pair, 6 grid jobs are submitted and 3 of them may be running in parallel: `Yas`, `Bal` and `PFM` or `PFR`). This workflow has been executed both on Grid'5000 dedicated clusters and on a VRes queue of EGEE.

5.2.1 Execution on dedicated clusters of Grid'5000

To benchmark the application on dedicated resources, we reserved nodes of Grid'5000 and deployed the OAR batch scheduler on this reservation to schedule jobs among these nodes. We

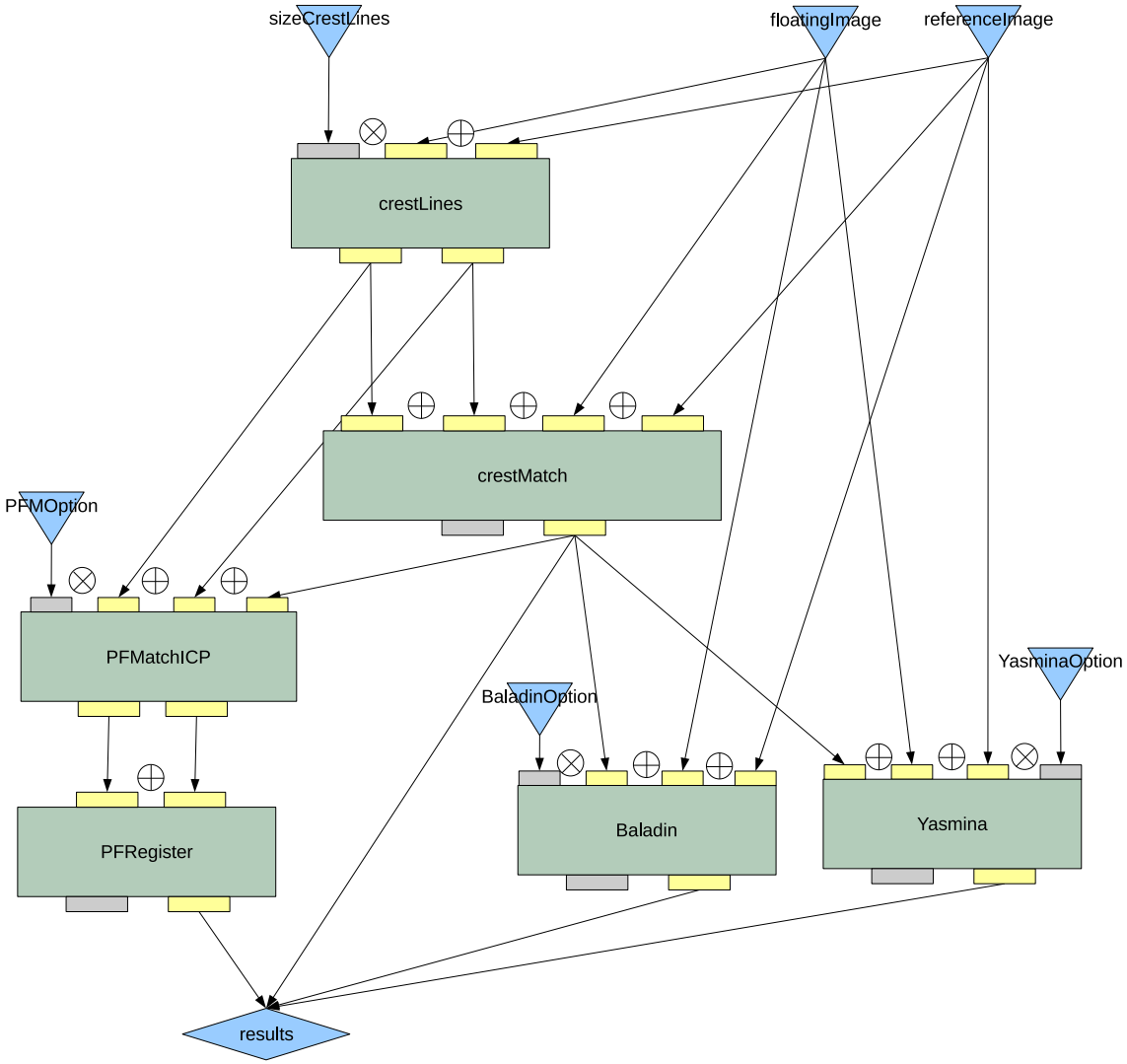

Figure 5.3: Representation of the bronze standard workflow used for the grid benchmark. The blue triangles represent inputs and the white boxes represent services to invoke. Each service invocation leads to the submission of a grid job. The input and output parameters of the services are represented by sub-divisions of the boxes. Yellow parameters denote files and the other ones are strings. Blue rectangles represent constants and blue diamonds are the outputs of the workflow. $\oplus$ and $\otimes$ correspond to Scufl iteration strategies (see chapter 2).

Services	Execution time			Input data (*i.e* data transferred)	Produced data
	Average	Stdev	Max		
CrestLines (CL)	54.87s	8.19s	67s	15MB	10MB
CrestMatch (CM)	26.74s	8s	42s	25MB	7.7MB
PFMatch (PFM)	24.12s	6.87s	44s	10.2MB	240kB
PFRegister (PFR)	5.37s	1.02s	8s	240kB	160kB
Yasmina (Yas)	146.02s	36.1s	236s	15.2MB	7.7MB
Baladin (Bal)	601.31s	53.35s	753s	15.2MB	7.7MB
Total	858.44s	65.8s	1150s	80.84MB	33.5MB
Critical path (CL+CM+Bal)	682.93s	54.57s	862s	xxx	xxx

Table 5.1: Execution times, input and produced data volumes of the services of the workflow of figure 5.3 for a single input image pair. Execution times have been obtained from the logs of the executions on Grid'5000 clusters. Averages and standard-deviations have been computed on 126 runs per service. The total volume of data transferred for the largest considered input data set (126 image pairs) is 80.84MB $\times$ 126 = 9.9 GB: the network is not a bottleneck for this experiment.

used three sites for the experiment, approximately distant of 1000 km from each other: Nancy (east of France), Rennes (west of France), and Sophia (south of France). All these sites are connected through a 10 Gbit/s backbone. A total of 60 computing nodes (AMD Opteron 64 bits 2GHz with 2GB RAM) were reserved (20 on each site). The number of computing nodes was selected so that it represents a reasonable number of resources, distributed all over the three sites, but still low enough as compared to the potential parallelism of the application (up to 380 concurrent tasks). A unique medical images repository was set up in Rennes.

An OAR 2.0 server managing the 60 computing nodes was set up on an additional node of Grid'5000. Each Web-Service of the workflow was submitting jobs to this server and was deployed on the same node. The bronze standard workflow was run by MOTEUR on the same node as well. Data transfers were handled at run time by auto-generated scripts copying images from the repository or from the place where the data has been generated using the *scp* UNIX command. When a task ends, it releases the resource of the OAR node and the data remains on this node for future use.

The makespan of the workflow is plotted in figure 5.4. The curve exhibits two linear phases. The y-intercept value of the first phase (below 40 image pairs) corresponds to the critical path of the application which is 652 seconds in this case. It differs from the average value reported in table 5.1 because of the variability of the execution times among the images. The slope of this linear approximation (5.7s/image pair) measures (i) the scalability of the whole deployment system (MOTEUR, OAR and data transfers) and (ii) the impact of the variability of the execution times on the makespan of the application. The latter may be the most important reason as

Service	Longest path from the service	Size of the longest path		Priority
		Average	Maximum	
CL	CL+CM+BAL	682.92s	1259s	1
CM	CM+BAL	628.05s	795s	2
BAL	BAL	601.31s	753s	3
YAS	YAS	146.02s	236s	4
PFM	PFM+PFR	29.49s	52s	5
PFR	PFR	5.37s	8s	6

Table 5.2: Priorities of the jobs of the workflow of figure 5.3 according to the longest paths. Time values refer to the benchmark displayed on table 5.1.

the former should be in the order of a few dozens of milliseconds according to the overhead quantification performed at the end of chapter 4. Indeed, increasing the number of input image pairs also increases the expected value of the critical path of the application.

Beyond 42 image pairs, the slope of the linear approximation grows from 5.7s/image pair to 16.5s/image pair. It comes from the saturation of the platform which is only 60 nodes whereas $3 \times 42 = 126$ jobs may be running in parallel for 42 image pairs. The total CPU time measured for 126 pairs of images is 29h 49min 12s and the corresponding makespan is 40min 12s. Thus, the speed-up obtained on those 60 nodes is 44.5. This speed-up is clearly sub-linear with respect to the number of processors but dependencies of the workflow have to be considered to determine the theoretical speed-up. To give an idea of this optimal value, we simulated the scheduling of the workflow using a list scheduling algorithm based on earliest finish times [Legrand and Robert, 2003]. This algorithm first assigns to each service of the workflow a priority depending on the weight of the longest path starting from it (see table 5.2). Tasks ready to be executed (*i.e* free from dependencies) are ordered according to their priorities. The first one is then assigned to the resource that provide the earliest finish time and the algorithm is iterated until the completion of the workflow. We assumed that services have fixed execution times that are set either to their average or to their maximum walltime values. The schedules obtained from this algorithm are plotted on figure 5.5. The finish time of the last task is the makespan of the workflow. In this simulation, the theoretical speed-up is 54.6 for average values and 43.3 for max values. The 44.5 experimental speed-up obtained in practice is in between those two ideal values. Even though a deeper analysis of the scheduling would probably reveal some possible improvements, the deployment system (MOTEUR invoking Web-Services submitting jobs to OAR on dedicated resources) seems scalable enough to provide a relevant practical reference for comparisons with production conditions. The overhead introduced by MOTEUR (see the end of chapter 4 for a detailed analysis) is negligible compared to the whole application performance and OAR behaves well at this scale.

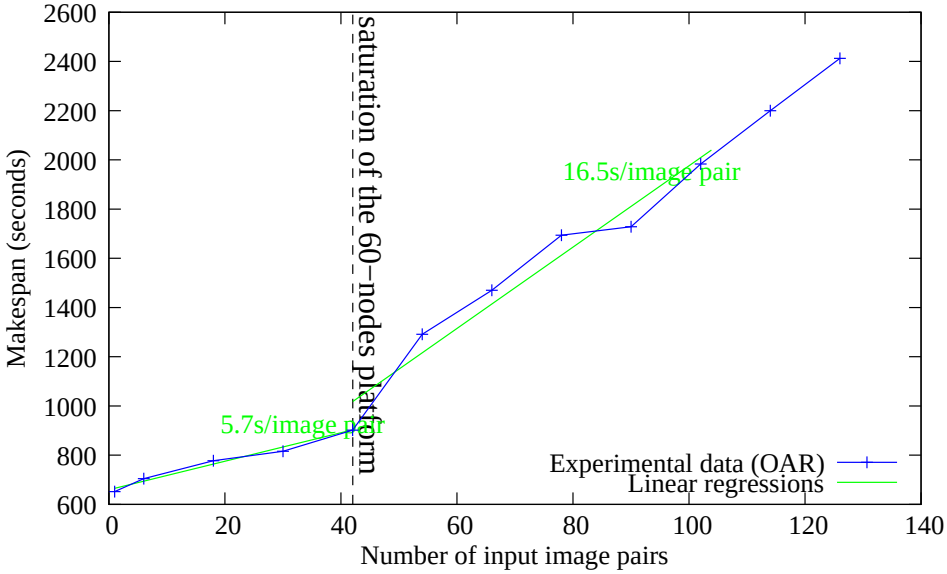

Figure 5.4: Evolution of the makespan of the workflow of figure 5.3 on 60 dedicated nodes of Grid'5000 for a growing size of the input data set (image pairs). The curve starts with a linear phase whose slope measures the scalability of the deployment system (MOTEUR+OAR+data transfers) as well as the impact of the variability of the execution times on this application. Beyond 40 image pairs, the 60-nodes platform is saturated and the slope of the fitted straight line increases from 5.7s/image pair to 16.5s/image pair.

5.2.2 Execution on the EGEE production grid

The workflow presented in figure 5.3 was then executed on the EGEE production grid. In this experiment, the virtual reservation mechanism described in section 5.1.1 has been activated on one EGEE site (LAL, Paris). A hundred slots for immediate execution were allocated. This is higher than the 60 computing nodes allocated in the experiments with Grid'5000. However, on EGEE, we have no control on the other users activities: these slots have been shared with other users exploiting the infrastructure for their computation needs. The LAL cluster hosts dual-processor (AMD Opteron, 2.2 GHz) machines with 1 GB of RAM per CPU (2 GB total), which is comparable to the Grid'5000 nodes that were used in the previous experiment. Medical images and results produced by the application were stored on a single EGEE storage server located in Clermont-Ferrand (center of France). MOTEUR ran directly on an EGEE user interface offering the client interface for job handling.

It may be argued that this experimental setup significantly differs from the one deployed on Grid'5000: the number of processors in the EGEE and Grid'5000 setups is not the same, the bandwidth of the network connections is not known, EGEE uses gridFTP for file transfers whereas the Grid'5000 setup uses *scp*, ... The point is that the EGEE grid imposes its own setup which is by no way configurable as it aims at being used for production purposes. The ultimate experiment to fully evaluate the EGEE middleware would be to deploy the gLite middleware on Grid'5000 nodes, setting up every component implied in an EGEE job life-cycle (Resource

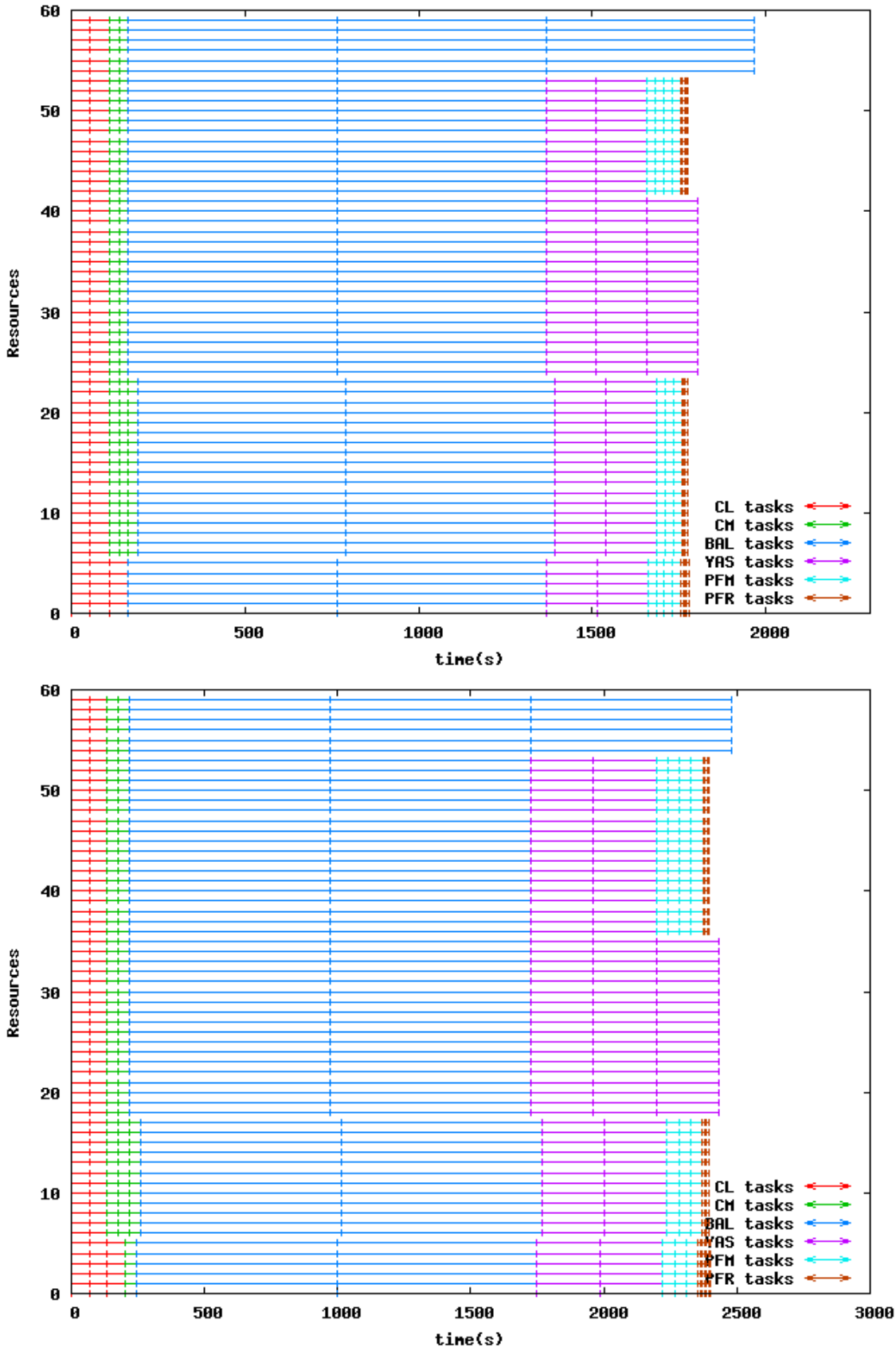

Figure 5.5: Gantt charts figuring the simulation of the scheduling of the bronze standard workflow on 126 image pairs on a 60-nodes dedicated platform, each service being assumed to have a constant execution time. Top: average values ; bottom: max values (see table 5.1). The makespan is 2477 seconds in the maximal case and 1967 seconds in the average case.

Broker, Logging and Bookkeeping service, ...) and to simulate realistic network links between them. Yet, this goes far beyond the scope of the experiment presented here which only aims at quantifying the gap between controlled and production conditions.

The makespan of the workflow for growing numbers of input image pairs is plotted in figure 5.6. On this infrastructure, the critical path of the application has grown to 1491 seconds despite the use of VRes. A comparison to the 652 seconds that were observed on dedicated resources (see figure 5.4) provides a measure of the high latency introduced by the infrastructure, which is about 280 seconds per job in this case (the critical path of the workflow is made of 3 services so that the average latency of the jobs is $\frac{1491s-652s}{3}$).

An easily measurable cause of latency on the EGEE experiment is the submission time. Figure 5.7 illustrates it with task-flow diagrams for the 3 experiments with 18 image pairs. Those graphs have been obtained with the VizDIET tool[9]. The first 18 workflow tasks were submitted every 0.17 seconds by the OAR middleware whereas the EGEE experiment revealed a submission rate of 1 task every 11.6 seconds. This poor performance comes from the time required by the workload management system to register the job in some of its components (such as the monitoring system and the matchmaker). It drastically limits the parallelism achieved by the application.

The linear approximation of the experimental data on figure 5.6 still provides an estimation of the overhead of the system. This value (80s/image pairs) is huge compared to the 5.7s/image pair measured on dedicated resources in the previous experiment. As the deployment system (MOTEUR invoking Web-Services submitting jobs) is the same as in the previous experiment, this difference may come from the impact of the latency on the makespan of the application. This latency may come from (i) the latency introduced by the gLite middleware itself and (ii) from the load imposed by other users that leads to jobs resubmissions (remember that a job is cancelled and resubmitted when no VRes slot is available on the site).

An effect of this variability is the fact that the makespan curve on EGEE is not monotonic. Load conditions may change between executions and some runs may be faster than other ones, even larger (see for instance the decrease between 114 and 126 image pairs on figure 5.6). A metric to quantify the load of the execution resources is the number of jobs rejected by the VRes queue when it saturates, which is detailed on table 5.3: it varies between 0 and 25% of resubmitted jobs.

Yet, the execution on EGEE still provides a speed-up of 9.8 with respect to the 29h 49min 12s sequential time for 126 input image pairs.

Partial conclusion. Experiments on the workflow of the bronze standard application have quantified the gap between the execution on dedicated resources and production conditions: the latter is 4.5 times slower than the former in similar conditions. The deployment of the

[9] http://graal.ens-lyon.fr/~diet/vizdiet.html

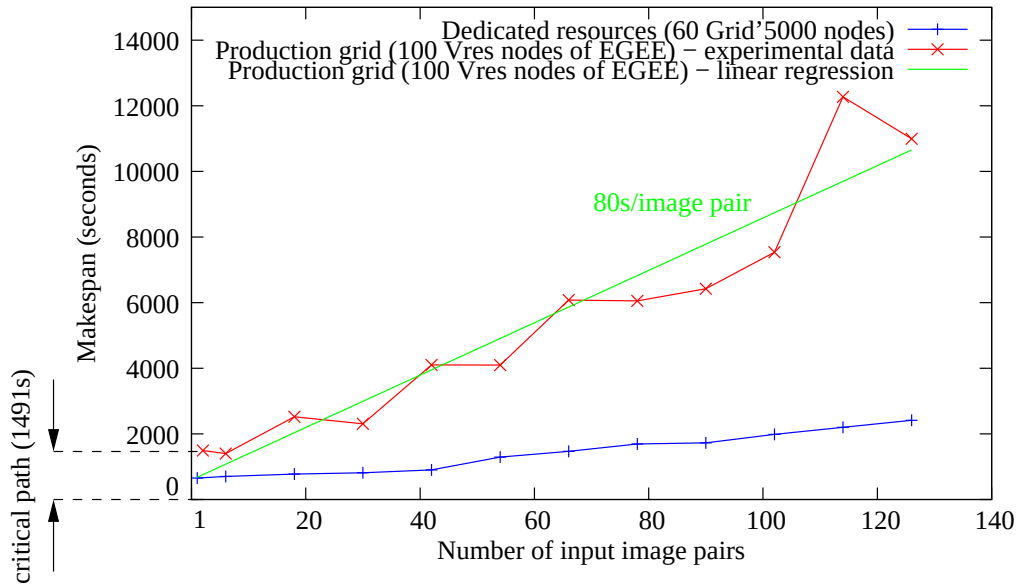

Figure 5.6: Makespan of the Bronze-Standard application on dedicated Grid'5000 resources VS a production VRes queue of EGEE. The larger critical path measured on EGEE is a consequence of the high latency observed on this production platform. The high slope of the linear approximation may come from the variability of the latency on the jobs (due to the load imposed by other users and middleware-intrinsic reasons.)

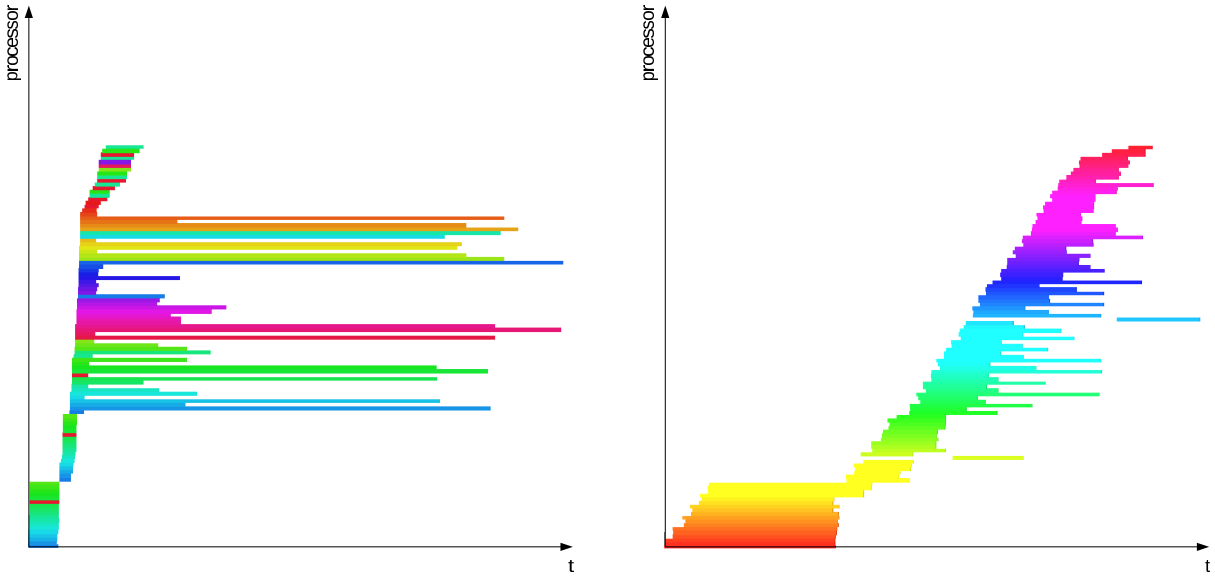

Figure 5.7: Task-flow graphs for 18 image pairs. Left: OAR. Right: EGEE. Each row of those diagrams corresponds to a particular job. Colored rectangles represent the task duration: they start once the corresponding task has been submitted and stop at the end of its execution. The slope of the left contour of these graphs gives a qualitative information on the job submission rate on the infrastructure. It characterizes the middleware submission performance. Colors are arbitrarily set and just help to distinguish the different tasks. Beware that time scales are different in those task-flows.

Number of image pairs	Total number of jobs submitted by the workflow	Resubmitted jobs	
		Number	Ratio
2	12	1	8%
6	36	0	0%
18	108	2	1.8%
30	180	1	0.5%
42	252	13	5.1%
54	324	1	0.3%
66	396	102	25.7%
78	468	31	6.6%
90	540	14	2.5%
102	612	15	2.9%
114	684	115	16.8%
126	756	82	10.8%

Table 5.3: Resubmissions performed on EGEE for the different sizes of input data. The resubmission ratio exhibits strong non-monotonic variations, which reflects the variability of the load on the VRes resources.

workflow with MOTEUR and OAR on dedicated nodes of Grid'5000 is satisfying, providing a speed-up close to the theoretical one that could be obtained on the execution platform. Even if the performance on the EGEE grid remains far lower than the one obtained on dedicated resources, it is still significant given that this grid is continuously under load and deployed at a very large scale. The job latency and its variability due to the load of other users are the main causes of performance drops in production. In the remaining of this chapter, we propose to benchmark the EGEE and Grid'5000 platforms by comparing their latencies.

5.3 Latency comparisons

The latency of a job is the duration between its submission date and the instant at which its execution really starts. It includes (i) latencies of the network infrastructure itself, (ii) the delay caused by the interaction of middleware components and (iii) the system load (queuing time in batch and services queues). The latency of production grids is known to be high and damageable for applications having short deadline constraints. However, a precise quantification of the impact of this latency on the application performance is missing. In this section, experiments are presented and latency quantification metrics are extracted. The latency faced by jobs on the EGEE production grid is compared to the one of Grid'5000 clusters that constitute the reference of the study.

5.3.1 Latency measures

In this section, the whole biomed VO of the EGEE grid will be compared to 2 Grid'5000 clusters: the "idpot" cluster of the Grenoble site, made of 20 2 GHz bi-processor nodes and a larger Grid5000 cluster in Sophia Antipolis, made of 105 bi-processor nodes.

Experimental setup. To measure the latency of the systems, the workload management system was progressively loaded by submitting an increasing number n of jobs. Each time a job completed, a new one was resubmitted so that the total load introduced by the experiment was constant. Jobs were short (t_{run} =1 minute long), sleeping for one minute to ensure constant execution time independently from the hardware on which they were running. This experimental setting favored a short turn-over of jobs and stressing conditions of the workload management system. Experiments were run over 3 hours periods (a long enough period compared to the jobs duration to capture the system behavior over a statistically significant number of measurements). The execution time t_{exec} of the jobs was measured and the system latency t_{lat} was obtained by computing the difference $t_{\text{exec}} - t_{\text{run}}$.

Results. Figure 5.8 displays the median of the latency for a growing number n of submitted jobs over the 3 studied systems. This figure also displays, for each measure, the inter-quartile range (IQR) of the latency. This metric measures the spread of the samples and gives an information about the variability of the system. Considering a sorted set of values, the IQR is the interval defined between the first quarter and the third quarter of the number of values. It represents the interval of the most relevant values, ignoring the 25% lowest and highest ones. We did not compute any means nor standard deviations in the analysis of the experimental results but rather medians and IQRs which are less sensitive to outliers. For this experiment, $20,000$ jobs were submitted to the EGEE infrastructure, $32,000$ to the Sophia cluster and $28,000$ to the Grenoble one.

5.3.2 Model and metrics

Those experimental results suggest an affine behavior of the median latencies with respect to the number of concurrently submitted jobs. Thus, an affine model $An + B$ was fitted by a linear regression to the median latency curve of each system. The lines obtained are plotted on figure 5.8. The parameters of this model are shown in table 5.4, where the systems are sorted from the smallest one to the widest. Those parameters will be used as metrics, to characterize the variation of the median of the latency with respect to the number of jobs for each system. The B parameter measures the *nominal latency* of the system. It corresponds to the latency introduced by the system without any load. A measures the *scalability* of the system with respect to the number of jobs. It represents the additional time generated by the submission of

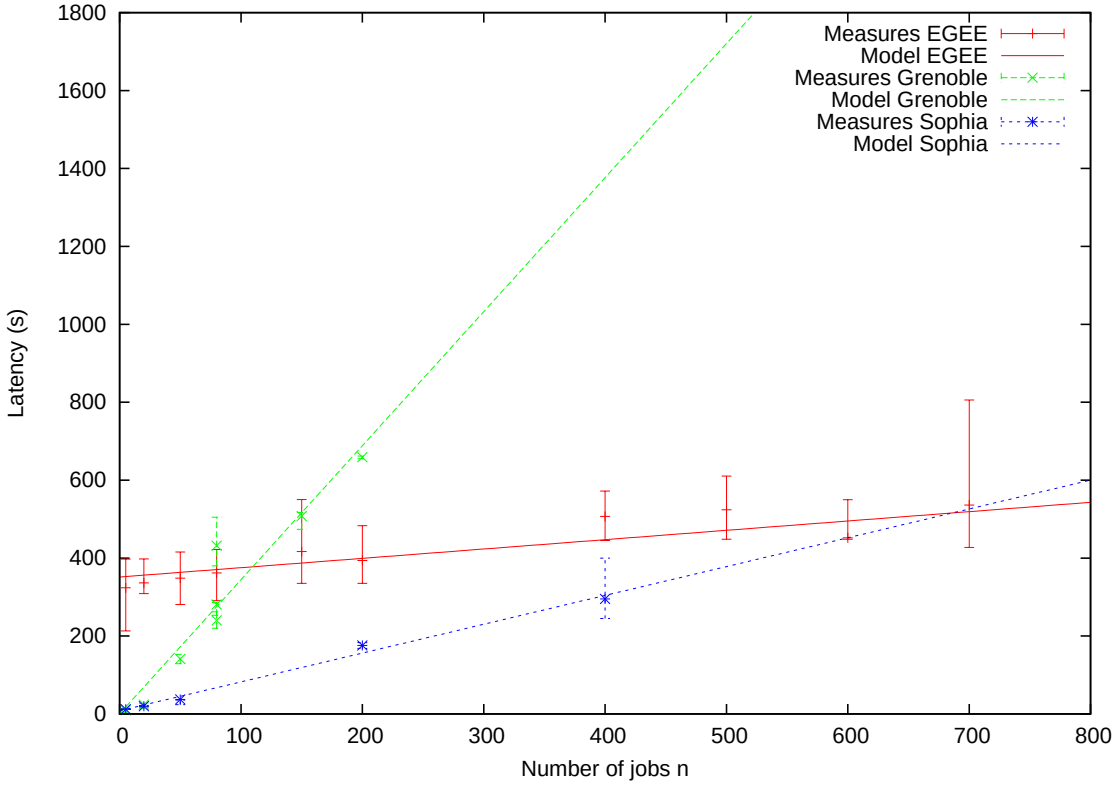

Figure 5.8: Latency time versus number of jobs maintained in the system. An affine behavior of the latency with respect to the number of jobs is suggested by those experimental results.

1 extra job to the system. Despite its simplicity, this model provides a relevant way to compare grid infrastructures, as detailed in the following discussion. Moreover, it will allow the design of the multi-grids model introduced in section 5.4.

Nominal latency. Nominal latencies (B metric) are growing with the size of the infrastructure, which is not surprising from a qualitative point of view. Quantitatively, the EGEE system has a very strong nominal latency. This value, close to 6 minutes (351 s) is mostly due to the concurrent usage of the grid by other users. This huge nominal latency is a characteristic

System	A (s/job)	B (s)
Grid5000 – Grenoble	3.44	0.48
Grid5000 – Sophia	0.74	8.25
EGEE – biomed VO	0.24	351.4

Table 5.4: Parameters of the latency model (latency = An+B). Systems are sorted from the smallest to the widest. A measures the scalability of the system whereas B corresponds to the nominal latency.

of production infrastructures. On the contrary, the nominal latency of the Grenoble cluster of Grid'5000 is far lower. Accessing the infrastructure requires less than a second. This performance comes from the relatively low load of the infrastructure and the reduced size of the infrastructure that makes communication costs lower. The Sophia cluster of Grid5000 is not very far from Grenoble, with a nominal latency of 8.25 seconds.

Scalability. Conversely, the scalability of the systems is improving with their size. The job scalability of EGEE constitutes its main advantage. The latency only grows by 3.5 minutes from the submission of 5 jobs to 1000 jobs and the latency due to the submission of one extra job is 0.24 second (A metric). The scalability of the Grenoble cluster of Grid5000 is weak. Submitting a single extra job leads to a latency growth of 3.44 seconds. Here again, the Sophia cluster stands in the middle: its scalability (A metric) is 0.74 second per job. It is three times worse than on EGEE and 4.65 better than on the Grenoble cluster.

Improving scalability. As already noticed in section 5.2.2, the submission procedure plays an important role in the growth of the median latency with respect to the number of jobs. Indeed, on all the evaluated systems, the submission is done from a single entry point (the user interface) to a central workload manager (OAR or RB host) through the network. These two hosts and the network connection may become bottlenecks beyond a critical stressing level. Computing again the scalability (A metric) on the values without submission time, we obtain 0.07 s/job for the EGEE system, 0.34 s/job for the Sophia cluster and 2.93 s/job for the Grenoble one. Those values are obtained by subtracting the submission times of the jobs to the measures presented on figure 5.8 and by fitting again the affine model. Comparing those values to the ones obtained in table 5.4, one can conclude that the submission procedure respectively leads to a 3.24, 2.15 and 1.18 slow-down ratios on the job scalability. A solution to improve scalability could therefore be to distribute the submission system, which is a real bottleneck on all the systems studied, as shown above. Many Resource Brokers are available on the EGEE system. Nevertheless, they do not communicate between each others and serious performance drops can be forecast in the scheduling when the load reaches a critical point. Conversely, solutions such as the one proposed by the DIET middleware [Caron and Desprez, 2005], where many collaborative schedulers are able to administrate the same pool of resources could provide an interesting improvement of the system.

Variability of the latency. On the Sophia and Grenoble clusters, the IQR of the measures for a low number of submitted jobs remains lower than 15 seconds. It then increases with the number of jobs maintained in the system because of the saturation of the platforms. On the EGEE infrastructure, the situation is quite different. Variability is around 3 minutes, even for a low number of submitted jobs. The order of magnitude of the variability remains constant

for less than 600 jobs and grows up to 5 minutes beyond this value. This high variability, even for a low number of concurrently submitted jobs leads to a problem specific to large-scale production infrastructures: a single job is likely to penalize the whole application performance if it remains blocked in the system. In part III of this thesis, strategies are proposed to reduce the impact of the variability of the latency on the application performance.

5.4 Choosing the best platform: a multi-grids model

Grid'5000 clusters and the EGEE grid exhibit different latency characteristics. The former has a lower nominal latency whereas the latter has a better scalability. Therefore, it is interesting to determine, given a number of jobs to process, the optimal fraction of these jobs that should be submitted to each infrastructure to minimize the total execution time. This is the goal of the multi-grids model proposed in this section. Based on the experiments of section 5.3, the analysis of this model provides additional metrics to compare the infrastructures. Multi-grids executions are for instance performable by workflow managers such as the P-Grade portal [Kacsuk et al., 2006a].

5.4.1 Principle of the model

Let us consider two systems and a total number n of jobs to submit in parallel. Let $\delta \in [0,1]$ be the fraction of jobs to submit on the first system. Let $t_{\text{lat}}^{(i)}(n)$ be the median latency time introduced by system i when it handles the submission of n concurrent jobs. The goal is to minimize the average latency time of the submitted jobs, which is:

$$R(\delta) = \delta t_{\text{lat}}^{(1)}(\delta n) + (1-\delta) t_{\text{lat}}^{(2)}((1-\delta)n)$$

The problem then resumes to the minimization of R with respect to δ. If we consider the affine model presented in section 5.3.1, then $R(\delta)$ becomes:

$$\begin{aligned} R(\delta) &= \delta(A_1 \delta n + B_1) + \\ &\quad (1-\delta)(A_2(1-\delta)n + B_2) \end{aligned}$$

where, A_i and B_i are the model parameters of the i^{th} system. R has a unique minimum reached for the optimal proportion of jobs $\hat{\delta}$ to submit on the first system:

$$\hat{\delta}(n) = \frac{B_2 - B_1 + 2A_2 n}{2n(A_2 + A_1)} \tag{5.1}$$

It must be determined when $\hat{\delta}(n)$ is in [0,1]. In the following, we suppose that system 1 is larger than system 2. According to section 5.3.1, it implies that $B_1 > B_2$. Indeed, the experimental

results showed that the nominal latency of the largest system is higher than the one of the smallest one. Conversely, $A_1 < A_2$ because the scalability of the largest system is better than the one of the smallest one. In this case, it is straightforward to prove that $\hat{\delta}(n) < 1$. It shows that the proportion of jobs to submit on the smallest infrastructure is never null: the smallest but fastest infrastructure has to be overwhelmed before starting submitting on the largest one. Moreover, we can show that $\hat{\delta}(n)$ is positive if and only if $n \geq n_0 = \frac{B_1 - B_2}{2.A_2}$. It highlights three phases of job submission. In the first one, when $n \leq n_0$, the number of jobs is low enough to submit all of them on the smallest infrastructure. It corresponds to an initialization phase. When n exceeds the critical value n_0, a transient phase begins: a proportion $\hat{\delta}(n)$ of jobs have to be submitted on the largest platform. During this second phase, another variable of interest is $n_{0.5}$, the number of jobs for which $\hat{\delta}(n)$ is 0.5, thus implying that the same number of jobs is submitted to both infrastructures $\left(n_{0.5} = \frac{B_1 - B_2}{A_2 - A_1}\right)$. Beyond this point, the largest system starts being preponderant. The model finally enters a saturation phase, where $\hat{\delta}$ tends to its asymptotic value $\hat{\delta}(\infty) = \frac{A_2}{A_1 + A_2}$. This value is inferior to 1 and denotes the remaining proportion of jobs that would always be submitted to the largest platform, even if the number of concurrently submitted jobs becomes very high.

5.4.2 Application to the studied systems

The variables of interest identifying the 3 phases described in the previous section are displayed, for each pair of systems, in table 5.5. The first line of this table compares EGEE to the Sophia cluster of Grid'5000. The value of n_0 indicates that there is no need for using EGEE if the number of jobs is lower than 232. The transient phase starts from this critical number of jobs. This value is twice as high as the number of processors of the Sophia cluster. On the next line, comparing EGEE to the Grenoble cluster, the critical number of jobs is 51, which is 4 times higher than the number of processors of the Grenoble cluster. Those values of n_0 are high, compared to the number of processors of the infrastructures. They are another way to perceive the difference between a production and an experimental infrastructure. On the contrary, the last line of this table indicates that the critical number of jobs from which it is necessary to submit to the Sophia cluster rather than only to the Grenoble one is 1. Indeed, even if those two clusters differ in their number of processors, the nominal latency of Sophia's cluster has the same order of magnitude as the one of Grenoble's cluster. Thus, it is not penalizing to submit jobs to the Sophia cluster even if the Grenoble one is not overwhelmed.

The $n_{0.5}$ value of the same table can lead to similar interpretations. This value corresponds to the abscissa where the lines cross on figure 5.8. The EGEE infrastructure and the Sophia cluster have the same latency if 686 jobs are submitted on each infrastructure. This number of jobs is 110 when comparing EGEE to the Grenoble cluster and 3 for the Sophia versus Grenoble comparison.

To have an idea of how the proportion of jobs to submit to the largest system grows, fig-

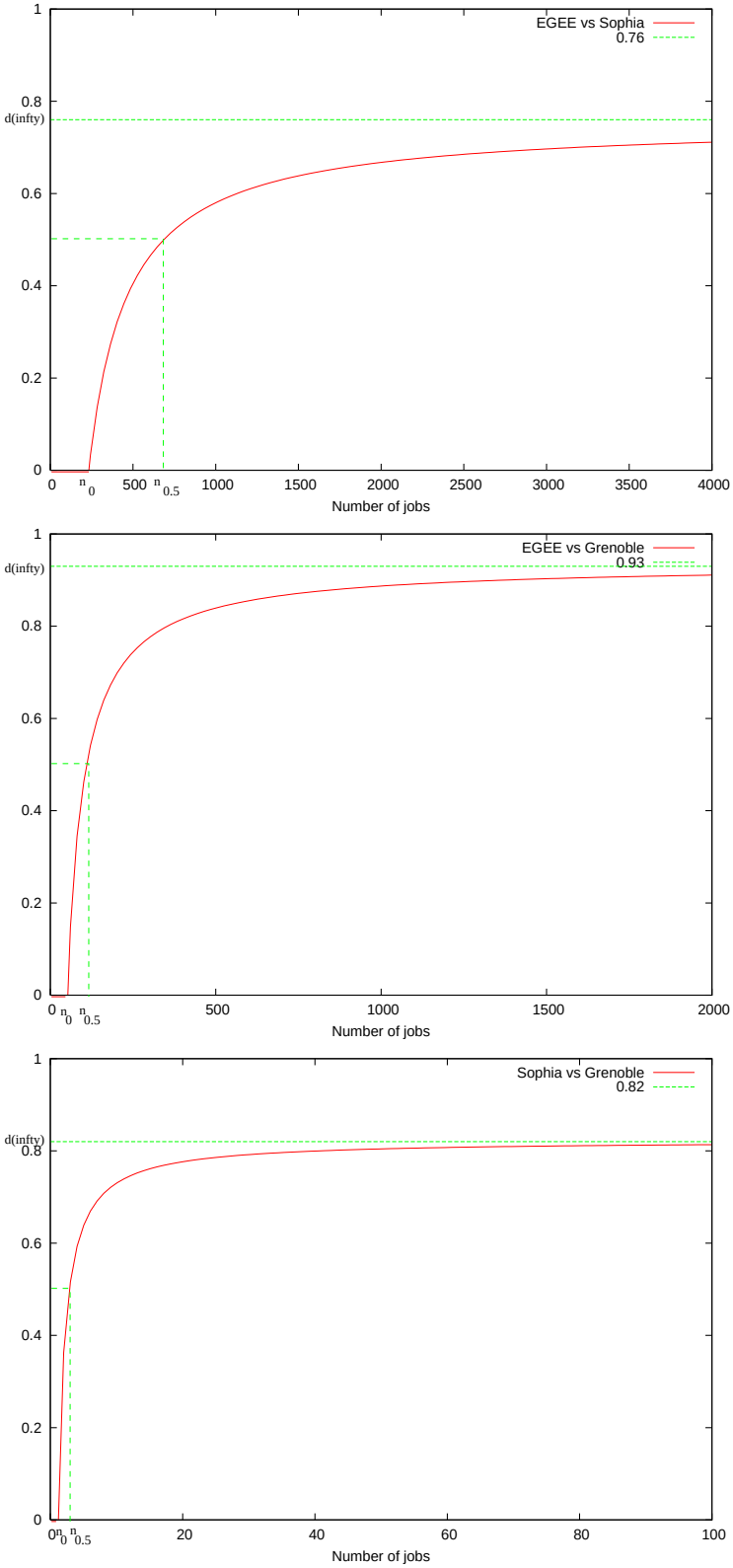

Figure 5.9: Evolution of the optimal proportion of jobs to submit on the largest system: from top to bottom: EGEE vs Sophia, EGEE vs Grenoble and Sophia vs Grenoble. Three different phases are visible: under a given number of jobs n_0, no job should be submitted on the largest system. Then this proportion is increasing and finally, the optimal proportion tends to an asymptotic value lower than 100%.

Largest system	Smallest system	n_0	$n_{0.5}$	$\hat{\delta}(\infty)$
EGEE	Sophia	232 jobs	686 jobs	76%
EGEE	Grenoble	51 jobs	110 jobs	93%
Sophia	Grenoble	1 job	3 jobs	82%

Table 5.5: Variables of the multi-grids model. Below n_0 jobs, no job should be submitted to the largest system. At $n_0.5$, jobs should be equally split among the two systems. $\hat{\delta}(\infty)$ denotes the maximal proportion of jobs to submit to the largest system.

ure 5.9 displays the evolution of $\hat{\delta}$ for each pair of systems. All those curves are growing with the number of jobs, as it could be predicted from equation 5.1. The bottom one, comparing the Sophia and the Grenoble clusters, grows rapidly and converges towards $\hat{\delta}(\infty) = 82\%$. This value characterizes the saturation phase. It indicates the proportion of jobs to submit to the Sophia cluster when the total number of jobs to submit is high. This result is close to the proportion of nodes on the Sophia cluster in the total number of nodes on the two systems: $\frac{105}{105+20} = 84\%$.

Looking at the two upper curves of figure 5.9, we can see that the larger the scale difference between the two compared systems, the faster the growth of the curves. Concerning the comparison between EGEE and Grenoble, the curve converges to $\hat{\delta}(\infty) = 93\%$. This limit is 76% for the comparison between EGEE and the Sophia cluster. This result indicates that whatever the number of concurrently submitted jobs is, there is no need to submit more than 76% of them to the EGEE infrastructure.

Validity of the results. The results presented above are all inferred from the experiment described in section 5.3.1, where all the submitted jobs are of identical running time (1 minute). It should be possible to extend those results to jobs with different running times. In this case, it is likely that the nominal latency (B metric) of the infrastructures will change only by little. Indeed, this value corresponds to the latency faced by a single job submitted to the system, which does not depend on its execution time in most cases. This assumption could eventually not be so realistic if the grid system uses a prediction of the execution time of a job in its scheduling policy. Yet, this case is not envisaged in this manuscript, given that most of the users of the EGEE grid do not specify such a prediction for the job submission. Concerning the scalability (A metric), we already noticed that the submission entity significantly contributes to it. It is reasonable to state that the submission time will not be correlated to the execution time of the submitted jobs. Consequently, only a restricted part of the scalability value (about 30% for the EGEE grid, 46% for the Sophia cluster and 85% for the Grenoble one) may be disturbed by a change in the job execution time. To have an idea of how the remaining part of the scalability value would be disturbed, one should identify the steps of the job life cycle that

could be delayed by the submission of longer jobs. Obviously, the most impacted step would be the waiting in the local queues, which would be lengthened by an increase of the job execution time. Yet, this waiting time would be disturbed only if many of the jobs are scheduled to the same queue, and if this queue lacks resources to execute all of them in parallel. Given the number of queues of the EGEE grid (about 150 in our VO) and the high number of resources per queue, it is probable that such interactions between our jobs will be limited on this system. The situation will probably be quite different on the clusters. In this case, the evolution of the waiting time with respect to the job duration could probably be predicted by quite simple models. Yet, the multi-grids model itself will remain valid. Only its parameters would have to be modified.

5.5 Conclusions

In this chapter, the deployment of the bronze standard workflow with MOTEUR enabled a fair comparison of a production grid (EGEE) and dedicated clusters of Grid'5000. Compared to theoretical predictions, the execution of the application on dedicated clusters exhibits reasonable performance so that the overhead of MOTEUR can be considered as negligible on this application. Even using specific (VRes) queues allowing immediate execution of jobs, the performance of the application is about 4.5 times lower on EGEE than on dedicated resources. The grid latency (*i.e* the duration between a job submission and the beginning of its execution) is the main causes of performance drops. For each system, the *median* of the latency has been shown to follow an affine model with respect to the number of jobs simultaneously submitted. A multi-grids model was derived from this measures and metrics measuring the scalability of the infrastructure and its nominal latency were extracted. Those metrics allow to further compare the EGEE grid to dedicated clusters. For instance, considering 1 minute-long jobs, it shows that there is no need to use the EGEE grid rather than the studied clusters below a threshold of 230 submitted jobs. Moreover, those results show that the submission entity is an important bottleneck of the infrastructures: the submission time reduces the scalability of the EGEE grid with a factor 3.

Beyond this median comparison of the latencies, the *variability* of the EGEE production grid has been shown to be superior of several minutes to the one of a cluster. It is suspected to be a major source of performance drops for the applications. Indeed, a grid user considers the completion time of a whole set of job (*i.e* the makespan of the application) rather than the throughput of the system that may be studied from the infrastructure's point of view. Consequently, the variability of the platform is critical because a single highly delayed job is able to dramatically penalize the whole application. Thus, in the next chapter, we focus on the analysis of the variability of the grid latency and on its impact on the performance of the application.

The experimental results shown in this chapter reveal that a hundred node dedicated cluster

provides a better performance than the EGEE production grid until a high threshold of computing time that was not reached with the bronze standard use-case. However, one should keep in mind that using a production grid provides computing power *for free* from a user point of view. In particular, in a medical context, setting up and maintaining a dedicated cluster is not always feasible and deploying applications on a production grid still provides a significant speed-up with a limited maintenance cost. Considering production grids may be a fixed constraint and it thus remains worth studying their characteristics.

Chapter 6

Analysis and impact of the latency variability on the EGEE grid

Contents

The goal of this chapter is to study the impact of the variability of the latency of the EGEE grid on the performance of applications. Based on a black-box model of the grid, we propose a probabilistic model of the workflow of an application. This model allows to quantify the impact of the variability of the latency on applications. In particular, its impact on the bronze standard application (see chapter 1) is shown to be of a factor 2. We show that the latency can be accurately modeled by a random variable with a mixed log-normal / Pareto distribution whose parameters are determined by fitting to experimental data. The impact of some job context parameters such as the execution site or the Resource Broker on the distribution of the latency is finally highlighted and quantified, which helps to refine the grid's latency distribution model.

Le but de ce chapitre est d'étudier l'impact de la latence de la grille EGEE sur les performances d'une application. Nous proposons un modèle probabiliste du flot de traitements d'une application fondé sur une vision "boîte noire" de la grille. Ce modèle permet de déterminer l'impact de la variabilité de la latence sur les applications. En particulier, nous montrons que la variabilité conduit à une perte de performance d'un facteur 2 sur l'application des étalons de bronze (voir chapitre 1). Nous montrons que la latence peut être modélisée correctement par une variable aléatoire de distribution mixte log-normale / Pareto dont les paramètres sont ajustés à des données expérimentales. Nous quantifions enfin l'influence de paramètres du contexte des tâches comme le site d'exécution ou le Resource Broker sur la distribution de la latence.

Schopf and Berman showed in [Schopf and Berman, 1999] that variability is an important source of performance drops for parallel applications and that it should be avoided, even if it reduces the mean performance of the infrastructure. On the EGEE production grid, the latency is not only high but also very variable (about 5 minutes with an inter-quartile range of 3 minutes), as demonstrated in the previous chapter. Thus, it is suspected to strongly penalize the performance of a workflow.

This variability is supposed to come from various factors, including the heterogeneity and volatility of the infrastructure (endogen factors) and the load imposed to it (exogen factor). Because a deterministic modelling of the system seems hardly tractable, the approach adopted in the remaining of this thesis is probabilistic. We propose to adopt a black-box model of the grid, where the latency is a random variable capturing all the sources of variability. The grid provides jobs submission and monitoring facilities and introduces a random delay before

the beginning of their execution. Beyond the submission gateway, all the parameters are let to the responsibility of the system administrators. We are only interested in the *observable* behavior of the system. For instance, the heterogeneity of processors, memory capacities, network bandwidth, I/O performance and other machine dependent factors will be included in the latency random variable. Similarly, the load and exogen factors are viewed as hidden variables influencing the grid latency. Consequently, in our grid model, the actual number of available grid resources and their characteristics will remain unknown. They are also viewed as hidden variables impacting the latency which is the only variable of interest.

In some cases, system flaws lead to huge latencies (several hours) largely prevailing on the ones faced by the other tasks of the application. Those latency values are obviously outside the distribution of the normal latency random variable and should be modeled separately, as outliers. The main causes of outliers are hardware failures, software bugs, locally heavy loads leading to tremendously high service response times and scheduling mistakes leading to jobs facing high queuing times. An important characteristics of production systems is the presence of a significant ratio of those outlier jobs.

The goal of the first section of this chapter is to quantify the impact of the latency variability on the performance of a workflow. To do that, we propose a model of the execution time (makespan) of a workflow taking into account the random variable modelling the latency. Determining an accurate predictive model of the distribution of the grid latency is a statistical problem which is not straightforward. Indeed, the grid latency is not stationary. The nature of its distribution, or at least its parameters, depend on time and on exogen factors. This dependency may not be easily modeled because it depends on factors such as the current number of jobs submitted by other users that are hardly predictable. In the second section of this chapter, we provide some experimental results analysing the status of the grid *on a given time period*. A log-normal / Pareto distribution is correctly fitted to those measures and the influence of some context parameters is studied in order to refine this distribution model.

6.1 Influence of the latency variability on a workflow

In this section, a probabilistic model of the makespan of a workflow is presented. It is used to estimate the impact of the variability of the latency on the bronze standard application. In the experiment presented in this section, the parameters of the distribution of the grid latency are estimated *a posteriori*, from the logs of the execution, in order to keep off the statistical problem of the estimation of up-to-date parameters. Such a model is demonstrated to correctly fit the experimental data. We derive from it a theoretical *non variable* system whose average latency is the same than the real one. A forecast of what would happen if the latency of the system were not variable is thus provided.

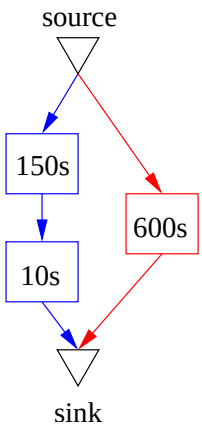

Figure 6.1: Because of the variability of the latency, the critical path of a service workflow depends on the number of data items put in its inputs. On this figure, if the latency of the grid is assumed Gaussian with mean 300 seconds and standard-deviation 200 seconds, and if a single data item is put in the source, then the critical path of the workflow is the red one which is expected to be 900 seconds (300 seconds for the expected latency + 600 seconds for the execution) whereas the expectation of the blue one is only 760 seconds (2 $\times$ 300 seconds latency + 160 seconds execution). But as soon as the number of data items is greater or equal to 3, then the critical path of the workflow becomes the blue one: for 3 data items, the expectation of the blue path is 1098 seconds whereas it is 1069 seconds for the red path. The computation of those values corresponds to the expectation of the distribution of the max of the execution times of the path on the data items, as stated in section 6.1.3 (data parallelism only – DP case). Numerical details about this computation are reported in appendix A.

6.1.1 Definitions

Critical path of the workflow. In the service workflow of an application (see chapter 2), a *path* denotes a set of services linking an input of the workflow to an output. A path is defined independently from the data to process: it will be instantiated at runtime on a set of data items. The *critical path* of the workflow denotes the longest path in terms of execution time. Because of the variability of the latency, this expected critical path depends on the number of data items on which the workflow is iterated, as suggested by figure 6.1.

Notations. n_W denotes the number of services on the critical path of the workflow and n_D denotes the number of data sets to be processed by the workflow. n_D corresponds to the degree of data parallelism that will be achieved by the workflow. $i \in [0, n_W - 1]$ denotes the index of the i^{th} service of the critical path of the workflow. Similarly, $j \in [0, n_D - 1]$ denotes the index of the j^{th} data set to be executed by the workflow. $T_{i,j}$ denotes the duration in seconds of the processing of the data set j by the service i. It corresponds to the total time from the job submission to its completion. $T_{i,j} = r_{i,j} + R_{i,j}$ is made of an application-dependent part

$r_{i,j}$ and the grid latency part $R_{i,j}$. $r_{i,j}$ corresponds to the computation time of service i on the data segment j. It is supposed to be a fixed value (predictable execution time) by opposition to $R_{i,j}$ which is a random variable. $R_{i,j}$ will model all the sources of variability coming from the infrastructure. For instance, the variability coming from the performance of the worker nodes or the network connection of the execution site will be included in this variable. $R_{i,j}$ does not take into account outliers. They have to be modeled separately. To study the impact of the variability of the grid on the performance of the application, the case where $R_{i,j}$ is a fixed value will also be considered in the following. Σ denotes the makespan of the workflow. The goal of the next sections is to express it with respect to n_D, n_W, $r_{i,j}$ and $R_{i,j}$ and to the parallelism configuration.

6.1.2 Hypotheses

Services are supposed not to simultaneously process all the data segments: in the following, workflows are assumed not to contain any synchronization barriers *on the data items*. Workflows containing such synchronization barriers may be analyzed as two sub workflows respectively corresponding to the parts of the initial workflow preceding and succeeding the synchronization barrier.

The n_D data items on which the application is iterated are assumed to be of equal size. For instance, in the context of medical image analysis, it means that all the processed images have the same dimensions. It is the case for the bronze standard application as well as for several medical image analysis applications that aim at processing a whole database of images acquired in similar conditions. Consequently, the execution times $r_{i,j}$ of the jobs can be assumed to be independent from the data: $\forall j, r_{i,j} = r_i$. One could have noticed that given the figures displayed on table 5.1 of chapter 5, this hypothesis is not strictly verified. Yet, the standard-deviation of the execution times of the services remains lower than 10 seconds for most of them and is lower than a minute for the other ones. This standard-deviation is far lower than the one of the EGEE latency, so that this hypothesis can be considered as holding in a first approximation. If the variability of the execution times of the services has to be taken into account, one should also consider the execution times of the services as random variables. Then, in the following, $T_{i,j}$ notations should not be expanded (into $r_i + R_{i,j}$) and the distribution of this random variable could be determined with respect to the distributions of the execution times and of the grid latency $R_{i,j}$. Yet, in this work, we concentrate on the variability introduced by the grid platform itself rather than on the intrinsic variability of the algorithms, which is more application-specific.

$R_{i,j}$ are assumed to be independent random variables: the jobs are supposed not to influence each other. Given the scale of the infrastructure, this hypothesis can be considered as realistic. What is assumed here is that the application itself does not impact the grid latency. Indeed, the submission of a couple of hundreds of jobs spread on a few hours should not disturb the grid

so much. Nevertheless, bottlenecks may trouble this hypothesis. For instance, the submission time of several jobs from the same machine is very likely to depend on the number of submitted jobs. Taking this phenomenon into account may not be easy from a general perspective: understanding how jobs interact with each other in the whole system seems difficult. Still, for specific steps such as the submission, some models could be integrated to take into account the interactions between jobs.

The grid latency is assumed not to depend on the nature of the submitted jobs, *i.e* on the command-line that will be executed on the resources. It is true that the queuing time of the job in the batch of a computing center is highly dependent on the expected duration of the task. However, as it is done by the huge majority of grid end-users, the expected wall-clock time of the job is assumed to be set to its default value, which is supposed to be largely superior to the effective duration of the submitted jobs. Consequently, the distribution of the grid latency is assumed to be independent from i. Similarly, the distribution of the grid latency is supposed to be independent from the data (*i.e* the distribution of $R_{i,j}$ is independent from j). Assuming that the distribution of the latency is independent from the service and from the data is not so critical. Considering applications handling large volumes of data (*i.e* applications for which data transfer times would be of several minutes), one could simply include it into the r_i value. Problems may only arise for applications for which the data impacts the job life cycle inside the system, *i.e* disturbs its submission, scheduling or queuing time. If they ever exist, such interactions should be of limited importance and still negligible with respect to the average grid latency. Thus, $R_{i,j}$ are assumed to be independent and identically distributed (iid) random variables.

6.1.3 Makespan of the application

Under those hypotheses, the expression of the makespan of the workflow for two different execution policies can be determined. We distinguish the case where only data parallelism is present (DP case) from the case where both data and service parallelism are enabled (DSP case). Definitions of those parallelisms are provided in chapter 4.

Case DP (Data Parallelism only). All the data segments are processed concurrently and the execution is synchronized after each service invocation.

$$\begin{aligned} \Sigma_{DP} &= \sum_{i<n_W} \max_{j<n_D} \left\{T_{i,j}\right\} = \sum_{i<n_W} \max_{j<n_D} \left\{r_i + R_{i,j}\right\} \\ &= \sum_{i<n_W} r_i + \sum_{i<n_W} \max_{j<n_D} \left\{R_{i,j}\right\} \end{aligned} \tag{6.1}$$

Case DSP (both Data and Service Parallelism). All the data segments are processed concurrently and the services are pipelined.

$$\Sigma_{DSP} = \max_{j<n_D} \left\{ \sum_{i<n_W} T_{i,j} \right\} = \max_{j<n_D} \left\{ \sum_{i<n_W} \left(r_i + R_{i,j}\right) \right\}$$

$$= \sum_{i<n_W} r_i + \max_{j<n_D} \left\{ \sum_{i<n_W} R_{i,j} \right\} \tag{6.2}$$

Deterministic case. If the latencies $R_{i,j}$ are fixed values, then for every i and every j, $R_{i,j} = \bar{R}$ and the above expression simplifies to:

$$\Sigma_{DP} = \Sigma_{DSP} = \sum_{i<n_W} r_i + n_W.\bar{R} \tag{6.3}$$

In this case, there is no difference between the DP and DSP cases. This deterministic model will be used to forecast the performance of the application in absence of variability of the latency. It corresponds to a theoretical non-variable system which has the same average latency as the real one.

Probabilistic case. The goal is to determine the expectation $E(\Sigma)$ of the makespan of the workflow and its standard deviation $\sigma(\Sigma)$. We could then have a prediction of the makespan of the workflow ($E(\Sigma)$) and an uncertainty on it ($\sigma(\Sigma)$)). In the following of this thesis, given a random variable X, f_X will denote the probabilistic density function (pdf) of X and F_X its cumulative density function (cdf).

DP case: thanks to the linearity of the expectation operator and to the fact that r_i is a fixed value, equation 6.1 gives:

$$E\left(\Sigma_{DP}\right) = \sum_{i<n_W} r_i + n_W E\left(\max_{j<n_D}\{R_{i,j}\}\right)$$

Given that the cumulative density function of the random variable $K = \max_{j<n_D}(R_{i,j})$ is $F_K = F_{R_{i,j}}^{n_D}$, we have:

$$E\left(\max_{j<n_D}\{R_{i,j}\}\right) = n_D \int_{-\infty}^{\infty} t f_{R_{i,j}}(t) F_{R_{i,j}}(t)^{n_D-1} dt, \tag{6.4}$$

We then have:

$$\begin{aligned} E(\Sigma_{DP}) &= \sum_{i<n_W} r_i + n_W E\left(\max_{j<n_D}\{R_{i,j}\}\right) \\ &= \sum_{i<n_W} r_i + n_W n_D \int_{-\infty}^{\infty} t f_{R_{i,j}}(t) F_{R_{i,j}}(t)^{n_D-1} dt \end{aligned} \tag{6.5}$$

Moreover given that two jobs are independent, equation 6.1 gives:

$$\sigma\left(\Sigma_{DP}\right)^2 = n_W \sigma\left(\max_{j<n_D}\{r_i + R_{i,j}\}\right)^2$$

And thus, because r_i are fixed values:

$$\sigma\left(\Sigma_{DP}\right)^2 = n_W \sigma\left(\max_{j<n_D}\{R_{i,j}\}\right)^2$$

Given that $\sigma(\max_{j<n_D}\{R_{i,j}\})^2 = E(\max_{j<n_D}\{R_{i,j}\}^2) - E(\max_{j<n_D}\{R_{i,j}\})^2$ and that $E(X^2) = \int_{-\infty}^{\infty} t^2 f_X(t)dt$, we have:

$$\begin{aligned}\sigma(\Sigma_{DP})^2 &= n_W \sigma\left(\max_{j<n_D}\{R_{i,j}\}\right)^2 \\ &= n_W\left[n_D\int_{-\infty}^{\infty} t^2 f_R(t)F_R(t)^{n_D-1}dt - n_D^2\left(\int_{-\infty}^{\infty} t\, f_R(t)F_R(t)^{n_D-1}dt\right)^2\right] \qquad (6.6)\end{aligned}$$

DSP case: The max operator prevent from simplifying the expressions of the expectation and standard-deviation of the makespan. Yet, those values can still be computed numerically, as it will be done in the following.

$$E(\Sigma_{DSP}) = \sum_{i<n_W} r_i + E\left(\max_{j<n_D}\left\{\sum_{i<n_W} R_{i,j}\right\}\right) \qquad (6.7)$$

$$\sigma(\Sigma_{DSP}) = \sigma\left(\max_{j<n_D}\left\{\sum_{i<n_W} R_{i,j}\right\}\right) \qquad (6.8)$$

6.1.4 Experimental results

The goal of this section is to present experimental results that:

1. evaluate the relevance of the model presented above to explain the makespan of the application;
2. study the impact of the latency variability on the execution on a production grid.

6.1.4.1 Experiments conditions

The workflow of the bronze standard application (see chapter 3) was executed on different input data sets sizes, ranging from 12 to 126 image pairs. Each of the input image pairs led to 6 job submissions. Thus, the amount of tasks submitted by the workflows ranged from 72 to 756. The workflow manager used for this experiment was MOTEUR (see chapter 4). Each data set was processed in the DP and DSP configurations. The workflow executions are not simultaneously submitted to the grid. Submitting all the executions simultaneously would not have been possible without introducing strong biases in the results. Indeed, the submission mechanism would have become a bottleneck and it is very likely that the executions would have disturbed each other. To avoid that, one should have used a different user interface and a different resource broker for each execution. Clearly, this would also have created different experimental conditions between the runs so that we rather executed successively the workflow runs. Changes in the grid status (number of available sites, average load from other users, ...) may thus happen between those runs. Those changes will be captured by the fitting of the

parameters of the latency distribution that is adapted to the execution conditions, as developed in the next sub-sections.

On a production grid infrastructure, setting a timeout to tasks is mandatory because a small fraction of tasks are likely to remain blocked for hours in a waiting queue or even to get lost: the timeout value prevents the application from facing outliers. Because of that, and taking into account failures that are likely to occur, tasks need to be resubmitted if necessary. For example, on the EGEE grid that, the tasks success rate was around 84% at the time of those experiments. In those experiments, the timeout value was arbitrarily set to 1 hour (which is far greater than the services walltime r_i - see table 5.1 of chapter 5) and no retry was performed in order to prevent the makespan to be influenced by resubmissions that are not modeled. Thus, timed-out jobs are neglected. A strategy to optimize the timeout value is described in chapter 8 of this thesis.

6.1.4.2 Gaussian assumption for R

In this experiment, the grid latency is assumed to be Gaussian. This latency model has been determined by searching the distribution that best fits the experimental results. It led to relevant results, correctly explaining the makespan of the application, as shown in section 6.1.4.4. However, this assumption differs from the heavy-tailed model that will be studied in section 6.2. At this point, a justification of this hypothesis may be the fact that the timeout has been set to 3600 seconds and that jobs that timed-out have been neglected. Consequently, the distribution of the latency can be assumed to have a lighter tail than the one that would be obtained by setting a higher time-out value or taking into account resubmissions.

6.1.4.3 Model computation

To compute the probabilistic model presented in section 6.1.3, the required parameters are (i) the deterministic part of the running time of each service on a single data set r_i and (ii) the mean μ and standard deviation σ of the grid latency R.

The r_i values were obtained as the mean values of the benchmark of the services presented in table 5.1 of chapter 5. Estimating μ and σ is more difficult: their values are likely to depend on the number of input data segments and to vary along time. The goal of this experiment is not to obtain an up-to-date model of the distribution of the grid latency. This is investigated in section 6.2. The point here is to validate a model of the *application*, assuming that the distribution of the latency is known. Thus, μ and σ were evaluated *a posteriori*, from the execution trace. Estimating them *a priori* requires a dedicated grid monitoring system, which is out of our scope here.

The first required step for the computation of the model is to determine the critical path of the workflow. As already noticed, the number of processed data sets n_D dramatically influence

the makespan of a given path of the workflow, particularly in case of high latency variances. Thus, we determined the critical path of the workflow separately for each number of data items.

The value of the makespan obtained from the deterministic model is an estimate of the performance that could be obtained in absence of variability. As suggested by equation 6.3, it is computed by considering that the latency is a fixed value. This value is set to the average of the observed latency.

6.1.4.4 Results

Figure 6.2 displays the experimental results. Two experiments are displayed and compared to the probabilistic and deterministic models: a data parallel execution (upper DP curves) and a data+service parallel execution (lower DSP curves). On both graphs, the experimental data is figured in red. Probabilistic models are figured with squares and deterministic ones with crosses. For the experimental and deterministic cases, a linear regression is superimposed, as already done in the experiments of chapter 5. For the probabilistic cases, intervals corresponding to $[\mu - 3\sigma, \mu + 3\sigma]$ are also drawn.

6.1.5 Discussion

6.1.5.1 Metrics for the analysis

To analyze performances, the first relevant metric from the user point of view is the speed-up, measured as the ratio of the execution time over the sequential execution time. The most interesting speed-up value is the maximal one obtained on the application, which in this case is the one obtained for the largest input data set.

To have a finer interpretation of the results, the global behavior of the application makespan with respect to the number of input data sets can be approximated with straight lines estimated through a linear regression. Those fitted straight lines are also plotted on figure 6.2. The relative error of this approximation with respect to the experimental data is 7.8% for the DSP case and 11.6% for the DP one.

The y-intercept and slope of the fitted lines can then be considered. The y-intercept value, expressed in seconds, measures the latency of the application on this infrastructure. This value corresponds to the nominal latency of the grid added to the execution time of a single data set by the application workflow: it is the incompressible amount of time required to access the infrastructure. The slope of the fitted line, expressed in seconds by jobs, is related to the throughput of the application. This value measures the data scalability of the infrastructure, that is to say its ability to process huge data sets with the same level of performance. Those metrics are similar to the one used in chapter 5 to compare the EGEE production grid to Grid'5000 clusters.

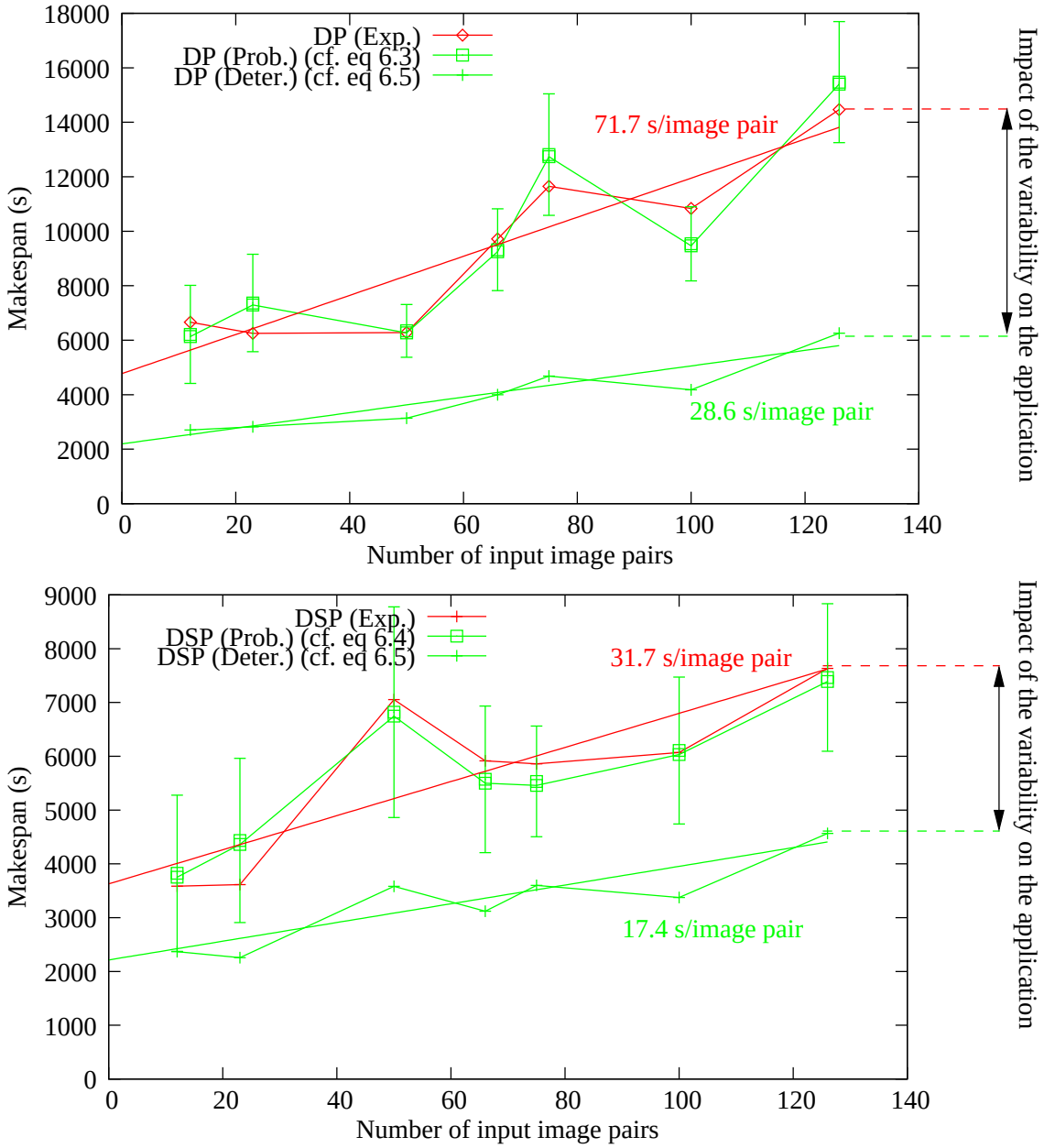

Figure 6.2: Comparison of the makespan of the application in the experimental and model cases. Top: DP case ; Bottom: DSP case. The DP case is less robust to the latency distribution tail, which explains its weaker performance. The impact of the variability of the latency can be noticed by comparing the deterministic (bottom green curves) case with the experimental (red) one. Variability leads to a factor 2 performance drop on this application.

	Experiment		Probabilistic Model		Deterministic Model	
	DP	DSP	DP(eq 6.5)	DSP(eq 6.7)	DP(eq 6.3)	DSP(eq 6.3)
y-intercept (seconds)	4778.0	3628.2	4921.6	4002.4	2195.2	2214.5
Slope (s/data sets)	71.7	31.7	72.2	26.0	28.6	17.4
Max speed-up	7.0	13.2	6.5	13.5	15.9	21.7

Table 6.1: Values of the metrics. The relevance of the probabilistic model can be noticed by comparing columns 4 and 5 to columns 2 and 3. The impact of the variability of the latency on the application can be quantified by comparing columns 6 and 7 to columns 2 and 3.

The values of those metrics are reported in table 6.1. The two first columns of this table correspond to the experimental values for the DP and DSP cases. The two next ones correspond to the values computed with the probabilistic model of section 6.1.3, from measured mean and standard-deviation of the latency. The two following columns correspond to the values computed with the deterministic model of section 6.1.3. Those values correspond to the ones that would have been obtained if the infrastructure were not variable.

6.1.5.2 Relevance of the probabilistic model

First, from a qualitative point of view, the results shown on figure 6.2 exhibit some singular behaviors. For instance, even if the global trend of the curves is to increase with the number of input image pairs, one can notice some local decreases, as between 50 and 75 input images for the DSP case and between 75 and 100 input images for the DP one. It is correctly explained by the model, thanks to the fitting of the parameters (mean and standard-deviation of the latency) to the experimental data. Actually, those local decreases can be explained by a diminution of the latency mean and standard-deviation between those values which do not correspond to simultaneous executions, as already mentioned.

Another singular behavior are the measures done for 50 input images pairs. Indeed, the DP case is there faster than the DSP one. Here again, this behavior can be explained by changes of the grid status between those two runs: it would not have happened if the execution were simultaneously submitted. However, the probabilistic model is again able to explain this behavior thanks to the *a posteriori* fitting of the Gaussian distribution to the observed one.

From a quantitative point of view, and as figure 6.2 shows, the probabilistic model is quite relevant and able to explain the experimental results (on this figure, experimental results are displayed in red and values from the probabilistic model are figured with squares). The mean relative error of the probabilistic model with respect to the experimental data is 6.7% for the DSP case and 8.4% for the DP one. The fact that this error is greater in the DP case than in the

DSP one is consistent because the makespan of the application is more affected by distribution tails in the DP case than in the DSP one. Indeed, in the former case, the processing of every data segment is depending on the processing of all the others because the execution is synchronized after each service invocation. It is also worth noticing that all the experimental values stay inside the $[\mu - 3\sigma, \mu + 3\sigma]$ interval. It shows that the model is able to provide bounds for the error it makes with respect to the experimental case.

The speed-up figures measured and displayed in table 6.1 (7.0 and 13.2 in the DP and DSP cases respectively) are very close to the probabilistic model estimates (6.5 and 13.5 respectively), showing that MOTEUR efficiently enables the workflow, data and service parallelism without introducing a significant performance loss.

6.1.5.3 Impact of the service parallelism

It has been explained in section 6.1.3 that in a deterministic system, the DP and DSP cases lead to identical performance. Considering the maximal experimental speed-up values, the DSP case was 1.8 times faster than the DP one. The y-intercept metric is 1.3 times higher in the DP case than in the DSP one. The slope ratio comparing those two cases is 2.3.

The fact that service parallelism does speed the execution up can be explained by the service parallelism making the application less sensitive to distribution tails. If no variability was possible (deterministic model), the impact of service parallelism would indeed be lower: the maximal speed-up ratio would be 1.4, the y-intercept ratio would be 1.0 and the slope ratio would be 1.6. It confirms the behavior described above: the more variable the infrastructure, the more interesting the service parallelism.

The impact of service parallelism is higher on the slope than on the y-intercept value: for the experimental case, table 6.1 shows that service parallelism reduces the slope with a factor 2.3, whereas it only leads to a factor 1.3 on the y-intercept. It is consistent that the benefit yielded by service parallelism mainly affects the data scalability of the application: the more important the number of submitted jobs, the more important the probability to lie in the distribution tail.

However, even in case of a non variable platform, there is still an impact of service parallelism on the slope of the straight lines and thus on the maximal speed-up, whereas there is no more on the y-intercept value. This can be explained by the fact that service parallelism reduces the mean grid latency due to sequential procedures such as the submission time. Indeed, if service parallelism is not present, waves of simultaneous job submissions occur, whereas submissions are more spread over time in case of service parallelism. This explains the impact of service parallelism on the scalability of the application.

6.1.5.4 Impact of variability

The impact of the variability of the grid latency on the makespan of the application is figured by the distance between the green and red curves on figure 6.2. Considering the values of table 6.1,

variability led to a maximum speed-up reduction factor of 2.4 for the DP case and 1.6 for the DSP one. If the infrastructure were deterministic, we would obtain a maximal speed-up of 21.7 in the DSP case, whereas it is only 13.2 there. Considering the y-intercept metric, variability leads to an increased factor of 2.24 for the DP case and this factor is 1.8 for the DSP one. Variability also introduces a 2.5 increase factor on the slope metric for the DP case and a 1.5 one for the DSP case. Variability has more impact on the DP case than on the DSP one. Indeed, as already mentioned before, the DP case is far less robust than the DSP one.

The estimates made for a deterministic system show that an additional speed-up in the order of 2 can be expected by adopting strategies to reduce the system variability.

6.1.5.5 Analysis of the grid's latency

The total mean latency introduced by the grid is slightly growing with the number of input data sets, as displayed on figure 6.3. This figure plots the mean latency obtained for the DP and DSP cases and identifies the different sources of latency, namely submission, scheduling and queuing times and the overhead added to the walltime. These values were obtained by subtracting the average benchmarked walltime to the average actual walltime of the tasks. The slow latency increase shows that we are far from saturating the large scale infrastructure.

Table 6.2 displays the mean values obtained for each entity of the infrastructure. The most important source of latency is the queuing time, as it is easily understandable on a multi-users platform. Then comes the overhead on the walltime, that includes data transfers and performance of the running hosts. Submission and scheduling times are the less important sources of overhead. The latency coming from the load of the infrastructure is distributed among those 4 entities. Yet, most of it may be included in the queuing latency. The latency coming from the walltime of the jobs covers the heterogeneity of the machines of the grid. Indeed, the services have been benchmarked on a particular machine and the perfomance of the grid worker nodes is unknown. All those values have been measured with the grid information system. They are thus highly dependent on its accuracy. In particular, too small update frequencies may disturb those measures. Yet, applications also rely on this information system so that those values are representative of what could be measured from the applications.

Entity	Mean latency (s)
Submission	182
Scheduling	110
Queuing	308
Walltime	279
Total	880

Table 6.2: Mean grid overhead for each component

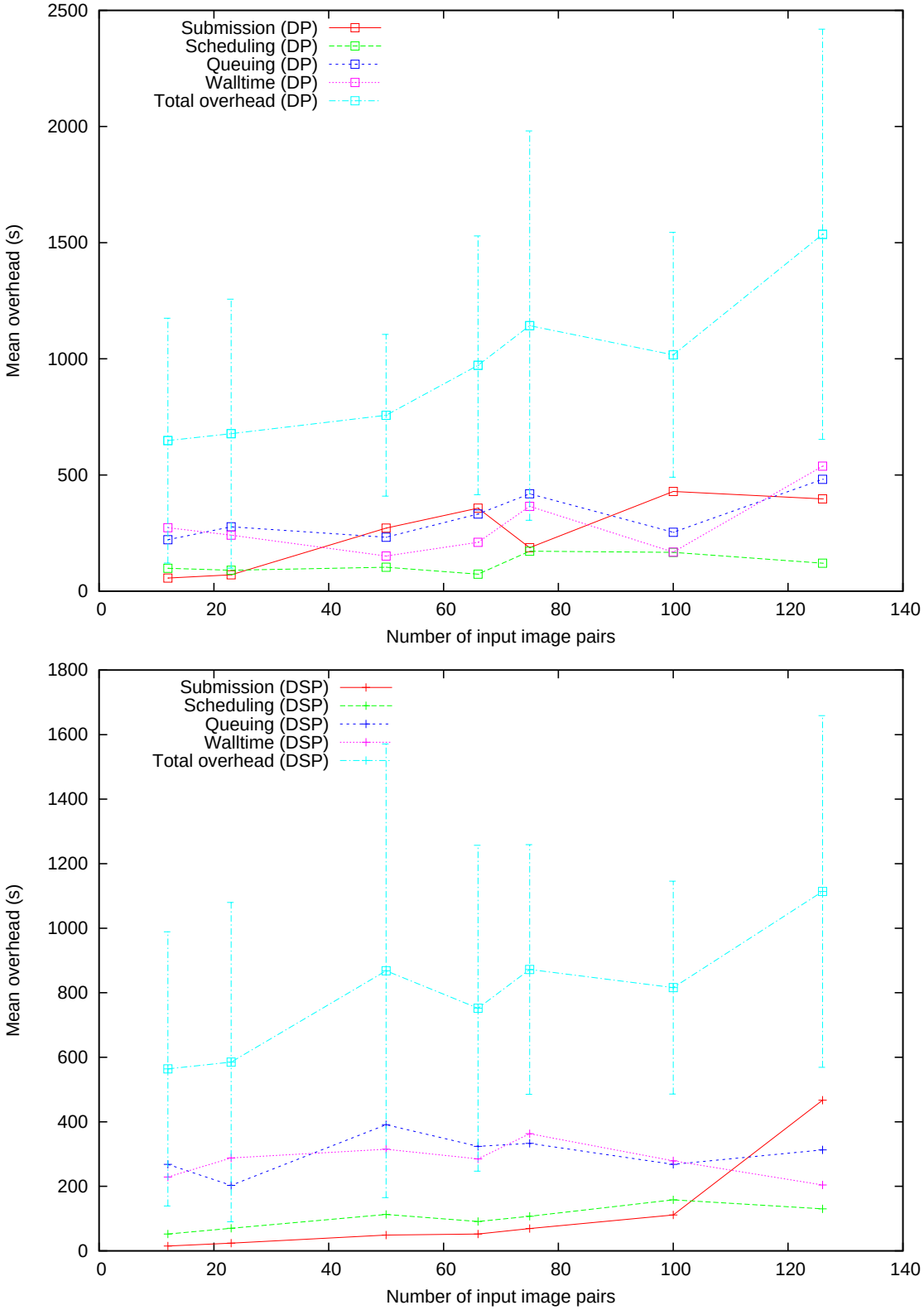

Figure 6.3: Mean overhead for each grid's component. The standard deviation of the total latency is plotted on the corresponding curve. Top: DP ; Bottom: DSP.

The variability of the overhead is hardly interpretable. The standard deviation of the total overhead varies from 390s to 890s but does not exhibit global trends.

6.2 Characterization of the latency variability

The goal of this section is to provide some information to characterize the distribution of the latency. Models of the grid latency are the basis of strategies to optimize the job submission parameters as presented in part III. Determining a precise and up-to-date model of the grid latency is the starting point to allow further optimizations to tackle the effects of the latency. The latency is modeled as a random variable and an outlier ratio. The random variable describes the latency variability in a *normal operation mode*. Outliers correspond to *system faults* that lead to huge latencies prevailing on the ones of the other tasks of the application. Those latency values can be considered as infinite. In this section, a global model of the distribution of the latency in the normal functioning mode is first presented and then the influence of some job context parameters is studied in order to make the model more accurate.

6.2.1 Model of the measured distribution

Data acquisition. To measure the distribution of the system latency on the EGEE grid, probe jobs that only consist in the execution of a `/bin/hostname` were submitted and their round-trip time was measured. A constant number of probes was maintained inside the system by submitting a new one as soon as one completed to avoid introducing any extra variability. This measure of the distribution of R gathers 2137 probe jobs involving 3 RBs. The maximal duration of those jobs was fixed to $t_{\max} = 10000$ seconds. Beyond this value, a job is considered as an outlier. Given those conditions, the measured outlier ratio was 2.5%. In normal operating mode, the measured distribution of R is plotted on figure 6.4 (plain red curve). Its expectation is 393 seconds and its standard deviation is 792 seconds. The corresponding latency histogram is shown on figure 6.5.

Modeling. The distribution of the experimental data shown on figure 6.4 appears to be close to a log-normal distribution for low values (up to 500 seconds) and a Pareto distribution beyond. Pareto distributions are used to model a large class of computer system measurements (jobs durations, size of the files, data transfers length on the Internet...) [Harchol-Balter and Balter, 2002]. Based on this observation we fitted the experimental data with the following distribution which is an interpolation of the log-normal and Pareto ones, for t in $[t_{\min}, t_{\max}]$:

$$F_R^m(t) = (1-\alpha(t))\,\Phi\left(\frac{\ln(t-t_{\min})-\mu}{\sigma}\right) + \alpha(t)\left(1-\left(\frac{a}{a+t}\right)^{\nu}\right) \tag{6.9}$$

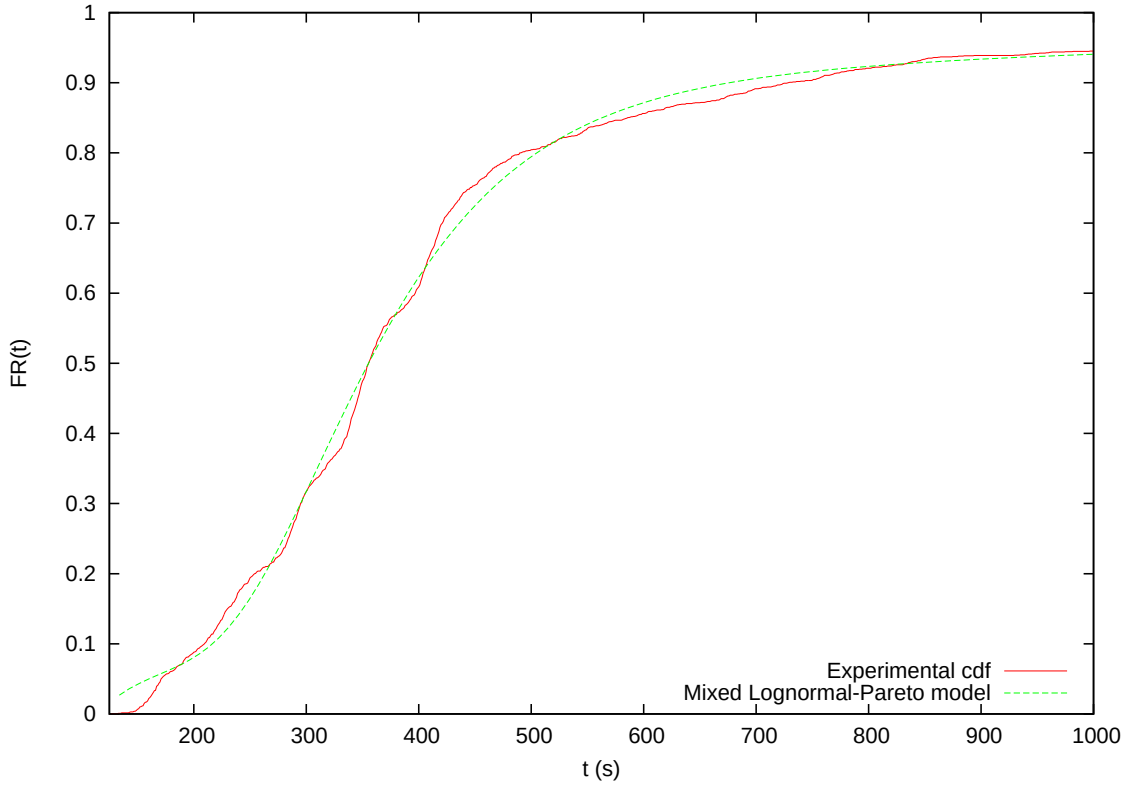

Figure 6.4: Measured data (plain) and best fitting Log-normal-Pareto model (dashed).

$$\text{with} \quad \alpha(t) = \left(\frac{t - t_{\min}}{t_{\max} - t_{\min}} \right)^k$$

$t_{\min}$ denotes the smallest latency measured among the data (the cdf is zero below this value) and $t_{\max}$ the highest one. There are thus five parameters fully describing this model (μ, σ, a, ν, and k). $\alpha(t)$ is a weight function designed so that $\alpha(t_{\min}) = 0$ and $\alpha(t_{\max}) = 1$. The model thus tends towards a log-normal distribution in $t_{\min}$ and towards a Pareto one in $t_{\max}$. The best fit of the model 6.9 with the experimental data was estimated by least-square minimization, minimizing the following criterion:

$$\arg \min_{(\mu,\sigma,a,\nu,k)} \left\{ \sum_{i=t_{\min}}^{t_{\max}} \left(F_R^{model}(i) - F_R^{exp}(i) \right)^2 \right\}$$

where $F_R^{exp}(i)$ is the value of the measured distribution at time i. The fitted model is displayed on figure 6.4 (dashed green curve). A Kolmogorov-Smirnov test was made to evaluate the quality of the model. When considering an under-sampling of up to 1000 measurements, the Kolmogorov-Smirnov test value is $D_{1000} = 1.35$ (we used $D_n = \sqrt{n} \sup |F_R^{exp} - F_R^m|$), which correspond to a p-value $p = 0.051$. The tests is thus positive. It shows that a simple model (5 parameters) can accurately model the distribution measured over a very complex grid system (EGEE grid infrastructure).

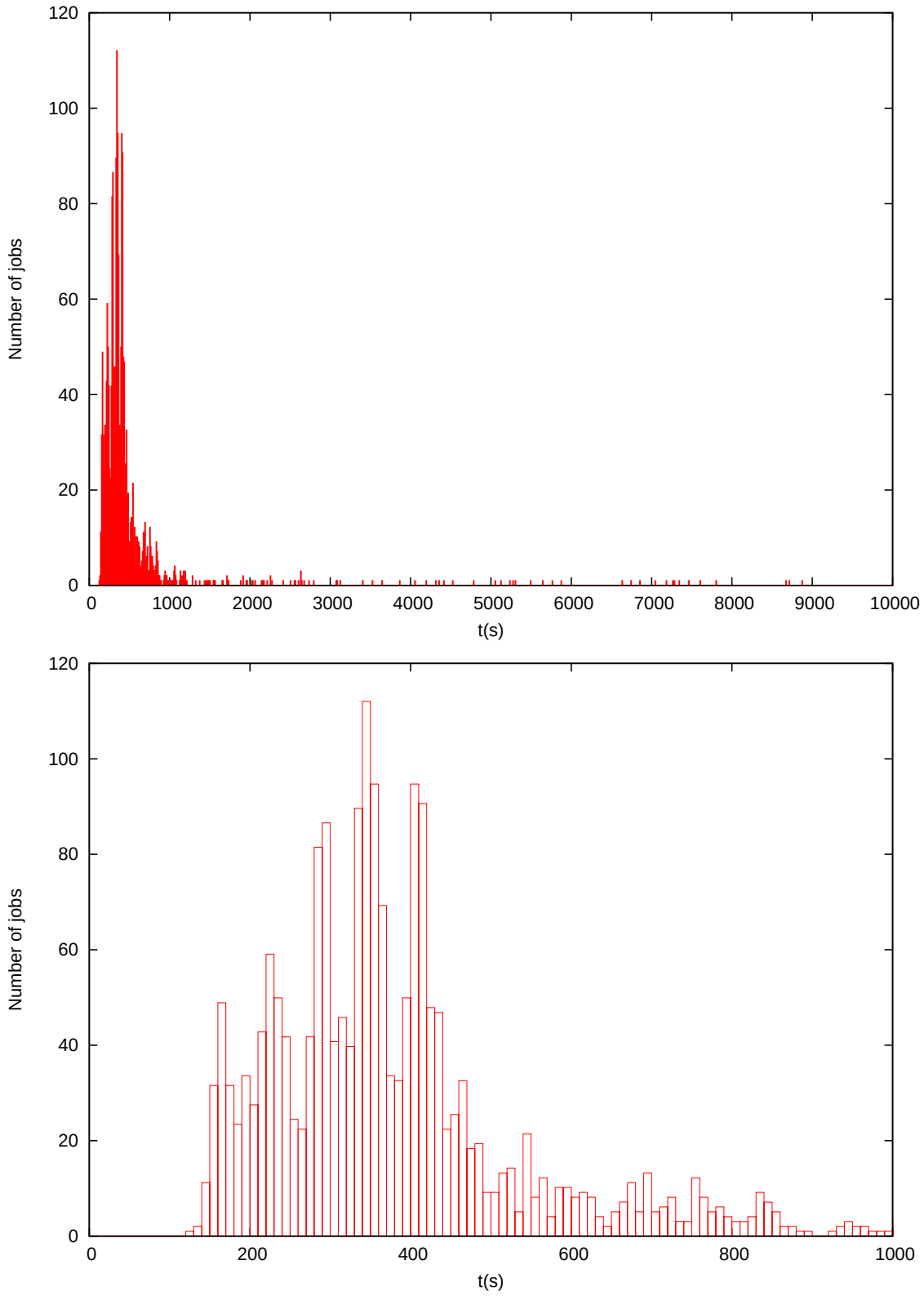

Figure 6.5: Histogram of the measured latencies. Values have been gathered into bins of 10 seconds. The y-axis denotes the number of jobs in each bin. The total number of submitted jobs is 2137. 2.5% of them were considered as outliers (latencies higher than 10,000 seconds) and thus do not appear on this histogram. The upper figure plots the remaining ones and the bottom one is a close-up on the [0s,1000s] interval.

6.2.2 Influence of job context parameters

Each job can be characterized by its execution context that depends on the grid status and may evolve during the job life-cycle. A promising way to refine the distribution model presented above is to adapt it to the evolutions of this job context. The ultimate goal of such a study would be to have the latency prediction of a job start with a rough estimation (*e.g* from the global model presented above) and be refined as the job status is evolving. For instance, once the submission and scheduling times are known, one could infer some information about the grid load and refine the latency estimation accordingly. Furthermore, as soon as the target Computing Element (CE) is known, one could switch to a particular latency model dedicated to this CE. The context parameters of a job are seen as hidden variables that are progressively discovered, thus refining the latency estimation. One could probably end-up with a complete probabilistic model of the grid.

The context of a job depends both on grid internal and external parameters. The internal context corresponds to parameters such as the hosts involved in the management of a specific job. It may not be completely known at the job submission instant. The external context is related to parameters such as the day of the week and may have an impact on the load imposed to the grid. Many parameters may have a direct influence on the jobs submitted to a grid infrastructure. We are here focusing on three of them which proved to have a particular impact as shown below: the Computing Element (CE), Resource Broker (RB), and the day of the week.

Data collection. The results presented here involve 4477 probe jobs acquired with the same method as in section 6.2.1. For each one, the job submission date, the User Interface (UI) used, the UI load at submission time, the RB used, the CE used and the jobs status duration (total duration t_{tot}, submission time t_{sub}, scheduling time t_{rb}, queuing time t_{q} and running time t_{run} as illustrated in figure 6.6) was logged. The median of this sampling is 363 seconds, its expectation is 559 seconds and its standard deviation is 850 seconds.

6.2.2.1 Site of computation parameter

The probe measures involved 90 different CEs of the infrastructure. Figure 6.7 plots the cumulative distribution of the grid latency for each CE involved in the experiment. To ensure statistical significance, CEs with less than 30 probe measures were removed from the study. 60 computing elements out of the 90 were remaining. Figure 6.7 suggests that 3 classes can be identified among the CEs. A k-means classification was thus done on the cumulative density functions of the CEs and the obtained classes are identified with distinct colors on the figure. Centroids of the classes are plotted in black.

The first class of CEs, pictured in blue, has the highest performance in average. The median of its centroid is 237 seconds. It is composed of 15 CEs. The second class of CEs, pictured in

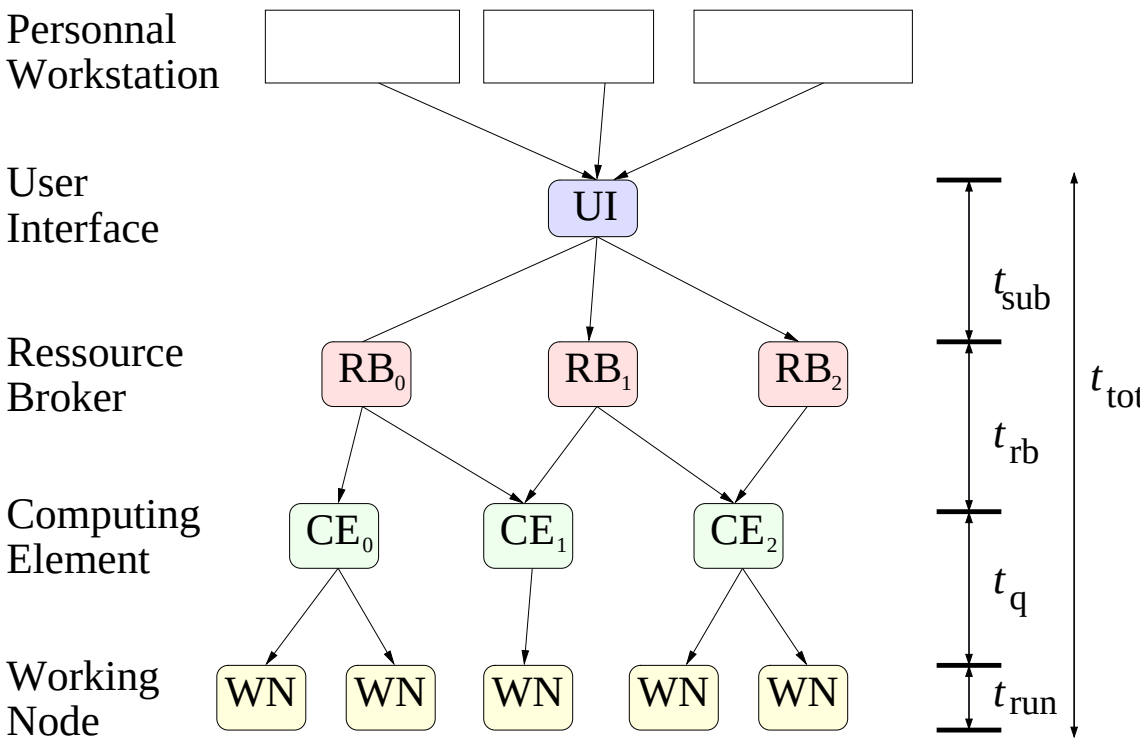

Figure 6.6: Job life-cycle inside EGEE and measured durations.

green, is composed of 35 CEs. The median of its centroid is 373 seconds, which corresponds to a 1.6 ratio with respect to the fastest class. Finally, the slowest class, pictured in red, is composed of 10 CEs and the median of its centroid is 652 seconds. Table 6.3 compares the median, expectation and standard-deviation of the grid latency for each CE class. It reveals that even if the first (blue) class of CEs has the highest performance in average, it is also more variable than the second (green) class. The third (red) class is the most variable. As shown in section 6.1, the impact of variability on the performance of an application depends on the number of submitted jobs. In some cases (high number of jobs), it would be better to submit jobs on a less variable CE class, even if it has a lower performance in average.

A noticeable feature of the green class is that almost all of its CEs contain the `lcgpbs` string in their names. In this class, the only CE whose name does not contain this string is plotted in cyan on figure 6.7 and is close to the border of this class. In the blue class, no CE contains this string in its name and in the slowest class, 7 CEs have this string in their name. It shows that the `lcgpbs` string name is informative in itself although the reasons are not necessarily known (it may correspond to a specific middleware version deployed on some of the CEs in this heterogeneous infrastructure).

Figure 6.8 displays the fitting of the mixed log-normal / Pareto model of equation 6.9 on the centroids of the 3 classes identified on figure 6.7 and table 6.4 shows the corresponding parameters of the model. This model is well fitted to the blue and green classes but is not so convincing for the red one. Actually, the CEs of the red class seem to be outliers: they

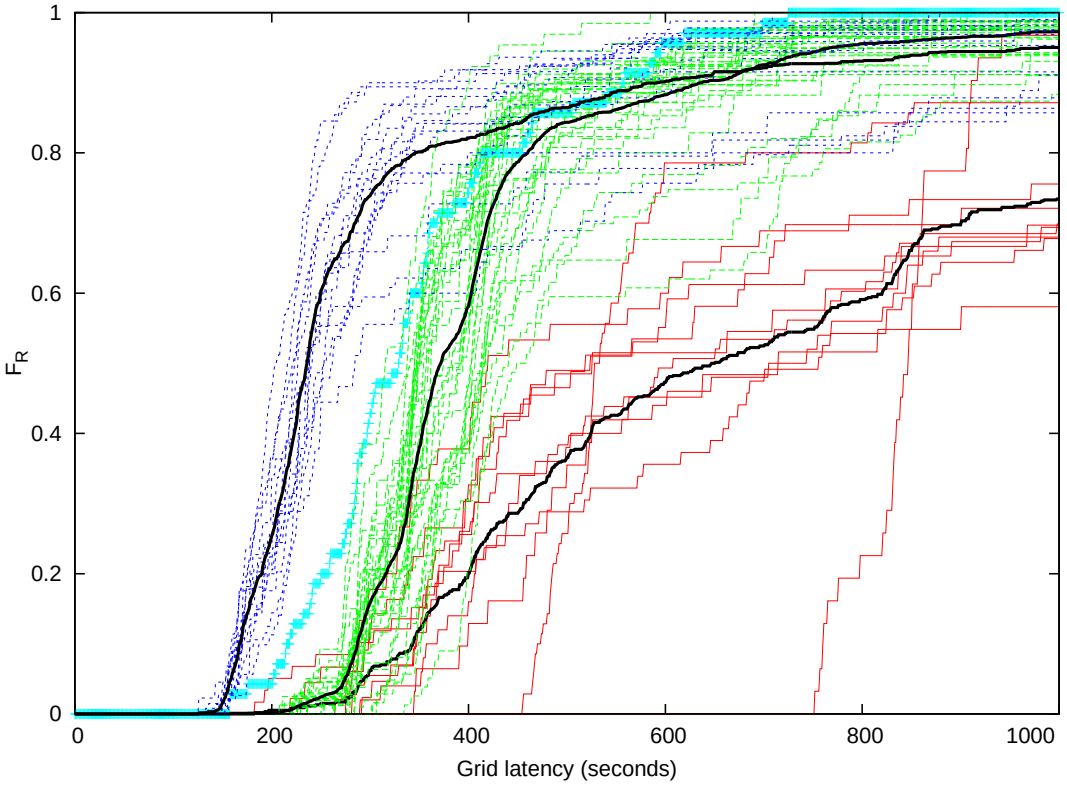

Figure 6.7: Classification in 3 classes of the cumulative density functions of the grid latencies by CE. Centroids of the k-means classes are plotted in black.

exhibit very poor average performance (expected latency close to 20 minutes) with a very high variability (the standard-deviation is more than 20 minutes). A dedicated model should be determined for such poor-performance CEs.

CE group	Median (s)	Expect. (s)	Stdev (s)
not lcgpbs (blue)	237	436	880
lcgpbs (green)	373	461	493
other (red)	652	1132	1396
Whole data	363	559	850

Table 6.3: First moments and median of the grid latency with respect to the execution CE class

The order of magnitude of the grid latency appears to be correlated to the execution CE. It is relevant because the CE is directly related to the job queuing time as a CE exactly corresponds to a batch queue. Variations of middleware and system versions may explain the differences observed among the 3 different classes while variations inside a given class may be coming from the load imposed by the users and the performance of CEs host hardware.

However, in general, the execution CE is only known after the job submission, during the

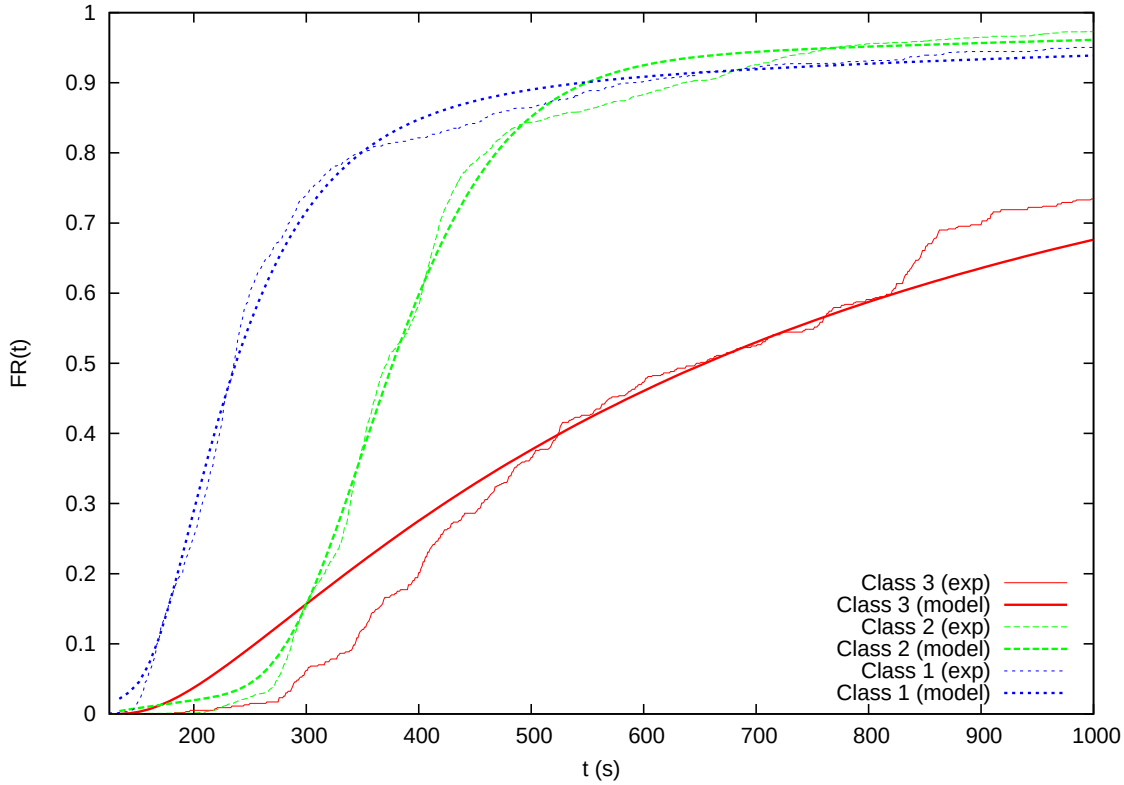

Figure 6.8: Fit of the model of equation 6.9 on the centroids of the 3 classes identified on figure 6.7

	k	μ (s)	σ (s)	a (s)	ν
Class 1 (blue)	0.38	4.7	0.63	663.5	2.04
Class 2 (green)	0.57	5.53	0.32	9306.7	18.24
Class 3 (red)	11.8	6.27	1.1	70.4	6.47

Table 6.4: Parameters of the model of equation 6.9 fitted on the centroids of the 3 classes identified on figure 6.7

scheduling procedure. Thus, this information could only be exploited for parameters that can be updated once the job has been submitted, as for instance the timeout value or the application completion prediction date, whereas parameters such as the granularity of the tasks to submit could not benefit from the CE information.

6.2.2.2 Resource Broker parameter

The probe measures were submitted to 3 different Resource Brokers (RBs). Figure 6.9 displays the cumulative density function of the submission time of the probe jobs sent to each of the RBs as well as the one of the submission time considering the whole experimental data set. First, the submission times seem to be quantified to a discrete set of values that correspond to the ones

Resource Broker	Expectation (s)	Stdev (s)
IFCA (red)	19	14
LAL (green)	25	24
SINP (blue)	22	16
Whole data	22	19

Table 6.5: First moments of the submission time with respect to the RB

were the cumulative density function is growing. As every curve correspond to more than 1400 probe measures, this phenomenon does not come from the lack of measures but rather from a characteristic of the submission system. Indeed, to ensure scalability, jobs are sequentially submitted to the RB, which could explain this behavior. The submission time is a multiple of the duration required to submit one job, which is about 4 seconds according to those measures.

The 3 RBs exhibit quite different behaviors. Two of them (red and blue curves) have equivalent tails that are smaller than the one of the third RB (green curve). It indicates that the latter RB is prone to have very high submission delays: on this RB, 30% of the jobs require more than 40 seconds to be submitted, whereas they are less than 10% for the two other RBs. On the other hand, many of the jobs of the green RB are submitted faster than on the two other ones. As a consequence, the median of the submission time on the green RB is only 12 seconds, whereas it is respectively between 19 and 20 seconds and between 15 and 16 seconds on the blue and red RBs.

Table 6.5 displays the expectation and standard-deviation of the submission time with respect to the RB. Knowing that a job is submitted to the red or blue RB reduces the variance of the submission time distribution. On the contrary, the standard-deviation of the green RB is higher than the one of the whole data.

6.2.2.3 Day of the week parameter

The day of the week is an important parameter of the external context which is likely to influence the load of the grid infrastructure. Figure 6.10 plots the cumulative density function of the grid latency with respect to the day of the week and table 6.6 displays the corresponding expectations and standard-deviations. For this experiment, 1364 probes submitted during the week-end were added to the previous 4477 ones.

The seven days exhibit similar behaviors for latencies lower than 500 seconds. Above this value, Saturday and Sunday have very similar cdf significantly lower than the ones of the other days. In average, those week-end days correspond to the ones when the latency is the highest, as shown by table 6.6. The variance also seems to be higher during the week-end than during the week.

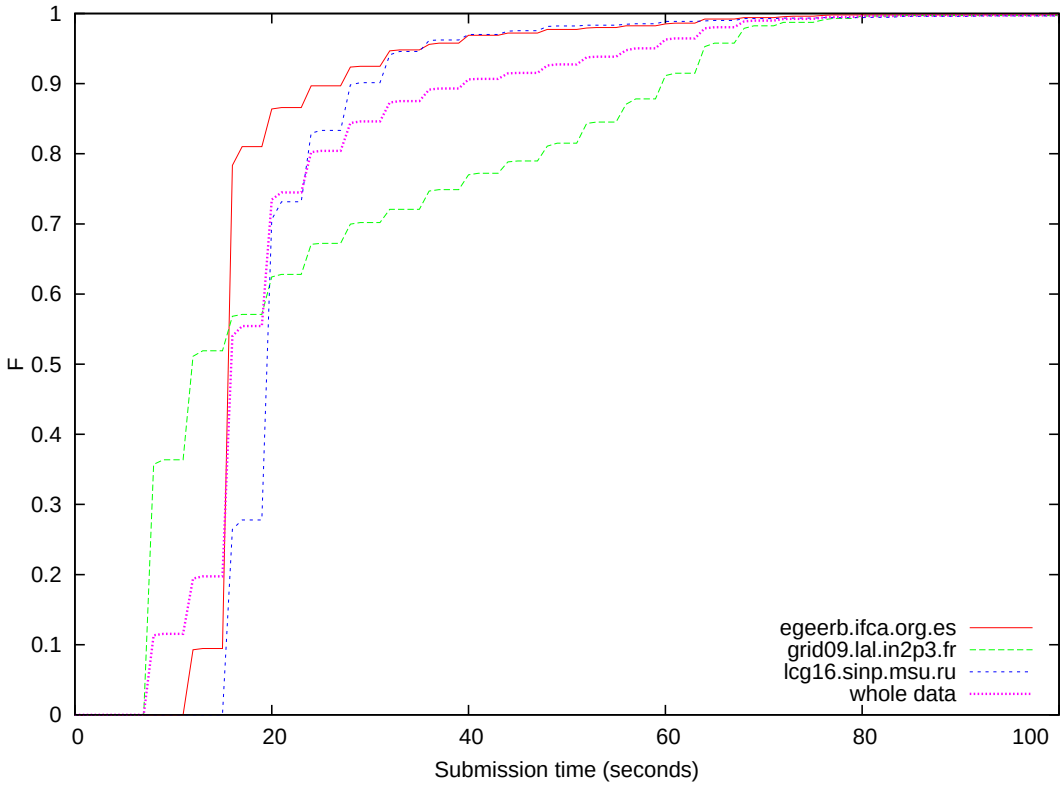

Figure 6.9: Cumulative density functions of the submission time by Resource Broker

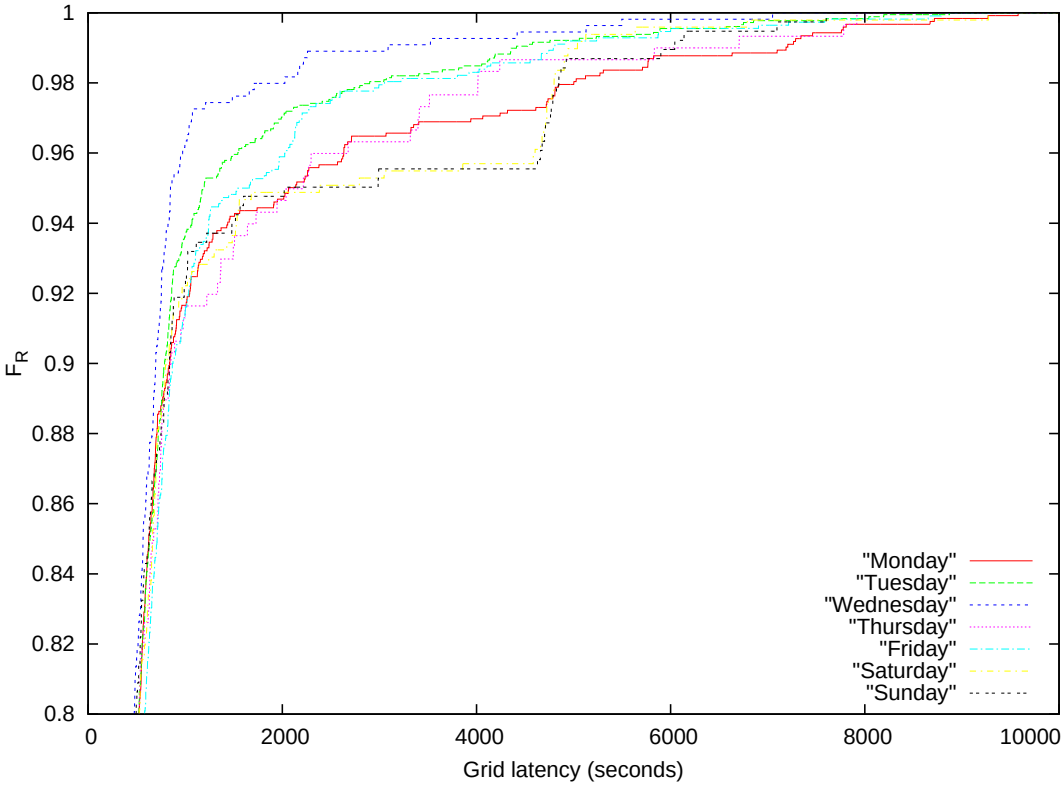

Figure 6.10: Cumulative density functions of the grid latency for each day of the week

Day	Expectation (s)	Standard-deviation (s)
Monday	622	1088
Tuesday	530	747
Wednesday	461	542
Thursday	609	972
Friday	569	808
Saturday	630	1035
Sunday	629	1066
Whole data	569	886

Table 6.6: First moments of the grid latency with respect to the day of the week

6.2.2.4 Discussion

These experiments revealed that the grid latency is related to the choice of a RB or a CE. More precisely, the middleware and system versions are probably involved in this phenomenon. Computers with older systems and middlewares are probably computers that were installed before newer ones, and not upgraded. The differences can either come from software performance improvement or the fact that newer computers have higher computing capabilities. This hypothesis could be confirmed by other experiments establishing what fraction of the latency is due to the computer hardware or to its software. It might be a valuable information for middleware developers. However, from our grid user point of view, the main interest is not necessarily the cause but rather its impact on the applications. A similar approach led Cieslak and co-authors [Cieslak et al., 2006] to propose performance analysis through grid log data mining. This can be very efficient in identifying point of failures or performance drops although it usually provides little information on their cause.

The last experiment made in this section shows also an interesting result: days from Monday to Friday are usually accepted as working days while Saturdays and Sundays are usually non-working days. This is the case for most western and eastern countries involved in the EGEE project. However, this assumption fails in some participating countries (for instance, in Algeria the week-end is on Thursday-Friday and in Israel on Friday-Saturday). This information on working days would thus need to be corrected by the geographical location of the grid sites handling the jobs. Working on such a large geographical area also implies to consider the time of the day. Working hours depend on the country we are dealing with and local habits. The dependency between latency and day of the week could be refined considering:

- Local meaning of week-end (*e.g.* Saturday/Sunday or Thursday/Friday).
- Local time of the day (day or night).
- Time zone: days start with significant time shifts in the EGEE infrastructure (from GMT+9 in Japan to GMT-8 in the USA).

- Local habits (*e.g.* working hours).

The dependency between latency and day of the week may be related to the system administrators activity (they are more frequently at work and system or services crashes are more rapidly fixed on week days). However, the fact that there is more activity during the week than during week-end, generating probably more faults should also be considered. These hypotheses need to be further tested by building a notion of time context with respect to time zones, working days and hours.

Similarly, a correlation between temperature and faults in southern countries may be investigated: in summer, air conditioning systems cooling down computing centers are more likely to break down, making large amounts of local resources unexpectedly unavailable. CPU's temperature is certainly the most accurate parameter to demonstrate this fact but is often difficult to obtain remotely. Considering cities temperatures could also gives indications on failures probability. This information is easily obtained for large cities through well known Web-Services.

Correlations between job latencies and parameters from the execution context such as the Resource Broker and batch systems involved in jobs management, or the week of the day have been demonstrated. These results encourage to perform more detailed studies in order to have a better understanding of the influence of these parameters. Depending on their influence and availability, they can be used to refine the model of job latency and thus to provide a better basis for the latency reduction strategies presented in part III.

6.3 Handling variability in grid models

The work presented in this chapter covers two different areas: the probabilistic modeling of workflows and the statistical parameters estimation of production grid systems. We review below the main contributions of the literature concerning those two aspects. The probabilistic modeling of applications has been investigated for quite a long time. However, the sources of variability were not the same and the application areas thus significantly differed from ours here. Consequently, statistical investigations about grid systems have only been introduced in the last years. A broad survey of such methods is reported in Feitelson's on going book[1] which synthesizes many of its papers [Feitelson, 2002, Feitelson, 2003]. Yet, as far as we know, such methods have only been introduced from the infrastructure's point of view so far. For instance, statistical attempts have been done to model the job inter arrival time of a cluster of the grid. The idea of considering the whole grid as a black box introducing a random latency on the jobs submitted by the user is original and leads to new parameters optimization methods in grid computing (see chapters 8 and 9).

[1] http://www.cs.huji.ac.il/~feit/wlmod/

6.3.1 Probabilistic approaches for application modeling

Probabilistic approaches to performance analysis have been used for quite a long time in parallel and distributed applications. Gelenbe *et al* [Gelenbe et al., 1986] and Mussi and Nain [Mussi and Nain, 1984] already considered the execution time of a task-graph as a random variable and determined its distribution from the graph parameters and topology. Sequential compositions are modeled as convolutions of the density function and parallel ones lead to exponentiation, as done in this chapter. They then determine the distribution of the execution time of the graph from the known ones of the tasks. Even if the motivating problem of those works is very different from ours (in [Gelenbe et al., 1986], the variability is related to the topology of the task graph and in [Mussi and Nain, 1984], only task trees are considered), the probabilistic tools employed are very similar, reinforcing the idea that they are adequate to model this kind of problem.

Later on, Gautama *et al* [Gautama, 1998] noticed that directly using the pdf to determine the execution time of the application leads to heavy computations preventing from any practical application. They thus propose an approach based on the four first moments of the distribution. The moments of the execution time of the application are expressed from moments of the tasks in the graph and from the graph topology, including the sequential operator, conditional branching and parallel composition. They also take into account more complex program patterns including for example random loops bounds which are difficult to model directly using the pdf. However, parallel operators raise problems in this framework because there is no relation linking the moments of the random variable $\max\{X_1, \ldots, X_n\}$ (which is the most common parallel pattern) to the ones of the X_i in the general case. In this case, the authors thus approximate density functions with generalized lambda distributions, characterized by four parameters only [Gautama and van Gemund, 2003]. Assuming that, the moments of the execution time of the graph are expressed from the ones of the tasks. Results concerning normal distributions show that the error made by the approximation remains under 1% for 1000 parallel tasks. However, only low mean and standard deviation values are presented due to numerical instabilities.

Close to this approach, Schopf and Berman use stochastic values, defined by their mean and standard deviation to model the execution time of an application [Schopf and Berman, 1998, Schopf and Berman, 2001]. They define arithmetic operations on them that comes from the arithmetic on normal distributions. As in Gautama's work, the definition of the max operation, that is critical in a parallel execution is not obvious and has to be "supplied by the model builder, scheduler or user". The application model presented in this work seems to be quite specific whereas using a workflow representation allows us to describe any workflow-based application in a more generic way.

Works such as [Manolache et al., 2001] and inside references propose performance analysis methods for task scheduling into embedded systems, considering probabilistic models of task execution times. In this work, the authors model task execution by a generalized contin-

uous probability distribution and propose a method not restricted to any specific scheduling policy. They consider both execution time and memory aspects. Their method is based on the construction of an underlying stochastic process and its analysis. Even if this approach is entirely probabilistic and makes no assumption on the nature of the probability function of the execution time, which well suits with our hypotheses, they assume all the tasks to be executed concurrently on a single processor.

In practice, the probabilistic approaches mentioned in the previous paragraphs have never been applied to production grid infrastructures at the scale we are demonstrating here. Even the recent work of Schopf and Berman described above exhibits very different orders of magnitude to ours. Results are showed on a cluster environment whereas the EGEE grid on which we conducted our experiments is much wider. Consequently, variability in [Schopf and Berman, 2001] is about 100 seconds whereas it can reach 900 seconds in our case. In our case, variability is related to the grid latency itself, which does not occur in such proportion on smaller platforms.

General considerations about features and architecture required for an efficient production grid (particularly focusing on data transfers) are discussed in [Laure et al., 2005] from the experience of the EU DataGrid project. This work focus on the large-scale multi-users grid that we are also targeting here. However, no detailed model to explain how the infrastructure behaves is proposed.

6.3.2 Statistical parameters estimation of grid systems

Several initiatives aim at modeling workload management systems. In [Li et al., 2004], correlations between job execution characteristics (job size or number of processors requested, job runtime and memory used) are studied on a multi-cluster supercomputer in order to build models of workloads, enabling comparative study on system design and scheduling strategies. Feitelson [Feitelson, 2002] has observed correlations between runtime and job size, number of cluster and time of the day.

In [Medernach, 2005], the author analyzes the usage of a cluster of the EGEE infrastructure. He studies several of the job parameters, such as the running, waiting and arrival times. A Markov-chains based model is then proposed and explains the observed data. Our approach is similar to those ones in the sense that a deterministic modeling of the studied parameters is not investigated. Yet, the adopted point of view is significantly different. Whereas those works focus on the infrastructure's point of view (and even on a particular cluster of the infrastructure) in order to provide realistic workloads modellings, we stand from the user's point of view, trying to model the global behavior of the grid. Consequently, the optimized parameters resulting from those studies may be quite different, concerning for instance particular configurations of the local batch schedulers of the clusters.

6.4 Conclusions

Based on the observations made in the previous chapter, we proposed here a model of the workflow of the application taking into account the variability of the grid latency. The originality of this model lie in the fact that the global behavior of the grid is modeled by a single random variable: the grid is viewed as a black box introducing a random latency on the jobs submitted by the user. This model is used to quantify the impact of the latency variability, which is shown to lead to a factor 2 performance drop on the workflow of the bronze standard application. Thus, strategies have to be developed in order to reduce the impact of the latency and of its variability on EGEE. This is the goal of the next part of this thesis.

In order to be predictive, such a probabilistic model has to rely on accurate statistical estimates of the latency distribution. We investigated such models in the second section of this chapter. Experiments demonstrate the relevance of a heavy-tailed distribution for modeling the grid latency in its normal functioning mode, with an outlier ratio capturing large latencies coming from system faults. In particular, a mixed log-normal/Pareto model has been shown to correctly fit to the measurement presented here and the influence of the computing element, resource broker and day of the week has been studied. Such a model is a rationale for the distinction between the tail weights that will be made in chapter 8.

Part III

E

In the previous part of this manuscript, the grid latency and its variability have been shown to drastically reduce the performance of a workflow running on a production grid. For instance, the performance gain that could be expected on the bronze standard application by reducing the impact of the latency variability is in the order of a factor 2. Outliers (*i.e* jobs whose latency can be considered as infinite) may also dramatically disturb applications and strategies have to be studied to reduce the risk of facing them and to deal with the inevitable ones. In this part, we investigate methods to achieve such performance improvements.

In the literature, some strategies aim at reducing the impact of the latency by pre-allocating resources with dedicated agents before the execution of the application and directly connecting to them (bypassing the middleware) when a job needs to be submitted [Garonne et al., 2004, Germain et al., 2005]. With such strategies, the latency penalty is actually shifted before the execution: reservation agents need to be launched sufficiently early before the execution (*i.e* at least one latency duration before), which may not be realistic for all users. Even if reasonable schemes can be set up, such reservations exploit a kind of middleware flaws as they keep resources busy for some time prior to the execution without computing anything. If a large number of users adopt them, then one could expect that the average wall-time of the jobs would grow, leading the grid latency duration to increase too. The global performance of the system would thus degrade, requiring the users to submit reservation agents even earlier and finally entering a vicious cycle. Besides, such pre-allocation also requires to forecast the amount of submitted jobs, which is not always possible in particular in functional or service workflows (see chapter 2). Yet, in some specific cases (in particular for applications involving a few jobs with a high priority), this kind of strategies would constitute an interesting alternate to the ones presented in this part.

In the following chapters, three strategies are envisaged to reduce the impact of the latency and outliers. Service grouping (chapter 7) and granularity optimization (chapter 9) act at the workflow level. They are both based on a reduction of the total number of jobs submitted by the application. The basic idea behind them is that the fewer the number of submitted jobs, the lower the probability to face high latencies. The timeout optimization (chapter 8) corresponds to a parameter optimization at the job level only. Its main interest is to efficiently deal with outliers. All those strategies but the service grouping are based on probabilistic models, assuming that the job latency is a random variable whose normal functioning mode is described by a probabilistic distribution and that may face outliers with a non null probability. They are a direct consequence of the probabilistic approach adopted in chapter 6.

Chapter 7

Service grouping

Contents

This chapter studies service grouping as a strategy to reduce the impact of the grid latency on the execution of a workflow. In a Service-Oriented Architecture (SOA, see chapter 2), grouping services is not possible because they are black boxes only exposing an implementation-independent interface to the outer world. Based on a dynamic wrapper, a service factory is presented and fulfills the conditions required to enable service grouping while still respecting the SOA principles. The grouping strategy itself is then described: it ensures that service grouping will not slow-down the workflow execution by breaking any kind of parallelism. Finally, experiments on the EGEE production grid are presented to evaluate the impact of this optimization on the execution time of a workflow.

Dans ce chapitre, nous étudions le groupement de services pour réduire l'impact de la latence de la grille sur l'exécution d'un flot de traitements. Dans une architecture orientée service (SOA, cf. chapitre 2), grouper les services n'est pas possible car ce sont des boîtes noires qui n'exposent qu'une interface indépendante de l'implémentation au monde extérieur. Une usine à services basée sur un procédé d'encapsulation dynamique est présentée et permet de mettre en œuvre le groupement de services tout en respectant les principes de SOA. La stratégie de groupement elle-même est ensuite décrite : elle assure que le groupement ne ralentira pas l'exécution du workflow en limitant le parallélisme. Enfin, des expériences sont présentées sur la grille de production EGEE pour évaluer l'impact de cette optimisation sur le temps d'exécution d'un flot de traitements.

7.1 Service grouping optimization strategy

On the one hand, grouping services of a workflow may reduce the total encountered latency by reducing the number of submitted jobs required to run the application. Consider for instance the simple workflow represented on the left side of figure 7.1. On top, services P_1 and P_2 are invoked independently. Data transfers are handled by each service and the connection between the output of P_1 and the input of P_2 is handled at the workflow engine level. On the bottom, P_1 and P_2 are grouped into a virtual single service. This service is capable of sequentially invoking the code embedded in both services, thus resolving the data transfer and independent code invocation issues.

On the other hand, grouping services may also reduce workflow parallelism (described in chapter 4) and we have to take care of the grouping strategy in order to avoid performance losses. In particular, grouping sequentially linked services is interesting because they do not benefit from any workflow parallelism. Those groupings can be done at the services level, *i.e* they will be available for each data item processed by the workflow. For example, considering the computational part of the workflow of the bronze standard application introduced in chapter 3 and recalled on figure 7.2, services `crestLines` and `crestMatch` can be grouped without parallelism loss as well as services `PFMatchICP` and `PFRegister`.

From the middleware point of view, grouping strategies may also be interesting because it reduces the total number of jobs to handle, thus decreasing the global load imposed on the infrastructure. Yet, grouping services leads to the submission of longer jobs, which may also increase the average queuing time as a damaging side effect. Consistently with the approach adopted in this thesis, we will not try to model the behavior of the middleware in order to be able to predict the impact of those side-effects. Rather, we will focus on the global system's behavior perceived by the grid end-user.

In practice, implementing service grouping is not straight forward given that:

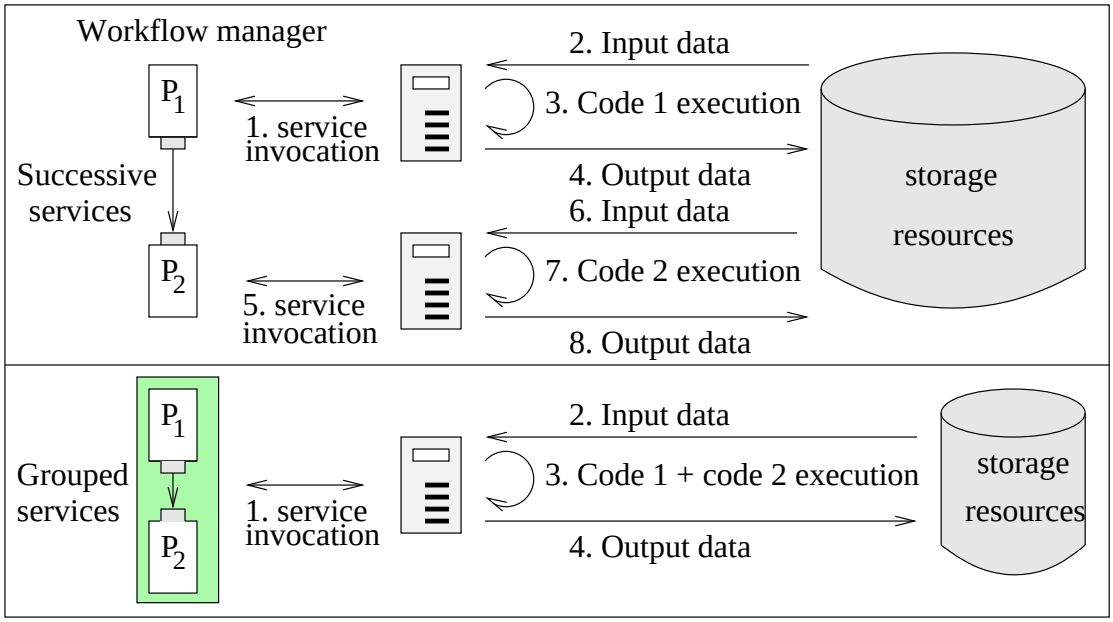

Figure 7.1: Classical services invocation (top) and service grouping (bottom).

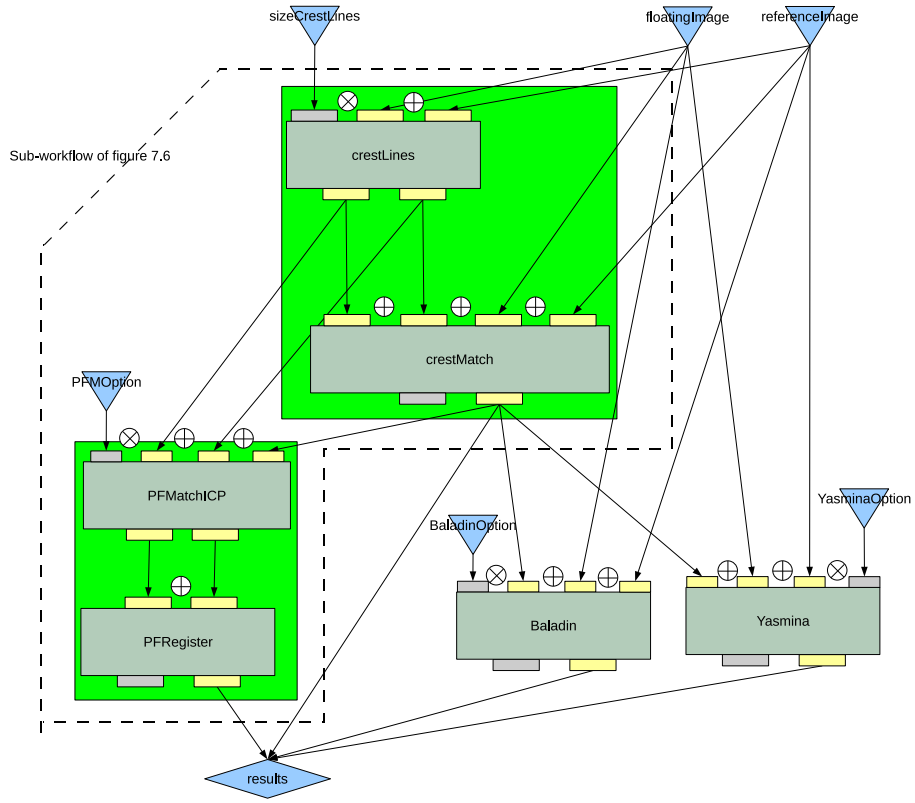

Figure 7.2: Workflow of the application. Services to be grouped are squared in green. The extracted sub-workflow is grouped into a single service, as detailed on figure 7.6.

1. The services composing the workflow are totally independent from each other: as explained in chapter 2, services are black boxes that only publish implementation-independent interfaces. Therefore, the workflow engine cannot access the details of the jobs submitted by the services.

2. The grid infrastructure handling the jobs does not have any information concerning the workflow and the job dependencies. The grouping cannot be handled at this level. Workflow learning solutions could eventually been developed, to have the middleware detect dependencies between jobs from historical information (as done for instance in [Shao et al., 2007]) but this would provide a middleware-specific service grouping, which is not suitable.

Thus, a specific architecture has to be designed to allow services grouping. In order to cope with the first problem described above, we propose below a Generic Application Service Wrapper (GASW).

7.1.1 The Grid Application Service Wrapper

We developed a generic grid submission Web-Service. This service is generic in the sense that it is unique and it does not depend on the executable code to submit. It exposes a standard interface that can be used by any Web-Service compliant client to trigger job submissions. It completely hides the grid infrastructure from the end user as it takes care of the interaction with the grid middleware.

To accommodate to any executable at runtime, this Generic Application Service Wrapper (GASW) is taking two different inputs: a descriptor of the executable command line format, and the input parameters and data of this executable. The production of the legacy code descriptor is the only extra work required from the application developer. It is a simple XML file which describes the legacy executable location, command line parameters and input / output data.

The main difference between this wrapper and related solutions (described in section 7.1.1.1) is that it is able to dynamically wrap code at runtime, thus allowing optimization strategies as the service grouping presented in this chapter. Indeed, as the workflow enactor has access to the descriptors of the executables, it is able to dynamically create a virtual service, composing the command lines of the codes to be invoked, and submitting a single job corresponding to a sequence of command lines invocations.

Legacy code descriptor. The command line description has to be complete enough to allow dynamic composition of the command line from the list of parameters at the service invocation time and to access the executable and input data files. As a consequence, the executable descriptor contains:

1. The name and access method of the executable. In our current implementation, access methods can be a URL or a Logical File Name (LFN). The wrapper is responsible for fetching the data according to those different access modes.
2. The access method and command-line option of the input data. The actual name of the input data files is not mandatory in the description. Those values will be defined at the execution time. This feature differs from various job description languages used in the task-based middlewares. The command-line option allows the service to dynamically build the actual command-line at the execution time.
3. The command-line option of the input parameters: parameters are values of the command-line that are not files and which do not have any access method.
4. The access method and command-line option of the output data. This information enables the service to register the output data in a suitable place after the execution. Here again, names of output data files cannot be statically determined because output file names are only generated at execution time.
5. The name and access method of the sandboxed files. Sandboxed files are external files such as dynamic libraries or scripts that may be needed for the execution although they do not appear on the command-line.

The wrapper is then able to build a dedicated job with the input data items provided at runtime and to submit and monitor it to the grid. It enables a complete decoupling of the grid concerns from the service providers (*e.g* the medical image analysis scientists) and the workflow users (*e.g* the clinicians). Ideally, this wrapper would be maintained by a grid "expert" who would configure and update it according to the middleware status and evolutions. The service providers would just release the descriptor of their codes that could be embedded in a workflow thanks to the Web-Service standard.

Example. An example of a legacy code description file is presented in figure 7.3. It corresponds to the description of the executable `crestLines` which is part of the bronze standard application (see its workflow in chapter 2). It describes the script `CrestLines.pl` which is available from the server `legacy.code.fr` and takes 3 input arguments: 2 files (options `-im1` and `-im2` of the command-line) that are already registered on the grid as LFNs at execution time and 1 parameter (option `-s` of the command-line). It produces 2 files that will be registered on the grid. It also requires 3 sandboxed files that are available from the same web server as the executable.

7.1.1.1 Comparison with related systems

Here, we briefly review systems that are used to wrap legacy code into services to be embedded in workflows.

```
<description>
  <executable name="CrestLines.pl">
    <access type="URL">
      <path value="http://legacy.code.fr"/>
    </access>
    <value value="CrestLines.pl"/>
    <input name="floating_image" option="-im1">
      <access type="LFN"/>
    </input>
    <input name="reference_image" option="-im2">
      <access type="LFN"/>
    </input>
    <input name="scale" option="-s"/>
    <output name="crest_reference" option="-c1">
      <access type="LFN"/>
    </output>
    <output name="crest_floating" option="-c2">
      <access type="LFN"/>
    </output>
    <sandbox name="convert8bits">
      <access type="URL">
        <path value="http://legacy.code.fr"/>
      </access>
      <value value="Convert8bits.pl"/>
    </sandbox>
    <sandbox name="copy">
      <access type="URL">
        <path value="http://legacy.code.fr"/>
      </access>
      <value value="copy"/>
    </sandbox>
    <sandbox name="cmatch">
      <access type="URL">
        <path value="http://legacy.code.fr"/>
      </access>
      <value value="cmatch"/>
    </sandbox>
  </executable>
</description>
```

Figure 7.3: Legacy code descriptor example for the Generic Application Service Wrapper. The location of the executable is first described. Then, inputs and outputs participating in the command-line generation are specified. Finally, external dependencies (such as dynamic libraries) are described in the sandbox section.

The Java Native Interface (JNI) has been widely adopted for the wrapping of legacy codes into services. Wrappers have been developed to automate this process. In [Huang et al., 2003], an automatic JNI-based wrapper of C code into Java and the corresponding type mapper with Triana [Taylor et al., 2005] is presented: JACAW generates all the necessary java and C files from a C header file and compiles them. A coupled tool, MEDLI, then maps the types of the obtained Java native method to Triana types, thus enabling the use of the legacy code into this workflow manager. Related to the ICENI workflow manager [Furmento et al., 2002], the wrapper presented in [Li et al., 2005] is based on code reengineering. It identifies distinct components from a code analysis, wrap them using JNI and adds a specific CXML interface layer to be plugged into an ICENI workflow.

The WSPeer framework [Harrison and Taylor, 2005], interfaced with Triana, aims at easing the deployment of Web-Services by exposing many of them at a single endpoint. It differs from a container approach by giving to the application the control over service invocation. The Soaplab system [Senger et al., 2003] is especially dedicated to the wrapping of command-line tools into Web-Services. It has been largely used to integrate bioinformatics executables in workflows with Taverna [Oinn et al., 2004]. It is able to deploy a Web-Service in a container, starting from the description of a command-line tool. This command-line description, referred to as the metadata of the analysis, is written for each application using the ACD text format file and then converted into a corresponding XML format. Among domain specific descriptions, the authors underline that such a command-line description format must include (i) the description of the executable, (ii) the names and types of the input data and parameters and (iii) the names and types of the resulting output data. As described latter, the format we used includes those features and adds new ones to cope with requirements of the execution of legacy code on grids.

The GEMLCA environment [Delaitre et al., 2005] addresses the problem of exposing legacy code command-line programs as Grid services. It is interfaced with the P-GRADE portal workflow manager [Kacsuk et al., 2003]. The command-line tool is described with the LCID (Legacy Code Interface Description) format which contains (i) a description of the executable, (ii) the name and binary file of the legacy code to execute and (iii) the name, nature (input or output), order, mandatory, file or command line, fixed and regular expressions to be used as input validation. A GEMLCA service depends on a set of target resources where the code is going to be executed. Architectures to provide resource brokering and service migration at execution time are presented in [Kecskemeti et al., 2005].

7.1.2 Implementing the grouping with a dynamic service factory

Our above-presented service wrapper separates the algorithm description from the grid details: grouping services is thus made possible by grouping their algorithm descriptions and submitting the resulting description to the wrapper. To do that, the workflow engine can dynamically enable services grouping by analyzing the workflow and generating grouped services on the

fly. An application wrapper factory service is added to the architecture. Its role is to instantiate both the code wrapping services and the grouped services. The complete architecture is diagrammed on figure 7.4 and owes a lot to the ideas and work of David Emsellem. GASW command-line descriptions are called MOTEUR descriptors and services described with it are called MOTEUR services. The MOTEUR factory is responsible for dynamically generating and deploying application services. The aim of this factory is to achieve two antagonist goals:

- To expose codes as autonomous Web-Services.
- To enable the grouping of two of these Web-Services as a unique one for optimizing the execution.

On one hand, the specific Web-Service implementation details (*i.e.* the execution of the wrapped code on a grid infrastructure) are hidden to the consumer. On the other hand, when the consumer is a workflow manager which can group jobs, it needs to be aware of the real nature the Web-Services (the encapsulation of a MOTEUR descriptor) so that it could merge them at run time. We choose to use the WSDL XML Format extension mechanism which allows to insert user defined XML elements in the WSDL content itself. We thus strictly conform to the WSDL standard while enabling our optimization strategy.

On figure 7.4, we exemplify the architecture through a usage scenario:

R.1 First, the legacy code provider registers a MOTEUR XML descriptor P1 to the MOTEUR factory.

G.1 The factory, then dynamically generates a Web-Service which wraps the submission of the legacy code to the grid via the generic service wrapper.

R.2 Another provider do the same with the descriptor of P2.

The resulting Web-Services expose their WSDL contracts to the external world with a specific extension associated with the WSDL operation. For instance, the WSDL contract resulting of the deployment of the `crestLines` legacy code described on figure 7.3 is printed on figure 7.5. This WSDL document defines two types (`CrestLines-request` and `CrestLines-response`) corresponding to the descriptor inputs and outputs and a single `Execute` operation. Notice that in the binding section, the WSDL document contains an extra `MOTEUR-descriptor` tag pointing to the URL of the legacy code descriptor file (`location`) and a binding to the Execute operation (`soap:operation`).

Suppose now that the workflow manager identifies a service grouping optimization (*e.g.* P1 and P2, displayed in green in figure 7.4). Because of its ability to discover the extended nature of these two services, the engine can retrieve the two corresponding MOTEUR descriptors.

C.1+2 The workflow manager can ask the factory to *combine* them and

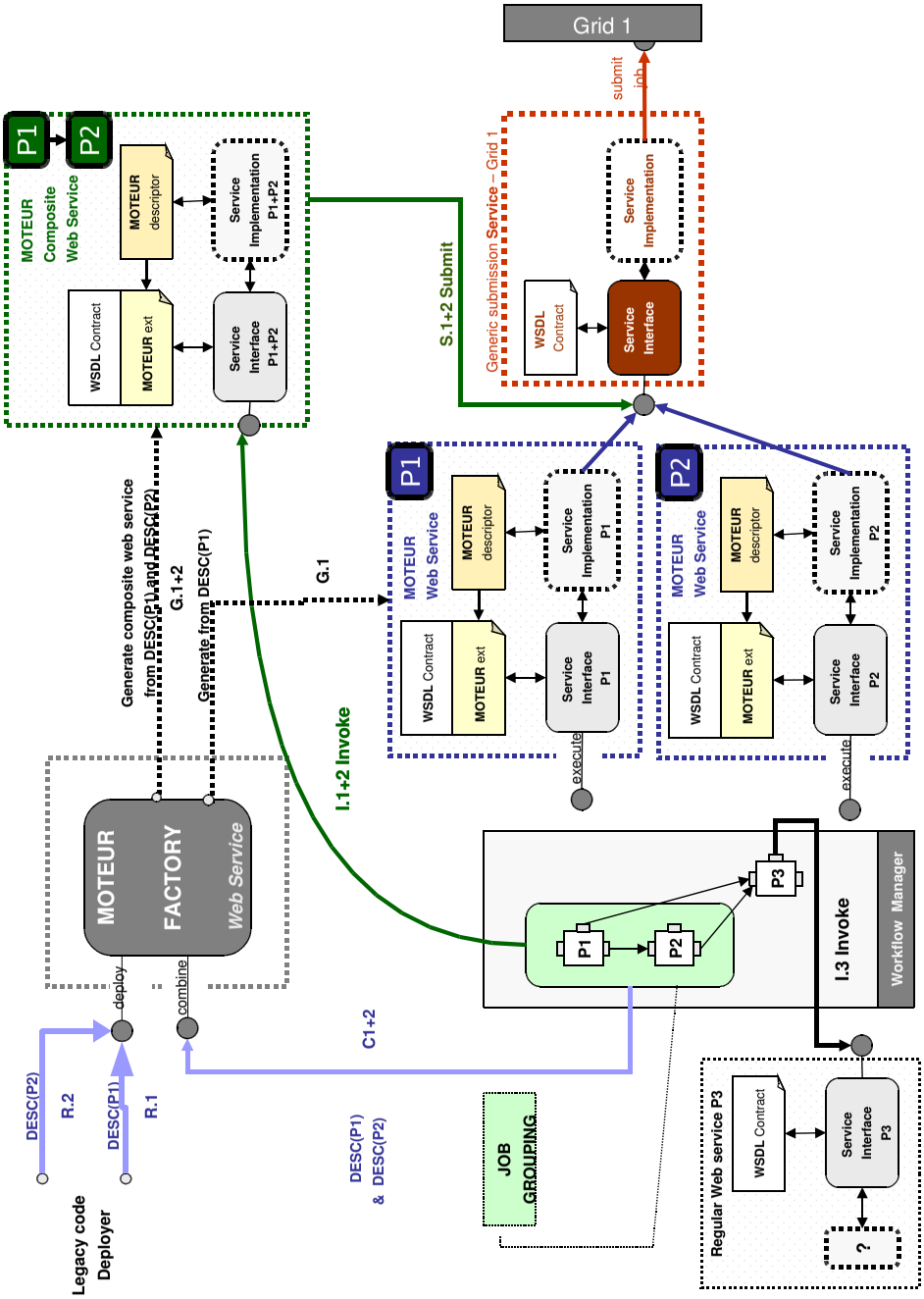

Figure 7.4: Services factory enabling service grouping. The MOTEUR factory is able to deploy a Web-Service from the description of an executable (see figure 7.3 for an example of such a description). To group services, the workflow engine (MOTEUR) dynamically invokes the services factory with the description of the algorithms to group (DESC(P1) and DESC(P2)). The factory then deploys a composite Web-Service P1+P2 that can be directly invoked by the workflow engine.

```
<?xml version="1.0" encoding="utf-8" ?>
<definitions ...>
 <types>
  <schema>
   <element name="CrestLines-request">
    <complexType>
     <sequence>
      <element name="floating_image"
              type="string"... />
      <element name="reference_image"
              type="string"... />
      <element name="scale" type="string"... />
     </sequence>
    </complexType>
   </element>
   <element name="CrestLines-response">
    <complexType>
      <sequence>
        <element name="crest_reference"
                type="string"... />
        <element name="crest_floating"
                type="string"... />
      </sequence>
    </complexType>
   </element>
  </schema>
 </types>
 <message name="ExecuteSoapIn">
  <part name="parameters"
       element="CrestLines.pl-request" />
 </message>
 <message name="ExecuteSoapOut">
  <part name="parameters"
       element="CrestLines.pl-response" />
 </message>
 <portType name="CrestLines.plSoap">
  <operation name="Execute">
   <input message="ExecuteSoapIn" />
   <output message="ExecuteSoapOut" />
  </operation>
 </portType>
 <binding ...>
  <soap:binding transport="http://..." />
   <operation name="Execute">
    <soap:operation soapAction="http://.../Execute"
         style="document" />
     <MOTEUR-descriptor xmlns="urn:...">
      <location>http://...</location>
     </MOTEUR-descriptor>
     ....
   </operation>
 </binding>
</definitions>
```

Figure 7.5: Extended WSDL generated by the factory for the code introduced in figure 7.3

G.1+2 generate a single composite Web-Service which exposes an operation taking its inputs from P1 (and P2 inputs coming from other external services) and returning the outputs defined by P2 (and P1 outputs going to other external services).

I.1+2 The workflow manager can invoke this composite Web-Service. It is of the same type than any regular legacy code wrapping service and it is accessible through the same interface.

S.1+2 It also delegates the grid submission to the generic submission Web-Service by sending the composite MOTEUR descriptor and the input link of P1 and P2 in the workflow.

7.1.3 Grouping strategy

In order to determine a grouping strategy that does not introduce any slow-down, neither from the user point of view, nor from the infrastructure one, we impose the two following constraints:

- The grouping strategy must not limit any kind of parallelism (user point of view) and
- During their execution, jobs cannot communicate with the workflow manager (infrastructure point of view).

The second constraint prevents a job from holding a resource just waiting for one of its ancestor to complete. An implication of this constraint is that if services A and B are grouped together, the results produced by A will only be available once B will complete. Moreover, a workflow may include both MOTEUR Web-Services (*i.e.* services that are able to be grouped) and classical ones, that could not be grouped. Assuming those constraints, we can prove the following rule:

> Let A be a MOTEUR service of the workflow and $\{B_0,...B_n\}$ its children in the service graph. Grouping B_i and A does not lead to any parallelism loss if and only if:
>
> 1. B_i is an ancestor of every B_j for every $i \neq j$ and
> 2. each ancestor C of B_i is an ancestor of A or A itself.

Let us first prove that (1) and (2) are *necessary* conditions to avoid parallelism loss. If (1) is not respected, then there exists a child B_j of A which is not a descendant of B_i. If A and B_i are grouped, then workflow parallelism is broken between B_i and B_j because B_j has to wait for B_i to complete before starting. Similarly, if (2) is not respected, then there exists an ancestor C of B_i that is not an ancestor of A and workflow parallelism is broken between A and C when A and B_i are grouped.

(1) and (2) are also *sufficient* to avoid any parallelism break in the workflow. Let us first notice that grouping services does not break **data parallelism** because this kind of parallelism

only concerns a single service of the workflow. Moreover, **service parallelism** relies on the independence of the processings of two different data segments by two successive services. As service grouping does not prevent B_i from processing a given piece of data while A is processing another one (assuming that data parallelism is not broken, which is the case here), service grouping does not break service parallelism. Thus, we are left to prove that (1) and (2) guarantee that **workflow parallelism** is not broken by grouping A and B_i. (1) guarantees that there is no workflow parallelism between B_i and every B_j. Workflow parallelism is thus likely to concern B_i only for services that are not children of A and thus cannot be broken by grouping A and B_i. Similarly, (2) guarantees that there is no workflow parallelism between A and every other ancestor of B_i. Workflow parallelism is thus likely to concern A only for services that are not ancestors of B_i and thus cannot be broken by grouping A and B_i. ■

Our grouping strategy tests this rule for each MOTEUR service of the workflow. Groups of more than two services may be recursively composed by successive matches of the grouping rule.

The constraints applied by the matching rule are illustrated on three different grouping examples in figure 7.6. This simplified workflow was extracted from the bronze standard workflow (see end of chapter 2 and chapter 5). It is made of 4 MOTEUR services. As it can be seen from the workflow graph, the data dependencies will enforce a sequential execution of these 4 services. It is therefore expected that the four services are grouped in a single one in order to minimize the job submission overhead. On this figure, notations nearby the services correspond to the ones introduced above in the grouping rule. For each of the 3 examples of figure 7.6, the grouping of the two services outlined by a green box is studied:

1. On the left of figure 7.6, the tested MOTEUR service A is `crestLines`. A is connected to the workflow inputs and it has two children: B_0 and B_1. B_0 is a father of B_1 and it only has as single ancestor which is A. Thus, the rule matches: A and B_0 can be grouped. If there were a service C ancestor of B_0 but not of A as represented on the figure, the rule would not match: A and C would have to be executed in parallel before starting B_0. Similarly, if there were a service D child of A but not of B_0, then the rule would not match as the workflow manager would need to communicate results during the execution of the grouped jobs in order to allow workflow parallelism between B_0 and D.
2. In the middle of figure 7.6, the tested service A is now `crestMatch`. A has a single child: B_0. B_0 has two ancestors, A and C. The rule matches because C is an ancestor of A. A and B_0 can then be grouped.
3. On the right of figure 7.6, A is the `PFMatch` service. It has only one child B_0 which only has a single ancestor, A. The rule matches and those services can thus be grouped.

Finally, when A is the `PFRegister` service, the grouping rule does not match because it does not have any child. Note that in this example, the recursive grouping strategy leads to a single

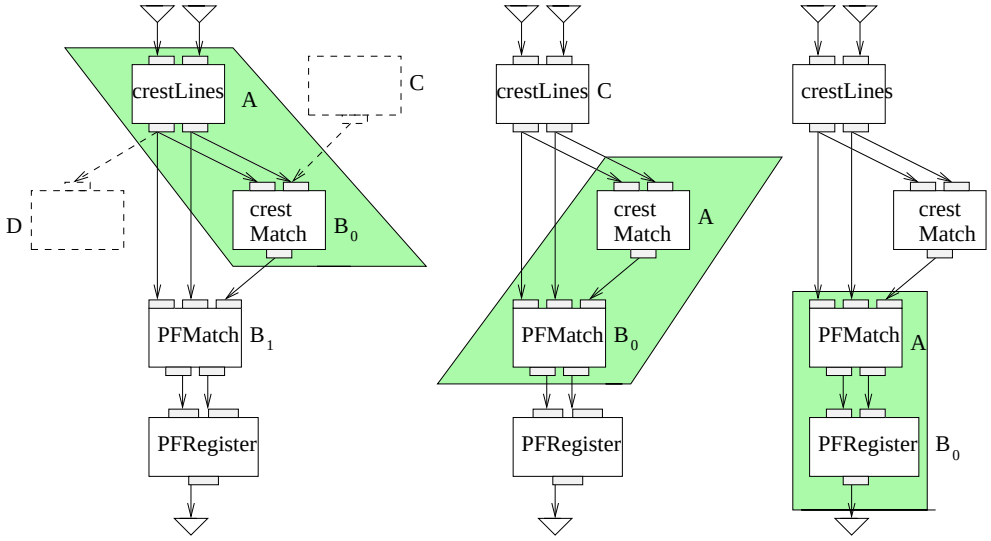

Figure 7.6: Service grouping examples. On this workflow, the grouping rule matches 3 times (once for each green box), thus resulting in a single service wrapping those 4. On the left part of the figure, service C or D would prevent the grouping between `crestLines` and `crestMatch` because it would break workflow parallelism between A and C and between B_0 and D.

job submission, as expected.

7.2 Experiments on the EGEE production grid

To quantify the speed-up introduced by service grouping on a real application workflow, we made experiments on the EGEE production grid infrastructure, using the whole biomed VO (see a description of this infrastructure in chapter 5).

7.2.1 Experimental workflows

First, we study the impact of service grouping on the workflow of the bronze standard application represented on figure 7.2. On this application, the grouping rule matches twice, as represented on the figure. Moreover, to show how service grouping is able to speed-up the execution on highly sequential applications, we also considered a sub-workflow of our application, as shown in figure 7.2 (dash-circled workflow) . It is made of 4 services that correspond to the `crestLines`, `crestMatch`, `PFMatchICP` and `PFRegister` ones in the application workflow. Our grouping rule groups those 4 services of the sub-workflow into a single one, as it has been detailed in the example of figure 7.6. It is important to notice that even if this sub-workflow is sequential, and thus does not benefit from workflow parallelism, its execution on a grid does make sense because of data and service parallelisms. To evaluate the impact of our grouping strategy on the performance, we compared the execution times of those workflows with and

Number of input image pairs	Sub-workflow (figure 7.6)			Whole application (figure 7.2)		
	Number of jobs		Speed-up	Number of jobs		Speed-up
	Regular	Grouping		Regular	Grouping	
12	48	12	2.91	72	48	1.42
66	264	66	1.72	396	264	1.34
126	504	126	2.30	756	504	1.23

Table 7.1: Grouping strategy speed-ups

without the grouping strategy.

7.2.2 Results

Table 7.1 presents the speed-ups induced by the grouping strategy for a growing number of input image pairs and for the two experimental workflows described above. This speed-up is computed as the ratio of a regular grid execution time (where each service invocation leads to a job submission) over the execution time using the grouping strategy. We can notice on this table that service grouping does effectively provide a significant speed-up on the workflow execution. This speed-up is ranging from 1.23 to 2.91.

The speed-up values are greater on the sub-workflow than on the whole application. Indeed, on the sub-workflow, 4 services are grouped into a single one, thus saving 3 job submissions for each input data set. On the whole application workflow, the grouping rule is applied only twice, thus only saving 2 job submissions for each input data set, as depicted on figure 7.2.

7.3 Conclusions

A service grouping rule ensuring that no parallelism is broken inside the workflow has been presented in this chapter. Coupled with a dynamic services factory that enables the implementation of services grouping while respecting the SOA principles, this rule allows a saving-up of job submissions and therefore a reduction of the impact of the grid latency on the workflow. On the workflow of the bronze standard application, speed-ups of 1.2 to 1.4 can be achieved and on a dedicated workflow, speed-up values can reach almost 3.

It is important to notice that this grouping strategy cannot slow down the application because it does not break any parallelism (even if it is true that some side-effects resulting from an increase of the job size may limit the expected speed-up). A next step in service grouping could be to limit parallelism at some point, thus further reducing the number of submitted jobs and the risk to face high latencies. In this case, a compromise would have to be found between parallelism loss and latency reduction. Such a strategy is investigated in chapter 9 where data

parallelism is restricted in order to limit the impact of the latency. Breaking workflow parallelism to reduce the number of submitted jobs may also be envisaged. In this case, a metric to foresee the interest of such a grouping could be provided by the workflow model presented in chapter 6.

Yet, grouping services does not prevent the workflow from facing outliers, which could still be damaging. To avoid them, setting a timeout value to the jobs is mandatory. A method to properly set this timeout value is investigated in the next chapter.

Chapter 8

Optimization of the timeout value

Contents

In this chapter, another solution for reducing the impact of the grid latency is investigated: setting a timeout value and resubmitting abnormally long jobs. The timeout value has to be properly set in order to prevent the job from facing too high latencies or to remain blocked somewhere in the grid because of a system failure. Otherwise, setting a timeout value will at least be useless and could even lead to considerable performance drops if the timeout value is to small and triggers overkilling cancellations and resubmissions. The probabilistic approach introduced in chapter 6 is considered. Through a theoretical study on classical latency distributions, typical behaviors are highlighted: in particular, the importance of the weight of the tail of the distribution of the latency is noticed. Finally, results obtained on an experimental distribution measured on EGEE give an idea of the performance gain that could be expected with such a method in practice.

Dans ce chapitre, une autre solution de réduction de l'impact de la latence est étudiée. Assigner un délai d'expiration aux tâches pour les resoumettre en cas de latence trop importante est une stratégie à double tranchant : d'un côté, si la valeur du délai d'expiration est fixée correctement, les latences excessivement importantes provenant d'une défaillance du système peuvent être évitées. Mais d'un autre, si cette valeur est mal fixée, assigner un délai d'expiration aux tâches sera au mieux inutile et dégradera considérablement les performances si le délai d'expiration est trop faible et conduit à des annulations et ressoumissions de tâches excessives. Une étude théorique des distributions classiques permet de mettre en évidence certaines propriétés comme l'importance du poids de la queue de la distribution de la latence pour l'existence ou non d'une valeur finie du délai d'expiration optimal. Enfin, des résultats obtenus sur une distribution expérimentale mesurée sur EGEE donnent une idée du gain de performance qui peut être espéré en pratique par l'utilisation d'une telle méthode.

Time-outing and resubmitting abnormally long jobs is a common strategy to reduce the impact of latency and outliers. However, choosing the timeout value is often let to the administrator or to the end user. A non trivial trade-off has to be found as a too long timeout will not prevent the job from facing excessively high latencies, while a too short one may be overkilling, causing the unnecessary resubmission of jobs that almost completed. This problem is often encountered when considering unreliable systems and timeout strategies have been designed in areas as different as TCP throughput optimization [Kesselman and Mansour, 2005], HTTP requests [Reinecke et al., 2004, Xie et al., 2002]

or power saving devices [Rong and Pedram, 2006].

From this chapter, two different kind of job definitions will be used. A *user* job will denote the computation that need to be performed by the user. It is composed of one or several *grid jobs* that are the actual jobs that will be submitted on the grid. For instance, in this chapter, the total latency faced by a user job is first modeled in section 8.1 in order to determine the optimal timeout value to set to the grid jobs. Then, section 8.2 presents some results of timeout optimization on classical distributions. To show how the optimization performs on a real infrastructure, the asymptotic behavior of the system and the impact of outliers are particularly studied. Some experimental results from a distribution of the latency measured on the EGEE production grid are finally presented in section 8.3. To facilitate legibility, many of the detailed proofs of the theoretical results are deferred to appendix B. The terms *latency* and *outliers* refer to the definitions already given: the latency is the duration from the job submission instant and the beginning of its execution and an outlier is a job whose latency is largely prevailing on the other ones (the latency faced by outliers is considered as infinite).

8.1 Model of the user job latency taking into account the timeout value

As motivated in chapter 6, a probabilistic modeling of the large-scale workload manager has been adopted. In this section, our goal is to determine the distribution of the latency faced by a user job taking into account timeout and resubmissions with respect to the timeout value and the grid latency. Let J be the total latency faced by a user job (including all its potential resubmissions) and t_∞ be a user defined timeout value. The system is seen as a black box introducing a positive latency R on the grid jobs. Consistently with the approach adopted in chapter 6, R is assumed to be a random variable. The outlier ratio is denoted by ρ. The case $\rho = 0$ corresponds to a reliable cluster management system: faults causing jobs loss are very unlikely (highly reliable LAN, robust schedulers). The case $\rho > 0$ is needed to model grid infrastructures where lower reliability of WANs, scale effects and scheduling errors lead to a significant number of outliers. For instance on the EGEE infrastructure, ρ is in the order of 2% to 3%.

q is the probability for a grid job to timeout. A grid job times-out either if it is an outlier or if it faces a latency which is superior to t_∞. Thus:

$$\begin{aligned} q &= \rho + (1-\rho)P(R>t_\infty) \\ &= 1 - (1-\rho)F_R(t_\infty). \end{aligned} \tag{8.1}$$

If a grid job times-out, then it is canceled and a new one is resubmitted. The cost of canceling a grid job and the resulting system load are very low: they are neglected in this

model. Moreover, the submission time of a resubmitted grid job is part of the grid latency and is included in R. Thus, consecutive submissions can be considered as independent.

Let J_i be the latency faced by a user job from the i^{th} grid job submission to its completion. J_i are independent and identically distributed random variables. They can be recursively defined as:

$$J_i = \begin{cases} R & \text{with probability } 1-q \\ t_\infty + J_{i+1} & \text{with probability } q. \end{cases} \tag{8.2}$$

The goal is to determine the distribution of $J = J_1$, the total latency faced by a user job, including all its resubmissions. The distribution of J has to be determined with respect to the grid latency R (*i.e* the latency faced by a grid job if no timeout is set) and t_∞. J is superior to nt_∞ if and only if n grid jobs timed-out. Thus:

$$P(J > nt_\infty) = q^n \quad \text{so that} \quad P(J < nt_\infty) = 1 - q^n. \tag{8.3}$$

Consequently, the cdf of J is known for every multiple of the timeout value. A complete expression of F_J now has to be obtained. Interpolating F_J in every $[nt_\infty, (n+1)t_\infty]$ is clearly not suitable. Indeed, those intervals can be quite large with respect to the total latency and the interpolation error is likely to produce inconsistent results. It is better to notice that **for all** t **in** $[nt_\infty, (n+1)t_\infty[$:

$$\begin{aligned} F_J(t) &= P(J < t | t \in [nt_\infty, (n+1)t_\infty]) \\ &= P(J < nt_\infty) + P(nt_\infty < J < t \mid t \leq (n+1)t_\infty) \end{aligned}$$

and thus, according to equation 8.3:

$$F_J(t) = 1 - q^n + P(nt_\infty < J < t \mid t \leq (n+1)t_\infty). \tag{8.4}$$

Given that $t \leq (n+1)t_\infty$, a user job latency J is in $[nt_\infty, t]$ if and only if n grid jobs timed-out (probability q^n) and the $(n+1)^{\text{th}}$ one succeeded, *i.e* it was not an outlier (probability $1-\rho$) and $R \leq t - nt_\infty$ (probability $F_R(t - nt_\infty)$). Therefore,

$$P(nt_\infty < J < t \mid t \leq (n+1)t_\infty) = q^n(1-\rho)F_R(t - nt_\infty)$$

We finally get, $\forall t \in [nt_\infty, (n+1)t_\infty[$:

$$F_J(t) = 1 - q^n + q^n(1-\rho)F_R(t - nt_\infty) \quad \text{with} \quad q = 1 - (1-\rho)F_R(t_\infty). \tag{8.5}$$

Given that R>0, it is clear that $F_J(0) = 0$ and $\lim_{t\to+\infty} F_J(t) = 1$ (note that $\lim_{t\to+\infty} n = +\infty$ and thus $\lim_{t\to+\infty} q^n = 0$) so that f_J is a pdf. It is important to notice that because it reveals that the timeout strategy is able to highly reduce the impact of outliers. With a finite timeout value, the probability for the latency J faced by the user job to be infinite is null, whereas without timeout, it is ρ.

Moreover, F_J is continuous at every nt_∞. Indeed, according to equation 8.5, the expression of F_J at the lower bound of the segment $[nt_\infty, (n+1)t_\infty]$ is:

$$F_J(nt_\infty) = 1 - q^n + q^n(1-\rho)F_R(0) = 1 - q^n$$

And at the upper bound of this segment, F_J is:

$$\begin{aligned} F_J((n+1)t_\infty) &= 1 - q^n + q^n(1-\rho)F_R((n+1)t_\infty - nt_\infty) = 1 - q^n + q^n(1-\rho)F_R(t_\infty) \\ &= 1 - q^n + q^n(1-q) \text{ (given equation 8.1)} \\ &= 1 - q^{n+1} \end{aligned}$$

However, in general, F_J is not differentiable in nt_∞.

Note that if $\rho = 0$, then equation 8.5 resumes to:

$$F_J(t) = 1 - q^n + q^n F_R(t - nt_\infty) \quad \text{with} \quad q = 1 - F_R(t_\infty).$$

whereas with outliers, it was:

$$F_J(t) = 1 - q^n + q^n(1-\rho)F_R(t - nt_\infty) \quad \text{with} \quad q = 1 - (1-\rho)F_R(t_\infty).$$

It means that the outlier case can be derived from the reliable case by replacing F_R with $(1-\rho)F_R$ in the expressions of F_J and q. We will use this property to simplify some interpretations in the following of this chapter.

8.1.1 Illustration for a reliable system

If no outlier is present, the choice of a timeout value can be evaluated by comparing the latencies faced by a user job with (J) and without (R) setting a timeout value. Figure 8.1 displays an example of a cdf for R (red curve) and J (green curve). Note the singularities at nt_∞ points. On the upper graph, the distribution of R is Gaussian with mean 300 seconds and standard deviation 100 seconds, truncated above zero to avoid negative latency values. The timeout value is equal to the mean of the original Gaussian (300 seconds). It is of course a very low timeout value leading to many resubmissions. We can graphically notice that for every t, $F_R(t) > F_J(t)$, which means that at any time t, there is a higher probability that $R < t$ than that $J < t$. In this case, it would thus have been better not to set any timeout value as it highly penalizes the execution.

On the other hand, the bottom of figure 8.1 displays an example of a cdf of R and J in a case where the timeout choice improves the execution. The timeout value is still 300 seconds but the distribution of R has a longer tail than in the former example. R is actually log-normal, with μ=15 seconds and σ=10 seconds. In this case, it seems that for every t, $F_R(t) < F_J(t)$, which means that this timeout value reduces the total latency faced by a user job.

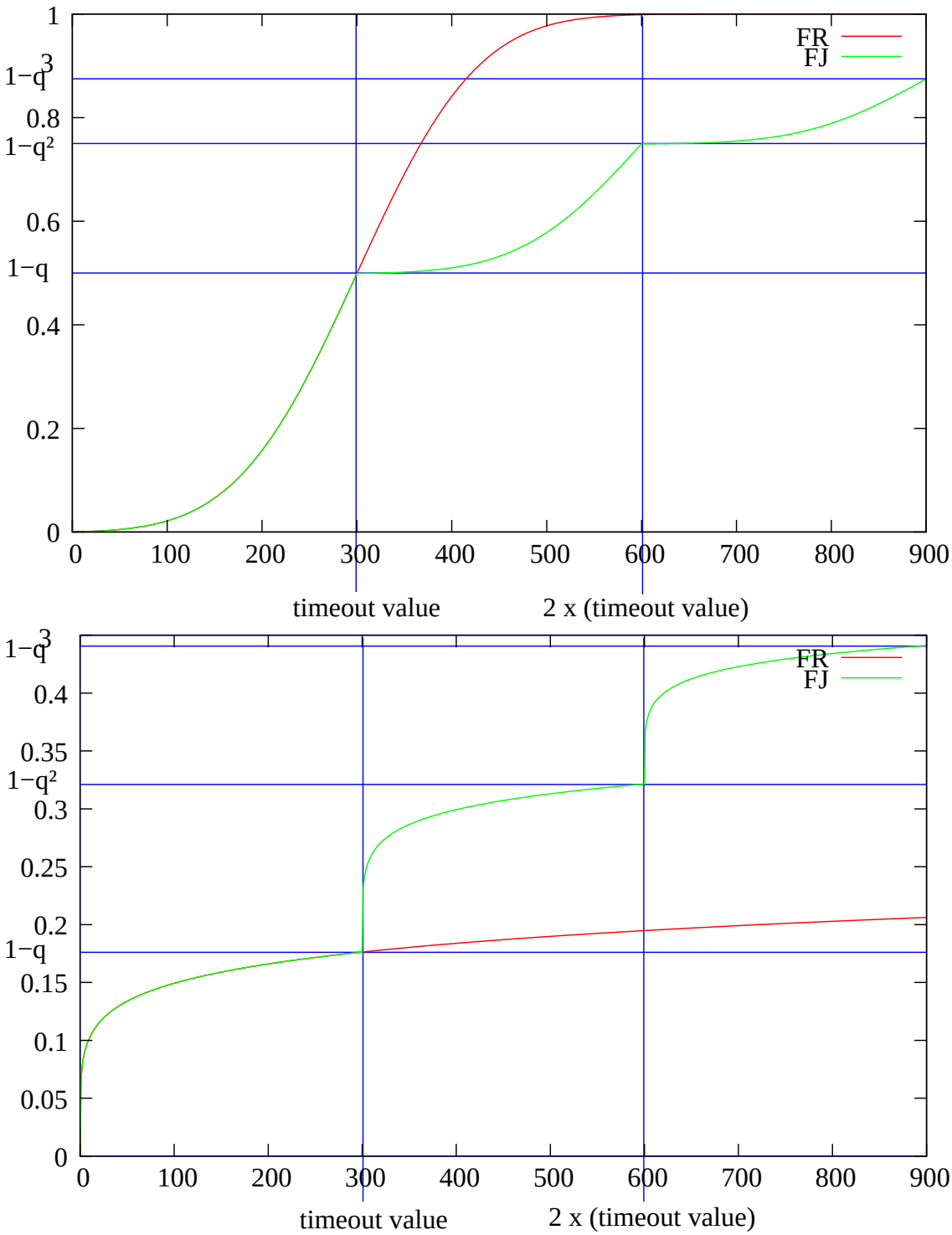

Figure 8.1: Example of cdfs of the latency without (R) and with timeout (J). Top: bad timeout choice ($F_R > F_J$). Bottom: good timeout choice ($F_R < F_J$).

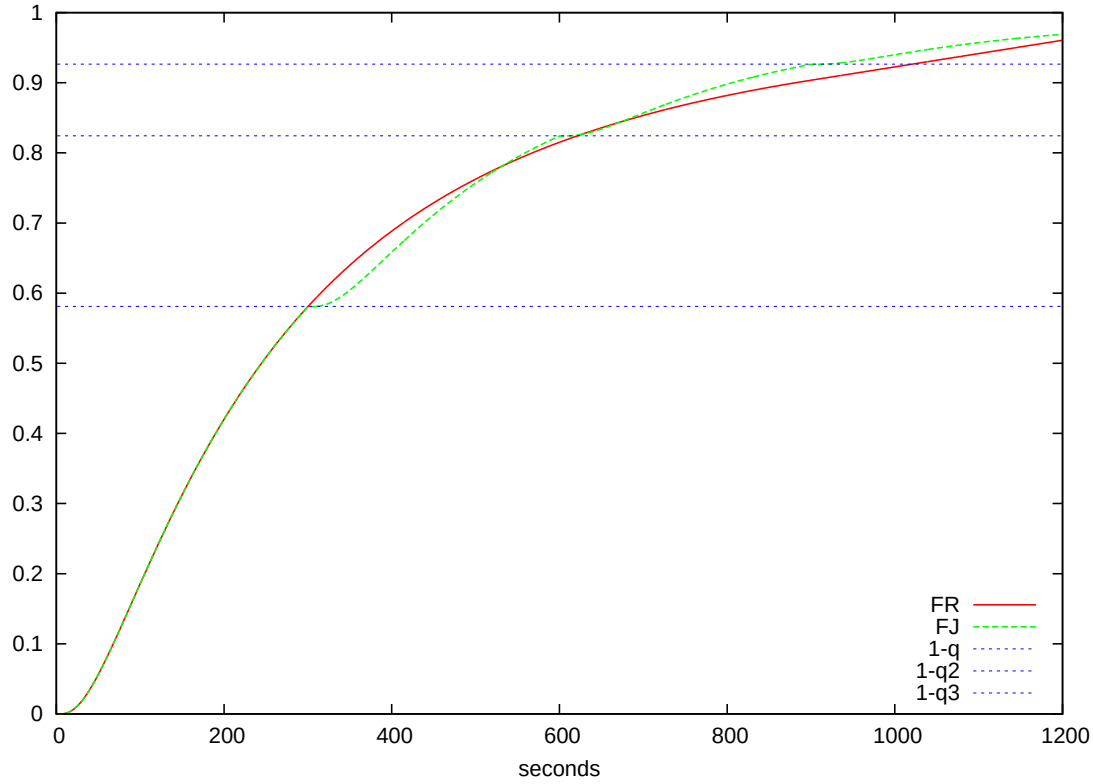

Figure 8.2: The cdfs of the latency without (F_R) and with (F_J) a timeout value cannot be compared for every time point. The optimal timeout value can be determined by expectation minimization.

As suggested by those graphical remarks, the impact of some timeout choices on the user job latency may be evaluated by comparing F_J and F_R only. However, apart from those particular cases, it is often not possible to have general results on the comparison between F_J and F_R at every time point t and the configuration displayed on figure 8.2 is observed. On this figure, the distribution of R is log-normal, with $\mu = 5.5$s, $\sigma = 1$s and a timeout value of 300s. In this case, minimizing the expectation of J with respect to t_∞ is a natural solution to optimize the timeout value.

8.1.2 Expectation of the latency J faced by a user job

Computing the expectation of the latency faced by a user job, general conclusions can be made on its behavior when the timeout value increases, independently from the system latency distribution. As shown in appendix B.1, The expectation of the latency J faced by a user job is:

$$E_J(t_\infty) = \frac{1}{F_R(t_\infty)} \int_0^{t_\infty} u f_R(u) du + \frac{t_\infty}{(1-\rho)F_R(t_\infty)} - t_\infty. \tag{8.6}$$

Equation 8.6 compares to similar expressions derived for modeling completion times probabilistically: equation 6 in [van Moorsel and Wolter, 2006] and equation 1

in [Libman and Orda, 2002]. In both cases, the authors introduced a fixed cost penalty to resubmission that is considered here to be included in the latency R. In [van Moorsel and Wolter, 2006], the authors also derives higher moments of J and some relevant properties about them (*e.g.* their existence). Our hypotheses are similar to theirs, except that they do not take into account outliers that are of major importance on the production infrastructures that are targeted in this chapter. This parameter is characteristic of unreliable systems and is needed to properly model a grid infrastructure. In [Libman and Orda, 2002], the authors do take into account the outlier ratio (denoted L) in the context of the retransmission of network packets. However, the studied hypotheses do not really match ours. In our case, a grid job is abandoned when it times-out (simple client) whereas it is still monitored in [Libman and Orda, 2002].

As shown in appendix B.2, E_J has the following limits:

$$\lim_{t_\infty \to \infty} E_J(t_\infty) = +\infty \quad \text{if } \rho \neq 0 \tag{8.7}$$

$$\text{and} \quad \lim_{t_\infty \to \infty} E_J(t_\infty) = E_R \quad \text{otherwise.} \tag{8.8}$$

Moreover, if $\rho \neq 0$ (with outliers), the straight-line $E_R + \frac{\rho}{1-\rho} t_\infty$ is an asymptote of $E_J(t_\infty)$. The first limit can be explained by noticing that if a single grid job is an outlier, then the latency faced by the user job is infinite. When $t_\infty \to +\infty$, the probability for encountering an outlier tends towards 1 and the expected latency faced by the user job tends towards infinity. It is thus mandatory to set a timeout value in case of outliers. The second limit is also intuitive: in absence of outliers, if no timeout value is set, then only a single grid job is submitted and the expectation of the latency faced by the user job resumes to the expectation of the grid latency.

8.2 Timeout optimization for classical latency distributions

In this section, some classical distributions of the latency R are studied from a theoretical point of view in order to understand how the timeout value impacts the expectation of the latency faced by the user job both with and without outliers. Distributions with light tails (uniform, truncated Gaussian and Weibull with shape parameter >1) are distinguished from heavy-tailed ones (log-normal, Weibull with shape parameter <1) and power tails (Pareto) to show how they exhibit different behaviors. The exponential distribution will constitute a transition between light and heavy-tailed distributions. Light-tailed distributions are the ones that decay faster than the exponential. In this case, there exists a such that: $\lim_{t\to+\infty} e^{at}(1 - F(t)) = 0$. On the contrary, heavy-tailed distributions decay slower than the exponential : $\lim_{t\to+\infty} e^{at}(1 - F(t)) = +\infty$. Power-tailed distributions are a subset of the heavy-tailed ones. In this case, there exists a and b such that $\lim_{t\to+\infty} \frac{1-F(t)}{t^a} = b$.

For each distribution, the goal is to determine the optimal timeout value:

$$\hat{t}_\infty = \arg\min_{t_\infty} \{E_J(t_\infty)\}.$$

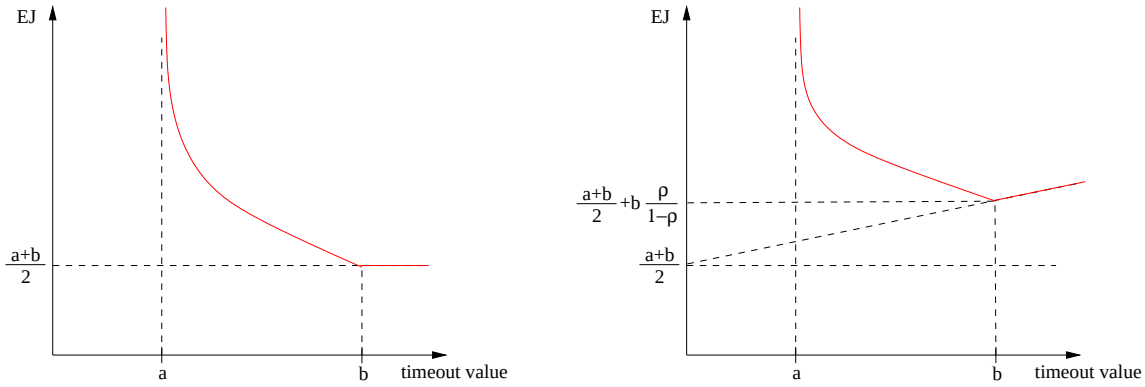

Figure 8.3: Behavior of the expectation of the latency $E_J(t_\infty)$ faced by a user job for the uniform distribution without (left) and with (right) outliers.

In case of very reliable systems (when no outliers are present), the optimal value of the timeout may be $+\infty$, which means that no timeout should be set. Another singular optimal timeout value is 0. This configuration occurs when the probability for a grid job to face a null latency is so high that it is interesting to resubmit a new one as soon as one knows that the current one is going to face a non null latency. This result would only be realistic if it was possible to resubmit an arbitrarily large number of jobs at no additional cost. Obviously, the overhead induced on any real system would finally slow down the process.

8.2.1 Uniform distribution

In this case, the pdf of the system latency is:

$$f_R(t) = \begin{cases} \frac{1}{b-a} & \text{if } t \in [a,b] \\ 0 & \text{otherwise.} \end{cases} \tag{8.9}$$

It is possible to derive from equation 8.6 the expectation of the latency J faced by a user job:

$$E_J(t_\infty) = \begin{cases} +\infty & \text{if } t_\infty \leq a \\ \frac{t_\infty+a}{2} + t_\infty \frac{b-t_\infty+\rho(t_\infty-a)}{(t_\infty-a)(1-\rho)} & \text{if } t_\infty \in [a,b] \\ \frac{b+a}{2} + t_\infty \frac{\rho}{1-\rho} & \text{otherwise.} \end{cases} \tag{8.10}$$

The curve of $E_J(t_\infty)$ is depicted on figure 8.3. The optimal timeout value is b both with and without outliers. Without outliers, setting the timeout to $+\infty$ is also optimal because the expectation of J is constant in [b,+∞[. If there is no outlier, it can be graphically noticed that setting a timeout always penalizes the execution. Indeed, as figure 8.4 shows, the cdf of the latency with timeout ($F_J(t)$) is lower than the cdf of the latency without any timeout ($F_R(t)$), for every timeout value t_∞ and every time point t.

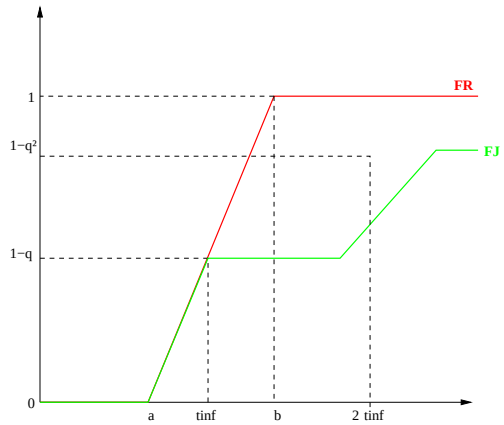

Figure 8.4: Behavior of the cdf F_J of the latency faced by a user job (green) and the cdf F_R of the grid latency (red) for a uniform distribution without outliers.

8.2.2 Truncated Gaussian distribution

Normal distributions are commonly used but they do not exclude negative values. In this case, the latency cannot be lower than 0. We are thus considering Gaussian distributions with mean μ and standard-deviation σ truncated above 0. In this case, the pdf and cdf of the system latency are:

$$f_R(t) = \begin{cases} \frac{1}{\Phi\left(\frac{\mu}{\sigma}\right)} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2} & \text{if} \quad t \geq 0 \\ 0 & \text{otherwise,} \end{cases}$$

$$F_R(t) = \frac{\Phi\left(\frac{\mu}{\sigma}\right) - \Phi\left(\frac{\mu-t}{\sigma}\right)}{\Phi\left(\frac{\mu}{\sigma}\right)} \quad \text{with} \quad \Phi(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{t} e^{-\frac{1}{2}u^2} du.$$

with Φ being the cdf of the normed and centered Gaussian distribution. Φ is linked to the error function (`erf`) as detailed in appendix B.9.

As shown in appendix B.5, the expectation of the latency faced by a user job is then:

$$E_J(t_\infty) = \mu + \sigma \frac{\phi\left(\frac{\mu}{\sigma}\right) - \phi\left(\frac{\mu-t_\infty}{\sigma}\right)}{\Phi\left(\frac{\mu}{\sigma}\right) - \Phi\left(\frac{\mu-t_\infty}{\sigma}\right)} + \frac{1}{1-\rho} t_\infty \left(\frac{\Phi\left(\frac{\mu-t_\infty}{\sigma}\right)}{\Phi\left(\frac{\mu}{\sigma}\right) - \Phi\left(\frac{\mu-t_\infty}{\sigma}\right)} + \rho \right)$$

with $\phi = \Phi'$ the pdf of the normed and centered Gaussian distribution.

The curve of E_J is plotted on figure 8.5. E_J exhibits different behaviors depending on the presence of outliers or not. If there is no outlier ($\rho = 0$), then E_J is decreasing towards its limit E_R when $t_\infty \to +\infty$. On the other hand, when $\rho \neq 0$, then E_J exhibits a global minimum reached for $\hat{t}_\infty < +\infty$. The corresponding proof is based on the fact that the fourth derivative of E_J is always positive, so that we can study the existence of a root in the lower order derivatives. It is reported in appendix B.6.

If the distribution of the system latency is Gaussian and there is no outlier, time-outing is not a solution to limit the impact of variability, regardless of the order of magnitude of the variability.

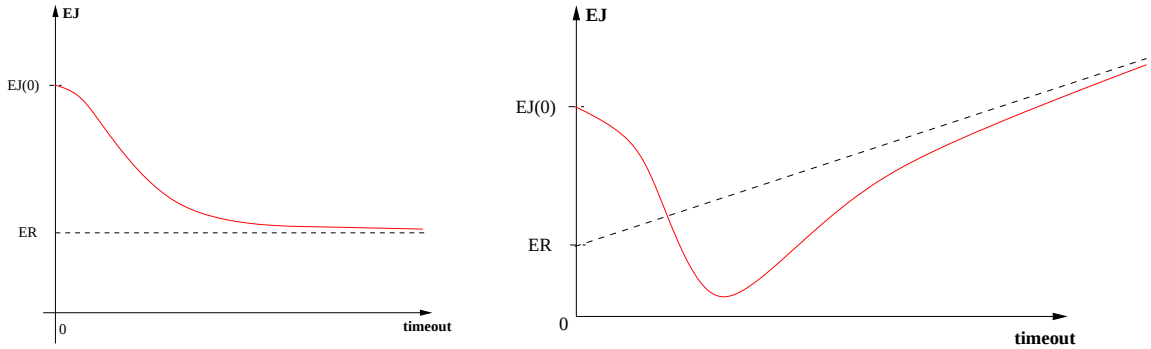

Figure 8.5: Behavior of the expectation E_J of the latency faced by a user job for a truncated Gaussian distribution without (left) and with (right) outliers.

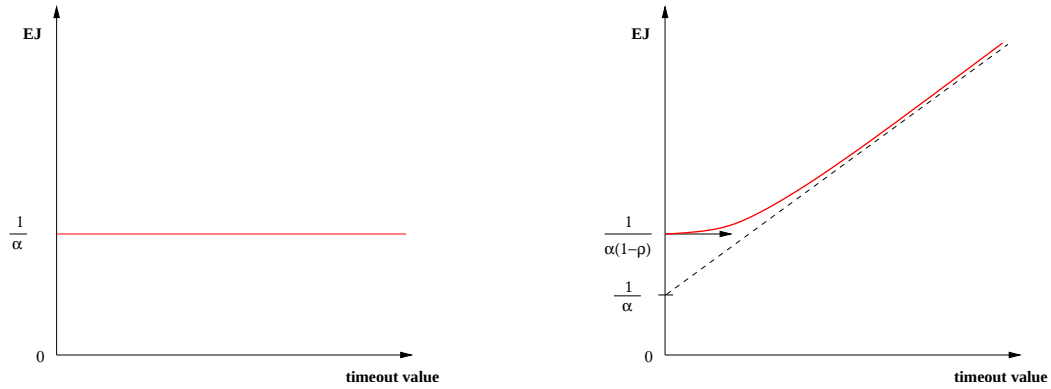

Figure 8.6: Behavior of the expectation E_J of the latency faced by a user job for an exponential distribution without (left) and with (right) outliers.

8.2.3 Exponential distribution

In this case, the cdf of the system latency is:

$$F_R(t) = 1 - e^{-\alpha t}.$$

And according to equation 8.6, the expectation of the latency J faced by a user job is:

$$E_J(t_\infty) = \frac{1}{\alpha} + \frac{\rho t_\infty}{(1-\rho)(1-e^{-\alpha t_\infty})}.$$

The curve of $E_J(t_\infty)$ is depicted on figure 8.6. In case of outliers, E_J is increasing and the best timeout value is $\hat{t}_\infty = 0$. If there are no outliers, the expectation of J is independent from t_∞, which is a singular behavior particular to the exponential distribution, as proved in appendix B.3. The exponential distribution is a particular case of the Weibull one which is studied in the next section.

8.2.4 Weibull distribution

The Weibull distribution is typically used to model the failure of technical devices. For this distribution, the cdf of the grid latency is:

$$F_R(t) = 1 - e^{-\left(\frac{t}{\lambda}\right)^k}$$

where k is a shape parameter and λ is a scale parameter of the distribution. In the context of failure modeling, k<1 means that the failure rate decreases over time, $k = 1$ means that the failure rate is independent from time and k>1 means that the failure rate increases over time. In this case, the random variable R can be seen as the instant at which a grid job completes, *i.e* R models the success of a grid job instead of its failure. Note that the exponential distribution of parameter $1/\lambda$ is a Weibull distribution with k=1.

In this case, the following results can be proved:

- If $k > 1$, then setting a timeout value always penalizes the execution, whatever this value is. The optimal timeout value is thus $+\infty$ (no timeout). This result is consistent with the fact that the Weibull distribution with $k = 3$ is often used to approximate the Gaussian one. In this case, it has been shown in section 8.2.2 that setting a timeout value always penalizes the execution.
- If $k < 1$, then the timeout value has to be as low as possible. The optimal timeout value is 0.
- If $k = 1$, then the distribution of R is exponential and the timeout value does not impact at all the latency faced by a user job.

The corresponding proofs are reported in appendix B.4.

The obtained results are consistent with the classical interpretation of the shape parameter of the Weibull distribution. Indeed, when k>1, the success rate of a grid job is increasing over time, which explains that time-outing will penalize the user job. On the contrary, when k<1, the success rate of a grid job is then decreasing over time, and time-outing as soon as possible becomes mandatory.

8.2.5 Log-normal distribution

The log-normal distribution is a typical example of heavy-tailed distribution. In [Li et al., 2004], it is used to fit job running times on clusters. In this section, the grid latency is assumed to have a log-normal distribution with parameters μ and σ. In this case, the cdf and pdf of the system latency are:

$$F_R(t) = \Phi\left(\frac{\ln t - \mu}{\sigma}\right) \quad \text{and} \quad f_R(t) = \frac{1}{t\sqrt{2\pi}\sigma} e^{-\frac{(\ln t - \mu)^2}{2\sigma^2}}.$$

The expectation and standard-deviation of the latency are:

$$E_R = e^{\mu+\frac{\sigma^2}{2}} \quad \text{and} \quad \sigma_R = \left(e^{\sigma^2}-1\right)e^{2\mu+\sigma^2}. \tag{8.11}$$

In this case, as reported in appendix B.7, the expectation of the latency J faced by a user job is:

$$E_J(t_\infty) = E_R\left(\frac{\Phi\left(x_\infty-\sigma\right)}{\Phi(x_\infty)} + e^{\sigma x_\infty-\frac{\sigma^2}{2}}\left(\frac{1}{(1-\rho)\Phi(x_\infty)}-1\right)\right) \tag{8.12}$$

$$\text{where} \quad x_\infty = \frac{\ln(t_\infty)-\mu}{\sigma}$$

This expression shows that the minimization of E_J can be performed independently from μ on the transformed variable x_∞. The obtained solution $\hat{x}_\infty(\sigma,\rho)$ only depends on σ and ρ. The optimal timeout value can then be written as:

$$\hat{t}_\infty(\mu,\sigma) = e^{\mu}K(\sigma,\rho) \quad \text{where} \quad K(\sigma,\rho) = e^{\sigma\hat{x}_\infty(\sigma,\rho)} \tag{8.13}$$

and:

$$\hat{x}_\infty(\sigma,\rho) = \arg\min_{x_\infty}\left(\frac{\Phi\left(x_\infty-\sigma\right)}{\Phi(x_\infty)} + e^{\sigma x_\infty-\frac{\sigma^2}{2}}\left(\frac{1}{(1-\rho)\Phi(x_\infty)}-1\right)\right).$$

$K(\sigma,\rho)$ is actually the optimal timeout value for $\mu = 0$.

We also have the following limit for $t_\infty = 0$:

$$\lim_{t_\infty\to 0} E_J(t_\infty) = \lim_{x_\infty\to-\infty} E_J(x_\infty) = +\infty.$$

This infinite limit proves that when $\rho \neq 0$ (with outliers), there exists a finite non null optimal timeout value that minimizes E_J. Indeed, in this case, the limit of E_J when t_∞ tends towards infinity is infinite, according to equation 8.7 and E_J thus has to reach a global minimum (because it is continuous on $]0,+\infty[$).

The existence of a global minimum of $E_J(t_\infty)$ when $\rho = 0$ is not straight-forward. Given the infinite limit of E_J when t_∞ tends towards 0 and given that $E_J(+\infty) = E_R$, it resumes to the existence of a t_∞ for which $E_J(t_\infty)<E_R$. If $\sigma \geq 1$, then $t_\infty = e^{\mu}$ satisfies this relation. Indeed, in this case, $x_\infty = 0$ and according to equation 8.12,

$$E_J(x_\infty = 0) = \left(2\Phi(-\sigma) + e^{-\frac{\sigma^2}{2}}\right)E_R$$

A numeric resolution then shows that $E_J<E_R$ if and only if $\sigma \gtrsim 0.9311$. Numeric simulations suggest that E_J has a global minimum even for lower values of σ. However, an analytic proof still has to be derived.

Figure 8.7 displays a simulation of the optimal timeout value for μ=0, several values of the outlier ratio and σ ranging from 1 to 2 seconds. We first can notice that $K(\sigma,\rho)$ seems to decrease with respect to ρ. The timeout value thus has to be reduced when the proportion of outliers is increasing, which is consistent. Moreover, given an outlier ratio, the optimal timeout value for $\mu = 0$ is decreasing as σ is growing. It is also consistent because the standard-deviation of the log-normal distribution is increasing with respect to σ (see equation 8.11). The optimal timeout value thus has to be reduced as the variability of the infrastructure is growing.

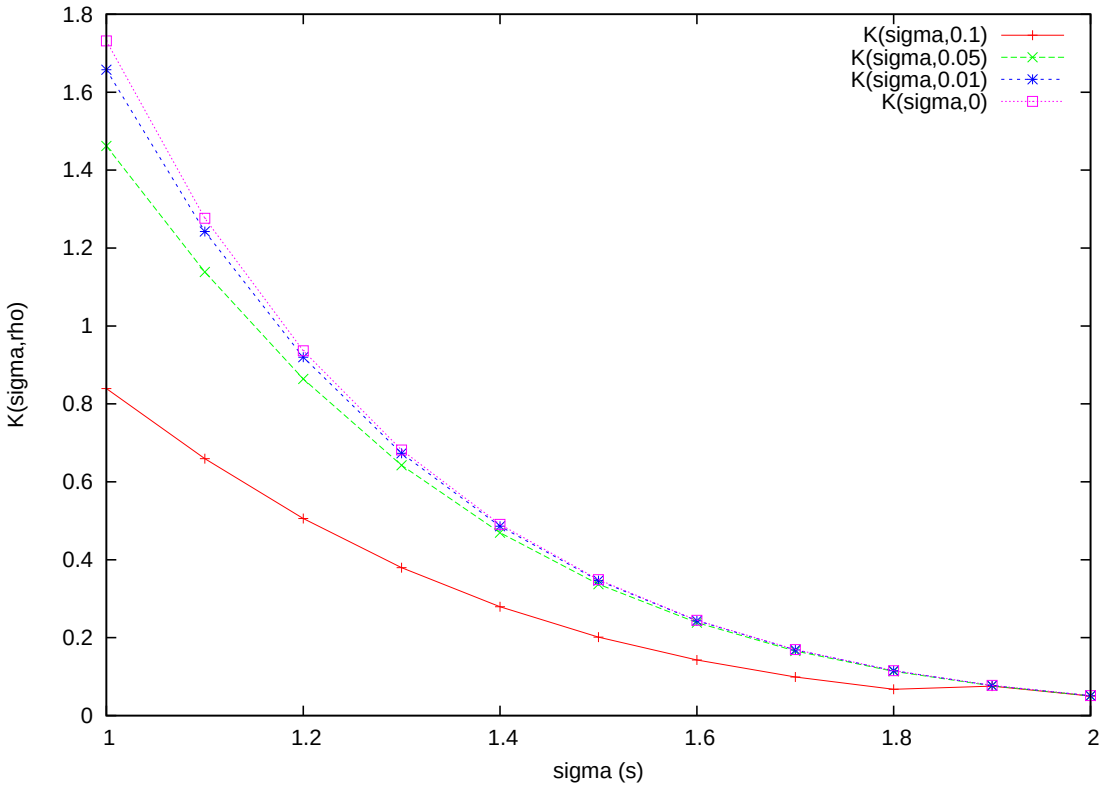

Figure 8.7: Evolution of the optimal timeout value for μ=0 in the log-normal case. $K(\rho, \sigma)$ is decreasing with respect to ρ and σ, which indicates that the timeout value has to be reduced when the variability of the latency and the outlier ratio increase.

Figure 8.8: Behavior of the expectation E_J of the latency faced by a user job for a Pareto distribution of the grid latency. Left: no outliers ($\rho = 0$); Right: $\rho \neq 0$.

8.2.6 Pareto distribution

The Pareto distribution was introduced to represent the distribution of wealth and proved to be very accurate to model a large class of computer systems measurements (jobs durations, size of the files, data transfers length on the Internet...) [Harchol-Balter and Balter, 2002]. It is an example of power tailed distribution. The cdf of the system latency is then:

$$F_R(t) = 1 - \left(\frac{a}{a+t}\right)^{\nu} \quad \text{with} \quad a \quad \text{and} \quad \nu > 0.$$

The expectation is only defined for ν>1. Then:

$$E_R = \frac{a}{\nu - 1}.$$

In this case, the expression of the expectation E_J of the latency faced by a user job can be directly derived from equation 8.6 and it is:

$$E_J(t_\infty) = \frac{a + t_\infty \nu - a\left(\frac{a+t_\infty}{a}\right)^{\nu}}{(1-\nu)\left[\left(\frac{a+t_\infty}{a}\right)^{\nu} - 1\right]} + \frac{t_\infty}{(1-\rho)\left[\left(\frac{a+t_\infty}{a}\right)^{\nu} - 1\right]} + \frac{\rho}{1-\rho} t_\infty.$$

We also have the following limit when the timeout value is null:

$$\lim_{t_\infty \to 0} E_J(t_\infty) = \frac{a}{\nu(1-\rho)}.$$

It can be shown that E_J is increasing with respect to the timeout value, regardless of the ρ value (see proof in appendix B.8). The optimal timeout value is thus 0. The behavior of $E_J(t_\infty)$ is depicted on figure 8.8.

8.2.7 Results summary and interpretation

Table 8.1 displays a summary of the results obtained for various distributions of the system latency. Those results suggest that the weight of the tail of the distribution of the system latency is a discriminatory parameter for the timeout optimization when outliers are not present.

Distribution of the latency (F_R)	Without outliers ($\rho = 0$)	With outliers (ρ>0)	Tail of F_R
Uniform	no timeout (or b)	b	Light
Trunc. Gaussian	no timeout	$0 < \hat{t}_\infty < +\infty$	Light
Weibull k>1	no timeout	?	Light
Exponential	any	0	Exp.
Weibull k<1	0	?	Heavy
Log-normal (μ,σ)	$\hat{t}_\infty = e^\mu K(\sigma) < +\infty$	$0 < \hat{t}_\infty < +\infty$	Heavy
Pareto (ν>1)	0	0	Power

Table 8.1: Optimal timeout values. The weight of the tail of the distribution is an important parameter of the problem.

Indeed, only heavy-tailed distributions such as the log-normal, or the Pareto ones lead to finite optimal timeout values. In this case, which corresponds to the most realistic one, the optimization speeds up the execution. On the other hand, when the distribution of the system latency decays faster than the exponential, (which is the case for the Gaussian truncated distribution, for the Weibull one with $k > 1$, and for the uniform one) then setting a timeout value always penalizes the execution and the optimal timeout is $+\infty$. The exponential distribution stands in the middle and is not affected by the timeout value.

As noticed in section 8.1, taking into account outliers corresponds to replacing F_R by $(1 - \rho)F_R$ in F_J and q (and thus in E_J). In this case, there is a probability to face an infinite latency, which makes the tail of the grid latency distribution heavy (mathematically, the distribution of the system latency becomes heavy-tailed because $\lim_{x\to+\infty} e^{ax}(1 - (1 - \rho)F_R(x)) = +\infty$ when $a > 0$). Consistently, the optimal timeout value is then always finite.

8.2.8 Performance improvement

In case of reliable systems (without outliers), the expectation of the latency faced by a user job without timeout equals to the one of the system latency. In this case, the ratio $\frac{E_R}{E_J(\hat{t}_\infty)}$ evaluates the speed-up yielded by the optimization. If the latency of the system is light-tailed, then setting a timeout value always penalizes the execution. The best strategy is thus to set the timeout value to infinity. In this case, the optimization does not provide any speed-up with respect to the expectation of the system latency. Concerning the limit case of an exponential distribution, the expectation of the latency faced by a user job is independent from the timeout value and the optimization does not lead to any speed-up.

The optimization becomes interesting for heavy-tailed distributions as already suggested. For the log-normal case, figure 8.9 displays a numerical simulation of the evolution of the speed-up of the optimization with respect to σ for a particular value of μ. It shows that the

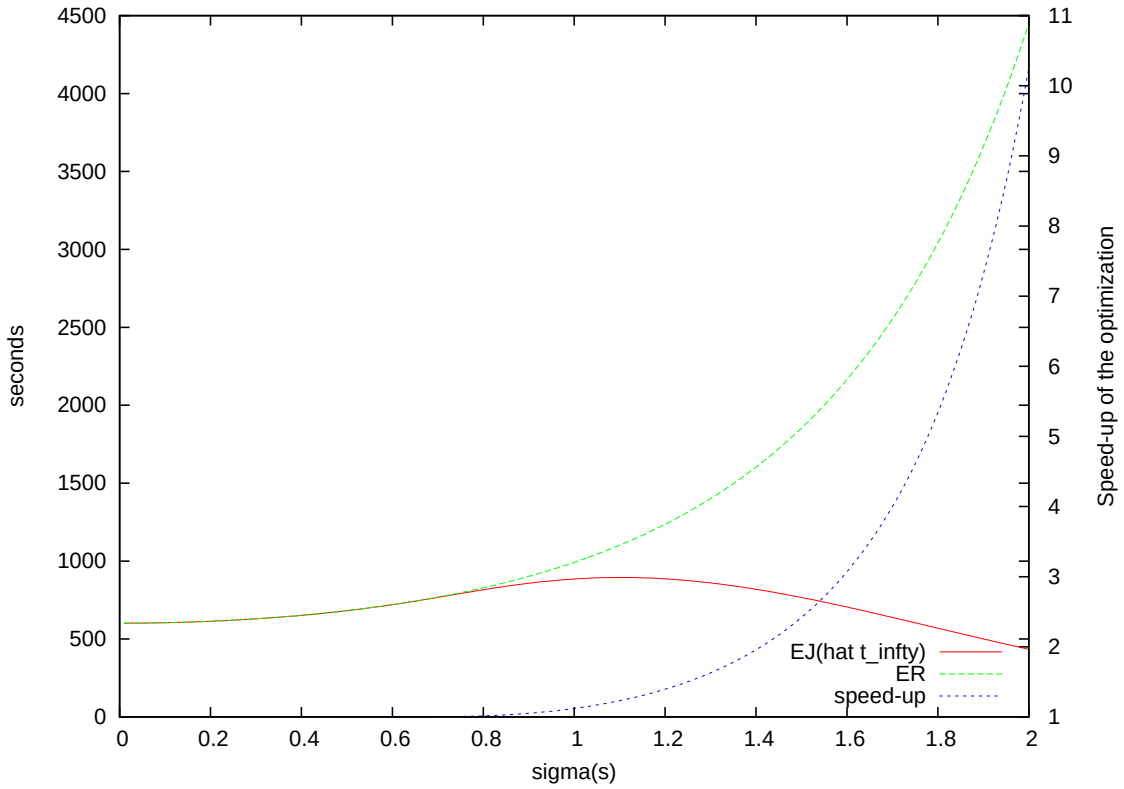

Figure 8.9: Evolution of the speed-up of the optimization for $\mu = 6.4$s in the log-normal case. The more variable the infrastructure, the more interesting the timeout optimization.

speed-up is growing with σ. In this case, both the expectation (E_R) and standard-deviation (σ_R) of the grid latency without any timeout are also growing with σ (see equation 8.11). Thus, the higher and the more variable the latency, the more interesting the timeout optimization.

Concerning the Pareto distribution, the optimized expectation of the latency faced by a user job without outliers is $\frac{a}{\nu}$, whereas the one obtained without setting any timeout is $E_R = \frac{a}{\nu-1}$. The speed-up obtained by the optimization is thus $\frac{\nu}{\nu-1}$. This value is maximal for $\nu = 1$ and decreases towards 1 when ν increases. Moreover, under Pareto assumption, the variance of the system latency $\left(\frac{\nu a^2}{(\nu-2)(\nu-1)^2}\right)$ is decreasing with respect to ν. Thus, the more variable the latency of the infrastructure, the higher the speed-up yielded by the optimization.

When outliers are present, the optimization of the timeout prevents the expectation of J to be infinite. The impact of the optimization can then be evaluated by comparing the optimized expectation of the latency faced by a user job to the one obtained without outliers. In case of a uniform distribution, outliers add the term $b\frac{\rho}{1-\rho}$ to the expectation of the latency faced by a user job. This term is increasing with respect to the outlier ratio and tends towards infinity when ρ tends towards 1. The exponential distribution and the Pareto one exhibit a similar behavior: the outliers introduce an extra $\frac{1}{1-\rho}$ factor on the expectation of the latency faced by a user job.

8.3 Experiments on the EGEE latency distribution

In this section, we present experimental results obtained by measuring the distribution of the latency of the EGEE grid infrastructure on a particular time period. This distribution was described in section 6.2.1 of chapter 6 and can be accurately modeled by a mixed log-normal and Pareto model. Thus, according to the theoretical study conducted above, the expectation of the latency with respect to the timeout value should exhibit a global finite minimum both with and without taking into account outliers.

If outliers are not taken into account, the evolution of the expectation E_J of the latency faced by a user job with respect to the timeout value is plotted on figure 8.10. E_J converges towards E_R as predicted by the theoretical analysis. It reaches a minimum for $\hat{t}_\infty = 360$s. At this optimal point, $\hat{E}_J(\hat{t}_\infty) = 289$s whereas $E_R = 393$s. The speed-up with respect to an execution without timeout is 1.36.

The evolution of $E_J(t_\infty)$ taking the outliers into account is also plotted on figure 8.10. E_J effectively tends towards its asymptote. The optimal timeout value $\hat{t}_\infty$ is now 358 seconds and $\hat{E}_J(\hat{t}_\infty)$ has grown to 300 seconds. Setting the optimal timeout value thus limits the impact of the outliers to a 11-seconds loss, whereas it would be much higher if the timeout value is not properly set, as suggested by figure 8.10. This figure also shows that the timeout value should better be overestimated than underestimated: both curves are rapidly decreasing to the optimal timeout value whereas they increase more smoothly after it.

Once the distribution of the grid latency is available (see chapter 6 for latency estimations), deriving the optimal timeout value with this method is easily automatable. The optimization criterion (*i.e.* E_J written in equation 8.6) is computable in a short time: it mainly includes the computation of an integral of $u\,f_R(u)$, which is a piecewise linear function when an empirical distribution is considered. For instance, plotting the curves of figure 8.10 takes less than 2 seconds on a modern PC.

8.4 Conclusions

In practice, setting a timeout value to the grid jobs is a relevant strategy to reduce the impact of the grid latency and it is required to keep the impact of outliers under control. In this chapter, a probabilistic model of the latency faced by a user job taking into account time-outing and resubmissions was presented. It can describe both job management systems prone to face outliers (grid) or not (cluster). The optimal timeout value highly depends on the distribution of the system latency. Without outliers, the heavy-tailed distributions lead to a finite optimal timeout value whereas for the light-tailed ones setting a timeout value always penalizes the execution. If outliers are present, the model predicts that the expectation of the latency faced by a user job with respect to the timeout value is diverging to $+\infty$ for every distributions,

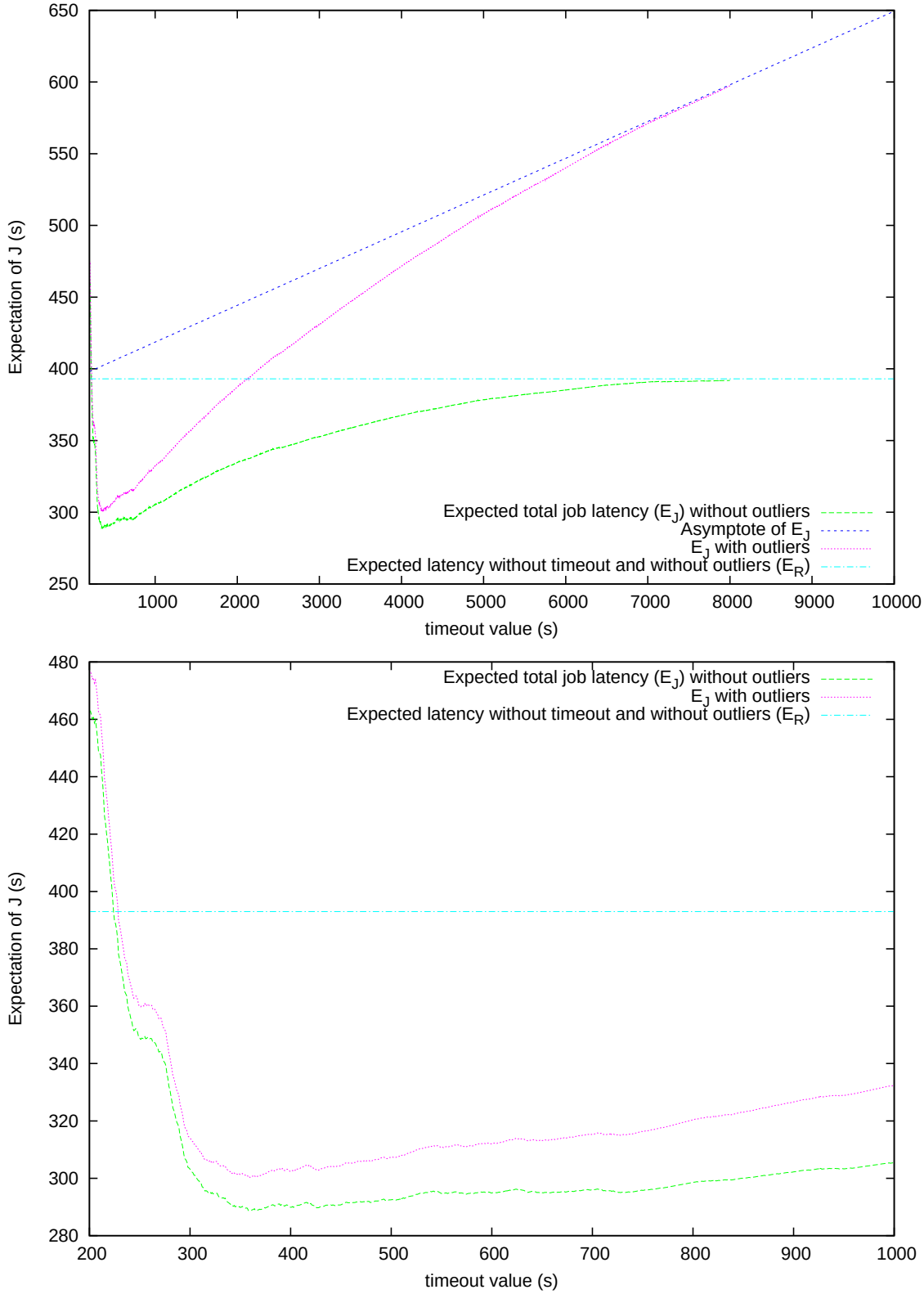

Figure 8.10: Evolution of the expectation of the total latency J of a user job in the experimental case (on EGEE). Top: experimental curves effectively tend towards their asymptotes. Bottom: close-up between 200s and 1000s. The impact of outliers (difference between the two curves) is limited to 11 seconds. Overestimating the timeout value seems better than underestimating it.

following an asymptote whose slope only depends on the outlier ratio. The optimal timeout value is finite for all the studied distributions since taking outliers into account lengthens the tail of the distribution. Some results were finally presented on an empirical distribution measured from the EGEE grid. It is heavy-tailed and modelable through a mixture of log-normal and Pareto distributions. Even without outliers, a 1.36 speed-up can be achieved by optimizing the timeout value. Considering outliers, optimizing the timeout value is even more critical and the resulting expectation of the latency faced by a user job is close to the one obtained without outliers.

Results presented in this chapter can be related to the field of operational research and in particular to the queuing theory [Kleinrock, 1975, Gross and Harris, 1985, Baynat, 2000]. For instance, the fact that the job timeout might be set to any value without any impact on the expected execution time given that the distribution of the grid latency is exponential could have been induced from the memory-less property of this distribution. This property made this distribution extremely popular in the queuing theory, in particular in the M/M/c model. Yet, given an arrival rate of the clients and a service rate, the goal of the queuing theory is to determine the distribution of the number of clients inside the system and the waiting time. It might be used at a finer grain to explain or even predict the distribution of the latency which is assumed here to be known. What we showed in this chapter is that considering the grid latency as a random variable (whose distribution is measurable or supposed to be known from another theory) and optimizing the jobs timeout value from it improves the performance of grid applications in practice. We believe that the adopted approach is new in the domain of distributed computing, as traditional systems do not exhibit the required conditions of variability for making such an optimization interesting. For instance, one can intuitively easily conceive that setting a timeout to jobs submitted to a single cluster would not be of any use. Indeed, in this case, cancelling and resubmitting the job to the same queue could only lead to an increase of the job waiting time.

As introduced in chapter 6, the grid is seen here as a black-box introducing a variable latency on the grid jobs. It is important to notice that this study is completely grounded by the highly variable nature of the grid latency, which was highlighted in chapters 5 and 6. This variability introduces a *risk* in the process of submitting a grid job: a job can be highly delayed or even not finish at all. Similarly, the problem addressed in the next chapter is motivated by this notion of risk associated to the submission of a grid job.

Chapter 9

Optimization of the job granularity

Contents

In this chapter, the optimization of the job granularity is studied as a solution to reduce the impact of the latency on an application. Given a divisible user job to compute and considering its total execution time as the criterion to optimize, a trade-off has to be found between the submission of a high number of short jobs (which maximizes parallelism but increases the risk that one of them penalizes the whole application by remaining blocked somewhere in the system) and lowering the number of jobs (which reduces parallelism as well as the risk to face high latencies). Similarly to the approach adopted in the previous chapter, this trade-off is formalized using a probabilistic model and studied on classical distributions. Finally, experimental results are presented on EGEE, showing that a significant speed-up as well as an important reduction of the number of jobs can be obtained using this method.

Dans ce chapitre, l'optimisation de la granularité des tâches est étudiée comme stratégie de réduction de l'impact de la latence sur une application. Pour une application partitionable, et en considérant son temps d'exécution total comme le critère à optimiser, un compromis doit être trouvé entre soumettre un grand nombre de tâches courtes (ce qui maximise le parallélisme mais augmente le risque que l'une d'entre-elles pénalise toute l'application en restant bloquée quelque part au cours de son cycle de vie) et diminuer le nombre de tâches (ce qui réduit le parallélisme mais aussi le risque de rencontrer de fortes latences). Comme dans les chapitres précédents, ce compromis est formalisé par un modèle probabiliste et étudié sur des distributions classiques. Enfin, des résultats expérimentaux sont présentés sur EGEE. Ils montrent qu'une accélération substantielle de l'application ainsi qu'une réduction importante du nombre total de tâches soumises peuvent être obtenues en utilisant cette méthode.

The impact of the grid latency on the performance of an application composed by a set of independent jobs is crucial. Ideally, a user would split her job in as many independent grid jobs as computing resources available in order to benefit from a maximal parallelism. Even in case of high but fixed latencies, this strategy would be optimal if the grid jobs do not communicate between each other. However, if the latency is assumed to be variable, which is the case on the EGEE infrastructure, as shown in chapters 5 and 6, a trade-off has to be found between the expected level of parallelization of the application and the risk to face high latencies. Indeed, the higher the number of submitted grid jobs, the higher the probability for one of them to be impacted by a high latency. It is important to notice here that the performance metric that is best considered from a workflow application point of view is the makespan rather than the throughput or the job fairness that may be taken into account from the infrastructure point of view. Thus, a single grid job is able to penalize the whole application performance if it is subject to an excessive latency. A strategy has to be found in order to minimize this risk. The

goal of such a strategy is to optimize the granularity of a user job in order to find the best compromise between parallelism and the risk of facing high latencies. The granularity of a user job is defined here as the number of grid jobs to submit given that the user job is supposed to be divisible into any number of chunks.

Optimizing the granularity of a user job also reduces the total number of jobs submitted to the infrastructure with respect to the default maximal partitioning strategy. Hence, a potential global improvement of the grid performance can be expected if every user adopts such a strategy. The goal of this study is to propose an optimization strategy for tuning the granularity of the user jobs. This strategy aims at :

- Lowering the total execution time of a user job (user's point of view) ;
- Reducing the total number of grid jobs submitted for a given user job (infrastructure's point of view).

A typical class of applications targeted by this strategy are embarrassingly parallel applications, that may correspond to parameter sweep studies and are very common in several scientific domains [Jacq et al., 2007] including medical image analysis [Sermesant et al., 2006] and where a large number of small jobs could be grouped into larger ones.

We consider a user job corresponding to a total execution time w supposed to be divisible into any number p of independent grid jobs. The grid infrastructure introduces a latency R_i on the grid job i. The goal is to minimize the makespan Σ of the user job defined as:

$$\Sigma = \max_{i \in [1,p]} \left(R_i + \frac{w}{p} \right) \tag{9.1}$$

If R_i are assumed to be constant fixed values ($\forall i, R_i = \bar{R}$), then the solution is straightforward and p has to be as high as possible. Thus, this problem is specific to highly variable infrastructures such as production grids. In the following of this chapter, the order of magnitude of the variability of the latency will be an important parameter of the problem. We will demonstrate that the more variable the infrastructure, the more interesting the granularity optimization. As it has already be discussed in chapter 6 (section 6.1.2), R_i will be assumed to be independent and identically distributed random variables.

The problem of splitting jobs over a set of a known amount of computing resources connected through a reliable and high performance network has been largely studied in the field of parallel computing [Chrétienne et al., 1995, Feitelson et al., 2004]. Several works address the job granularity issue, noticing that there is an optimal number of processors to determine to minimize the total execution time, taking into account both the computation time and the communication time. In [Weissman and Zhao, 1998], the authors use heuristics to determine a close to optimal configuration, in which jobs are assigned to specific processors to reduce communication overhead induced by routing and contention. Even if it provides good results in their scope, their solution is strictly deterministic and models the communication function

linearly in the number of processors, which cannot properly describe the latency R considered here. In [Montagnat et al., 2004b], the authors determine the optimal number of jobs to submit by determining an analytical model of the latency of the grid submission and queuing systems in a batch architecture. Such an analytical model is very hard to determine in a complex dynamic multi-users grid infrastructure. In those works, splitting a user job in a high number of grid jobs is damageable because the grid jobs are going to communicate between each other. The cost of these communications is increasing with the number of submitted jobs. Is is fundamental to notice that our problem does not come from the same issue. In our problem, the point is that the submission of a single job introduces a risk of penalizing the whole application.

In the following, the behaviour of the makespan Σ of the user job defined by equation 9.1 is investigated. Its expectation is derived and studied for various classical distributions of the latency in section 9.1. Then, an experiment on EGEE evaluates the gain that could be achieved by this method.

9.1 Model of the execution time of a user job allowing granularity tuning

The goal here is to determine the expectation of the makespan Σ of a user job with respect to its execution time w, its granularity p and the grid latency F_R in order to study its minimization for various distributions. First, the cdf of Σ can easily be determined:

$$\begin{aligned} F_\Sigma(t) &= P(\Sigma < t) = \prod_{i=1}^{p} P\left(R_i + \frac{w}{p} < t\right) \\ &= P\left(R_i < t - \frac{w}{p}\right)^p = F_R\left(t - \frac{w}{p}\right)^p \\ \text{Then } f_\Sigma(t) &= \frac{dF}{dt} = p\, f_R\left(t - \frac{w}{p}\right) F_R\left(t - \frac{w}{p}\right)^{p-1} \end{aligned}$$

The expectation of Σ can then be derived:

$$\begin{aligned} E_\Sigma(p) &= \int_{\mathbb{R}} t.f_\Sigma(t)dt = \int_{\mathbb{R}} tp\, f_R\left(t - \frac{w}{p}\right) F_R\left(t - \frac{w}{p}\right)^{p-1} dt \\ &= \int_{\mathbb{R}} p\left(t + \frac{w}{p}\right) f_R(t) F_R(t)^{p-1} dt \\ &= \int_{\mathbb{R}} pt\, f_R(t) F_R(t)^{p-1} dt + \frac{w}{p} \end{aligned} \tag{9.2}$$

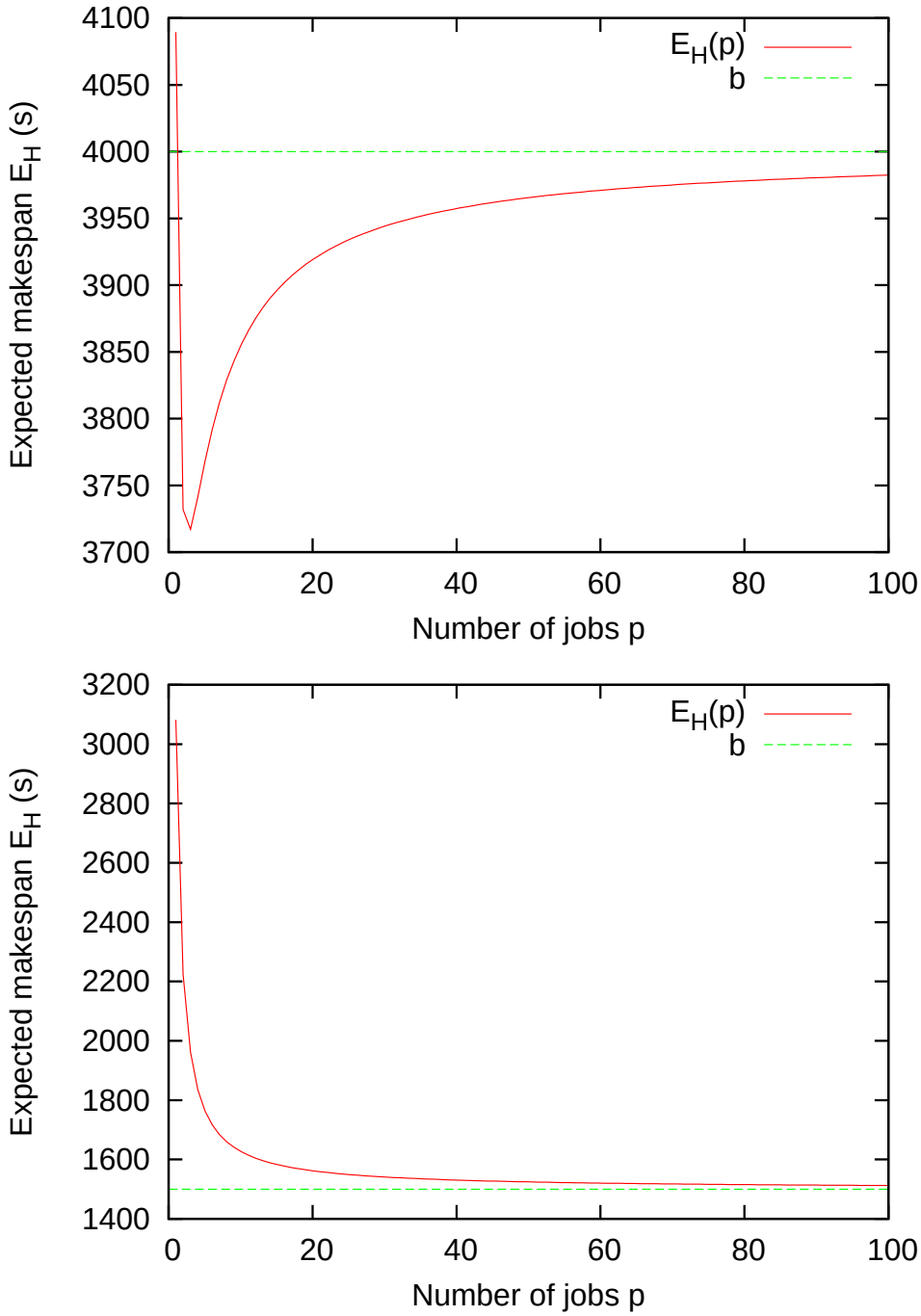

Figure 9.1: Representation of the expectation E_Σ of the makespan of a user job with respect to its granularity p for a uniform distribution of the grid latency with $a = 200$s, $b = 4000$s and $w = 2000$s (top) and $a = 700$s, $b = 1500$s and $w = 2000$s (bottom). The existence of a finite optimal granularity value is conditioned by the magnitude of $\frac{b-a}{w}$.

9.1.1 Uniform distribution

If the grid latency R is assumed to be uniformly distributed between a minimum value a and a maximum value b, then an explicit solution can be provided. Indeed, we then have:

$$f_R(t) = \begin{cases} \frac{1}{b-a} \text{ if } t \in [a,b] \\ 0 \text{ otherwise} \end{cases}$$

and

$$F_R(t) = \begin{cases} 0 \text{ if } t < a \\ \frac{t-a}{b-a} \text{ if } t \in [a,b] \\ 0 \text{ if } t > b \end{cases}$$

Thus, according to equation 9.2:

$$\begin{aligned} E_\Sigma(p) &= \int_a^b pt \frac{1}{b-a} \left(\frac{t-a}{b-a}\right)^{p-1} dt + \frac{w}{p} \\ &= \frac{(p+1)w + bp^2 + ap}{p(p+1)} \end{aligned}$$

It can be noticed that $E_\Sigma(1) = w + \frac{a+b}{2}$: it is consistent with the fact that the execution time on a single machine is w and the execution suffers from a $\frac{a+b}{2}$ penalty that is the mean latency introduced by the infrastructure. Moreover, $\lim(E_\Sigma(p))_{p\to+\infty} = b$: if an infinite amount of resources is used, the worst possible latency is faced (b) but the best computation time (0) is also obtained. Indeed, as the number of submitted grid jobs increases, the probability for one of them to suffer from a high latency increases. Finally, $\lim(E_\Sigma(p))_{p\to 0} = +\infty$: the limit of E_Σ towards zero corresponds to the execution of the user job on zero machine. In this case, the makespan of the user job consistently tends towards infinity.

The next step is the minimization of the expectation of Σ with respect to the granularity p. At the optimum, the derivative of the expectation of Σ should be null:

$$\frac{dE_\Sigma(p)}{dp} = -\frac{p^2w + 2pw + w - bp^2 + ap^2}{p^2(p+1)^2} = 0$$

Thus, if $w \neq b-a$, then the optimal number of grid jobs to submit is:

$$\begin{cases} p_1 = -\frac{\sqrt{(b-a)w}+w}{w-(b-a)} \\ or \\ p_2 = \frac{\sqrt{(b-a)w}-w}{w-(b-a)} \end{cases}$$

p_1 is positive if $(b-a) > w$ and negative otherwise whereas p_2 is always negative. Given that p has to be positive, there is a unique optimal number of grid jobs p_{opt} minimizing $E_\Sigma(p)$ if $(b-a) > w$ and we have: $p_{opt} = -\frac{\sqrt{(b-a)w}+w}{w-(b-a)}$. Such a configuration is represented on the upper

graph of figure 9.1 where $E_\Sigma(p)$ is plotted for a uniform distribution with $a = 200$s, $b = 4000$s and $w = 2000$s. On the other hand, if $(b-a) < w$ then $\frac{dE_\Sigma(p)}{dp} < 0$ so that E_Σ is strictly decreasing and the optimal number of grid jobs corresponds to the maximal one. Such a configuration is represented on the bottom graph of figure 9.1 where $E_\Sigma(p)$ is plotted for a uniform distribution with $a = 700$s, $b = 1500$s and $w = 2000$s. If $w = b - a$, then $\frac{dE_\Sigma(p)}{dp} = -\frac{2pw+w}{p^2.(p+1)^2}$: it has no positive root and here again, the optimal number of grid jobs corresponds to the maximal one.

We can conclude from this particular example that the ratio $\frac{b-a}{w}$ plays a strong role into the optimization procedure. Actually, this ratio corresponds to a comparison between the standard-deviation ($\frac{b-a}{\sqrt{12}}$) of the grid latency and w, the execution time of the user job. We define the relative variability of the latency for the user job as the ratio $V = \frac{\sigma}{w}$ where σ is the standard-deviation of the latency. This parameter will play an important role in the following. For a uniform distribution, whatever the actual mean of R is, if V is low enough, then looking for an optimal user job partitioning is straight forward (maximum partitioning).

9.1.2 Gaussian distribution

If the distribution of the grid latency R is supposed to be Gaussian, with mean μ and standard deviation σ, then:

$$f_R(t) = \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right)$$
$$\text{and} \quad F_R(t) = \frac{1}{\sqrt{2\pi}\sigma}\int_{-\infty}^{t} exp\left(-\frac{(u-\mu)^2}{2\sigma^2}\right)du$$

Then the expectation of the makespan of the user job is:

$$E_\Sigma(p) = \int_{\mathbb{R}} pt\frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right)\left(\frac{1}{\sqrt{2\pi}\sigma}\int_{-\infty}^{t} exp\left(-\frac{(u-\mu)^2}{2\sigma^2}\right)du\right)^{p-1} dt + \frac{w}{p}$$

Minimizing $E_\Sigma(p)$ is hardly analytically feasible but a minimum can be estimated numerically. For example, figure 9.2 displays the evolution of $E_\Sigma(p)$ with respect to p for different values of the relative variability V ranging from 0.015 to 0.6 and with $\mu = 600$s and $\sigma = 300$s. It can be noticed from those graphs that the higher the relative variability V, the deeper the minimum of $E_\Sigma(p)$. One can here again conclude that the optimization procedure is particularly suitable for environments with a high variability with respect to the execution time w of the user job.

9.1.3 Experimental EGEE distribution

Figure 9.3 presents the granularity optimization of a 2000 seconds user job considering the experimental distribution measured on EGEE in section 6.2 of chapter 6. The expectation

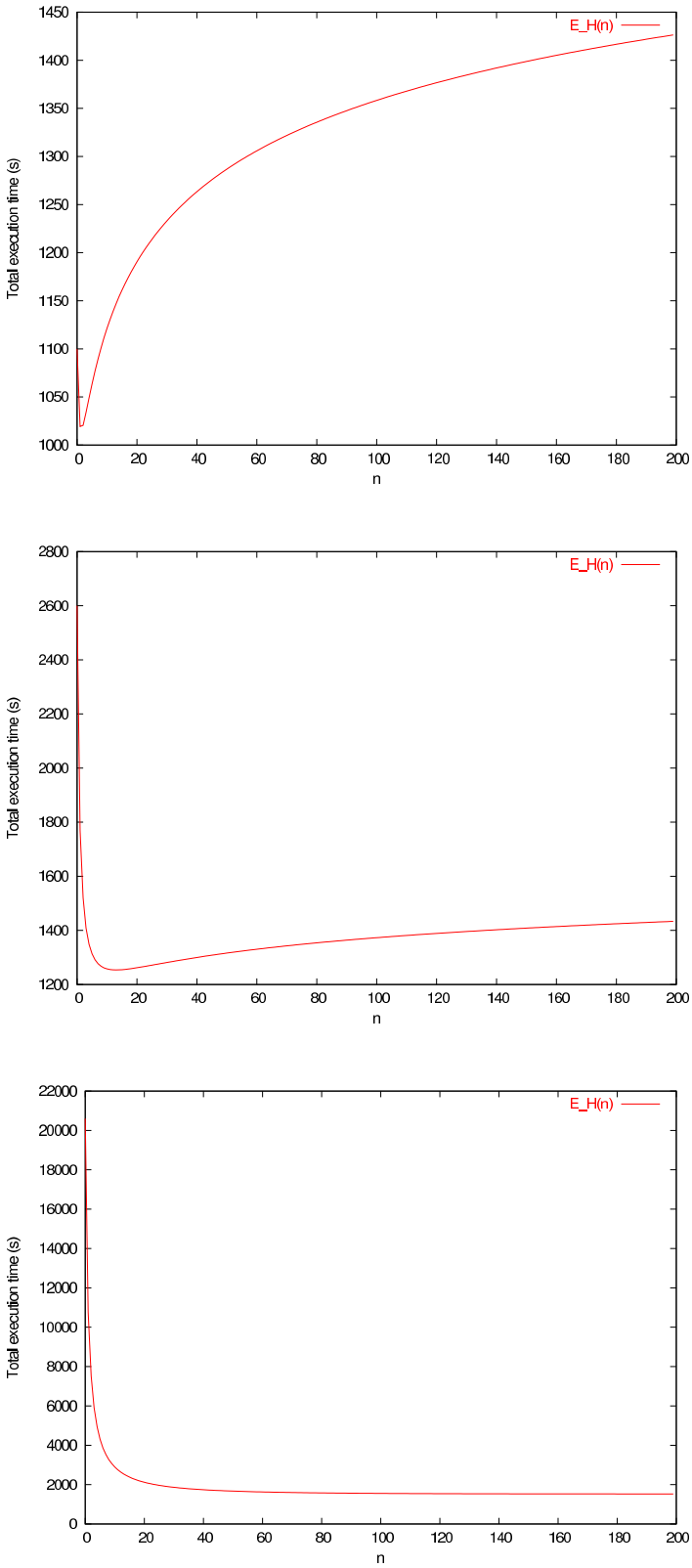

Figure 9.2: $E_\Sigma(p)$ for a Gaussian distribution with $\sigma = 300$s and $\mu = 600$s. From top to bottom: $V = 0.6$, $V = 0.15$ and $V = 0.015$

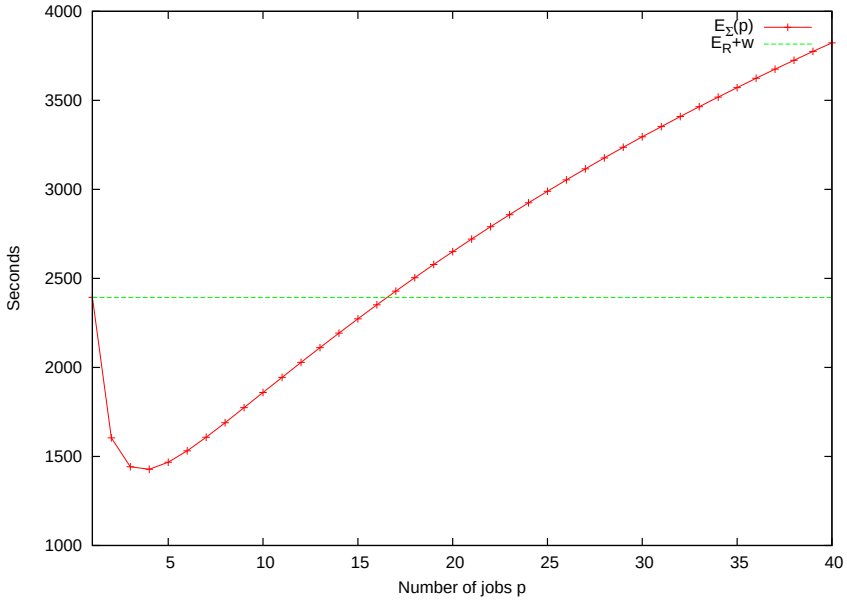

Figure 9.3: Evolution of the expected execution time of a 2000s user job with respect to the number of submitted grid jobs with the experimental distribution measured on EGEE (in section 6.2 of chapter 6). In this case, the optimal granularity is to submit 4 grid jobs of 500s. Increasing this number rapidly leads to a strong penalty.

E_R of this latency distribution is 393 seconds and it has been correctly fitted by a mixed log-normal / Pareto model. The expectation of the makespan of the user job (E_Σ) exhibits a global minimum reached for 4 submitted jobs, which corresponds to grid jobs of 8min 20s. After this optimum, the expected makespan is rapidly growing. Above 16 grid jobs, it is even higher than the time expected for the submission of a single 2000s grid job (green line). The minimum is deep enough (it is about 1000 seconds between the value of the expected time at the minimum and the green line) to conclude that the the optimization of the granularity could effectively be interesting on EGEE.

Going further in the theoretical analysis of the optimization of the granularity of a user job would for sure be an interesting perspective of this section. In particular, it could be worth studying the influence of the tail of the distribution of the grid latency on the existence of a global minimum. Indeed, it is an important parameter for the optimization of the timeout value presented in chapter 8 of this manuscript.

9.2 Experimental evaluation on EGEE

9.2.1 Distribution acquisition

Our strategy to optimize the granularity of a user job was assessed on EGEE. To do that, the optimal granularity was determined on-line, from an experimental distribution periodically up-

	Min	Max	Avg	Median
δ (seconds)	10	960	258.94	215
$\delta_{normalized}$	0.04	12.64	2.1	1.16

Table 9.1: Errors between the model and the measures.

dated through waves of dedicated probe jobs submitted to the infrastructure. Those jobs do not process anything and were used as probes to measure the grid latency. The main problem raised by this distribution acquisition is the fact that the status of the infrastructure may be disrupted by such a measure. Indeed, submitting waves of measure jobs would cause an additional load on the infrastructure, leading to inconsistent measures. To face this problem, a limited set of probe jobs was initially submitted and then, a new one was submitted each time a probe job completed, so that the total number of measure jobs running on the infrastructure remained constant, leading to a fixed perturbation.

Even if a grid provides a huge number of resources, thus theoretically allowing a large number of job submissions, the EGEE infrastructure is actually limited by the maximum number of simultaneous connections from the submission entity and the maximum number of grid jobs on the Resource Broker. The number of probe jobs was empirically tuned to 50, as a trade-off between the accuracy of the measure and the induced overhead on the submission system. It is true that this kind of method is quite unfair because it introduces a significant overload on the infrastructure. But ultimately, the middleware should provide to the users such statistics computed from all the submitted jobs so that the method would not be invasive.

Setting a timeout to grid jobs is required to avoid unreasonable waiting times because of outliers. Taking into account timed-out grid jobs into the optimization procedure would require to propose a fault-tolerant model handling grid job resubmissions and so on. This is investigated in chapter 8 and some remarks about the joint optimization of the granularity and the timeout value are done in section 9.3.2. Here, timed-out grid jobs are neglected, both in the measure scope and in the validation study. In those experiments, the timeout value was fixed to the total execution time w of the user job, so that timed-out grid jobs are the ones which would lead to a slowing down of the user job with respect to a sequential execution.

Once latency measures are acquired, the next step is to determine the pdf of the grid latency R. It was done by considering the 50 last probe measures and gathering them into 5 seconds bins. The computation and minimization of $E_\Sigma(p)$ is straightforward from equation 9.2. $E_\Sigma(p)$ is then computed for the granularity p ranging from 1 to a maximum value corresponding to the maximum number of grid jobs submittable to the infrastructure from a single user interface.

	Min	Max	Avg
Expected	0	671	162.5
Measured	-775	1308	198.1

Table 9.2: Time difference in seconds between maximal and optimal strategies (gain of the optimization).

9.2.2 Experimental set-up

Two experiments were conducted in order to evaluate the model on the EGEE infrastructure. First, the capability of the model to correctly predict the makespan of a user job was evaluated. The makespan of a user job was measured and compared to the value given by $E_\Sigma(p)$. The user job was composed of 30 grid jobs, 67 seconds long each, thus leading to a total execution time w of 2000 seconds.

Second, the performance obtained with the model (*optimal strategy*) was compared to the naive strategy consisting in submitting a maximal number of grid jobs (*maximal strategy*). A user job corresponding to a $w = 2000$s execution time was submitted, on the one hand using the optimal number of grid jobs resulting from the minimization of $E_\Sigma(p)$, and on the other hand using a fixed number of 30 grid jobs (this corresponds to the maximum number of grid jobs that we can submit concurrently on the infrastructure without hitting some performance loss). To avoid any bias resulting from an evolution of the grid status between the two submission processes, the two strategies were alternatively repeated up to 88 times, on various day times (mornings, afternoons, nights) spread over one week and using 3 different Resource Brokers.

9.2.3 Results

Experiment 1: model versus measures. On its upper line, table 9.1 shows statistics concerning the difference δ in seconds between the model prediction and the effective measure: $\delta = |\Sigma_{predicted} - \Sigma_{measured}|$. In order to quantify the accuracy of the model, this error was normalized with the predicted standard-deviation of the random variable Σ : $\delta_{normalized} = \frac{|\Sigma_{predicted} - \Sigma_{measured}|}{\sigma_\Sigma}$. On its lower line, the table shows the minimum, maximum, average and median of $\delta_{normalized}$. One can notice that the median ratio is close to 1, that is to say that the measured error is close to the standard-deviation of Σ, which is consistent. Thus, the model provides a good estimation of the makespan of a user job as well as a confidence interval on it.

Experiment 2: optimal strategy versus maximal strategy. Two different conclusions can be made from this experiment.

- Job saving: the total number of submitted grid jobs is 2580 for the maximal strategy and 1756 for the optimal one. The optimal strategy leads to a total saving of 824 grid jobs,

representing 32% of the grid jobs submitted with the maximal strategy.

- Time gain: table 9.2 shows statistics on the differences (in seconds) between the maximal and the optimal strategies. Those statistics have been computed over the 42% of the cases for which the optimal number of grid jobs differed from the maximal one. The remaining 58% correspond to cases for which the computed optimal number of grid jobs is superior or equal to the maximal value (30). One can notice that the average time gain yielded by our optimization strategy is about 200s, which represents 10% of the execution time w of the user job.

9.3 Extensions of the method

9.3.1 Taking into account outliers

The reader may wonder why outliers were not taken into account in this chapter, whereas their importance was highlighted in the previous one. Actually, the optimization of the granularity of a user job is not able to prevent an application from facing outliers. If we take outliers into account, then the probability for the makespan Σ of a user job corresponding to an execution time w composed of p grid jobs to be lower than a given value t is the probability for each grid job (i) to be an inlier (probability $1-\rho$) and (ii) to have a total execution time $R_i + \frac{w}{p}$ inferior to t. Therefore:

$$\begin{aligned} P(\Sigma < t) &= \prod_{i=1}^{p} (1-\rho) P\left(R_i + \frac{w}{p} < t\right) \\ &= (1-\rho)^p F_R\left(t - \frac{w}{p}\right)^p \end{aligned}$$

We can notice that $P(\Sigma = +\infty) > 0$. Consequently, if outliers are taken into account, then the expectation of Σ will be infinite whatever the number of submitted jobs. On the contrary, as noticed by equation 8.5 of section 8.1 of the previous chapter, setting a timeout value prevents the expectation of the latency faced by a user job to be infinite: after a sufficiently high (potentially infinite) number of resubmission, the user job will complete.

9.3.2 Joint timeout and granularity optimization

As shown above, setting a timeout value is the only strategy that guarantees that the application will not be impacted by outliers. Still, as it has be shown in this chapter, optimizing the job granularity provides significant performance gains. An interesting approach could be to set an optimal timeout value to each grid job resulting from the optimization of the user job granularity. Yet, there is no rationale to state that the timeout value of the grid jobs and the user job

granularity can be determined independently from each other. The timeout value and the granularity should probably be jointly optimized. To do that, the random variable $\Sigma = \max_{i=1...p} J_i$ has to be considered, where J_i are defined similarly as in equation 8.5 of chapter 8, yet including a wall-clock time depending on the granularity of the user job. For every t in $[nt_\infty, (n+1)t_\infty]$:

$$F_J(t) = 1 - q^n + q^n (1-\rho) F_R\left(t - nt_\infty - \frac{w}{p}\right) \quad \text{with} \quad q = 1 - (1-\rho) F_R\left(t_\infty - \frac{w}{p}\right)$$

The expectation of the makespan Σ of the user job can then be expressed according to the one of R:

$$E_\Sigma(p, t_\infty) = \sum_{n=0}^{+\infty} \int_{nt_\infty}^{(n+1)t_\infty} ptq^n f_R\left(t - nt_\infty - \frac{w}{p}\right)(1-\rho)\left[1 - q^n + q^n(1-\rho)F_R\left(t - nt_\infty - \frac{w}{p}\right)\right]^{p-1} dt$$

The minimization of E_Σ with respect to p and t_∞ and for various distributions of R could then be studied. The major interest to study this minimization is to be able to limit the impact of the outliers while optimizing the job granularity.

9.4 Conclusions

A strategy to optimize the job partitioning on a real grid infrastructure was proposed and studied in this chapter. The method was evaluated taking into account the dynamic and probabilistic nature of such an infrastructure by perpetually refreshing the pdf of the grid latency and minimizing the expectation of the makespan of the user job. Experimental results demonstrate that (i) a significant speed-up and (ii) a substantial grid job saving can be obtained using this method.

In practice, exploiting the methods presented in chapters 8 and 9 requires (i) to estimate of the distribution of the grid latency and (ii) the derivation of the optimal parameters values. The first issue cannot reasonably be addressed by an end-user. Collecting real-time statistics about the grid latency implies the submission of several probe jobs that may disturb the grid operation without any production usage of the resources. However, such an information should easily be available from logs of the grid workload management system. A production grid such as EGEE already includes a logging service which would be able to compute and update the cdf of the grid latency over time. Yet, more fundamental problems such as the handling of non-stationarities of the workload still remain.

Considering the whole grid as a black box characterized by the random variable capturing its latency seems to be a powerful approach. Important problems such as the ones presented in this part can be modeled in this way. The fundamental idea behind those models is that the submission of a grid job introduces a *risk* which is likely to impact the whole application. The strategies presented in this part aim at reducing this risk without limiting too much the parallelism of the application. A major interest of the resulting optimizations lies in the fact

that they only depend on the distribution of the grid latency, hiding the internal complexity of the grid. Several other optimizations may be targeted using the same approach. For instance, submitting redundant grid jobs [Casanova, 2006] or further grouping the services of a workflow would also reduce the risk associated to a user job.

Conclusions and future directions

1 Summary of the contributions

In this thesis, we studied the deployment and the results of an archetypal medical image analysis application on the EGEE production grid: the bronze standard. This is a precise and scalable method to assess the accuracy of rigid medical image registration. Results have demonstrated that this statistical procedure is powerful enough to detect subtle image impairments such as tilts of less than 2 degrees. It is also generic enough to assess the influence of external parameters such as the lossy compression ratio of the images. The bronze standard application exhibits requirements that characterize many medical image analysis applications. In particular, the need for computing power and algorithms and data sharing motivate the study of a grid workflow deployment of such applications. Based on a taxonomy of existing workflow approaches, we highlighted the suitability of service workflows for this scientific area. We thus presented the service workflow of the bronze standard application and we demonstrated that the selected Scufl language is expressive enough to describe similar applications.

Because of the lack of existing fully parallel service workflow enactor, we developped MOTEUR, a Scufl workflow enactor enabling all the parallelism levels that could be achieved on a grid. In this context, the handling of Scufl iteration strategies is not straightforward, because the order of data items going through the workflow is completely disturbed. We thus proposed a novel algorithm to deal with this problem. This development facilitates the execution of the application in comparable experimental conditions on several grid platforms. The application was considered as a grid benchmark and we compared its execution on a production grid versus on dedicated clusters. The execution was shown to be 4 times slower on production grids than on dedicated clusters. To further compare those two kinds of infrastructures, we proposed a multi-grids model which is able to determine, given an amount of CPU time to compute, the proportion of jobs to submit on each of those systems. We determined that the main cause of performance drop on production grids was the high latency that was imposed by such systems. Based on an appropriate model of the execution time of a workflow, the *variability* of the latency was demonstrated to have a strong impact on applications: in particular, it slows down the execution of the bronze standard application by a factor 2. This is one of the most important remark that guided the following developments.

Those conclusions rationalized the adoption of a probabilistic latency model that was used to develop strategies aiming at reducing the impact of the latency on the workflow. The adopted approach focuses on the modeling of the global behavior of the grid which is seen as a black box introducing a random latency on its jobs. In addition to a job grouping strategy aiming at reducing the mean latency faced by the application, we proposed two strategies based on such a probabilistic model: both the job granularity and the timeout value can be optimized by considering such a model and significant speed-up results were demonstrated. Moreover, the optimization of the timeout value allows to properly control the impact of the outliers on the application. All those strategies are based on a limitation of the number of grid jobs submitted by the applications: even if it may reduce the parallelism achieved by the application, reducing the number of jobs limits the risk to face outliers and high latencies. Further exploitating this statement seems to be a very promising way for a deeper understanding and optimization of production grids.

2 Future directions in grid workflows

Latency reduction strategies at a workflow-level. Latency reduction strategies proposed in chapters 8 and 9 are relying on probabilistic models based on statistical estimations of the grid latency. Even if they provide relevant strategies for optimizing the timeout value and the granularity of the jobs, they are still limited to the job-level or to very simple workflows (for the granularity). On the other hand, the probabilistic model of the makespan of a workflow presented in chapter 6 does not include the timeout and granularity parameters and thus cannot be used for optimization yet. Extending those probabilistic models to the case of a complete workflow (or integrating parameters to the workflow model) is a required perspective towards the application of those strategies to complete applications. Comparing job-level and workflow-level optimization results would also be interesting: determining whether the position of a job inside a whole workflow influences its submission parameters such as the timeout value and to what extent scaling up to the workflow-level could speed up the execution would motivate such investigations.

Probabilistic methods. Generally speaking, we believe that investigating probabilistic methods to optimize the execution of workflows is particularly tailored to production grids conditions. This could include optimization of other parameters of the workflow (such as the degree of redundant job submissions or the Resource Brokers to use) or even be extended to the introduction of variability in well-known algorithms such as scheduling ones. To properly tune the grid parameters of an application, one should keep in mind that the submission of a single grid job introduces a significant risk on the whole performance.

Scheduling of functional workflows. As detailed in chapter 2, functional workflows are characterized by the impossibility to predict before runtime the number of tasks that are going to be generated by the application. In this thesis, jobs generated by the workflow are submitted to the grid independently from each other: the scheduling is done at the job level only. It could be interesting to study (i) to what extent existing workflow-level scheduling algorithms could improve the performance of the application in production (in particular w.r.t latency reduction) and (ii) scheduling strategies dedicated to functional workflows. Current approaches for the scheduling of such partially unknown workflows are based on the clustering of the workflow into independent tasks-graphs that could be completely expressed and scheduled using classical heuristics. Nonetheless, taking into account `foreach`, `if` and other unpredictable constructs in the scheduling (for instance using branching prediction strategies or history information about services or simply expectation minimization) may improve the execution. Yet, workflow-level scheduling may still remain problematic in a Service-Oriented Architecture: if services are assumed to be completely black boxes (which guarantees an implementation independent description and eases their integration in an application), service developers could choose to hide the grid deployment of their applications (for instance for security reasons, or to keep the interface simple). In this case, the workflow engine cannot access the job submission system and it is consequently unable to perform any workflow-level scheduling. A very practical strategy to cope with this problem would be that the grid submission system learns the workflow topologies from the submitted jobs as suggested in [Shao et al., 2007]. Dependencies could be detected between jobs submitted by a given user (or even a community of users), thus detecting workflow being currently executed on the infrastructure.

3 Future directions in production grids modeling

Latency distribution monitoring. Our probabilistic methods would be completely useless without a satisfying statistical estimate of the grid latency. The latency reduction strategies presented in this thesis are based on a probabilistic model of the execution time according to the distribution function of the latency and allowing the optimization of a given parameter (the timeout value or the job granularity). Along this manuscript, many models have been envisaged for the distribution of the latency (mixed Log-Normal and Pareto, Gaussian, bi-uniform) and the nature of this distribution is not really known at that time. The grid remains a non-stationary system, constantly impacted by the load imposed by other users and by intrinsic evolutions (middleware updates, site connection/downtimes,...). Even if promising results have been obtained by a real-time monitoring of the latency in chapter 7, prediction errors remain high (200 seconds in the presented experiment). The use of such strategies in a real system first requires to be able to predict the latency that will be faced by a job with a reasonably low error. Several approaches could be envisaged in that purpose:

- Purely agnostic probabilistic approach (as done in this thesis): the latency is considered as a random variable. The (statistical) correlation of other variables is then investigated and modeled.
- Causes identification: a large-scale modelization of the grid middleware could help to understand what are some of the causes of the latency. In particular, determining to what extend the grid latency is due to external load and which part comes from intrinsic middleware design would be a great step towards a latency model. Simulations of the middleware or even benchmarks in controlled conditions could be done on experimental grids such as Grid'5000, under realistic load conditions injected from traces and logs obtained in production.
- Comparisons with other production grids to identify common patterns and features. Results remain specific to the EGEE grid yet. Comparing them to other production grids could help to characterize them more generically.

4 Future directions in service computing for medical image analysis applications

Medical image analysis application would for sure be impacted by the perspectives presented above. Yet, several initiatives could also be conducted to promote the use of grids in this particular scientific field. There is still a little adoption of grids among medical image analysis scientists. Even if such infrastructures have proven to be able to support science in many domains, they are still not a daily tool for the medical image analyst, as highlighted by the low number of papers using the grid in medical image computing conferences such as MICCAI (in 2007, the only paper mentioning the grid in its title is [Yang et al., 2007]). Yet, there is a quite strong algorithms sharing culture among the medical image analysis community, as demonstrated by the success of standard developments toolkits such as ITK[1], which is currently being interfaced with Condor grids[2]. The development of grid services repositories dedicated to medical image analysis could help to foster algorithms sharing one step further. Several use-cases could benefit from such repositories. For instance, one could tests one's data with the algorithm of other scientists or compare several of them on some data.

Even if such repositories seem technically feasible with the existing technologies, interoperability problems among the algorithms will arise (for instance data type conversions) and providing ontologies such as the one developed in [Gibaud et al., 2004] in Neuroimaging would probably be required to address such problems. Such ontologies would also allow a semantic browsing of such an algorithms repository, facilitating the finding of a suitable service for a

[1] http://www.itk.org/
[2] http://wiki.na-mic.org/Wiki/index.php/NAC_Grid_Enabled_ITK

given problem. Connecting such a medical image analysis services repository to a computing grid could then help to answer the potentially huge computing demand of such a sharing facility.

5 Future directions towards a clinical use of the grid

The question of the use of grids in a clinical context still remains a long term perspective to this work. Even if the solutions proposed in this manuscript are a step to bridge the gap between still low level grid middlewares and their usage for a daily medical activity, we remain completely aware that this ultimate goal would require further developments to reach an operational level quality of service and to cover all end users expectations. The technical part of the research towards this objective requires the development of stable and approved user-friendly interfaces completely hiding the grid to their users, as operating system now offer for hardware architectures. We believe that the development of MOTEUR (whose principle, design and applications are presented in chapters 1 to 5 of this thesis) is a step in this direction. Nevertheless, it remains that the immediate audience of this software are medical image scientists rather than clinicians. Such scientists can now benefit from it to significantly leverage the gridification of their applications. MOTEUR could also constitute a *part* of a future clinical system using the grid.

Another blocking point limiting the grid adoption by medical users is the fact that production grids are still not completely autonomous systems. In most cases, baby-sitting the grid is still required at some point of the execution. To address this problem, the methods presented in chapters 6 to 9 of this manuscript can be considered. We showed that they can improve fault-tolerance and guarantee a quality of service at a user-level, in spite of terrible highly variable execution conditions. The future directions reported in the previous sections of this conclusion reveal how they might still be developed.

Finally, we keep in mind that a clinical use of the grid would imply changes in clinical or even legal practices regarding patient data. However, it has to be recognized that the subsequent restrictions (for instance in terms of data/metadata storage locations or clinical computations performed outside of the hospitals) remain grounded by limitations of grid systems that are for instance unable to provide a sufficient security guarantees for medical applications for now, even if some solutions are under investigation [Montagnat et al., 2007, Erberich et al., 2007]. It keeps the window excitingly opened for further research in this domain.

Appendix A

Determination of the numerical values of the path of the workflow of figure 6.1

According to equation 6.5 the expectation of a path made from n_w services with runtimes r_i and iterated on n_D data items is, in DP mode:

$$E(\Sigma_{DP}) = \sum_{i<n_W} r_i + n_W n_D \int_{-\infty}^{\infty} t f_{R_{i,j}}(t) F_{R_{i,j}}(t)^{n_D-1} dt$$

In the example of figure 6.1, the latency is assumed to be Gaussian so that:

$$f_{R_{i,j}}(t) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right)$$

Thus:

$$E(\Sigma_{DP}) = \sum_{i<n_W} r_i + \frac{n_W n_D}{\left(\sqrt{2\pi}\sigma\right)^{n_D}} \int_{-\infty}^{\infty} t \exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right)\left(\int_{-\infty}^{t} \exp\left(-\frac{(u-\mu)^2}{2\sigma^2}\right) du\right)^{n_D-1} dt$$

The example assumes that n_D=3, μ=300s and σ=200s. We can then numerically compute that:

$$\frac{n_D}{\left(\sqrt{2\pi}\sigma\right)^{n_D}} \int_{-\infty}^{\infty} t \exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right)\left(\int_{-\infty}^{t} \exp\left(-\frac{(u-\mu)^2}{2\sigma^2}\right) du\right)^{n_D-1} dt = 468.8\text{s}$$

Along the blue path, we have n_W=2, r_0=150s and r_1=10s, so that $E(\Sigma_{DP})$=160+2 ×468.8=1098s. Similarly, along the red path, n_W=1 and r_0=600 so that $E(\Sigma_{DP})$=600+468.8=1069s.

Appendix B

Proofs of the timeout results of chapter 8

B.1 Expectation of J in the general case

We have:

$$\begin{aligned}
E_J(t_\infty) &= \int_0^\infty t f_J(t)dt = \sum_{n=0}^{\infty} \int_0^\infty t f_J^{[n,n+1]}(t)dt \\
&= (1-\rho)\sum_{n=0}^{\infty} q^n \int_{nt_\infty}^{(n+1)t_\infty} t f_R(t-nt_\infty)dt \\
&= (1-\rho)\sum_{n=0}^{\infty} q^n \int_0^{t_\infty} (u+nt_\infty) f_R(u)du \\
&= \frac{1-\rho}{1-q}\int_0^{t_\infty} u f_R(u)du + \frac{(1-\rho)qt_\infty}{(1-q)^2}\int_0^{t_\infty} f_R(u)du \\
&= \frac{1}{F_R(t_\infty)}\int_0^{t_\infty} u f_R(u)du + \frac{(1-\rho)\left(1-(1-\rho)F_R(t_\infty)\right)t_\infty}{(1-\rho)^2 F_R(t_\infty)^2} F_R(t_\infty) \\
&= \frac{1}{F_R(t_\infty)}\int_0^{t_\infty} u f_R(u)du + \frac{(1-(1-\rho)F_R(t_\infty))t_\infty}{(1-\rho)F_R(t_\infty)} \\
&= \frac{1}{F_R(t_\infty)}\int_0^{t_\infty} u f_R(u)du + \frac{t_\infty}{(1-\rho)F_R(t_\infty)} - t_\infty
\end{aligned}$$

B.2 Limits of E_J

$$\lim_{t_\infty \to +\infty} \frac{1}{F_R(t_\infty)}\int_0^{t_\infty} u f_R(u)du = \int_0^{+\infty} u f_R(u)du = E_R$$

And, when $\rho \neq 0$:

$$\lim_{t_\infty \to +\infty} \frac{t_\infty}{(1-\rho)F_R(t_\infty)} - t_\infty = +\infty$$

Whereas when $\rho = 0$:

$$\begin{aligned} \frac{t_\infty}{(1-\rho)F_R(t_\infty)} - t_\infty = \frac{t_\infty(1-F_R(t_\infty)}{F_R(t_\infty)} &= \frac{t_\infty \int_{t_\infty}^{\infty} f_R(u)du}{F_R(t_\infty)} \\ \text{Thus,} \quad \frac{t_\infty}{(1-\rho)F_R(t_\infty)} - t_\infty &\leq \frac{\int_{t_\infty}^{\infty} u\, f_R(u)du}{F_R(t_\infty)} \\ \text{So that } \lim_{t_\infty \to +\infty} \left(\frac{t_\infty}{(1-\rho)F_R(t_\infty)} - t_\infty \right) &= 0 \end{aligned}$$

B.3 Distributions for which the timeout value does not impact E_J when $\rho = 0$

Let F be the cdf of a distribution which has this property. Then:

$$\begin{aligned} &\forall n \in \mathbb{N}, \forall t_\infty \in \mathbb{R}^+, \\ &1 - (1 - F(t_\infty))^n + (1 - F(t_\infty))^n F(t - nt_\infty) = F(t) \\ &\Rightarrow \forall n \in \mathbb{N}, \forall t_\infty \in \mathbb{R}^+, \\ &G(t_\infty)^n G(t - nt_\infty) = G(t) \quad \text{with} \quad G = 1 - F \\ &\Rightarrow \forall n \in \mathbb{N}, \forall t_\infty \in \mathbb{R}^+, \\ &nH(t_\infty) + H(t - nt_\infty) = H(t) \quad \text{with} \quad H = ln\,(G) \\ &\Rightarrow \forall n \in \mathbb{N}, \forall t_\infty \in \mathbb{R}^+, \\ &H'(t - nt_\infty) = H'(t) \end{aligned}$$

H' is thus periodic, with period nt_∞ for every t_∞ and every n. It is thus constant and we have $H'(t) = \alpha$. Thus, $H(t) = \alpha t + \beta$. We thus have $F(t) = 1 - e^{\beta} e^{\alpha t}$. Moreover, the limit of $F(t)$ has to be 1 when $t \to +\infty$, so that α<0 and $\beta = 0$, which demonstrates that the distribution of R has to be exponential.

B.4 Behavior of E_J in the Weibull case without outliers

We have:

$$\begin{aligned} &\forall n \in \mathbb{N}, \forall t \in [nt_\infty, (n+1)t_\infty], \\ F_J(t) &= 1 - q^n + q^n F_R(t - nt_\infty) \\ &= 1 - e^{-n\left(\frac{t_\infty}{\lambda}\right)^k} + e^{-n\left(\frac{t_\infty}{\lambda}\right)^k} \left(1 - e^{-\left(\frac{t-nt_\infty}{\lambda}\right)^k}\right) \\ &= 1 - e^{-\left(n\left(\frac{t_\infty}{\lambda}\right)^k + \left(\frac{t-nt_\infty}{\lambda}\right)^k\right)} \end{aligned}$$

We are going to compare F_J and F_R. To do that, we will actually compare $\ln\left(\frac{1}{1-F_J}\right)$ and $\ln\left(\frac{1}{1-F_R}\right)$. The comparison resumes to study the sign of the following function $f_{t_\infty,n}$, for every n and every t_∞ ($f_{t_\infty,n} = \ln\left(\frac{1}{1-F_J}\right) - \ln\left(\frac{1}{1-F_R}\right)$):

$$\begin{aligned} & \forall n \in \mathbb{N}, \forall t_\infty > 0, \\ f_{t_\infty,n} : [nt_\infty, (n+1)t_\infty] & \rightarrow \mathbb{R} \\ t & \mapsto n\left(\frac{t_\infty}{\lambda}\right)^k + \left(\frac{t-nt_\infty}{\lambda}\right)^k - \left(\frac{t}{\lambda}\right)^k \end{aligned}$$

If $f_{t_\infty,n}$ is positive, then setting a timeout improves the execution. The derivative of $f_{t_\infty,n}$ with respect to t is: $f'_{t_\infty,n}(t) = \frac{k}{\lambda}\left(\left(\frac{t-nt_\infty}{\lambda}\right)^{k-1} - \left(\frac{t}{\lambda}\right)^{k-1}\right)$.

If $k > 1$: $f_{t_\infty,n}(nt_\infty) = \left(\frac{t_\infty}{\lambda}\right)^k \left(n - n^k\right) < 0$ and $\forall t \in [nt_\infty, (n+1)t_\infty]$, $f'_{t_\infty,n}(t) < 0$. $f_{t_\infty,n}(t)$ is thus negative on this interval and we have:

$$\begin{aligned} & \forall t > 0, \forall t_\infty > 0, \\ \ln\left(\frac{1}{1-F_J(t)}\right) - \ln\left(\frac{1}{1-F_R(t)}\right) & < 0 \\ \text{thus,} \quad F_J(t) & < F_R(t) \end{aligned}$$

It proves that every timeout value penalizes the execution. The timeout thus has to be infinite.

If $k < 1$: $f_{t_\infty,n}(nt_\infty) = \left(\frac{t_\infty}{\lambda}\right)^k \left(n - n^k\right) > 0$ and $\forall t \in [nt_\infty, (n+1)t_\infty]$, $f'_{t_\infty,n}(t) > 0$. $f_{t_\infty,n}(t)$ is thus positive on this interval and we have:

$$\begin{aligned} & \forall t > 0, \forall t_\infty > 0, \\ \ln\left(\frac{1}{1-F_J(t)}\right) - \ln\left(\frac{1}{1-F_R(t)}\right) & > 0 \\ \text{thus,} \quad F_J(t) & > F_R(t) \end{aligned}$$

Moreover, $n\left(\frac{t_\infty}{\lambda}\right)^k + \left(\frac{t-nt_\infty}{\lambda}\right)^k$ is decreasing with respect to t_∞. Thus, $\forall t_\infty > 0, \forall t'_\infty > t_\infty$, $F_{J,t'_\infty}(t) < F_{J,t_\infty}(t)$ for every t. The optimal timeout value is thus 0.

B.5 Expression of $E_J(t_\infty)$ in the truncated Gaussian case

According to equation 8.6,

$$\begin{aligned} E_J(t_\infty) &= \frac{1}{F_R(t_\infty)} \int_0^{t_\infty} u f_R(u) du + \frac{t_\infty}{(1-\rho)F_R(t_\infty)} - t_\infty \\ &= \frac{1}{\Phi\left(\frac{\mu}{\sigma}\right) - \Phi\left(\frac{\mu-t_\infty}{\sigma}\right)} \frac{1}{\sqrt{2\pi}\sigma} \int_0^{t_\infty} u e^{-\frac{1}{2}\left(\frac{u-\mu}{\sigma}\right)^2} du + \frac{t_\infty \Phi\left(\frac{\mu}{\sigma}\right)}{(1-\rho)\left(\Phi\left(\frac{\mu}{\sigma}\right) - \Phi\left(\frac{\mu-t_\infty}{\sigma}\right)\right)} - t_\infty \end{aligned}$$

Moreover,

$$\int_0^{t_\infty} u e^{-\frac{1}{2}\left(\frac{u-\mu}{\sigma}\right)^2} du = \sigma^2 \int_{-\frac{\mu}{\sigma}}^{\frac{t_\infty-\mu}{\sigma}} v e^{\left(-\frac{1}{2}v^2\right)} dv + \sigma\mu \int_{-\frac{\mu}{\sigma}}^{\frac{t_\infty-\mu}{\sigma}} e^{\left(-\frac{1}{2}v^2\right)} dv$$
$$= \sigma^2 \sqrt{2\pi}\left(\phi\left(\frac{\mu}{\sigma}\right) - \phi\left(\frac{\mu - t_\infty}{\sigma}\right)\right) + \sigma\mu\sqrt{2\pi}\left(\Phi\left(\frac{\mu}{\sigma}\right) - \Phi\left(\frac{\mu - t_\infty}{\sigma}\right)\right)$$

Thus,

$$E_J(t_\infty) = \mu + \sigma\frac{\phi\left(\frac{\mu}{\sigma}\right) - \phi\left(\frac{\mu-t_\infty}{\sigma}\right)}{\Phi\left(\frac{\mu}{\sigma}\right) - \Phi\left(\frac{\mu-t_\infty}{\sigma}\right)} + \frac{1}{1-\rho}t_\infty\left(\frac{\Phi\left(\frac{\mu-t_\infty}{\sigma}\right)}{\Phi\left(\frac{\mu}{\sigma}\right) - \Phi\left(\frac{\mu-t_\infty}{\sigma}\right)} + \rho\right)$$

B.6 Behavior of $E_J(t_\infty)$ in the truncated Gaussian case

Let us consider the following transformed variables:

$$v_\infty = \frac{\mu - t_\infty}{\sigma} \quad \text{and} \quad \lambda = \frac{\mu}{\sigma}$$

According to equation 8.11, we then have:

$$\frac{E_J(v_\infty)}{\sigma} = \lambda + \frac{\phi(\lambda) - \phi(v_\infty)}{\Phi(\lambda) - \Phi(v_\infty)} + (\lambda - v_\infty)\left(\frac{\Phi(v_\infty)}{(1-\rho)(\Phi(\lambda) - \Phi(v_\infty))} + \frac{\rho}{1-\rho}\right)$$

To study the behavior of E_J, we are going to consider the function $f^{(1)}(v_\infty) = \frac{(1-\rho)(\Phi(\lambda)-\Phi(v_\infty))^2}{\sigma}\frac{\partial E_J(v_\infty)}{\partial v_\infty}$ which has the same sign as $\frac{\partial E_J(v_\infty)}{\partial v_\infty}$. We are going to show that the third derivative of $f^{(1)}$ is positive. We will then be able to study the sign of $f^{(1)}$. We have:

$$\begin{aligned} f'(v_\infty) &= \phi(v_\infty)\left[k(\lambda) - k(v_\infty) + \rho v_\infty(\Phi(v_\infty) - \Phi(\lambda)) + \rho(\phi(v_\infty) - \phi(\lambda))\right] \\ &+ \Phi(v_\infty)(\Phi(v_\infty) - \Phi(\lambda)) - \rho(\Phi(\lambda) - \Phi(v_\infty))^2 \\ \text{with} \quad & k(v) = v\Phi(v) + \phi(v) \end{aligned}$$

k is actually a primitive of Φ and is thus increasing. Derivating f' with respect to v, we obtain:

$$\begin{aligned} f^{(2)}(v) &= \phi(v)g^{(2)}(v) \\ \text{with } g^{(2)}(v) &= v(k(v) - k(\lambda)) + \rho v^2(\Phi(\lambda) - \Phi(v)) \\ &\quad +\rho v(\phi(\lambda) - \phi(v)) + (1-\rho)(\Phi(v) - \Phi(\lambda)) \end{aligned}$$

$f^{(2)}$ has the same sign of $g^{(2)}$. By successive derivation of this function, we obtain:

$$\begin{aligned} g^{(3)}(v) &= v\Phi(v)(1-2\rho) + \phi(v)(1-2\rho) + k(v) - k(\lambda) + 2\rho v\Phi(\lambda) + \rho\phi(\lambda) \\ &\text{and} \\ g^{(4)}(v) &= 2(1-\rho)\Phi(v) + 2\rho\Phi(\lambda) \end{aligned}$$

$g^{(4)}(v)$ is thus positive on $]-\infty, \lambda]$, so that $g^{(3)}$ is increasing on this interval. Moreover, $g^{(3)}(-\infty) = -\infty$ and $g^{(3)}(\lambda) = \lambda\Phi(\lambda) + (1-\rho)\phi(\lambda) > 0$. $g^{(3)}$ thus has a single root v_0 on $]-\infty, \lambda]$. $g^{(3)}$ is negative for $v < v_0$ and positive otherwise.

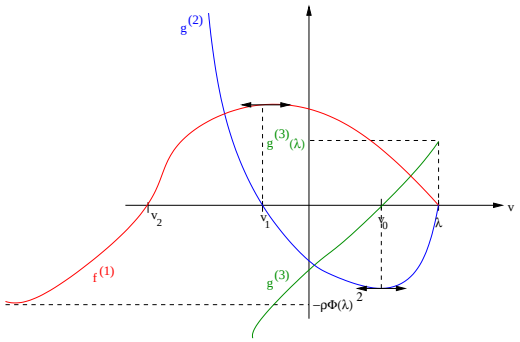

Figure B.1: Behavior of $f^{(1)}$, $g^{(2)}$ and $g^{(3)}$ derivatives of $E_J(v_\infty)$ in the truncated Gaussian case.

$g^{(2)}$ is thus decreasing on $]-\infty, v_0]$ and increasing on $[v_0, \lambda]$. Moreover, $g^{(2)}(-\infty) = +\infty$ and $g^{(2)}(\lambda) = 0$. $g^{(2)}$ thus has a single root v_1 on $]-\infty, \lambda]$ ($v_1 < v_0$). $g^{(2)}$ is positive for $v < v_1$ and negative otherwise.

$f^{(1)}$ is thus increasing on $]-\infty, v_1]$ and decreasing on $[v_1, \lambda]$ Moreover, $f^{(1)}(\lambda) = 0$ and $f^{(1)}(-\infty) = -\rho\Phi(\lambda)$. Two cases then have to be studied:

1. $\rho = 0$: in this case, $f^{(1)}(-\infty) = 0$. $f^{(1)}$ is thus positive on $]-\infty, \lambda]$.

2. $\rho \neq 0$: in this case, $f^{(1)}(-\infty) < 0$. $f^{(1)}$ thus has a single root v_2 on $]-\infty, \lambda]$ ($v_2 < v_1$). $f^{(1)}$ is negative for $v < v_2$ and positive otherwise.

The behavior of $f^{(1)}$, $g^{(2)}$ and $g^{(3)}$ is plotted on figure B.6.

We can then conclude on the behavior of $E_J(t_\infty)$ by noticing that:

$$\begin{aligned} \frac{\partial E_J(t_\infty)}{t_\infty} &= \frac{\partial E_J(v_\infty)}{v_\infty}\frac{\partial v_\infty}{t_\infty} \\ &= -\frac{1}{\sigma}\frac{\partial E_J(v_\infty)}{v_\infty} \end{aligned}$$

The two cases described above thus resume to:

1. $\rho = 0$: $E_J(t_\infty)$ is decreasing on $[0, +\infty[$

2. $\rho \neq 0$: $E_J(t_\infty)$ is decreasing on $[0, t_2]$ and increasing on $[t2, +\infty[$ (with $t_2 = \mu - \sigma v_2$).

B.7 Expression of E_J in the log-normal case

If we consider the transformed variable $x_\infty = \frac{\ln(t_\infty)-\mu}{\sigma}$, then equation 8.6 gives:

$$E_J(x_\infty) = \frac{1}{\Phi(x_\infty)}\int_0^{e^{\sigma x_\infty+\mu}} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{\ln(u)-\mu}{\sigma}\right)^2} du + e^{\sigma x_\infty+\mu}\left(\frac{1}{(1-\rho)\Phi(x_\infty)} - 1\right)$$

If we then perform the variable change $v = \frac{\ln(u)-\mu}{\sigma} - \sigma$ in the integral term, we obtain:

$$E_J(x_\infty) = \frac{1}{\Phi(x_\infty)} e^{\mu+\frac{\sigma^2}{2}} \Phi(x_\infty - \sigma) + e^{\sigma x_\infty + \mu} \left(\frac{1}{(1-\rho)\Phi(x_\infty)} - 1 \right)$$

B.8 Behavior of E_J in the Pareto case

Let us consider the transformed variable $Z = \frac{a+t_\infty}{a}$. We then have, according to equation 8.14 and after some manipulations:

$$\frac{E_J(Z)}{a} = \frac{\frac{1}{1-\nu}(Z - Z^\nu) + \frac{\rho}{1-\rho}(Z^{\nu+1} - Z^\nu)}{Z^\nu - 1}$$

By derivation, we then have:

$$\begin{aligned} \frac{(Z^\nu - 1)^2}{a} \frac{\partial E_J(Z)}{\partial Z} &= \frac{1}{1-\nu} \left((1-\nu)Z^\nu + \nu Z^{\nu-1} - 1 \right) \\ &\quad + \frac{\rho}{1-\rho} Z^{\nu-1} \left(Z^{\nu+1} - (\nu+1)Z + \nu Z \right) \end{aligned}$$

A(Z)=$(1-\nu)Z^\nu + \nu Z^{\nu-1} - 1$ is negative, for every ν>1 and for every Z >1. Indeed, A(1)=0 and A'(Z)=$\nu(1-\nu)Z^{\nu-2}(Z-1)$ is negative. Moreover, B(Z)=$Z^{\nu+1} - (\nu+1)Z + \nu Z$ is positive, for every ν>1 and for every Z >1. Indeed, B(1)=0 and B'(Z)=$(\nu+1)(Z-1)$ is positive. $\frac{\partial E_J(Z)}{\partial Z}$ is thus positive, as well as $\frac{\partial E_J(t_\infty)}{\partial t_\infty}$ and E_J is thus increasing.

B.9 Properties of Φ and link with erf

$$\begin{aligned} \phi(t) &= \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}t^2} \\ \phi'(t) &= -t\phi(t) \\ \Phi(-t) &= 1 - \Phi(t) \\ erf(t) &= 2\Phi(\sqrt{2}t) - 1 \\ \Phi(t) &= \frac{1}{2} + \frac{1}{2} erf\left(\frac{t}{\sqrt{2}} \right) \end{aligned}$$

Bibliography

[Agerwala, 1974] Agerwala, T. (1974). A complete model for representing the coordination of asynchronous processes. Technical Report 32, John Hopkins University, Hopkins Computer Science Program, Baltimore.

[Ajmone Marsan et al., 1984] Ajmone Marsan, M., Conte, G., and Balbo, G. (1984). A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Transactions on Computer Systems*, 2(2):93–122.

[Amar et al., 2006] Amar, A., Bolze, R., Bouteiller, A., Chis, A., Caniou, Y., Caron, E., Kaur Chouan, P., Le Mahec, G., Dail, H., Depardon, B., Desprez, F., Gay, J.-S., and Su, A. (2006). DIET: New Developments and Recent Results. Technical Report RR-6027, INRIA Rhône-Alpes.

[Antonio et al., 2004] Antonio, B., Canal, C., Pimentel, E., and Vallecillo, A. (2004). Formalizing Web Services Choreographies. *Electronic Notes in Theoretical Computer Science*, 105:73–94.

[Arbel et al., 2004] Arbel, T., Morandi, X., M Comeau, R., and Louis Collins, D. (2004). Automatic non-linear MRI-ultrasound registration for the correction of intra-operative brain deformations. *Comput Aided Surg*, 9(4):123–36.

[Armstrong et al., 1999] Armstrong, R., Gannon, D., Geist, A., Keahey, K., Kohn, S., McInnes, L., Parker, S., and Smolinski, B. (1999). Toward a Common Component Architecture for High-Performance Scientific Computing. In *Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC'99)*, pages 115–124.

[Arsigny et al., 2005] Arsigny, V., Pennec, X., and Ayache, N. (2005). Polyrigid and Polyaffine Transformations: a Novel Geometrical Tool to Deal with Non-Rigid Deformations - Application to the registration of histological slices. *Medical Image Analysis (MedIA)*, 9(6):507–523.

[Atkinson et al., 2007] Atkinson, C., Bostan, P., Hummel, O., and Stoll, D. (2007). A Practical Approach to Web Service Discovery and Retrieval. In *International Conference on Web-Services (ICWS'07)*, Salt-Lake City, Utah, USA.

[Baduel et al., 2006] Baduel, L., Baude, F., Caromel, D., Contes, A., Huet, F., Morel, M., and Quilici, R. (2006). *Grid Computing: Software Environments and Tools*, chapter Programming, Deploying, Composing, for the Grid. Springer-Verlag.

[Bankman, 2000] Bankman, I. (2000). *Handbook of Medical Imaging*. Academic Press.

[Barais et al., 2006] Barais, O., Lawall, J., Le Meur, A.-F., and Duchien, L. (2006). Safe Integration of New Concerns in a Software Architecture. In *13th Annual IEEE International Conference on Engineering of Computer Based Systems (ECBS'06)*, Potsdam, Germany. IEEE.

[Barga and Gannon, 2007] Barga, R. and Gannon, D. (2007). *Scientific versus Business Workflows*, chapter 2, pages 9–16. In [Taylor et al., 2007].

[Baynat, 2000] Baynat, B. (2000). *Théorie des files d'attente*. Hermes.

[Benoit-Cattin et al., 2005] Benoit-Cattin, H., Collewet, G., Belaroussi, B., Saint-Jalmes, H., and Odet, C. (2005). The SIMRI project : a versatile and interactive MRI simulator. *Journal of Magnetic Resonance Imaging (JMRI)*, (173):97–115.

[Blanquer Espert et al., 2005] Blanquer Espert, I., Hernández García, V., and Segrelles Quilis, J. (2005). Creating Virtual Storages and Searching DICOM Medical Images through a GRID Middleware based in OGSA. *Journal of Clinical Monitoring and Computing*, 19(4-5):295–305.

[Blay-Fornarino et al., 2004] Blay-Fornarino, M., Charfi, A., Emsellem, D., Pinna-Déry, A.-M., and Riveill, M. (2004). Software interaction. *Journal of Object Technology (ETH Zurich)*, 3(10):161–180.

[Bobrow et al., 1988] Bobrow, D. G., DeMichiel, L. G., Gabriel, R. P., Keene, S. E., Kiczales, G., and Moon, D. A. (1988). Common Lisp Object System specification. *ACM SIGPLAN Notices*, 23:1–142.

[Bowers et al., 2006] Bowers, S., McPhillips, T., Ludäscher, B., Cohen, S., and Davidson, S. (2006). A Model for User-Oriented Data Provenance in Pipelined Scientific Workflows. In *International Provenance and Annotation Workshop (IPAW)*, LNCS.

[Box, 1997] Box, D. (1997). *Essential COM*. Addison-Wesley Longman Publishing, Boston, MA, USA.

[Brookshier et al., 2002] Brookshier, D., Govoni, D., Govoni, D., Krishnan, N., and Soto, J. (2002). *JXTA: Java P2P Programming*. Sams publishing, Indianapolis.

[Bruneton et al., 2004] Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., and Stefani, J.-B. (2004). An open component model and its support in java. In *CBSE*, pages 7–22.

[Buyya and Murshed, 2002] Buyya, R. and Murshed, M. (2002). GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *Concurrency and Computation: Practice and Experience (CCPE)*, 14(13-15):1175–1220.

[Capit et al., 2005] Capit, N., Da Costa, G., Georgiou, Y., Huard, G., and Marti, C. (2005). A batch scheduler with high level components. In *Cluster computing and Grid 2005 (CCGrid'05)*, volume 2, pages 776– 783.

[Cappello and Bal, 2007] Cappello, F. and Bal, H. (2007). Towards an International Computer Science Grid. In *7th IEEE Int. Symp. on Cluster Computing and the Grid (CCGrid'07)*, pages 3–12, Rio de Janeiro, Brazil.

[Cappello et al., 2005] Cappello, F., Desprez, F., Dayde, M., Jeannot, E., Jegou, Y., Lanteri, S., Melab, N., Namyst, R., Vicat-Blanc Primet, P., Richard, O., Caron, E., Leduc, J., and Mornet, G. (2005). Grid'5000: A Large Scale, Reconfigurable, Controlable and Monitorable Grid Platform. In *6th IEEE/ACM International Workshop on Grid Computing (Grid'2005)*, Seattle, Washington, USA.

[Caromel et al., 2006] Caromel, D., Delbe, C., Costanzo, A., and Leyton, M. (2006). Proactive: an integrated platform for programming and running applications on grids and p2p systems. *Computational Methods in Science and Technology*, (12).

[Caromel et al., 2008] Caromel, D., Henrio, L., and Leyton, M. (2008). Type safe algorithmic skeletons. In *Proceedings of the 16th Euromicro International Conference on Parallel, Distributed and network-based Processing*, Toulouse, France. To appear.

[Caromel and Leyton, 2007] Caromel, D. and Leyton, M. (2007). Fine tuning algorithmic skeletons. In *13th International Euro-par Conference: Parallel Processing*, volume 4641, pages 72–81.

[Caron and Dail, 2005] Caron, E. and Dail, H. (2005). GoDIET: a tool for managing distributed hierarchies of DIET agents and servers. Technical Report RR-5520, Institut National de Recherche en Informatique et en Automatique (INRIA).

[Caron and Desprez, 2005] Caron, E. and Desprez, F. (2005). DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid. *International Journal of High Performance Computing Applications*.

[Casanova, 2001] Casanova, H. (2001). Simgrid: A Toolkit for the Simulation of Application Scheduling. In *1st International Symposium on Cluster Computing and the Grid (CCGrid'01)*, page 430.

[Casanova, 2006] Casanova, H. (2006). On the Harmfulness of Redundant Batch Requests. In *15th IEEE International Symposium on High Performance Distributed Computing (HPDC'06)*, pages 255–266, Paris, France.

[Casanova and Dongarra, 1997] Casanova, H. and Dongarra, J. (1997). NetSolve: A Network-Enabled Server for Solving Computational Science Problems. *International Journal of High Performance Computing and Applications (IJHPCA)*, 11(3):212–223.

[Casanova et al., 2003] Casanova, H., Legrand, A., and Marchal, L. (2003). Scheduling Distributed Applications: the SimGrid Simulation Framework. In *Proceedings of the Third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)*. IEEE Computer Society Press.

[Cheung-Foo-Wo et al., 2006] Cheung-Foo-Wo, D., Tigli, J.-Y., Lavirotte, S., and Riveill, M. (2006). Wcomp: a Multi-Design Approach for Prototyping Applications using Heterogeneous Resources. In *17th IEEE International Workshop on Rapid System Prototyping (RSP)*, Chania, Crete.

[Chrétienne et al., 1995] Chrétienne, P., Coffman, E., Lenstra, J., and Liu, Z. (1995). *Scheduling theory and its applications*. John Wiley and Sons.

[Cieslak et al., 2006] Cieslak, D., Thain, D., and Chawla, N. (2006). Troubleshooting Distributed Systems via Data Mining. In *IEEE International Symposium on High Performance Distribut ed Computing (HPDC'06)*, Paris, France.

[Cole, 1991] Cole, M. (1991). *Algorithmic skeletons: structured management of parallel computation*. MIT press, Cambridge, MA, USA.

[Commowick and Malandain, 2006] Commowick, O. and Malandain, G. (2006). Evaluation of Atlas Construction Strategies in the Context of Radiotherapy Planning. In *SA2PM Workshop (From Statistical Atlases to Personalized Models)*, Copenhagen.

[Commowick et al., 2005] Commowick, O., Stefanescu, R., Fillard, P., Arsigny, V., Ayache, N., Pennec, X., and Malandain, G. (2005). Incorporating Statistical Measures of Anatomical Variability in Atlas-to-Subject Registration for Conformal Brain Radiotherapy. In *MICCAI 2005, Part II*, pages 927–934, Palm Springs, CA, USA. Springer Verlag.

[Deelman et al., 2003] Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Blackburn, K., Lazzarini, A., Arbree, A., Cavanaugh, R., and Koranda, S. (2003). Mapping

Abstract Complex Workflows onto Grid Environments. *Journal of Grid Computing (JGC)*, 1(1):9–23.

[Delaitre et al., 2005] Delaitre, T., Kiss, T., Goyeneche, A., Terstyanszky, G., Winter, S., and Kacsuk, P. (2005). GEMLCA: Running Legacy Code Applications as Grid Services. *Journal of Grid Computing (JGC)*, 3(1-2).

[Delannoy et al., 2006] Delannoy, O., Emad, N., and Petiton, S. (2006). Workflow Global Computing with YML. In *7th IEEE/ACM International Conference on Grid Computing*, pages 25–32, Barcelona, Spain.

[Delannoy and Petiton, 2004] Delannoy, O. and Petiton, S. (2004). A Peer to Peer Computing Framework ; Design and Performance Evaluation of YML. In *Third International Workshop on Algorithms, Models, and Tools for Parallel Computing on Heterogeneous Networks*, pages 362– 369.

[Duan et al., 2006] Duan, R., Prodan, R., and Fahringer, T. (2006). Data Mining-based Fault Prediction and Detection on the Grid. In *Proceedings of the IEEE International Symposium on High Performance Distributed Computing (HPDC-15)*, pages 305–308, Paris, France.

[Dugas-Phocion, 2006] Dugas-Phocion, G. (2006). *Segmentation d'IRM Cérébrales Multi-Séquences et Application à la Sclérose en Plaques*. Thèse de sciences, École des Mines de Paris.

[Emmerich et al., 2005] Emmerich, W., Butchart, B., Chen, L., Wassermann, B., and Price, S. (2005). Grid Service Orchestration Using the Business Process Execution Language (BPEL). *Journal of Grid Computing (JGC)*, 3(3-4):283 – 304.

[Erberich et al., 2007] Erberich, S., Silverstein, J., Chervenak, A., Schuler, R., Nelson, M., and Kesselman, C. (2007). Globus MEDICUS - Federation of DICOM Medical Imaging Devices into Healthcare Grids. *Studies in Health Technology and Informatics*, 126:269–278.

[Fahringer et al., 2007] Fahringer, T., Prodan, R., Duan, R., Hofer, J., Nadeem, F., Nerieri, F., Podlipnig, S., Qin, J., Siddiqui, M., Truong, H.-L., Villazon, A., and Wieczorek, M. (2007). *ASKALON: a development and grid computing environment for scientific workflows*, chapter 27, pages 450–471. In [Taylor et al., 2007].

[Feitelson, 2002] Feitelson, D. (2002). *Workload modeling for performance evaluation*, pages 114–141. Springer-Verlag - LNCS vol 2459.

[Feitelson, 2003] Feitelson, D. (2003). Metric and workload effects on computer systems evaluation. *Computer*, 36(9):18–25.

[Feitelson et al., 2004] Feitelson, D., Rudolph, L., and Schwiegelshohn, U. (2004). Parallel Job Scheduling – A Status Report. In *10th Workshop on Job Scheduling Strategies for Parallel Processing*, New-York, NY, USA.

[Foster, 2005] Foster, I. (2005). Globus Toolkit Version 4: Software for Service-Oriented Systems. In *International Conference on Network and Parallel Computing (IFIP)*, volume 3779, pages 2–13. Springer-Verlag LNCS.

[Foster and Kesselman, 1997] Foster, I. and Kesselman, C. (1997). Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128.

[Foster et al., 2002] Foster, I., Voeckler, J., Wilde, M., and Zhao, Y. (2002). Chimera: A Virtual Data System for Representing, Querying and Automating Data Derivation. In *Scientific and Statistical Databases Management*, Edinburgh, UK.

[Furmento et al., 2002] Furmento, N., Mayer, A., McGough, S., Newhouse, S., Field, T., and Darlington, J. (2002). ICENI : Optimisation of component applications within a Grid environment. *Journal of Parallel Computing*, 28(12):1753–1772.

[Gannon, 2007] Gannon, D. (2007). *Component Architectures and Services: From Application Construction to Scientific Workflows*, chapter 12, pages 174–189. In [Taylor et al., 2007].

[Garonne et al., 2004] Garonne, V., Tsaregorodtsev, A., and Stokes-Rees, I. (2004). A scalable lightweight architecture for high throughput computing. In *5th International Workshop on Grid Computing (GRID'04)*, pages 19–25, Pittsburgh, PA, USA. IEEE Computer Society.

[Gautama, 1998] Gautama, H. (1998). A probabilistic approach to the analysis of program execution time. Master's thesis, Delft University of Technology, Information Technology and Systems.

[Gautama and van Gemund, 2003] Gautama, H. and van Gemund, A. J. C. (2003). Symbolic Performance Estimation Of Speculative Parallel Programs. *Parallel Processing Letters*, 13(4):513–524.

[Gelenbe et al., 1986] Gelenbe, E., Montagne, E., Suros, R., and Woodside, C. (1986). A performance model of block structured parallel programms. In *International Workshop on Parallel Algorithms and Architectures*, pages 127–138, Luminy, France. Elsevier Science B.V (North Holland).

[Germain et al., 2005] Germain, C., Breton, V., Clarysse, P., Gaudeau, Y., Glatard, T., Jeannot, E., Legré, Y., Loomis, C., Magnin, I., Montagnat, J., Moureaux, J.-M., Osorio, A., Pennec, X., and Texier, R. (2005). Grid-enabling medical image analysis. *Journal of Clinical Monitoring and Computing*, 19(4–5):339–349.

[Germain et al., 2006] Germain, C., Texier, R., Osorio, A., and Loomis, C. (2006). Grid Scheduling for Interactive Analysis of Medical Images. In *Healthgrid'06*, pages 25–33, Valencia, Spain.

[Gholipour et al., 2007] Gholipour, A., Kehtarnavaz, N., Briggs, R., Devous, M., and Gopinath, K. (2007). Brain Functional Localization: A Survey of Image Registration Techniques. *IEEE Transactions on Medical Imaging (TMI)*, 26(4):427–451.

[Gibaud et al., 2004] Gibaud, B., Dojat, M., Benali, H., Dameron, O., Matsumoto, J.-P., Pellegrini-Issac, M., Valabregue, R., and Barillot, C. (2004). Towards an Ontology for Sharing Neuroimaging Data and Processing Tools: Experience Learned from the Development of a Demonstrator. In *DiDaMIC'04 Workshop. Satellite of the MICCAI conference*, pages 15–23, St Malo, France.

[Gil, 2007] Gil, Y. (2007). *Workflow Composition: Semantic Representation for Flexible Automation*, chapter 16, pages 244–257. In [Taylor et al., 2007].

[Glatard et al., 2006a] Glatard, T., Emsellem, D., and Montagnat, J. (2006a). Generic web service wrapper for efficient embedding of legacy codes in service-based workflows. In *Grid-Enabling Legacy Applications and Supporting End Users Workshop (GELA'06)*, pages 44–53, Paris, France.

[Glatard et al., 2007a] Glatard, T., Lingrand, D., Montagnat, J., and Riveill, M. (2007a). Impact of the execution context on Grid job performances. In *International Workshop on Context-Awareness and Mobility in Grid Computing (WCAMG07)*, pages 713–718, Rio de Janeiro. IEEE.

[Glatard et al., 2008a] Glatard, T., Montagnat, J., Emsellem, D., and Lingrand, D. (2008a). A Service-Oriented Architecture enabling dynamic services grouping for optimizing distributed workflows execution. *Future Generation Computer Systems*. to appear.

[Glatard et al., 2008b] Glatard, T., Montagnat, J., Lingrand, D., and Pennec, X. (2008b). Flexible and efficient workflow deployement of data-intensive applications on grids with MOTEUR. *International Journal of High Performance Computing and Applications (IJHPCA)*. to appear.

[Glatard et al., 2005] Glatard, T., Montagnat, J., and Pennec, X. (2005). Grid-enabled workflows for data intensive medical applications. In *18th IEEE International Symposium on Computer-Based Medical Systems (CBMS)*, pages 537–542, Dublin, Ireland.

[Glatard et al., 2006b] Glatard, T., Montagnat, J., and Pennec, X. (2006b). An experimental comparison of Grid5000 clusters and the EGEE grid. In *Workshop on Experimental Grid*

testbeds for the assessment of large-scale distributed applications and tools (EXPGRID'06), Paris, France.

[Glatard et al., 2006c] Glatard, T., Montagnat, J., and Pennec, X. (2006c). Efficient services composition for grid-enabled data-intensive applications. In *IEEE International Symposium on High Performance Distributed Computing (HPDC'06)*, pages 333–334, Paris, France.

[Glatard et al., 2006d] Glatard, T., Montagnat, J., and Pennec, X. (2006d). Medical image registration algorithms assesment: Bronze Standard application enactment on grids using the MOTEUR workflow engine. In *HealthGrid conference (HealthGrid'06)*, pages 93–103, Valencia, Spain. IOS Press.

[Glatard et al., 2006e] Glatard, T., Montagnat, J., and Pennec, X. (2006e). Probabilistic and dynamic optimization of job partitioning on a grid infrastructure. In *14th euromicro conference on Parallel, Distributed and network-based Processing (PDP06)*, pages 231–238, Montbéliard-Sochaux, France.

[Glatard et al., 2007b] Glatard, T., Montagnat, J., and Pennec, X. (2007b). Optimizing jobs timeouts on clusters and production grids. In *International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 100–107, Rio de Janeiro. IEEE.

[Glatard et al., 2006f] Glatard, T., Pennec, X., and Montagnat, J. (2006f). Performance evaluation of grid-enabled registration algorithms using bronze-standards. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI'06)*, LNCS 4191, pages 152–160, Copenhagen, Denmark. Springer.

[Glatard et al., 2007c] Glatard, T., Sipos, G., Montagnat, J., Farkas, Z., and Kacsuk, P. (2007c). *Workflow Level Parametric Study Support by MOTEUR and the P-GRADE Portal*, chapter 18, pages 279–299. In [Taylor et al., 2007].

[Gross and Harris, 1985] Gross, D. and Harris, C. (1985). *Fundamentals of Queueing Theory*. John Wiley and Sons, second edition.

[Grova et al., 2001] Grova, C., Biraben, A., Scarabin, J.-M., Jannin, P., Buvat, I., Benali, H., and Gibaud, B. (2001). A methodology to validate MRI / SPECT registration methods using realistic simulated SPECT data. In *MICCAI*, volume 2208 of *LNCS*, pages 275–282, Utrecht (The Netherlands). Springer.

[Hajnal et al., 2001] Hajnal, J., Hill, D., and Hawkes, D. (2001). *Medical Image Registration*. Crc press edition.

[Harchol-Balter and Balter, 2002] Harchol-Balter, M. and Balter, R. (2002). Task Assignment with Unknown Duration. *Journal of the ACM (JACM)*, 49(2):260–288.

[Harrison and Taylor, 2005] Harrison, A. and Taylor, I. (2005). Dynamic Web Service Deployment Using WSPeer. In *Proceedings of 13th Annual Mardi Gras Conference - Frontiers of Grid Applications and Technologies*, pages 11–16.

[Heining et al., 2006] Heining, S. M., Wiesner, S., Euler, E., and Navab, N. (2006). Pedicle screw placement under video-augmented fluoroscopic control. first clinical application in a cadaver study. *International Journal of Computer Assisted Radiology and Surgery*, 1(Supplement 1):189–190.

[Hellier and Barillot, 2004] Hellier, P. and Barillot, C. (2004). A hierarchical parametric algorithm for deformable multimodal image registration. *Computer Methods and Programs in Biomedicine*, 75(2):107–115.

[Hellier et al., 2003] Hellier, P., Barillot, C., Corouge, I., Gibaud, B., Le Goualher, G., Collins, D., Evans, A., Malandain, G., Ayache, N., Christensen, G., and Johnson, H. (2003). Retrospective evaluation of intersubject brain registration. *IEEE Transactions on Medical Imaging (TMI)*, 22(9):1120–1130.

[Hoheisel and Alt, 2007] Hoheisel, A. and Alt, M. (2007). *Petri Nets*, chapter 13, pages 190–207. In [Taylor et al., 2007].

[Holden et al., 2000] Holden, M., Hill, D., Denton, E., Jarosz, J. M., Cox, T. C., Rohlfing, T., Goodey, J., and Hawkes, D. (2000). Voxel Similarity Measures for 3-D Serial MR Brain Image Registration. *IEEE Transactions on Medical Imaging (TMI)*, 19(2):94–102.

[Huang et al., 2003] Huang, Y., Taylor, I., Walker, D. M., and Davies, R. (2003). Wrapping Legacy Codes for Grid-Based Applications. In *17th International Parallel and Distributed Processing Symposium (IPDPS)*, page 139. IEEE Computer Society.

[Irani and Bashna, 2002] Irani, R. and Bashna, S. J. (2002). *AXIS: Next Generation Java SOAP*. Wrox Press.

[Jacq et al., 2004] Jacq, N., Blanchet, C., Combet, C., Cornillot, E., Duret, L., Kurata, K., Nakamura, H., Silvestre, T., and Breton, V. (2004). Grid as a bioinformatic tool. *Journal of Parallel Computing*, 30(9-10):1093–110.

[Jacq et al., 2007] Jacq, N., Salzeman, J., Jacq, F., Legré, Y., Medernach, E., Montagnat, J., Maass, J., Reichstadt, M., Schwichtenberg, H., Sridhar, M., Kasam, V., Zimmermann, M., Hofmann, M., and Breton, V. (2007). Grid-enabled Virtual Screening against malaria. *Journal of Grid Computing (JGC)*.

[Jannin et al., 2002] Jannin, P., Fitzpatrick, J., Hawkes, D., Pennec, X., Shahidi, R., and Vannier, M. (2002). Validation of Medical Image Processing in Image-guided Therapy. *IEEE Transactions on Medical Imaging (TMI)*, 21(12):1445–1449.

[Kacsuk et al., 2003] Kacsuk, P., Dózsa, G., Kovács, J., Lovas, R., Podhorszki, N., Balaton, Z., and Gombás, G. (2003). P-GRADE: A Grid Programing Environment. *Journal of Grid Computing (JGC)*, 1(2):171–197.

[Kacsuk et al., 2006a] Kacsuk, P., Farkas, Z., Sipos, G., Toth, A., and Herrmann, G. (2006a). Workflow-level parameter study management in multi-grid environments by the P-GRADE portal. In *Grid Computing Environments 2006 (GCE'06)*, Tampa, USA.

[Kacsuk et al., 2006b] Kacsuk, P., Kiss, T., and Sipos, G. (2006b). Solving the Grid Interoperability Problem by P-GRADE Portal at Workflow Level. In Collet, P. and Lahire, P., editors, *Journées du groupe Objets, Composants et Modèles (OCM'2006)*, Nmes, France. I3S/RR-2006-06-FR.

[Kacsuk and Sipos, 2005] Kacsuk, P. and Sipos, G. (2005). Multi-Grid, Multi-User Workflows in the P-GRADE Grid Portal. *Journal of Grid Computing (JGC)*, 3(3-4):221 – 238.

[Kassim et al., 2005] Kassim, A., Yan, P., Yan, P., Lee, W., and Sengupta, K. (2005). Motion Compensated Lossy-to-Lossless Compression of 4D Medical Images Using Integer Wavelet Transforms. *IEEE Transactions on Information Technology In Biomedicine (TITB)*, 9(1):132–138.

[Kecskemeti et al., 2005] Kecskemeti, G., Zetuny, Y., Kiss, T., Sipos, G., Kacsuk, P., Terstyanszky, G., and Winter, S. (2005). Automatic deployment of Interoperable Legacy Code Services. In *UK e-Science All Hands Meeting*, Nottingham, UK.

[Kesselman and Mansour, 2005] Kesselman, A. and Mansour, Y. (2005). Optimizing TCP Retransmission Timeout. In *International Conference on Networking*, volume 3421 of *LNCS*, Saint-Denis de la Réunion. Springer.

[Khalaf et al., 2003] Khalaf, R., Mukhi, N., and Weerawarana, S. (2003). Service-Oriented Composition in BPEL4WS. In *International World Wide Web Conference (WWW)*, Budapest, Hungary. W3C.

[Kiczales et al., 1997] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., and Irwin, J. (1997). Aspect-Oriented Programming. In *Proceedings of the European Conference on Object-Oriented Programming*, volume 1241, pages 220–242.

[Kiepuszewski, 2003] Kiepuszewski, B. (2003). *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows*. PhD thesis, Queensland University of Technology, Brisbane, Australia.

[Kim and Pearlman, 1997] Kim, B.-J. and Pearlman, W. (1997). An Embedded Wavelet Video Coder Using Three-Dimensional Set Partitioning in Hierarchical Trees. In *IEEE Data Compression Conference (DCC'97)*, pages 251–260, Snowbird, Utah, USA.

[Kleinrock, 1975] Kleinrock, L. (1975). *Queueing Systems*.

[Kovács et al., 2007] Kovács, L., Micsik, A., and Pallinger, P. (2007). Handling User Preferences and Added Value in Discovery of Semantic Web Services. In *2007 IEEE International Conference on Web Services ((ICWS 2007))*, pages 225–232, Salt-Lake City, Utah, USA. IEEE Computer Society.

[Krishnan and Gannon, 2004] Krishnan, N. and Gannon, D. (2004). XCAT3: A Framework for CCA Components as OGSA Services. In *9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS'2004)*.

[Kristensen et al., 1998] Kristensen, L. M., Christensen, S., and Jensen, K. (1998). The practitioner's guide to coloured Petri nets. *International Journal on Software Tools for Technology Transfer (STTT)*, 2(2):98–132.

[Laure et al., 2006] Laure, E., Fisher, S., Frohner, Á., Grandi, C., and Kunszt, P. (2006). Programming the Grid with gLite. *Computational Methods in Science and Technology*, 12(1):33–45.

[Laure et al., 2005] Laure, E., Stockinger, H., and Stockinger, K. (2005). Performance Engineering in Data Grids. *Concurrency and Computation: Practice & Experience*, 17(2-4).

[Lee and Neuendorffer, 2000] Lee, E. A. and Neuendorffer, S. (2000). MoML - A Modeling Markup Language in XML, Version 0.4. Technical Report UCB/ERL M00/12, University of California, Berkeley, CA 94720.

[Legrand et al., 2006] Legrand, A., Quinson, M., Fujiwara, K., and Casanova, H. (2006). The SimGrid Project - Simulation and Deployment of Distributed Applications. In *Proceedings of the IEEE International Symposium on High Performance Distributed Computing (HPDC-15)*, Paris, France. IEEE Computer Society Press.

[Legrand and Robert, 2003] Legrand, A. and Robert, Y. (2003). *Algorithmique parallèle*. Dunod edition.

[Lewis and Papadimitriou, 1981] Lewis, H. and Papadimitriou, C. (1981). *Elements of the theory of computation*. Prentice-Hall, Englewood Cliffs, New Jersey.

[Li et al., 2004] Li, H., Groep, D., and Walters, L. (2004). Workload Characteristics of a Multicluster Supercomputer. In *Job Scheduling Strategies for Parallel Processing*, pages 176–193. Springer Verlag.

[Li et al., 2005] Li, J., Zhang, Z., and Yang, H. (2005). A Grid Oriented Approach to Reusing Legacy Code in ICENI Framework. In *IEEE International Conference on Information Reuse and Integration (IRI'05)*, pages 464– 469, Las Vegas, Nevada, USA.

[Libman and Orda, 2002] Libman, L. and Orda, A. (2002). Optimal Retrial and Timeout Strategies for Accessing Network Resources. *IEEE/ACM Transactions on Networking (TN)*, 10(4):551–564.

[Little et al., 1997] Little, J. A., Hill, D., and Hawkes, D. (1997). Deformations incorpotationg rigid structures. *Computer Vision and Image Understanding*, 66(2):223–232.

[Lucchi and Mazzara, 2007] Lucchi, R. and Mazzara, M. (2007). A pi-calculus based semantics for WS-BPEL. *Journal of Logic and Algebraic Programming*, (70):96–118.

[Ludäscher et al., 2005] Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E. A., Tao, J., and Zhao, Y. (2005). Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice & Experience*, 18(10):1039 – 1065.

[Magott, 1984] Magott, J. (1984). Performance evaluation of concurrent systems using Petri nets. *Information Processing Letters*, 18(1):7–13.

[Maintz and Viergever, 1998] Maintz, J. and Viergever, M. (1998). A survey of medical registration. *Medical Image Analysis (MedIA)*, 1(2):1–36.

[Makela et al., 2002] Makela, T., Clarysse, P., Sipila, O., Pauna, N., Pham, Q., Katila, T., and Magnin, I. (2002). A review of cardiac registration methods. *IEEE Transactions on Medical Imaging*, 21(9):1011–1021.

[Makela et al., 2003] Makela, T., Pham, Q., Clarysse, P., Lotjonen, J., Sipila, O., Hanninen, H., Lauerma, K., Knutti, J., Katila, T., and Magnin, I. (2003). A 3d model-based registration approach for the pet, mr and mcg cardiac data fusion. *Medical Image Analysis*, 7(3):377–389.

[Manolache et al., 2001] Manolache, S., Eles, P., and Peng, Z. (2001). Memory and Time-Efficient Schedulability Analysis of Task Sets with Stochastic Execution Time. In *Euromicro Conference on Real-Time Systems*, Delft, The Netherlands.

[Mayer et al., 2004] Mayer, A., McGough, S., Furmento, N., Lee, W., Gulamali, M., Newhouse, S., and Darlington, J. (2004). Workflow Expression: Comparison of Spatial and Temporal Approaches. In *Workflow in Grid Systems Workshop, GGF-10*, Berlin.

[Mazzara and Govoni, 2005] Mazzara, M. and Govoni, S. (2005). A Case Study of Web Services Orchestration. In *Coordination Models and Languages*, LNCS 3454, pages 1–16. Springer Verlag.

[Mazziotta et al., 1995] Mazziotta, J., Toga, A., Evans, A. C., Fox, P., and Lancaster, J. (1995). A probabilistic atlas of the human brain: theory and rationale for its development. The International Consortium for Brain Mapping. *NeuroImage*, 2(2):89–101.

[Mc Ilraith and Mandell, 2002] Mc Ilraith, S. and Mandell, D. (2002). Comparison of DAML-S and BPEL4WS. Technical report, Knowledge Systems Lab, Standford.

[McGough et al., 2006] McGough, S., Cohen, J., Darlington, J., Katsiri, E., Lee, W., Panagiotidi, S., and Patel, Y. (2006). An End-to-end Workflow Pipeline for Large-scale Grid Computing. *Journal of Grid Computing (JGC)*, pages 1–23.

[McGough et al., 2007] McGough, S., Lee, W., Cohen, J., Katsiri, E., and Darlington, J. (2007). *ICENI*, pages 395–415. In [Taylor et al., 2007].

[McGough et al., 2004] McGough, S., Young, L., Afzal, A., Newhouse, S., and Darlington, J. (2004). Workflow Enactment in ICENI. In *UK e-Science All Hands Meeting*, pages 894–900, Nottingham, UK.

[McIlroy, 1968] McIlroy, M. (1968). Mass-produced software components. In *Proceedings of the NATO Conference on Software Engineering*, Garmisch, Germany. NATO Science Committee.

[McPhillips and Bowers, 2005] McPhillips, T. and Bowers, S. (2005). An Approach for Pipelining Nested Collections in Scientific Workflows. *SIGMOD Record*, 35(3).

[Medernach, 2005] Medernach, E. (2005). Workload Analysis of a Cluster in a Grid Environment. In *Job Scheduling Strategies for Parallel Processing*.

[Mendling and Müller, 2003] Mendling, J. and Müller, M. (2003). A Comparison of BPML and BPEL4WS. In *1st Conference Berliner XML-Tage*, pages 305–316, Berlin.

[Menegaz and Thiran, 2002] Menegaz, G. and Thiran, J.-P. (2002). Lossy to lossless object-based coding of 3-D MRI data. *IEEE Transactions on Image Processing (TIP)*, 11(9):1053–1061.

[Micu et al., 2006] Micu, R., Jakobs, T., Urschler, M., and Navab, N. (2006). A new registration/visualization paradigm for ct-fluoroscopy guided rf liver ablation. In *Proc. Int'l Conf. Medical Image Computing and Computer Assisted Intervention (MICCAI)*, Lecture Notes in Computer Science. Springer.

[Milner, 1999] Milner, R. (1999). *Communicating and Mobile Systems: The Pi-Calculus*. Cambridge University Press, Cambridge, UK.

[Monson-Haefel, 2001] Monson-Haefel, R. (2001). *Enterprise JavaBeans*. O'Reilly, third edition.

[Montagnat, 2006] Montagnat, J. (2006). *Processing and analyzing large medical image sets*. Hdr thesis, University of Nice-Sophia Antipolis, Sophia Antipolis, France.

[Montagnat et al., 2004a] Montagnat, J., Bellet, F., Benoit-Cattin, H., Breton, V., Brunie, L., Duque, H., Legré, Y., Magnin, I., Maigne, L., Miguet, S., Pierson, J.-M., Seitz, L., and Tweed, T. (2004a). Medical images simulation, storage, and processing on the european datagrid testbed. *Journal of Grid Computing (JGC)*, 2(4):387–400.

[Montagnat et al., 2005] Montagnat, J., Breton, V., and Magnin, I. (2005). Partitionning medical image databases for content-based queries on a grid. *Methods of Information in Medicine (MIM)*, 44(2):154–160.

[Montagnat et al., 2007] Montagnat, J., Frohner, Á., Jouvenot, D., Pera, C., Kunszt, P., Koblitz, B., Santos, N., Loomis, C., Texier, R., Lingrand, D., Guio, P., Brito Da Rocha, R., Sobreira de Almeida, A., and Farkas, Z. (2007). A Secure Grid Medical Data Manager Interfaced to the gLite Middleware. *Journal of Grid Computing (JGC)*.

[Montagnat et al., 2006] Montagnat, J., Glatard, T., and Lingrand, D. (2006). Data composition patterns in service-based workflows. In *Workshop on Workflows in Support of Large-Scale Science (WORKS'06)*, Paris, France.

[Montagnat et al., 2004b] Montagnat, J., Magnin, I., and Breton, V. (2004b). Medical image databases content-based queries partitioning on a grid. In *HealthGrid'04*, Clermont-Ferrand, France.

[Murata, 1989] Murata, T. (1989). Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580.

[Mussi and Nain, 1984] Mussi, P. and Nain, P. (1984). Evaluation of parallel execution of program tree structures. In *Proceedings of the 1984 ACM SIGMETRICS conference on Measurement and modeling of computer systems (SIGMETRICS '84)*, pages 78–87, New York, NY, USA. ACM Press.

[Nakada et al., 2005] Nakada, H., Matsuoka, S., Seymour, K., Dongarra, J., Lee, C., and Casanova, H. (2005). A GridRPC Model and API for End-User Applications. Technical report, Global Grid Forum (GGF).

[Nemo et al., 2007a] Nemo, C., Blay-Fornarino, M., Kniesel, G., and Riveill, M. (2007a). SEMANTIC ORCHESTRATIONS MERGING - Towards Composition of Overlapping Orchestrations. In Filipe, J., editor, *9th International Conference on Enterprise Information Systems (ICEIS'2007)*, Funchal, Madeira.

[Nemo et al., 2007b] Nemo, C., Glatard, T., Blay-Fornarino, M., and Montagnat, J. (2007b). Merging overlapping orchestrations: an application to the Bronze Standard medical application. In *International Conference on Services Computing (SCC 2007)*, Salt Lake City, Utah, USA. IEEE Computer Engineering.

[Nicolau et al., 2003] Nicolau, S., Pennec, X., Soler, L., and Ayache, N. (2003). Evaluation of a New 3D/2D Registration Criterion for Liver Radio-Frequencies Guided by Augmented Reality. In *International Symposium on Surgery Simulation and Soft Tissue Modeling (IS4TM'03)*, volume 2673 of *LNCS*, pages 270–283, Juan-les-Pins, France. INRIA Sophia Antipolis, Springer-Verlag.

[Oinn et al., 2004] Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M. R., Wipat, A., and Li, P. (2004). Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics journal*, 17(20):3045–3054.

[Ourselin et al., 2000] Ourselin, S., Roche, A., Prima, S., and Ayache, N. (2000). Block Matching: A General Framework to Improve Robustness of Rigid Registration of Medical Images. In *Third International Conference on Medical Image Computing And Computer-Assisted Intervention (MICCAI'00)*, LNCS, pages 557–566, Pittsburgh, Pennsylvania USA. Springer Verlag.

[P. Anderson et al., 2002] P. Anderson, D., Cobb, J., Korpela, E., Lebofsky, M., and Werthimer, D. (2002). SETI@home: an experiment in public-resource computing. *Communications of the ACM (CACM)*, 45(11):56–61.

[Pennec, 2006a] Pennec, X. (2006a). Intrinsic Statistics on Riemannian Manifolds: Basic Tools for Geometric Measurements. *Journal of Mathematical Imaging and Vision (JMIV)*, 1(25):127–154.

[Pennec, 2006b] Pennec, X. (2006b). *Statistical Computing on Manifolds for Computational Anatomy*. Hdr thesis, Université Nice Sophia-Antipolis.

[Pennec et al., 2000] Pennec, X., Ayache, N., and Thirion, J.-P. (2000). Landmark-based registration using features identified through differential geometry. In Bankman, I., editor, *Landmark-based registration using features identified through differential geometry*, chapter 31, pages 499–513. Academic Press.

[Pennec et al., 1998] Pennec, X., Guttman, R. G., and Thirion, J.-P. (1998). Feature-Based Registration of Medical Images: Estimation and Validation of the Pose Accuracy. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI98)*, volume 1496 of *LNCS*, pages 1107–1114, Cambridge, USA. Springer.

[Pennec and Thirion, 1997] Pennec, X. and Thirion, J.-P. (1997). A Framework for Uncertainty and Validation of 3D Registration Methods based on Points and Frames. *International Journal of Computer Vision (IJCV)*, 25(3):203–229.

[Petri, 1962] Petri, C. A. (1962). *Kommunikation mit Automaten*. PhD thesis, Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 3, Bonn.

[Pfister and Szyperski, 1996] Pfister, C. and Szyperski, C. (1996). Why Objects are Not Enough. In *International Component Users Conference*, Munich, Germany. SIGS.

[Puhlmann, 2006] Puhlmann, F. (2006). Why do we actually need the Pi-Calculus for Business Process Management? In *9th International Conference on Business Information Systems (BIS06)*, Klagenfurt, Austria.

[Puhlmann and Puhlmann, 2005] Puhlmann, F. and Puhlmann, F. (2005). Using the pi-Calculus for Formalizing Workflow Patterns. In *Third International Conference on Business Process Management (BPM'05)*, LNCS 3649, pages 153–168, Nancy, France.

[Raffy et al., 2006] Raffy, P., Gaudeau, Y., Miller, D., Moureaux, J.-M., and Castellino, R. (2006). Computer-aided Detection of Solid Lung Nodules in Lossy Compressed Multidetector Computed Tomography Chest Exams. *Academic Radiology*, 13(10):1194–1203.

[Ramakrishnan et al., 2007] Ramakrishnan, A., Singh, G., Zhao, H., Deelman, E., Sakellariou, R., Vahi, K., Blackburn, K., Meyers, D., and Samidi, M. (2007). Scheduling Data-Intensive Workflows onto Storage-Constrained Distributed Resources. In *7th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07)*, pages 401–409, Rio de Janeiro, Brazil. IEEE Computer Society Press.

[Reinecke et al., 2004] Reinecke, P., van Moorsel, A., and Wolter, K. (2004). A Measurement Study of the Interplay between Application Level Restart and Transport Protocol. In *International Service Availability Symposium (ISAS)*, volume 3335 of *LNCS*, pages 86–100, Munich, Germany.

[Rex et al., 2003] Rex, D., Ma, J., and Toga, A. W. (2003). The LONI Pipeline Processing Environment. *NeuroImage*, 3(19):1033–1048.

[Roche et al., 1998] Roche, A., Malandain, G., Pennec, X., and Ayache, N. (1998). The Correlation Ratio as a New Similarity Measure for Multimodal Image Registration. In *First International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI'98)*, volume 1496 of *LNCS*, pages 1115–1124, Cambridge, USA. Springer Verlag.

[Roche et al., 2001] Roche, A., Pennec, X., Rudolph, L., Auer, D., Malandain, G., Ourselin, S., Auer, L., and Ayache, N. (2001). Rigid Registration of 3D Ultrasound with MR Images: a New Approach Combining Intensity and Gradient Information. *IEEE Transactions on Medical Imaging (TMI)*, 20(10):1038–1049.

[Rong and Pedram, 2006] Rong, P. and Pedram, M. (2006). Determining the optimal timeout values for a power-managed system based on the theory of Markovian processes: offline and online algorithms. In *Design, Automation and Test in Europe (DATE'06)*, pages 1128–1133, Munich, Germany.

[Sarrut et al., 2006] Sarrut, D., Boldea, V., Miguet, S., and Ginestet, C. (2006). Simulation of 4d ct images from deformable registration between inhale and exhale breath-hold ct scans. *Medical Physics*, 33(3):605–617.

[Schestowitz et al., 2006] Schestowitz, R., Twining, C. J., Cootes, T., Petrovic, V. S., Taylor, C., and Crum, W. R. (2006). Assessing the accuracy of non-rigid registration with and without ground truth. In *Third IEEE International Symposium on Biomedical Imaging (ISBI'06)*, pages 836–839.

[Schopf and Berman, 1998] Schopf, J. and Berman, F. (1998). Performance prediction in production environments. In *12th International Parallel Processing Symposium*, pages 647–653, Orlando, Florida, USA.

[Schopf and Berman, 1999] Schopf, J. and Berman, F. (1999). Stochastic Scheduling. In *Supercomputing (SC'99)*, Portland, USA.

[Schopf and Berman, 2001] Schopf, J. and Berman, F. (2001). Using Stochastic Information to Predict Application Behavior on Contended Resources. *International Journal of Foundations of Computer Science*, 12(3):341–364.

[Senger et al., 2003] Senger, M., Rice, P., and Oinn, T. (2003). Soaplab - a unified Sesame door to analysis tool. In *UK e-Science All Hands Meeting*, pages 509–513, Nottingham.

[Sermesant et al., 2006] Sermesant, M., Delingette, H., and Ayache, N. (2006). An Electromechanical Model of the Heart for Image Analysis and Simulation. *IEEE Transactions on Medical Imaging (TMI)*, 25(5):612–625.

[Shao et al., 2007] Shao, Q., Kinsy, M., and Chen, Y. (2007). Storing and Discovering Critical Workflows from Log in Scientific Exploration. In *IEEE Congress on Services, International Workshop on Scientific Workflows (Services'07)*, pages 209–212, Salt-Lake City, Utah, USA.

[Shields, 2007] Shields, M. (2007). *Control- Versus Data-Driven Workflows*, chapter 11. In [Taylor et al., 2007].

[Slominski, 2007] Slominski, A. (2007). *Adapting BPEL to Scientific Workflows*, chapter 14, pages 208–226. In [Taylor et al., 2007].

[Smith and Fingar, 2003] Smith, H. and Fingar, P. (2003). Workflow is jus a Pi process.

[Song et al., 2007] Song, H., Cheng, D., Messer, A., and Kalasapur, S. (2007). Web Service Discovery Using General-Purpose Search Engines. In *2007 IEEE International Conference on Web Services (ICWS 2007)*, Salt-Lake City, Utah, USA. IEEE Computer Society.

[Spanoudakis et al., 2007] Spanoudakis, G., Mahbub, K., and Zisman, A. (2007). A Platform for Context Aware Runtime Web Service Discovery. In *2007 IEEE International Conference on Web Services (ICWS 2007)*, pages 233–240, Salt-Lake City, Utah, USA. IEEE Computer Society.

[Stef-Praun et al., 2007] Stef-Praun, T., Clifford, B., Foster, I., Hasson, U., Hategan, M., Small, S., Wilde, M., and Zhao, Y. (2007). Accelerating Medical Research using the Swift Workflow System. In *HealthGrid*.

[Szyperski, 2002] Szyperski, C. (2002). *Component Software: Beyond Object-Oriented Programming*. Addison Wesley, second edition.

[Tanaka et al., 2003] Tanaka, Y., Nakada, H., Sekiguchi, S., Suzumura, T., and Matsuoka, S. (2003). Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing. *Journal of Grid Computing (JGC)*, 1(1):41–51.

[Taylor et al., 2007] Taylor, I., Deelman, E., Gannon, D., and Shields, M. (2007). *Workflows for e-Science*. Springer-Verlag.

[Taylor et al., 2005] Taylor, I., Wand, I., Shields, M., and Majithia, S. (2005). Distributed computing with Triana on the Grid. *Concurrency and Computation: Practice & Experience*, 17(1–18).

[Topcuoglu et al., 2002] Topcuoglu, H., Hariri, S., and Min-You, W. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *International Journal of Supercomputer Applications*, 13(3):260–274.

[Unser et al., 2003] Unser, M., Aldroubi, A., and Laine, A. (2003). Special Issue on Wavelets in Medical Imaging (editorial). *IEEE Transactions on Medical Imaging (TMI)*, 22(3):285–288.

[van der Aalst, 2004] van der Aalst, W. M. (2004). Why workflow is NOT just a Pi-process.

[van der Aalst and ter Hofstede, 2002] van der Aalst, W. M. and ter Hofstede, A. H. (2002). Workflow Patterns: On the Expressive Power of (Petri-net-based) Workflow Language. In

Proceedings of the Fourth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2002), volume 560 of *DAIMI*. Jensen, K.

[van der Aalst and ter Hofstede, 2005] van der Aalst, W. M. and ter Hofstede, A. H. (2005). YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275.

[van der Aalst et al., 2003] van der Aalst, W. M., ter Hofstede, A. H., Kiepuszewski, B., and Barros, A. P. (2003). Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51.

[Van Engelen and Gallivan, 2002] Van Engelen, R. A. and Gallivan, K. A. (2002). The gSOAP Toolkit for Web Services and Peer-to-Peer Computing Networks. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '02)*, page 128, Washington DC, USA. IEEE Computer Society.

[van Moorsel and Wolter, 2006] van Moorsel, A. and Wolter, K. (2006). Analysis of Restart Mechanisms in Software Systems. *IEEE Transactions on Software Engineering (TSE)*, 32(8):547–558.

[Veldhuizen, 2003] Veldhuizen, T. (2003). C++ Templates are Turing Complete. Technical report, Indiana University.

[von Laszewski et al., 2004] von Laszewski, G., Kaizar, A., Hategan, M., Zaluzec, N. J., Hampton, S., and Rossi, A. (2004). GridAnt: A Client-Controllable Grid Workflow System. In *37th Hawai'i International Conference on System Science*, Island of Hawaii, Big Island.

[Wagstrom et al., 2002] Wagstrom, P., Krishnan, S., and von Laszewski, G. (2002). GSFL: A Workflow Framework for Grid Services. In Austvoll, I., editor, *Scandinavian Conference on Image Analysis*, Bergen, Norway.

[Warfield et al., 2004] Warfield, S. K., Zou, K., and Wells, W. (2004). Simultaneous truth and performance level estimation (STAPLE): an algorithm for the validation of image segmentation. *IEEE Transactions on Medical Imaging (TMI)*, 23(7):903– 921.

[Weissman and Zhao, 1998] Weissman, J. and Zhao, X. (1998). Scheduling parallel applications in distributed networks. *Cluster Computing (CC)*, 1(1):109–118.

[West et al., 1997] West, J., Fitzpatrick, J. M., Wang, M. Y., Dawant, B. M., Maurer, Jr., C. R., Kessler, R. M., Maciunas, R. J., Barillot, C., Lemoine, D., Collignon, A., Maes, F., Suetens, P., Vandermeulen, D., van den Elsen, P. A., Napel, S., Sumanaweera, T. S., Harkness, B., Hemler, P. F., Hill, D. L. G., Hawkes, D. J., Studholme, C., Maintz, J. B. A., Viergever, M. A., Malandain, G., Pennec, X., Noz, M. E., Maguire, Jr., G. Q., Pollack, M., Pelizzari, C. A., Robb, R. A., Hanson, D., and Woods, R. P. (1997). Comparison and evaluation

of retrospective intermodality brain image registration techniques. *Journal of Computer Assisted Tomography*, 21(4):554–566.

[White, 2006] White, S. A. (2006). Using BPMN to model BPEL process. Technical report, IBM Corp.

[Wohed et al., 2003] Wohed, P., van der Aalst, W. M., Dumas, M., and ter Hofstede, A. H. (2003). Analysis of Web Services Composition Languages: The Case of BPEL4WS. In *22nd International Conference on Conceptual Modeling*, Chicago.

[Woodman et al., 2007] Woodman, S., Parastatidis, S., and Webber, J. (2007). *Protocol-Based Integration Using SSDL and pi-Calculus*, pages 227–243. In [Taylor et al., 2007].

[Xie et al., 2002] Xie, W., Sun, H., Cao, Y., and Trivedi, K. (2002). Optimal Webserver Session Timeout Settings for Web Users. In *Computer Measurement Group Conference (CMGC)*, pages 799–820, Reno, NV, USA.

[Xiong et al., 2003] Xiong, Z., Wu, X., Cheng, S., and Hua, J. (2003). Lossy-to-lossless compression of medical volumetric data using three-dimensional integer wavelet transforms. *IEEE Transactions on Medical Imaging (TMI)*, 22(3):459–470.

[Yang et al., 2007] Yang, L., Chen, W., Meer, P., Salaru, G., Feldman, M., and Foran, D. J. (2007). High Throughput Analysis of Breast Cancer Specimens on the Grid. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI'07)*, Brisbane, Australia.

[Yu and Buyya, 2004] Yu, J. and Buyya, R. (2004). A novel architecture for realizing grid workflow using tuple spaces. In *Proceedings. Fifth IEEE/ACM International Workshop on Grid Computing*, pages 119–128.

[Yu and Buyya, 2005a] Yu, J. and Buyya, R. (2005a). A taxonomy of scientific workflow systems for grid computing. *ACM SIGMOD records (SIGMOD)*, 34(3):44–49.

[Yu and Buyya, 2005b] Yu, J. and Buyya, R. (2005b). A Taxonomy of Workflow Management Systems for Grid Computing. *Journal of Grid Computing (JGC)*, 3(3-4):171 – 200.

[Zhang et al., 2004] Zhang, K., Damevski, K., Venkatachalapathy, V., and Parker, S. (2004). SCIRun2: A CCA Framework for High Performance Computing. In *9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS)*, pages 72–79, Los Alamitos, CA, USA.

[Zhao et al., 2007a] Zhao, Y., Hategan, M., Clifford, B., Foster, I., von Laszewski, G., Raicu, I., Stef-Praun, T., and Wilde, M. (2007a). Swift: Fast, Reliable, Loosely Coupled Parallel Computation. In *IEEE International Workshop on Scientific Workflows*, Salt-Lake City.

[Zhao et al., 2007b] Zhao, Y., Wilde, M., and Foster, I. (2007b). *Virtual Data Language: A Typed Workflow Notation for Diversely Structured Scientific Data*, chapter 17, pages 258–275. In [Taylor et al., 2007].