THÈSE

présentée à

L'INSTITUT NATIONAL DES SCIENCES APPLIQUÉES DE LYON

pour obtenir

LE GRADE DE DOCTEUR

ÉCOLE DOCTORALE : INFORMATIQUE ET MATHÉMATIQUES

Présentée par

Rafael FERREIRA DA SILVA

A science-gateway for workflow executions: online and non-clairvoyant self-healing of workflow executions on grids

Jury

Rapporteurs:	Eric RUTTEN	-	CR INRIA Grenoble
	Thomas FAHRINGER	-	Professeur, University of Innsbruck
Directeur:	Frédéric DESPREZ	-	DR INRIA, Laboratoire LIP
Co-Directeur:	Tristan GLATARD	-	CR CNRS, Laboratoire CREATIS
Examinateurs:	Silvia D. OLABARRIAGA	-	Assistant Professor, University of Amsterdam
	Johan MONTAGNAT	-	DR CNRS, Laboratoire I3S
	Hugues BENOIT-CATTIN	-	Professeur, INSA-Lyon
	Martin QUINSON	-	Maître de Conférences, Laboratoire LORIA

INSA Direction de la Recherche - Ecoles Doctorales – Quinquennal 2011-2015

SIGLE	ECOLE DOCTORALE	NOM ET COORDONNEES DU RESPONSABLE
CHIMIE	**CHIMIE DE LYON** **http://www.edchimie-lyon.fr** Insa : R. GOURDON	M. Jean Marc LANCELIN Université de Lyon – Collège Doctoral Bât ESCPE 43 bd du 11 novembre 1918 69622 VILLEURBANNE Cedex Tél : 04.72.43 13 95 directeur@edchimie-lyon.fr
E.E.A.	**ELECTRONIQUE, ELECTROTECHNIQUE, AUTOMATIQUE** **http://edeea.ec-lyon.fr** Secrétariat : M.C. HAVGOUDOUKIAN eea@ec-lyon.fr	M. Gérard SCORLETTI Ecole Centrale de Lyon 36 avenue Guy de Collongue 69134 ECULLY Tél : 04.72.18 65 55 Fax : 04 78 43 37 17 Gerard.scorletti@ec-lyon.fr
E2M2	**EVOLUTION, ECOSYSTEME, MICROBIOLOGIE, MODELISATION** **http://e2m2.universite-lyon.fr** Insa : H. CHARLES	Mme Gudrun BORNETTE CNRS UMR 5023 LEHNA Université Claude Bernard Lyon 1 Bât Forel 43 bd du 11 novembre 1918 69622 VILLEURBANNE Cédex Tél : 06.07.53.89.13 e2m2@ univ-lyon1.fr
EDISS	**INTERDISCIPLINAIRE SCIENCES-SANTE** **http://www.ediss-lyon.fr** Sec : Samia VUILLERMOZ Insa : M. LAGARDE	M. Didier REVEL Hôpital Louis Pradel Bâtiment Central 28 Avenue Doyen Lépine 69677 BRON Tél : 04.72.68.49.09 Fax :04 72 68 49 16 Didier.revel@creatis.uni-lyon1.fr
INFOMATHS	**INFORMATIQUE ET MATHEMATIQUES** **http://infomaths.univ-lyon1.fr** Sec :Renée EL MELHEM	Mme Sylvie CALABRETTO Université Claude Bernard Lyon 1 INFOMATHS Bâtiment Braconnier 43 bd du 11 novembre 1918 69622 VILLEURBANNE Cedex Tél : 04.72. 44.82.94 Fax 04 72 43 16 87 infomaths@univ-lyon1.fr
Matériaux	**MATERIAUX DE LYON** **http://ed34.universite-lyon.fr** Secrétariat : M. LABOUNE PM : 71.70 –Fax : 87.12 Bat. Saint Exupéry Ed.materiaux@insa-lyon.fr	M. Jean-Yves BUFFIERE INSA de Lyon MATEIS Bâtiment Saint Exupéry 7 avenue Jean Capelle 69621 VILLEURBANNE Cedex Tél : 04.72.43 83 18 Fax 04 72 43 85 28 Jean-yves.buffiere@insa-lyon.fr
MEGA	**MECANIQUE, ENERGETIQUE, GENIE CIVIL, ACOUSTIQUE** **http://mega.ec-lyon.fr** Secrétariat : M. LABOUNE PM : 71.70 –Fax : 87.12 Bat. Saint Exupéry mega@insa-lyon.fr	M. Philippe BOISSE INSA de Lyon Laboratoire LAMCOS Bâtiment Jacquard 25 bis avenue Jean Capelle 69621 VILLEURBANNE Cedex Tél :04.72 .43.71.70 Fax : 04 72 43 72 37 Philippe.boisse@insa-lyon.fr
ScSo	**ScSo*** **http://recherche.univ-lyon2.fr/scso/** Sec : Viviane POLSINELLI Brigitte DUBOIS Insa : J.Y. TOUSSAINT	M. OBADIA Lionel Université Lyon 2 86 rue Pasteur 69365 LYON Cedex 07 Tél : 04.78.77.23.86 Fax : 04.37.28.04.48 Lionel.Obadia@univ-lyon2.fr

*ScSo : Histoire, Géographie, Aménagement, Urbanisme, Archéologie, Science politique, Sociologie, Anthropologie

Acknowledgments

First of all, I would like to give my heartfelt dedication of this thesis to my precious girlfriend Audrey for all of her love and understanding during these years. Without her support, inspiration, confidence, and encouragement I would never be able to accomplish this work.

I would like to express my deepest and sincerest gratitude to my two thesis advisors, Tristan Glatard, and Frédéric Desprez for their valuable guidance and consistent encouragement throughout this work. I would like to specially thank Tristan Glatard for giving me the opportunity to be a software engineer of the VIP project, that led me to an interesting research topic which is the subject of this work. Moreover, he taught me how to be a great researcher. Special thanks go to Frédéric Desprez to have accepted to be my supervisor, for his advices, and to open doors in my way.

I would like to express my gratitude to all my colleagues from Creatis, Sorina Camarasu, Frédéric Cervenansky, Michaël Sdika, Delphine Charpigny, and Nouha Boujelben. Many thanks go in particular to my good friends Ting Li, Sarra Ben Fredj, Ibrahim Kallel, and William Romero for their collaboration and encouragements over these years.

I want to thank VIP ANR and ER-Flow projects for their financial support for my PhD. I also would like to thank Ewa Deelman for the opportunity to be a visitor researcher in her group for 3 months. Thanks for all my colleagues at ISI-USC, Gideon Juve, Karan Vahi, Mats Rynge, Rajiv Mayani, Weiwei Chen, and Sepideh Azarnoosh.

Finally, I would like to acknowledge my familly, my parents Célia and Wellington, my sister Danielly, my brother Sandro, and my three nieces Amanda, Teresa, and Maria Letícia.

Rafael FERREIRA DA SILVA

Abstract

Science gateways, such as the Virtual Imaging Platform (VIP), enable transparent access to distributed computing and storage resources for scientific computations. However, their large scale and the number of middleware systems involved in these gateways lead to many errors and faults. In practice, science gateways are often backed by substantial support staff who monitors running experiments by performing simple yet crucial actions such as rescheduling tasks, restarting services, killing misbehaving runs or replicating data files to reliable storage facilities. Fair quality of service (QoS) can then be delivered, yet with important human intervention.

Automating such operations is challenging for two reasons. First, the problem is online by nature because no reliable user activity prediction can be assumed, and new workloads may arrive at any time. Therefore, the considered metrics, decisions and actions have to remain simple and to yield results while the application is still executing. Second, it is non-clairvoyant due to the lack of information about applications and resources in production conditions. Computing resources are usually dynamically provisioned from heterogeneous clusters, clouds or desktop grids without any reliable estimate of their availability and characteristics. Models of application execution times are hardly available either, in particular on heterogeneous computing resources.

In this manuscript, we propose a general self-healing process for autonomous detection and handling of operational incidents in workflow executions. Instances are modeled as Fuzzy Finite State Machines (FuSM) where state degrees of membership are determined by an external healing process. Degrees of membership are computed from metrics assuming that incidents have outlier performance, e.g. a site or a particular invocation behaves differently than the others. Based on incident degrees, the healing process identifies incident levels using thresholds determined from the platform history. A specific set of actions is then selected from association rules among incident levels.

This manuscript is composed by seven chapters organized in two parts. In the first part, we address the design of a science-gateway and its components, and we introduce a workload archive that provides fine-grained information about application executions. The Virtual Imaging Platform (VIP) is an open accessible platform to support the execution of workflow applications on distributed computing infrastructures. We present a complete overview of the architecture, describing the tools and strategies used to exploit computing and storage resources. The platform currently has 441 registered users who consumed 379 years of CPU time since January 2011.

The science-gateway workload archive provides fundamental fine-grained information for the development of our self-healing methods. Archives of distributed workloads acquired at the infrastructure level lack information about users and application-level middleware. In this thesis, we show the added value of this archive acquired in the science-gateway level on several case studies related to user account, pilot jobs, fine-grained task analysis, bag of tasks, and workflows.

In the second part of this thesis, we first introduce our self-healing process for autonomous detection and handling of these operational incidents, and then we instantiate the healing process to late task executions and task granularity incidents, and unfairness among workflow executions incident. We present two methods to cope with the long-tail effect problem, and a method to control task replication. No strong assumption is made on the task duration or resource characteristics. Experimental results show that both methods properly detect blocked activities and speed up workflow executions up to a factor of 4.5.

To optimize task granularity in distributed workflows, we present a method that groups tasks when the fineness degree of the application becomes higher than a threshold determined from execution traces. The algorithm also de-groups task groups when new resources arrive. Results showed that under stationary load, our fineness control process significantly reduces the makespan of all applications. Under non-stationary load, task grouping is penalized by its lack of adaptation, but our de-grouping algorithm corrects it in case variations in the number of available resources are not too fast.

Finally, we present a method to address unfairness among workflow executions. We define a novel metric that quantifies unfairness based on the fraction of pending work in a workflow. It compares workflow activities based on their ratio of queuing tasks, their relative durations, and the performance of resources where tasks are running. Results show that our method can very significantly reduce the standard deviation of the slowdown, and the average value of our metric.

Keywords: error detection and handling, workflow execution, production distributed systems.

Résumé

Les *science-gateways*, telles que la Plate-forme d'Imagerie Virtuelle (VIP), permettent l'accès à un grand nombre de ressources de calcul et de stockage de manière transparente. Cependant, la quantité d'informations et de couches intergicielles utilisées créent beaucoup d'échecs et d'erreurs de système. Dans la pratique, ce sont souvent les administrateurs du système qui contrôlent le déroulement des expériences en réalisant des manipulations simples mais cruciales, comme par exemple replanifier une tâche, redémarrer un service, supprimer une exécution défaillante, ou copier des données dans des unités de stockages fiables. De cette manière, la qualité de service fournie est correcte mais demande une intervention humaine importante.

Automatiser ces opérations constitue un défi pour deux raisons. Premièrement, la charge de la plate-forme est en ligne, c'est-à-dire que de nouvelles exécutions peuvent se présenter à tout moment. Aucune prédiction sur l'activité des utilisateurs n'est donc possible. De fait, les modèles, décisions et actions considérés doivent rester simples et produire des résultats pendant l'exécution de l'application. Deuxièmement, la plate-forme est non-clairvoyante à cause du manque d'information concernant les applications et ressources en production. Les ressources de calcul sont d'ordinaire fournies dynamiquement par des grappes hétérogènes, des *clouds* ou des grilles de volontaires, sans estimation fiable de leur disponibilité ou de leur caractéristiques. Les temps d'exécution des applications sont difficilement estimables également, en particulier dans le cas de ressources de calculs hétérogènes.

Dans ce manuscrit, nous proposons un mécanisme d'auto-guérison pour la détection autonome et traitement des incidents opérationnels dans les exécutions des chaînes de traitement. Les objets considérés sont modélisés comme des automates finis à états flous (FuSM) où le degré de pertinence d'un incident est déterminé par un processus externe de guérison. Les modèles utilisés pour déterminer le degré de pertinence reposent sur l'hypothèse que les erreurs, par exemple un site ou une invocation se comportant différemment des autres, sont rares. Le mécanisme d'auto-guérison détermine le seuil de gravité des erreurs à partir de l'historique de la plate-forme. Un ensemble d'actions spécifiques est alors sélectionné par règle d'association en fonction du niveau d'erreur.

Ce manuscrit complet comporte sept chapitres organisés en deux parties. Dans la première partie, nous abordons la conception d'une *science-gateway* et de ses composants, et nous présentons une archive des traces d'exécutions acquisent auprès de la *science-gateway*, fournissant des informations détaillées sur les exécutions de l'application. La Plate-forme d'Imagerie Virtuelle (VIP) est une plate-forme ouverte et accessible pour l'exécution de workflows sur des ressources de calcul distribuées. Nous présentons un apperçu complet de l'architecture, décrivant les outils et les stratégies utilisées pour exploiter les ressources de calcul et de stockage. La plate-forme compte actuellement 441 utilisateurs enregistrés qui ont consommés 379 années de temps de CPU depuis Janvier 2011.

L'archive des traces d'exécutions acquises auprès du *science-gateway* fournit des informations fondamentales pour le développement de nos méthodes d'auto-guérison. Les archives de traces d'exécution acquises au niveau des infrastructures manquent d'information sur les utilisateurs et les applications. Dans cette thèse, nous montrons la valeur ajoutée de cette archive sur plusieurs études de cas : le suivi de l'activité des utilisateurs, les tâches pilotes, l'analyse détaillée des tâches, les tâches indépendantes (*bag of tasks*) et workflows.

Dans la deuxième partie de cette thèse, nous présentons d'abord notre processus d'auto-guérison pour la détection autonome et traitement des incidents opérationnels, et puis on instancie le processus pour traiter le retard et la granularité des tâches, et l'inégalité entre les exécutions de workflow. Nous présentons deux méthodes pour traiter le problème du retard de tâches, et une méthode pour contrôler la réplication des tâches. Aucune hypothèse forte n'est faite sur la durée de la tâche ou sur les caractéristiques des ressources. Les résultats expérimentaux montrent que la méthode détecte correctement les activités bloquées et accélère les exécutions jusqu'à 4,5 fois.

Pour optimiser la granularité des tâches des workflows, nous présentons une méthode que regroupe les tâches lorsque le degré de finesse de l'application devient supérieur à un seuil déterminé à partir des traces d'exécution. L'algorithme dégroupe également des groupes de tâches lorsque de nouvelles ressources arrivent. Les résultats montrent que sous charge stationnaire, notre processus de contrôle de finesse réduit considérablement le temps d'exécution total de toutes les applications. En cas de charge non stationnaire, le regroupement des tâches est pénalisé par son manque d'adaptation, mais notre algorithme de regroupement le corrige tant que les variations du nombre de ressources disponibles ne sont pas trop rapides.

Enfin, nous présentons une méthode pour traiter l'inégalité entre les exécutions de workflow. Nous définissons un modèle qui quantifie l'inégalité basée sur la fraction de travail en attente d'un workflow. Il compare les activités de workflow à partir de leur ratio de tâche en attente, leur durée moyenne, et la performance des ressources dans lesquelles les tâches sont en cours. Les résultats montrent que notre méthode peut très significativement réduire l'écart type du ralentissement.

Mots-clés: détection autonome et traitement des erreurs, exécutions des chaînes de traitement, systèmes distribués en production.

Contents

List of Figures

List of Tables

List of Algorithms

Introduction

Distributed computing infrastructures (DCI) such as grids and clouds are becoming daily instruments of scientific research. As collections of independent computers linked by a network presented to the users as a single coherent system, they enable easy collaboration among organizations, enhanced reliability and availability, and high performance computing. For instance, in the last decade, the European Grid Infrastructure (EGI[1]) and the Extreme Science and Engineering Discovery Environment (XSEDE[2]) demonstrated their ability to support heavy scientific computations with a high throughput [Gagliardi et al., 2005, Romanus et al., 2012].

Computational simulation is commonly used by researchers, and it usually involves long simulation times ranging from a few hours to years. A definition of computational simulation can be found in [fis, 2013]:

> A computational simulation is the discipline of designing a model of an actual or theoretical physical system, executing the model on a computer, and analyzing the execution output. Simulation embodies the principle of "learning by doing" — to learn about the system we must first build a model of some sort and then operate the model.

For instance, medical imaging research increasingly relies on simulation for studying image analysis, treatment planing or instrumentation prototyping, and simulation codes generally need specific computing infrastructures to produce realistic results [Wang et al., 2012b, Grevillot et al., 2012, Leporq et al., 2013].

Although distributed computing infrastructures help support the computing load and store the generated data, using them should not become an additional burden to simulation users. Therefore, high-level interfaces should allow their transparent exploitation, together with high performance.

Science gateways are emerging as user-level platforms to facilitate the access to distributed infrastructures. Their high-level interface allows scientists to transparently run their analyses on large sets of computing resources. They combine a set of authentication, data transfer, and workload management tools to deliver computing power as transparently as possible. While

[1]http://www.egi.eu
[2]http://www.xsede.org

these platforms provide important amounts of resources, their large scale and the number of middleware systems involved lead to many errors and faults, such as application crashes, hardware faults, network partitions, and unplanned resource downtime [Zhang et al., 2004]. Easy-to-use interfaces provided by these gateways exacerbate the need for properly solving operational incidents encountered on DCIs since end users expect high reliability and performance with no extra monitoring or parametrization from their side. In practice, science gateways are often backed by substantial support staff who monitors running experiments by performing simple yet crucial actions such as rescheduling tasks, restarting services, killing misbehaving runs or replicating data files to reliable storage facilities. Fair quality of service (QoS) can then be delivered, yet with important human intervention.

Application optimization in such platforms is complex. Science gateways have no *a-priori* model of the execution time of their applications because (*i*) task costs depend on input data with no explicit model, and (*ii*) characteristics of the available resources, in particular network and RAM, depend on background load. Modeling application execution time in these conditions requires cumbersome experiments which cannot be conducted for every new application in the platform. As a consequence, such platforms operate in *non-clairvoyant* conditions, where little is known about executions before they actually happen. Such platforms also run in *online* conditions, i.e. users may launch or cancel applications at any time and resources may appear or disappear at any time too.

A science gateway is considered here as a platform where users can process their own data with predefined applications workflows. Workflows are compositions of activities defined independently from the processed data and that only consist of a program description. At runtime, activities receive data and spawn invocations from their input parameter sets. Invocations are assumed independent from each other (bag of tasks) and executed on the DCI as single-core tasks which can be resubmitted in case of failures. This model fits several existing gateways such as e-bioinfra [Shahand et al., 2012], WS-PGRADE/gUSE [Kacsuk et al., 2012], and the Virtual Imaging Platform [Ferreira da Silva et al., 2011, Glatard et al., 2013]. We also consider that files involved in workflow executions are accessed through a single file catalog but that storage is distributed. Files may be replicated to improve availability and reduce load on servers.

The gateway may take decisions on file replication, resource provisioning, and task scheduling on behalf of the user. Performance optimization is a target but the main point is to ensure that correctly-defined executions complete, that performance is acceptable, and that misbehaving runs (e.g. failures coming from user errors or unrecoverable infrastructure downtimes) are quickly detected and stopped before they consume too many resources.

An autonomic manager can be described as a so-called MAPE-K loop [Kephart and Chess, 2003] which consists of monitoring (M), analysis (A), planning (P), execution (E), and knowledge (K). Self-healing techniques, generally implemented as MAPE-K loops, provide an interesting framework to cope with online non-clairvoyant problems. They address non-clairvoyance by using *a-priori* knowledge about the platform (e.g. extracted from traces), detailed monitoring, and analysis of its current behavior. They can also cope with online problems by periodical monitoring updates. Our ultimate goal is to reach a general model of such a scientific gateway that could autonomously detect and handle operational incidents, and control the behavior of non-clairvoyant, online platforms to limit human intervention required for their operation.

This manuscript is organized in two parts. Part I (Chapters 2 and 3) addresses the design of a science-gateway and its components, as well as issues and limitations related to the execution infrastructure. We also introduce a workload archive that provides fine-grained information about application executions. Part II (Chapters 4, 5, 6, and 7) addresses the development of automated methods to handle operational incidents in workflow executions. We first introduce our self-healing process for autonomous detection and handling of these operational incidents, and then we instantiate the healing process to late task executions and task granularity incidents, and unfairness among workflow executions incident.

Chapter 1. Chapter 1 presents the state of the art of distributed computing infrastructures and self-healing of workflow executions. As in this manuscript, it is organized in two parts. In the first part we present an overview of science-gateway components from the infrastructure level up to the interface, and in the second part we highlight strategies and mechanisms to handle operational issues at different component levels of a science-gateway.

Chapter 2. This chapter introduces the architecture of the Virtual Imaging Platform (VIP), a platform to support the execution of workflow applications on distributed computing infrastructures. A complete overview is presented, describing the tools and strategies used to exploit computing and storage resources. The system relies on the MOTEUR engine for workflow execution and on the DIRAC pilot-job system for workload management. The platform architecture and main functionalities were published as a conference paper [Ferreira da Silva et al., 2011], and a journal article [Glatard et al., 2013]. VIP have supported several research studies, such as [Rojas Balderrama et al., 2011, Forestier et al., 2011, Marion et al., 2011, Wang et al., 2012a, Camarasu-Pop et al., 2013, Rogers et al., 2013], and have been referenced in the research community as in [Shahand et al., 2012, Caan et al., 2012, Balderrama et al., 2012, De Craene et al., 2013, Prakosa et al., 2013].

Chapter 3. In this chapter, we describe a workload archive acquired at the science-gateway level, and we show its added value, compared to an archive acquired at the infrastructure level, on several case studies related to user accounting, pilot jobs, fine-grained task analysis, bag of tasks, and workflows. The science-gateway workload archive provides fundamental fine-grained information for the development of our self-healing methods. Results of this work were published in the CoreGRID/ERCIM workshop [Ferreira da Silva and Glatard, 2013].

Chapter 4. This chapter presents a self-healing process for workflow executions. The process quantifies incident degrees from metrics that can be computed online. These metrics make little assumptions on the application or resource characteristics. From their degree, incidents are classified in levels and associated to sets of healing actions. The healing process is parametrized on real application traces acquired in production on the European Grid Infrastructure (EGI). We also present a first example of this process on 7 basic incidents related to task errors. The self-healing process was presented in the CCGrid conference [Ferreira da Silva et al., 2012], and thereafter published as a journal article [Ferreira da Silva et al., 2013b].

Chapter 5. In this chapter, we propose a new algorithm to handle the long-tail effect and to control task replication. The long-tail effect is characterized by executions on slow machines, poor network connections or communication issues, which lead to substantial speed-up reductions. The healing process presented in Chapter 4 is parametrized on real application traces acquired in production on EGI. Results of this work were also presented in the CCGrid conference [Ferreira da Silva et al., 2012], and thereafter published as a journal article [Ferreira da Silva et al., 2013b].

Chapter 6. In this chapter, we propose a granularity control algorithm for platforms where such clairvoyant and offline conditions are not realistic. Controlling the granularity of workflow activities executed on grids is required to reduce the impact of task queuing and data transfer time. Most existing granularity control approaches assume extensive knowledge about the applications and resources (e.g. task duration on each resource), and that both the workload and available resources do not change over time. Our method groups tasks when the fineness degree of the application, which takes into account the ratio of shared data and the queuing/round-trip time ratio, becomes higher than a threshold determined from execution traces. The algorithm also de-groups task groups when new resources arrive. The application's behavior is constantly monitored so that the characteristics useful for the optimization are progressively discovered. This work was recently accepted to be presented in the Euro-Par conference [Ferreira da Silva et al., 2013a].

Chapter 7. In this chapter, we propose an algorithm to fairly allocate distributed computing resources among workflow executions to multi-user platforms. We consider a non-clairvoyant, online fairness problem where the platform workload, task costs and resource characteristics are unknown and not stationary. We propose a fairness control loop which assigns task priorities based on the fraction of pending work in the workflows. Workflow characteristics and performance on the target resources are estimated progressively, as information becomes available during the execution. Our method is implemented and evaluated on 4 different applications executed in production conditions on EGI. This work was recently accepted to be presented in the Euro-Par conference [Ferreira da Silva et al., 2013c].

Chapter 1

State of the art: distributed computing infrastructures and self-healing of workflow executions

Contents

In this chapter we present an overview of science-gateway components from the infrastructure level up to the interface, and then we highlight strategies and mechanisms to handle operational issues at different component levels of a science-gateway. Our survey shows that none or few studies consider the online problem and make little assumptions on the characteristics of the application or resources.

Dans ce chapitre nous présentons un aperçu des composants des « science-gateways » depuis l'infrastructure jusqu'à l'interface, puis nous présentons des stratégies et des mécanismes pour gérer les échecs opérationnelles à différents niveaux du « science-gateway ». Notre étude montre que aucun ou peu d'études considèrent le problème en ligne et font peu d'hypothèses sur les caractéristiques de l'application ou de ressources.

1.1 Infrastructure and software for science-gateways

Science gateways are a domain-specific frameworks (or toolsets) which incorporates applications, data, and tools to enable running application in computing environments such as grids and clouds. They are typically accessed via a web portal, that provides a scientific community with end-to-end support for a particular scientific workflow. They also provide services to support, upload, search, manage and download (or share) applications and data[1]. In this section we present the state of the art of science-gateways components, from the infrastructure to the interface, for the execution of applications on grid infrastructures.

1.1.1 Grid computing

Computational grids emerged in the middle of the past decade as a paradigm for high-throughput computing for scientific research and engineering, through the federation of heterogeneous resources distributed geographically in different administrative domains [Foster et al., 2002]. Resource sharing is governed by *virtual organizations* (VO) [Foster et al., 2001], which are a set of individuals or institutions defined around a set of resource-sharing rules and conditions.

Infrastructure. Grid computing infrastructures are federations of cooperating resource infrastructure providers, working together to provide computing and storage services for research communities. These infrastructures can be characterized into research and production infrastructures. Research infrastructures are designed to support computer-science experiments related to parallel, large-scale or distributed computing, and networking. Examples of such infrastructures are Grid'5000 [Cappello et al., 2005], Future Grid [von Laszewski et al., 2010], and DAS-3[2]. Production infrastructures, on the other hand, are designed to support large scientific experiments. They can be classified into HPC (high-performance computing) and HTC (high throughput computing). HPC systems focuses on tightly coupled parallel tasks, while

[1]http://www.hpcwales.co.uk/glossary, https://sites.google.com/site/iwsglife2012
[2]http://www.cs.vu.nl/das3

HTC focuses on the efficient execution of a large number of loosely-coupled tasks[3]. The main HPC infrastructures are XSEDE (USA) and PRACE (Europe). XSEDE[4] (Extreme Science and Engineering Discovery Enviroment) provides a single virtual system that scientists can use to interactively share computing resources, data, and expertise. It provide access to some dozens HPC resources and about 1.5 PB of storage capacity. PRACE[5] (Partnership for Advanced Computing in Europe) provides access to high-performance computing resources to researchers and scientists from academia and industry from around the world through a peer review process. Its infrastructure is composed by 6 leading-edge high performance computing systems of average performance of 3 Petaflops/s each. The main HTC infrastructures are OSG, NorduGrid, and EGI. The Open Science Grid (OSG[6]) provides common service and support for resource providers and scientific institutions using a distributed fabric of high throughput computational services. It provides access to computing resources of more than 100 computing centers in America. The NorduGrid Collaboration[7] provides a production-level grid infrastructure for research tasks. They provide access to computing resources of 11 partners from 10 countries. The European Grid Infrastructure (EGI[8]) gives European scientists access to more than 320,000 logical CPUs, 152 PB of disk space and data. It is a federation of over 350 resource centers across more than 50 countries. EGI is the infrastructure used in this manuscript and is presented in details in Section 2.2.5 of Chapter 2.

Middleware. Grid middleware enables users to submit tasks to store data and execute computation on grid infrastructures. Task scheduling, resources management, data storage, replication, and transfers are handled by the middleware. It also enables security functions, such as authentication and authorization. Several grid middleware were developed. We use the taxonomy presented in [Krauter et al., 2002] to classify grid systems according to their types: computational, data, or service grid, with centralized or decentralized scheduler. For instance, the Globus Toolkit [Foster, 2005] enables computing power, databases, and other tools sharing across corporate, institutional, and geographic boundaries. The toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. It is packaged as a set of components that can be used either independently or together to develop applications. Globus fits several classifications in the taxonomy, such as computational, data, and service grids, with a decentralized scheduler. HTCondor [Thain et al., 2005] is a specialized workload management system for compute-intensive jobs. It provides a job queueing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. HTCondor handles cluster of dedicated resources as well

[3]https://wiki.egi.eu/wiki/Glossary_V1
[4]http://www.xsede.org
[5]http://www.prace-project.eu
[6]https://www.opensciencegrid.org
[7]http://www.nordugrid.org
[8]http://www.egi.eu

idle desktop workstations. It falls into the category of computational grids with a centralized scheduler. DIET (Distributed Interactive Engineering Toolbox) [Caron et al., 2002] is a hierarchical set of components to build network-enabled server applications in a grid environment. As a middleware, DIET provides transparent access to a pool of computational resources, and is designed to take into account the data location when scheduling jobs. DIET can be classified as a computational and data grid with decentralized scheduler. ARC (used by NorduGrid) and UNICORE are classified as computational grids with decentralized scheduler. ARC (Advanced Resource Connector) [Ellert et al., 2007] provides a reliable implementation of the fundamental grid services, such as information services, resource discovery and monitoring, job submission and management, logging, brokering, data and resource management. It integrates computing resources (commodity computing clusters managed by a batch system or standalone workstations) and storage facilities, making them available via a secure common grid layer. UNICORE (Uniform Interface to Computing Resources) [Erwin, 2002] offers a ready-to-run grid system including client and server software. It provides access to distributed computing and data resources in a seamless and secure way. GridWay and gLite are classified as computational grids with decentralized scheduler. The gLite[9] middleware (used by EGI) provides a framework for building applications tapping into distributed computing and storage resources across the Internet. A gLite grid is composed by a set of resource centers running services to provide remote access to local dedicated computational resources and central services that form the backbone of the service grid. It is the middleware used in this manuscript and is presented in Section 2.2.5 of Chapter 2. The GridWay metascheduler [Huedo et al., 2010] enables large-scale, reliable, and efficient sharing of computing resources over different grid middleware, such as Globus Toolkit or gLite. Computing resources can be managed by different resource management systems within a single organization or scattered across several administrative domains. GridWay provides a single point of access to the computing resources, from in-house systems to partner grid infrastructures and public Cloud providers. Finally, BOINC (Berkeley Open Infrastructure for Network Computing) [Anderson, 2004] is a platform for public-resource distributed computing. Although it was originally designed for volunteer computing (computer owners donate their computing resources to one or more applications), it also works for grid computing. BOINC allows vast number of single PCs to be connected in a grid-like system (a.k.a. desktop grid), adding up their processing power. BOINC falls in a new category that is not listed in the taxonomy: opportunistic grids. Examples of such grids also include the OurGrid [Cirne et al., 2006], and InteGrade [Goldchleger et al., 2004].

[9]`http://glite.cern.ch`

1.1.2 Pilot-job systems

Most of the large-scale experiments conducted on large VOs of production grids now use a pilot-job system. Pilot jobs run special agents that fetch user tasks from the task queue, set up their environment and steer their execution. This late binding of pilot jobs to processing tasks prevents latencies and failure modes in resource acquisition. They also improve the reliability and efficiency of task execution by providing automatic load balancing, fine-grained scheduling and failure recovery. The importance of the use of pilot jobs is detailed in Section 2.2.4 of Chapter 2. Frameworks can be roughly classified as *lightweight* or *heavyweight* pilot job systems. Lightweight frameworks use simple master-slave system, but lack features to manage large communities of users, or large pools of resources. Examples of such frameworks include DIANE, and ToPoS. In contrast, heavyweight frameworks use complex systems that can support an entire virtual organization. Condor glideIns, SAGA BigJob, PanDA, and DIRAC are examples of such frameworks. DIRAC (Distributed Infrastructure with Remote Agent Control) [Tsaregorodtsev et al., 2009] is the pilot job system used in this manuscript and is presented in Section 2.2.4 of Chapter 2.

DIANE [Korkhov et al., 2009] is a lightweight task execution control framework for parallel scientific applications. ToPoS[10] (A Token Pool Server for Pilot Jobs) introduces the concept of token pools. Tokens uniquely identifies a task and can be files, parameters, numbers, etc. Users have to create tokens and to submit pilot jobs. Pilot jobs contact the token pool server, request for a token and execute it. The system assumes that pilot jobs have the application executable, and tokens are the input data for them.

Condor glideIns [Thain et al., 2003] is a multi-level scheduling technique where glideIns are submitted as user tasks via grid protocols to a remote cluster. The glideIns are configured to contact a central manager controlled by the user where they can be used to execute the user's jobs on the remote resources. BigJob [Luckow et al., 2010] is a SAGA[11]-based pilot job implementation. It provides support to parallel applications (e.g. based on MPI) and work across different distributed infrastructures. BigJob is composed by three main components, the manager, that provides the pilot job abstraction and manages the orchestration and scheduling of BigJobs; the agent, that represents the pilot jobs and thus, the application-level resource manager running on the respective resource; and the advert service that is used for communication between the manager and the agent. PanDA (Production and Distributed Analysis) [Nilsson et al., 2011, Zhao et al., 2011] is the workload management system of the ATLAS experiment[12], used to run managed production and user analysis jobs on the grid. Pilot factories send jobs containing a wrapper to the grid computing sites through Condor-G. The local batch system sends the wrapper to a worker node where it downloads and executes the

[10] `https://grid.sara.nl/wiki/index.php/Using_the_Grid/ToPoS`
[11] `http://saga-project.github.io`
[12] `http://atlas.ch`

PanDA pilot code. The pilot fetches a user job, executes it, and uploads the outputs to a local storage element.

1.1.3 Scientific workflows

Scientific workflows allow users to easily express multi-step computational tasks, for example retrieve data from an instrument or a database, reformat the data, and run an analysis. A scientific workflow describes the dependencies between the tasks. In most cases the workflow is described as a directed acyclic graph (DAG), where the nodes are tasks (or group of tasks) and the edges denote the task (or group of tasks) dependencies. Sometimes control structures (e.g. loops, ifs) are also used to describe workflows.

Workflow description language. Scientific workflows are described as high-level abstraction languages which conceal the complexity of execution infrastructures to the user. A workflow language formalism is a formalism expressing the causal/temporal dependencies among a number of tasks to execute[13]. A formalism can underpin different languages. For instance, AGWL (Abstract Grid Workflow Language) [Fahringer et al., 2005b] is a XML-based workflow language for composing a workflow application from atomic units of work called activities interconnected through control flow and data flow dependencies. Activities are represented by type levels, simplified abstract descriptions of functions or semantics of an activity, and deployment levels, references to executables or deployed web services. Workflows can be modularized and invoked as sub-workflows. YAWL (Yet Another Workflow Language) [van der Aalst and ter Hofstede, 2005] is a Petri-net based workflow language defined according to some predefined workflow patterns[14]. A YAWL model is made of tasks, conditions and a flow relation between tasks and conditions. Each YAWL model has one start condition and one end condition. It supports three kinds of split and three corresponding kinds of join: AND, XOR, and OR. YAWL provides direct support for cancellation regions. If a task is within the cancellation region of another task, it may be prevented from being started or its execution may be terminated. Scufl (Simplifed conceptual workflow language) [Oinn et al., 2006] is a data-flow centric language, defining a graph of data interactions between different services. Its purpose is to present the workflows from a problem-solving oriented view, hiding the complexity of the interoperation of the services. Gwendia (Grid Workflow Efficient Enactment for Data Intensive Applications) [Montagnat et al., 2009] is a workflow language that targets the coherent integration of: (*i*) a data-driven approach to achieve transparent parallelism; (*ii*) arrays manipulation to enable data parallel application in an expressive and compact framework; (*iii*) conditional and loop control structures to improve expressiveness; and (*iv*) asynchronous execution to optimize execution on a distributed infrastructure. Gwendia is the work-

[13]http://gridworkflow.org/snips/gridworkflow/space/Workflow+Description+Languages
[14]http://www.workflowpatterns.com

flow language used in this manuscript. IWIR (Interoperable Workflow Intermediate Representation) [Plankensteiner et al., 2011] is a common workflow language for use as an intermediate exchange representation by multiple workflow systems. It has a graph-based structure, mixing data-flows and an expressive set of sequential and parallel control structures, specified as an XML representation.

Workflow execution engine. Workflow interpretation and execution are handled by a workflow engine that manages the execution of the application on the infrastructure through the middleware. We use the taxonomy presented in [Yu and Buyya, 2005] to classify workflow engines regarding the structure of workflow they support. In general, the workflow can be represented as a Directed Acyclic Graph (DAG), and non-DAG. For instance, Pegasus [Deelman et al., 2005] is a framework for mapping and executing DAG workflows on distributed computational resources. It takes as input an abstract workflow and converts it into an executable workflow by mapping tasks to grid resources, transferring the task executables to those resources, discovering sources for input data and adding data transfer nodes to the workflow. The final executable workflow can be executed on local or remote resources. Pegasus can also reduce workflows based on the data produced by previous runs in the grid. Taverna [Missier et al., 2010] is a suite of tools used to design and execute scientific workflows. A Taverna workflow specification is compiled into a multi-threaded object model, where processors are represented by objects, and data transfers from output to input ports of downstream processor objects are realized using local method invocations between objects. One or more activities are associated to each processor. These activities may consist of entire sub-workflows, in addition to executable software components. DIET manages workflows applications structured as DAG represented by an XML document that defines the nodes and data-dependencies. DIET can work in two modes: one in which it defines a complete scheduling of the workflow (ordering and mapping), and one in which it defines only an ordering for the workflow execution. Mapping is then done in the next step by the client, using a master agent to find the server where the workflow services should be run. Kepler [Altintas et al., 2004] is a free, open-source system for designing, executing, reusing, evolving, archiving, and sharing scientific DAG workflows. In Kepler, workflow authors use a graphical user interface to implement an analytical procedure by connecting a series of workflow components, called Actors, through which data are processed. Actors that may contain a hierarchy of other actors are called Composites. Parts of actors that receive Tokens, which encapsulate single or multiple data or messages, are called Ports. Directors control the execution of workflows, and in a typical, simple workflow, one director manages the execution of one set of actors. Parameters are settings that a user may create and configure, e.g. to serve as arguments to an actor. Triana [Taylor et al., 2007] is a workflow-based graphical problem solving environment and an underlying subsystem. Workflows are described as dataflows. The underlying subsystem consists of a collection of interfaces that bind to different types of middleware and services, such as grid middleware and web services. The P-GRADE Grid Portal [Farkas and Kacsuk, 2011] is a web based, service rich environment for the development, execution and monitoring of workflows and workflow based parameter studies on various grid platforms. Currently, P-GRADE evolved to WS-PGRADE/gUSE and is described in the next section. Example of workflow engines that provide support to non-DAG workflow executions include ASKALON and MOTEUR. ASKALON [Fahringer et al., 2005a] provides a suite of middleware services that support the execution of scientific workflows on the grid. Workflow specifications are converted into executable forms, mapped to grid resources, and executed. Workflow executions consist of coordinating control flow constructs and resolving data flow dependencies specified by the application developer. ASKALON also manages grid resources; it provides grid resource discovery, advanced reservation and virtual organization-wide authorization along with a dynamic registration framework for the grid activities. MOTEUR [Glatard et al., 2008, Rojas Balderrama et al., 2010] is a workflow engine enabling both the simple description of complex and large-scale applications, and the efficient execution on distributed computing infrastructures. It is the workflow engine used in this manuscript and is described in Section 2.2.3 of Chapter 2.

1.1.4 Science-gateways

Some software-as-a-service platforms, commonly called scientific gateways, integrate application software with access to computing and storage resources via web portals or desktop applications. Science-gateways are used in different scientific domains such as multi-disciplinary, climate, and medical imaging. Examples of multi-disciplinary platforms include the GISELA portal and the WS-PGRADE/gUSE framework. The CCSM portal is an example of a climate gateway. Science-gateways used for medical imaging include the e-bioinfra portal, the neuGRID project, and the Virtual Imaging Platform (VIP). In the remainder of this section we present the main characteristics of such gateways. VIP [Ferreira da Silva et al., 2011, Glatard et al., 2013] is the gateway used in this manuscript and is presented in Chapter 2.

The GISELA portal [Barbera et al., 2011] is a multi-disciplinary science-gateway for research communities in Latin America. Authentication is based on identity federations through the Grid IDentity Pool (GrIDP[15]). Users are mapped to a robot certificate used for all grid authentications. Applications have no formal description; instead, customized web pages are available for each predefined application. Tasks are directly submitted to gLite and executed on the *prod.vo.eu-eela.eu* VO of EGI. Science-gateways such as EUMEDGrid[16] and DECIDE[17] are built with a similar model.

WS-PGRADE/gUSE [Kacsuk, 2011, Kacsuk et al., 2012] is a gateway framework that pro-

[15]http://gridp.garr.it
[16]http://applications.eumedgrid.eu
[17]http://applications.eu-decide.eu

vides a generic purpose, workflow-oriented graphical user interface to create and run workflows on various infrastructures including clusters, grids, desktop grids and clouds. Applications are described as workflows using its own XML-based workflow language. Authentication is based on login and password (or identity federations) and mapped to a robot certificate when executing in grids, desktop grids, and clouds; and is based on SSH keys for the cluster case.

The CCSM portal (Community Climate System Modeling) [Basumallik et al., 2007] is a coupled climate modeling framework for simulating the earth's climate system. Authentication is based on login and password. Predefined applications are ported as scripts to run on supercomputing clusters of the XSEDE infrastructure. Files are transferred through a FTP client. This architecture fits several science-gateways such as VLab[18] (Virtual Laboratory for Earth and Planetary Materials) and CIMA[19] (Common Instrument Middleware Architecture).

The e-bioinfra portal [Shahand et al., 2012] is a science-gateway for medical image and genomics analysis where users access predefined applications through a web portal interface. Authentication is based on login and password. Users are mapped to a robot certificate used for all grid authentications. Data management is twofold. First, files are transferred from the user's machine to a data server through a FTP client or the web portal; then, files are automatically copied to grid resources. Applications are described as workflows and enacted by MOTEUR workflow engine. Tasks are submitted to DIANE pilot job system and executed on the *vlemed* VO of EGI.

The neuGRID project [Frisoni et al., 2011] targets neuroimaging data sharing and analysis using grid computing infrastructures. In neuGRID, users can start remote desktop sessions on machines where image analysis tools and clients are pre-installed to access resources on the *vo.neugrid.eu* VO of EGI. Users are authenticated by X.509 certificates imported into their web browsers. An applet is available to create proxies directly from the browser keystore and upload them to a MyProxy[20] server. All the subsequent grid operations are done with the user proxy.

1.2 Self-healing of workflow executions on grids

The second part of this thesis presents our self-healing mechanism and its application on several operational incidents. In this section, we present the state of the art regarding strategies to address these incidents: (*i*) task resubmission (Chapter 4), (*ii*) task and file replication (Chapters 4 and 5), (*iii*) task grouping (Chapter 6), and (*iv*) fairness among workflow executions (Chapter 7).

[18]http://www.vlab.msi.umn.edu/
[19]http://cimaportal.indiana.edu:8080/gridsphere/gridsphere
[20]http://grid.ncsa.illinois.edu/myproxy

1.2.1 Task resubmission

Task resubmission is one of the most common technique for handling failures. Several scheduling strategies can be used when resubmitting a task. The most naive approach is to randomly select a resource, assuming that the probability to drop in the same problematic resource is minimal. More complex strategies include resource blacklisting or resource selection based on successful task completion rates. In particular, task resubmission has been widely used for the execution of scientific workflows on grids. For instance, in [Wieczorek et al., 2008], the authors present a taxonomy of the multi-criteria grid workflow scheduling problem where task resubmission is considered for the problem of lost tasks due to full queues. In [Kandaswamy et al., 2008, Zhang et al., 2009, Montagnat et al., 2010], failed tasks are resubmitted to increase the probability of have a successful execution in another computing resource, while in [Pandey et al., 2009], they are resubmitted to resources that do not have failure history for those tasks. [Plankensteiner et al., 2009] proposes an algorithm based on the impact of task resubmission to decide whether to resubmit a task or replicate it. Their approach reduces resource waste compared to conservative task replication and resubmission techniques.

1.2.2 Task and file replication

Task replication, a.k.a. redundant requests is commonly used to address non-clairvoyant problems [Cirne et al., 2007], but it should be used sparingly, to avoid overloading the middleware and degrading fairness among users [Casanova, 2006]. For instance, [Litke et al., 2007] propose a task replication strategy to handle failures in mobile grid environments. Their approach is based on the Weibull distribution to estimate the number of replicas to guarantee a specific fault-tolerance level. In [Ramakrishnan et al., 2009], task replication is enforced as fault-tolerant mechanism to increase the probability to complete a task successfully. Recently, [Ben-Yehuda et al., 2012] proposed a framework for dynamic selection of Pareto-efficient scheduling strategy, where tasks are replicated only in the tail phase when task completion rate is low. All the proposed approaches make strong assumptions on task and resource characteristics, such as the expected duration and resource performance.

An important aspect to be evaluated when replicating task is the resource waste, a.k.a. the cost of task replication. Cirne et al. [Cirne et al., 2007] evaluate the waste of resources by measuring the percentage of wasted cycles among all the cycles required to execute the application. In this manuscript, we also consider the cost of task replication as detailed in Section 5.2.2 of Chapter 5.

File replication strategies also often assume clairvoyance on the size of produced data, file access pattern and infrastructure parameters [Bell et al., 2003, Elghirani et al., 2008]. In practice, production systems mostly remain limited to manual replication strategies [Rehn et al., 2006]. Ma et al. [Ma et al., 2013] proposed a taxonomy of files and resources models, optimization criterions, replication processes, and replication validation methods. They reviewed and classified 30 replication method studies according to these taxonomies.

1.2.3 Task grouping

The low performance of *fine-grained* tasks is a common problem in widely distributed platforms where the scheduling overhead and queuing times are high, such as Grid and Cloud systems. Several works have addressed the control of task granularity of bag of tasks. For instance, Muthuvelu et al. [Muthuvelu et al., 2005] proposed an algorithm to group bag of tasks based on their granularity size—defined as the processing time of the task on the resource. Resources are ordered by their decreasing values of capacity (in MIPS) and tasks are grouped up to the resource capacity. This process continues until all tasks are grouped and assigned to resources. Then, Ng et al. [Ng et al., 2006] and Ang et al. [Ang et al., 2009] extended the work of Muthuvelu et al. by introducing bandwidth in the scheduling framework to enhance the performance of task scheduling. Resources are sorted in descending order of bandwidth, then assigned to grouped tasks downward ordered by processing requirement length. The size of a grouped task is determined from the task cost in millions instructions (MI).

Later, Muthuvelu et al. [Muthuvelu et al., 2008] extended [Muthuvelu et al., 2005] to determine task granularity based on QoS requirements, task file size, estimated task CPU time, and resource constraints. Meanwhile, Liu & Liao [Liu and Liao, 2009] proposed an adaptive fine-grained job scheduling algorithm (AFJS) to group lightweight tasks according to processing capacity (in MIPS) and bandwidth (in Mb/s) of the current available resources. Tasks are sorted in descending order of MI, then clustered by a greedy algorithm. To accommodate with resource dynamicity, the grouping algorithm integrates monitoring information about the current availability and capability of resources. Afterwards, Soni et al. [Soni et al., 2010] proposed an algorithm to group lightweight tasks into coarse-grained tasks (GBJS) based on processing capability, bandwidth, and memory-size of the available resources. Tasks are sorted into ascending order of required computational power, then, selected in first-come, first-served (FCFS) order to be grouped according to the capability of the resources. Zomaya and Chan [Zomaya and Chan, 2004] studied limitations and ideal control parameters of task clustering by using genetic algorithm. Their algorithm performs task selection based on the earliest task start time and task communication costs; it converges to an optimal solution of the number of clusters and tasks per cluster.

Although the reviewed works significantly reduce communication and processing time, neither of them are non-clairvoyant and online at the same time. Recently, Muthuvelu et al. [Muthuvelu et al., 2010, Muthuvelu et al., 2013] proposed an online scheduling algorithm to determine the task granularity of compute-intensive bag-of-tasks applications. The granularity optimization is based on task processing requirements, resource-network utilisation constraint (maximum time a scheduler waits for data transfers), and users QoS requirements (user's budget and application deadline). Submitted tasks are categorised according to their file sizes, estimated CPU times, and estimated output file sizes, and arranged in a tree structure. The scheduler selects a few tasks from these categories to perform resource benchmarking. Tasks are grouped according to predefined objective functions of task granularity, and submitted to resources. The process restarts upon task arrival. In a collaborative work [Chen et al., 2013], we present three balancing methods to address the load balance problem when clustering workflow tasks. We defined three imbalance metrics to quantitative measure workflow characteristics based on task runtime variation (HRV), task impact factor (HIFV), and task distance variance (HDV). Although these are an online approach, the solutions are still clairvoyant.

1.2.4 Fairness among workflow executions

Fairness among workflow executions has been addressed in several studies considering the scheduling of multiple workflows, but to the best of our knowledge, no algorithm was proposed in a non-clairvoyant and online case. For instance, Zhao and Sakellariou [Zhao and Sakellariou, 2006] address fairness based on the slowdown of Directed Acyclic Graph (DAG); they consider a clairvoyant problem where the execution time and the amount of data transfers are known.

Similarly, N'Takpé and Suter [N'Takpe and Suter, 2009] propose a mapping procedure to increase fairness among parallel tasks on multi-cluster platforms; they address an offline and clairvoyant problem where tasks are scheduled according to one of the following three characteristics: critical path length, maximal exploitable task parallelism, or amount of work to execute. Casanova et al. [Casanova et al., 2010] evaluate several scheduling online algorithms of multiple parallel task graphs (PTGs) on a single, homogeneous cluster. Fairness is measured through the maximum stretch (a.k.a. slowdown) defined by the ratio between the PTG execution time on a dedicated cluster, and the PTG execution time in the presence of competition with other PTGs.

Hsu et al. [Hsu et al., 2011] propose an online HEFT-like algorithm to schedule multiple workflows; they address a clairvoyant problem where tasks are ranked based on the length of their critical path, and tasks are mapped to the resources with the earliest finish time. Sommerfeld and Richter [Sommerfeld and Richter, 2011] present a two-tier HEFT-based grid workflow scheduler with predictions of input-queue waiting times and task execution times; fairness among workflow tasks is addressed by preventing HEFT to assign the highest ranks to the first tasks regardless of their originating workflows.

Hirales-Carbajal et al. [Hirales-Carbajal et al., 2012] schedule multiple parallel workflows on a grid in a non-clairvoyant but offline context, assuming dedicated resources. Their multi-stage scheduling strategies consist of task labeling and adaptive allocation, local queue prioritization and site scheduling algorithm. Fairness among workflow tasks is achieved by task labeling based on task run time estimation.

Recently, Arabnejad and Barbosa [Arabnejad and Barbosa, 2012] proposed an algorithm addressing an online but clairvoyant problem where tasks are assigned to resources based on their rank values; task rank is determined from the smallest remaining time among all remaining tasks of the workflow, and from the percentage of remaining tasks. Finally, in their evaluation of non-preemptive task scheduling, Sabin et al. [Sabin et al., 2004] assess fairness by assigning a fair start time to each task. A fair start time of a task is defined by the start time of the task on a complete simulation of all tasks whose queue time is lower than that one. Any task which starts after its fair start time is considered to have been treated unfairly. Results are trace-based simulations over a period of one month, but the study is performed in a clairvoyant context.

1.3 Conclusions

In this chapter, we presented an overview of science-gateway components from the infrastructure level up to the interface, and then we highlighted strategies and mechanisms to handle operational issues at different component levels of a science-gateway. Usually, these strategies and mechanisms cannot be computed during the execution of the application, i.e. they are *offline*, or they make strong assumptions about resource and application characteristics, i.e., they are *clairvoyant*.

As described in the next chapters, the Virtual Imaging Platform (VIP) is designed to easily integrate autonomic methods, which detect and handle failures online with little assumptions on the application or resource characteristics. MOTEUR was selected as the workflow engine because of its ability to handle complex workflows (support to control structures) described with the GWENDIA language. The choice for DIRAC is because it provides a complete solution for managing distributed computing resources, and mainly due to the support provided by the development team. EGI is the largest grid infrastructure in Europe, which provides a significant amount of computing resources for the execution of scientific applications.

In the following chapters of this manuscript, we introduce VIP in details, and then we present the self-healing process, based on the MAPE-K loop, that quantifies incident degrees of workflow executions from metrics measuring the long-tail effect, task granularity, and fairness among workflow executions. Our approach differs from the ones presented in this chapter because our metrics are simple enough to be computed *online*, and little assumptions are made on the characteristics of the application or resources, i.e. they are *non-clairvoyant*. Moreover, our healing process is parametrized on real application traces acquired in production on the European Grid Infrastructure (EGI), and it is implemented and experimental evaluated in a production science-gateway (VIP).

Part I

A SCIENCE-GATEWAY FOR WORKFLOW EXECUTIONS ON GRIDS

Chapter 2

A science-gateway for workflow executions on grids

Contents

This chapter introduces the architecture of the Virtual Imaging Platform (VIP), a platform accessible at http://vip.creatis.insa-lyon.fr to support the execution of workflow applications on distributed computing infrastructures. A complete overview is presented, describing the tools and strategies used to exploit computing and storage resources. The system relies on the MOTEUR engine for workflow execution and on the DIRAC pilot-job system for workload management. The platform currently has 441 registered users who consumed 379 years of CPU time since January 2011. This level of activity demonstrates VIP usability, availability and reliability.

Ce chapitre présente l'architecture de la plate-forme VIP, une plate-forme accessible depuis http://vip.creatis.insa-lyon.fr pour l'exécution de workflows sur des ressources de calcul distribuées. Un aperçu complet est présenté, décrivant les outils et les stratégies utilisées pour exploiter les ressources de calcul et de stockage. Le système utilise le moteur d'exécution de workflow MOTEUR et le système de tâches pilotes DIRAC pour la gestion des calculs. La plate-forme compte actuellement 441 utilisateurs enregistrés qui ont consommés 379 années de temps de CPU depuis Janvier 2011. Ce niveau d'activité démontre la facilité d'utilisation, la disponibilité et la fiabilité de la plate-forme VIP.

2.1 Introduction

Grid computing infrastructures are becoming daily instruments of scientific research, in particular through scientific gateways [Gesing and van Hemert, 2011] developed to allow scientists to transparently run their analyses on large sets of computing resources. Quoting the definition of science-gateways on XSEDE[1]: "A science-gateway is a community-developed set of tools, applications, and data that are integrated via a graphical interface, that is further customized to meet the needs of a specific community". This intersects the definition of software as a service (SaaS). SaaS is a software delivery model in which software is integrated and hosted by the SaaS provider and which users access over the internet, usually through a web portal. A scientific gateway is considered here as a software as a service platform where users can process their own data with predefined applications.

This chapter describes the science-gateway designed and developed in this thesis: the Virtual Imaging Platform (VIP) [Ferreira da Silva et al., 2011, Glatard et al., 2013]. VIP is an openly-accessible platform for workflow executions on a production grid. In VIP, users authenticate to a web portal with login and password, and they are then mapped to X.509 robot credentials. From the portal, users transfer data and launch applications workflows to be executed on the grid. Workflows are compositions of activities defined independently from the processed data and that only consist of a program description and requirements. At runtime, activities receive data and spawn invocations from their input parameter sets. Invocations are independent from each other (bag of tasks) and executed on the computing resource as single-core tasks which can be resubmitted in case of failures. Thanks to a modular architecture, the platform can be easily extended to provide other features, such as tools that facilitate the sharing of object models for medical simulation [Forestier et al., 2011] and facilitate simulator integration [Marion et al., 2011]. VIP applications are executed on the biomed virtual organization (VO) of the European Grid Infrastructure (EGI).

The chapter is organized as follows. In Section 2.2, we show methods and techniques

[1]`https://www.xsede.org/gateways-overview`

employed to support workflow execution in VIP: Section 2.2.1 describes the web interface, Section 2.2.2 depicts the process of file transfers between user machines and the grid, Section 2.2.3 introduces the workflow engine, Section 2.2.4 presents the workload management system, Section 2.2.5 describes the execution infrastructure, and Section 2.2.6 summarizes the workflow execution process. Usage statistics are presented in Section 2.3, and challenges and limitations of the platform are presented in Section 2.4. Section 2.5 concludes the chapter.

2.2 VIP architecture for workflow execution

Figure 2.1 shows the overall VIP architecture for workflow execution. It is composed of (*i*) a web portal which interfaces users to applications described as workflows, (*ii*) a data management tool to handle transfer operations between users machines and the grid storage, (*iii*) a workflow engine to process user inputs and spawn computational tasks, (*iv*) a workload management system for resource provisioning and task scheduling, and (*v*) an execution infrastructure. This section describes each component of the architecture in details. In the context of the VIP project[2], we fully developed the web portal interface (Section 2.2.1) and the data management tool (Section 2.2.2). The workflow engine and service wrapper (Section 2.2.3), the workload management system (Section 2.2.4), and the infrastructure (Section 2.2.5) are software developed by partners with which we contribute.

2.2.1 Interface

In VIP web portal, authentication is done with login and password. Users are mapped to a regular X.509 robot certificate used for all grid authentications. In science-gateways, robot certificates make user authentication easier, in particular for those who are not familiar with personal digital certificates and the technical aspects of the Grid Security Infrastructure (GSI) [Barbera et al., 2009]. Robot certificates are used to perform automated tasks on grids on behalf of users. They identify a service and are bounded to a person or institution.

Although robot certificates enable transparent access to computational resources, they raise security challenges to science-gateways: users are mapped to a unique certificate preventing user authentication at infrastructure level. Therefore, the platform should keep track of all user operations, in particular file transfers and application executions. VIP adopts an open access policy where accounts are created based on valid email addresses only. New accounts are mapped to a *Beginner* level where users can execute one application at a time, and cannot write in shared folders. Advanced users have extended rights, but they must briefly describe their activities, and have a grid certificate registered in a virtual organization. To avoid technical problems, the certificate is not used to log in to the portal but only for administrators to check

[2]ANR-09-COSI-03

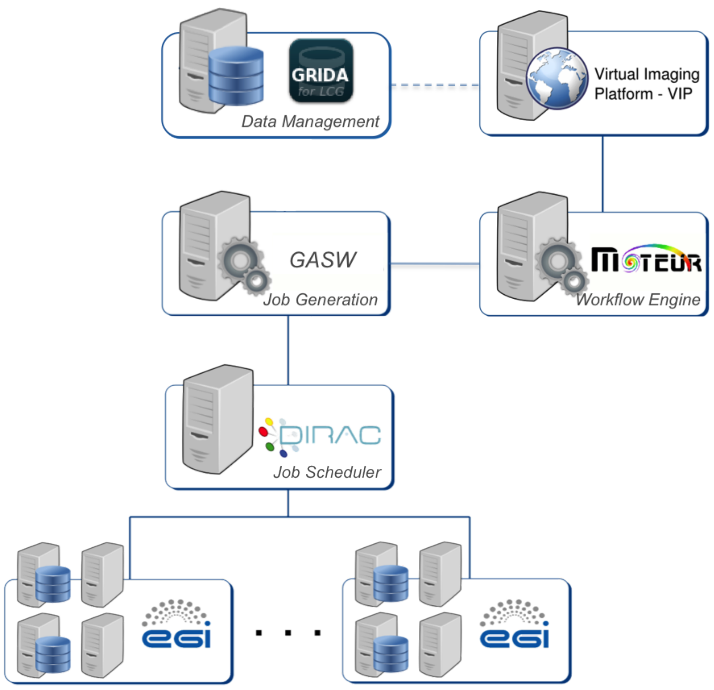


Figure 2.1: VIP architecture for workflow execution.*

* Figure courtesy of William A. Romero R.

user identity. Users are organized in groups defining access rights to applications and data. Applications are organized in classes which are associated to users groups. Groups can have private or public access. Public groups can be joined by anyone but access to private groups is restricted. Figure 2.2 shows the user↔group↔class↔application association.

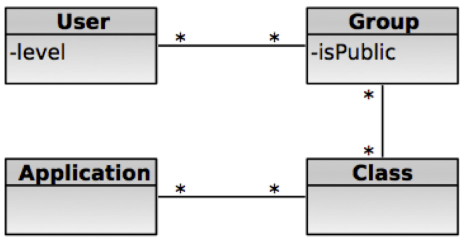
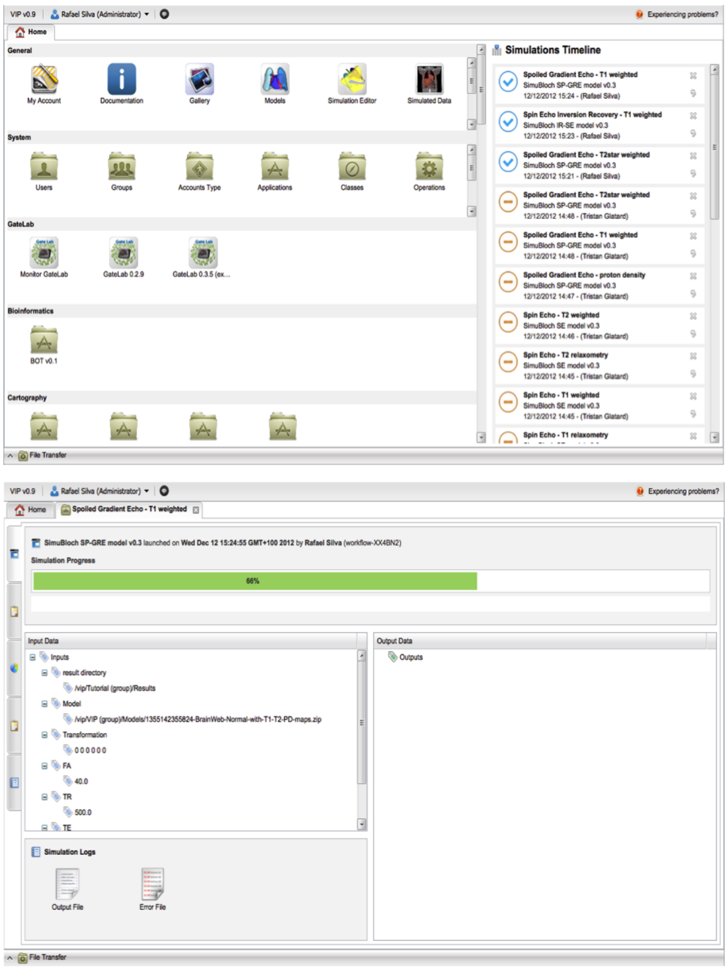


Figure 2.2: Association among users, groups, classes and applications.

Figure 2.3 (top) shows a view of the home page once logged into the platform. On the left side, applications are listed by class. Only classes associated to user groups are displayed. To launch an execution, a user selects an application, fills its input parameters (including files) and clicks the launch button. A timeline, on the right side, allows application monitoring. From the monitoring panel (Figure 2.3–bottom) users can follow application evolution, monitor performance, and retrieve output data once the execution is finished.

The web portal interface enables users to focus on their domain-specific problem. It is achieved by providing transparent access to computing and storage resources, i.e., users have



Spoiled Gradient Echo - T1 weighted
SimuBloch SP-GRE model v0.3
12/12/2012 15:24 - (Rafael Silva)

Spin Echo Inversion Recovery - T1 weighted
SimuBloch IR-SE model v0.3
12/12/2012 15:23 - (Rafael Silva)

Spoiled Gradient Echo - T2star weighted
SimuBloch SP-GRE model v0.3
12/12/2012 15:21 - (Rafael Silva)

Spoiled Gradient Echo - T2star weighted
SimuBloch SP-GRE model v0.3
12/12/2012 14:48 - (Tristan Glatard)

Spoiled Gradient Echo - T1 weighted
SimuBloch SP-GRE model v0.3
12/12/2012 14:48 - (Tristan Glatard)

Spoiled Gradient Echo - proton density
SimuBloch SP-GRE model v0.3
12/12/2012 14:47 - (Tristan Glatard)

Spin Echo - T2 weighted
SimuBloch SE model v0.3
12/12/2012 14:46 - (Tristan Glatard)

Spin Echo - T2 relaxometry
SimuBloch SE model v0.3
12/12/2012 14:45 - (Tristan Glatard)

Spin Echo - T1 weighted
SimuBloch SE model v0.3
12/12/2012 14:45 - (Tristan Glatard)

Spin Echo - T1 relaxometry

SimuBloch SP-GRE model v0.3 launched on Wed Dec 12 15:24:55 GMT+100 2012 by Rafael Silva (workflow-XX4BN2)

- Inputs
  - result directory
    - /vip/Tutorial (group)/Results
  - Model
    - /vip/VIP (group)/Models/1355142355824-BrainWeb-Normal-with-T1-T2-PD-maps.zip
  - Transformation
    - 0 0 0 0 0 0
  - FA
    - 40.0
  - TR
    - 500.0
  - TE

Figure 2.3: VIP screenshots: home page with applications (top) and application execution monitoring (bottom).

no information about resources, and no control on scheduling. This approach also reduces the probability of users errors. However, resource management issues should be handled by the science-gateway. Moreover, the problem is *online*: users may launch new workflows or cancel existing ones at any time. The problem is also *non-clairvoyant*: a model of user behavior or workload prediction is extremely cumbersome.

2.2.2 Data management

The GRId Data management Agent (GRIDA)[3] enables file transfer between local user machines and grid storage. The upload process consists in (*i*) uploading the file from the local machine to the portal and (*ii*) transferring the file to the grid through an asynchronous pool of transfers processed sequentially, in first-come, first-served (FCFS) order. Download is performed similarly, in the opposite direction. Figure 2.4 illustrates this process. Downloaded data are kept on the platform for 2 months, until the operation is removed by the user, or until the platform has

[3] http://vip.creatis.insa-lyon.fr:9002/projects/vletagent

less than 5% of disk space left.

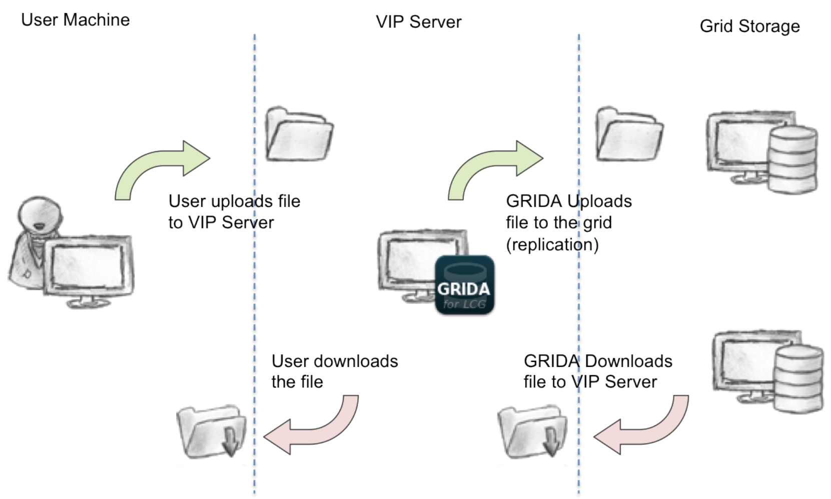

User Machine
VIP Server
Grid Storage
User uploads file to VIP Server
GRIDA Uploads file to the grid (replication)
GRIDA
User downloads the file
GRIDA Downloads file to VIP Server

Figure 2.4: File uploading and downloading process in VIP.

This two-step process avoids connectivity issues between user machines and distributed grid hosts. A transfer pool manages the load of concurrent transfers performed by the server to avoid network clogging on the portal machine. GRIDA also replicates files to ensure availability, and caches a local copy of them.

An optimized file browser is available to interact with the transfer service. It caches grid directory browsing. File permissions are also enforced by the browser: users have a private folder readable only by themselves, and each group has a shared folder.

2.2.3 Workflow engine

VIP integrates applications based on their workflow description, and without modifying their code so that parallelization does not require any thorough validation step. The motivation for using workflows is twofold. First, it provides a parallel language that enables execution on distributed computing platforms. Second, it facilitates application management by assembling dependencies on deployment, and by enabling automatic interface generation in the portal.

In VIP, workflows are interpreted and executed using MOTEUR workflow engine [Glatard et al., 2008, Rojas Balderrama et al., 2010], which provides an asynchronous grid-aware enactor. Workflows are described using the Gwendia (Grid Workflow Efficient Enactment for Data Intensive Applications) workflow language [Montagnat et al., 2009].

Figure 2.5 illustrates a workflow built with MOTEUR. Triangle components denote input data, and diamond components denote output data; arrows denote dependencies among components. Rectangular and oval components denote activities. Rectangular activities receive data and spawn tasks to be executed on grids, and oval activities are *beanshells*, a piece of Java code to be executed locally.

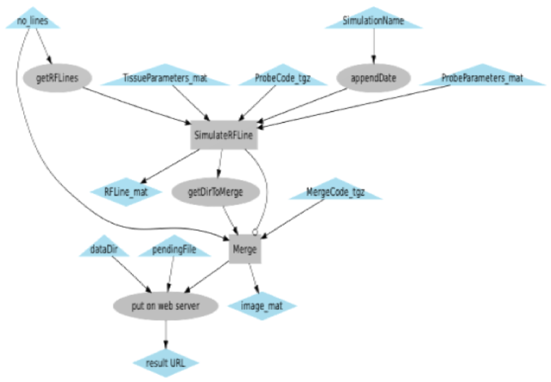


Figure 2.5: Example of a MOTEUR workflow.

The MOTEUR enactor is architected as a two-level engine. At the upper level, the core engine interprets the workflow representation, evaluates the resulting data flows and produces invocations ready for remote execution through a generic interface. At the lower level, invocations are converted into tasks targeting a specific computing infrastructure. The interface between the core engine and the task generation is implemented by a component named GASW (Generic Application Service Wrapper)[4]. GASW provides a mechanism to package an application and its dependencies, to deploy it and to publish it. When an application wrapped with GASW receives data, it generates tasks in the form of bash scripts wrapped in grid tasks and submits to the workload management system. Bash scripts first test file upload, then download the input data, launch the application command-line and finally upload the results. Statistics information are gathered from finished executions for accounting.

GASW has a pluggable architecture. It supports three categories of plugins: (*i*) database, (*ii*) executor and (*iii*) listener plugins. Database plugins specify which back-end should be used to persist information about workflow executions. Executor plugins submit and monitor tasks on computing infrastructures. Available executors include plugins for the Application Hosting Environment[5], gLite, DIRAC (presented in the next section), and local servers. Listener plugins receive notifications concerning task execution such as task submission, status changes, and task completeness. They are useful collecting feedback about executions, for instance, for the development of healing process described in the second part of this manuscript.

2.2.4 Workload management system

The DIRAC (Distributed Infrastructure with Remote Agent Control) [Tsaregorodtsev et al., 2009] workload management system (WMS) is represented in Figure 2.6. It implements a late binding between tasks and resources. User tasks are submitted to a central task queue which sends pilot jobs to execution nodes at the same time.

[4]http://vip.creatis.insa-lyon.fr:9002/projects/gasw
[5]http://http//www.realitygrid.org/AHE

Pilot jobs run special agents that fetch user tasks from the task queue, set up their environment and steer their execution. Advantages of this approach are summarized below.

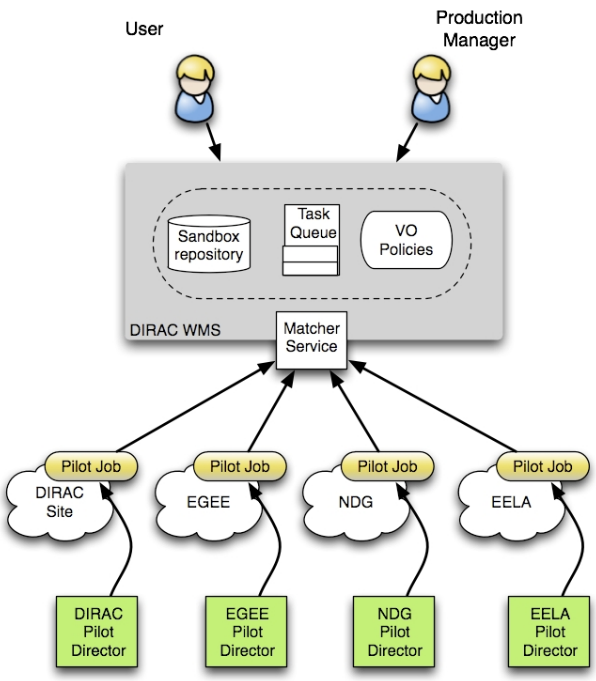


Figure 2.6: DIRAC Workload Management with pilot jobs.*

* Extracted from [Tsaregorodtsev et al., 2009].

First of all, pilot jobs check their execution environment before retrieving user tasks, which reduces the failure rate of user tasks and increases the efficiency of the WMS. Besides, pilot jobs effectively reserve their execution node for the time slot allocated by the batch system, which reduces considerably the load on the grid middleware—a pilot can execute more than one task in sequence. In addition, the centralization of user task in the task queue enables community policy enforcements while decentralized priority rules defined at the site level are imprecise and suffer scalability problems. Finally, pilot jobs can easily incorporate computing resources of different nature and belonging to various administrative domains. Indeed, once running, pilot jobs behave the same regardless of the type of resources.

The relevance of using pilot jobs have been experimentally demonstrated in several works, such as in [Camarasu-Pop et al., 2011]. In this study, the difficulty to complete an application execution comes from recurrent errors and high latencies of the execution infrastructure. For instance, Figure 2.7 shows the result of an experiment where an application is executed with a pilot jobs system (`DS`) and without (`gLite`). Pilot jobs outperforms the classical job submission system, by achieving 100% of the results and by significantly lowering the completion time.

Most of the large-scale experiments conducted on large shared infrastructures now use a pilot job system. Currently, VIP uses DIRAC's French national instance provided by France-

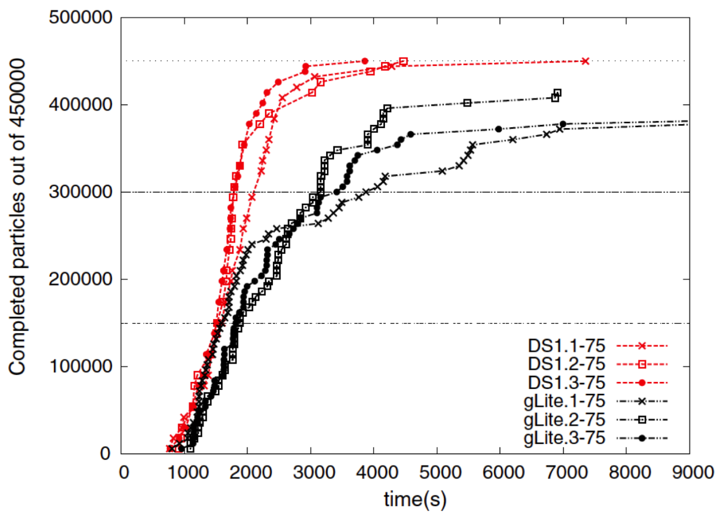


Figure 2.7: Application execution with (DS) and without (gLite) pilot jobs.*

* Extracted from [Camarasu-Pop et al., 2011].

Grilles[6], in which the administration is shared among VIP and other communities.

2.2.5 Infrastructure

Application tasks are executed on the *biomed* virtual organization (VO) of the European Grid Infrastructure (EGI). EGI is a federation of over 350 resources centers (sites) across more than 50 countries which has access to more than 320,000 logical CPUs and 152 PB of disk space. EGI has more than 230 VOs from about 10 disciplines composed of more than 22,000 members worldwide. The biomed VO, as of January 2013 , has access to some 90 computing sites of 22 countries, offering 190 batch queues and approximately 4 PB of disk space. Table 2.1 shows the distribution of sites per country supporting the biomed VO.

EGI uses the gLite middleware. The gLite middleware was born from the collaborative efforts of more than 80 people in 12 different academic and industrial research centers as part of the EGEE series of Projects[7]. A gLite grid is composed by a set of resource centers running services to provide remote access to local dedicated computational resources and central services that form the backbone of the service grid. The gLite stack combines low level core services with a range of higher level services. gLite grid services can be thematically grouped into 4 groups: access and security, information and monitoring, job management and data services.

Access and security services identifies users, allowing or denying access to services, on the basis of agreed policies. Authentication is based on the public key infrastructure (PKI) X.509 technology with certification authorities as trusted third parties. The information and monitoring services provide information about gLite resources and their status. Published information

[6]https://dirac.france-grilles.fr
[7]http://www.eu-egee.org

Country	Number of sites	Number of batch queues
UK	13	50
Italy	12	30
France	12	31
Greece	9	11
Spain	5	7
Germany	5	14
Portugal	4	7
Turkey	3	3
Poland	3	4
Netherlands	3	12
Croatia	3	6
Bulgaria	3	3
FYROM	2	2
Brazil	2	3
Vietnam	1	1
Slovakia	1	1
Russia	1	2
Other (.org)	1	1
Moldova	1	1
Mexico	1	1
Cyprus	1	1
China	1	1

Table 2.1: Distribution of sites and batch queues per country in the biomed VO (January 2013).

is used to locate resources and for monitoring and accounting purposes. Job management services handle the execution of tasks on the grid. The computing element (CE) represents a set of computing resources (worker nodes) localized at a resource center (e.g. cluster) and is responsible for the local task management. The data services manage the location and movement of data among the resource centers of the grid. The storage element (SE) provides a common interface to the storage backend available at the resource center. Meta-data information of each file in the SEs are mapped by a logical file catalog (LFC).

EGI evaluates the quality of its grid services by measuring the availability and reliability of resources. Service availability denotes the ratio of time that a service was up and running; service reliability is the ratio of time that a service was supposed to be up and running, excluding scheduled downtime and maintenance. Figure 2.8 shows the measured values for availability and reliability of EGI resource centers from May 2010 to January 2012. Although the infrastructure shows increasing evidence of availability and reliability, fault-tolerance mechanisms should still be developed by science-gateways.

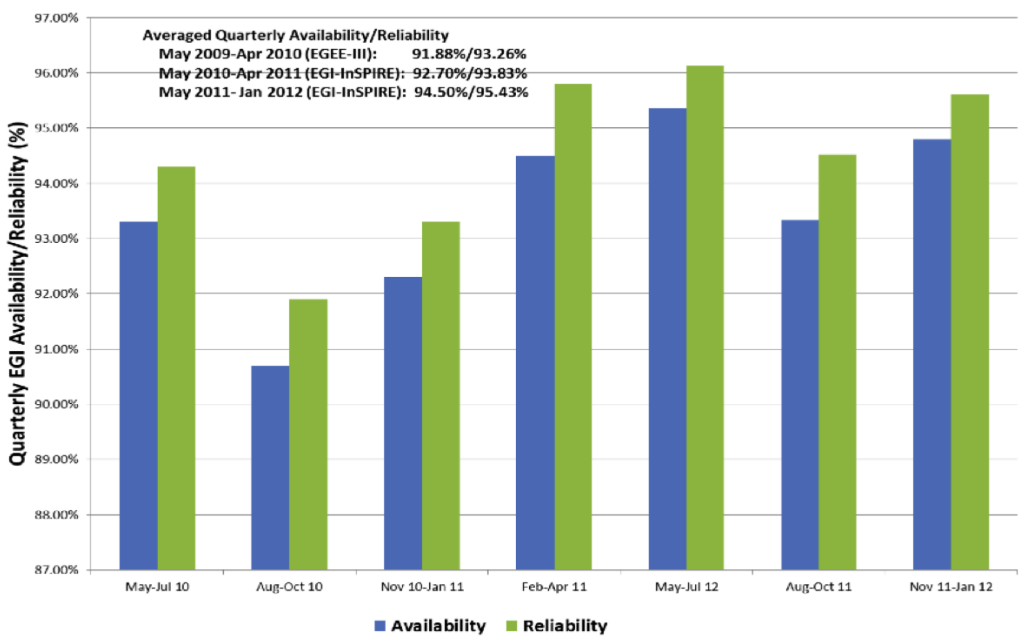


Figure 2.8: Resource centers availability and reliability across EGI from May 2010 to January 2012.*

* Extracted from "Annual Report on the EGI Production Infrastructure" (https://documents.egi.eu/public/ShowDocument?docid=1059)

2.2.6 Workflow execution

For a user, an application execution consists of select an application, upload input data, launch a workflow, and download results (steps 1, 2 and 11 from Figure 2.9). For the platform, it consists of performing a workflow execution. A workflow description and a set of input parameters is received and processed by MOTEUR, which produces invocations; from invocations GASW generates grid tasks, and submits to the workload management system. DIRAC deploys pilot jobs on computing resources; pilot jobs fetch user tasks and execute them; task execution consists of downloading input data, executing the application, and uploading results. Figure 2.9 summarizes this process.

2.3 Platform usage

The Virtual Imaging Platform was put in production on January 2011. To date, 441 users from 49 countries are registered in VIP, among which 62 logged in in July 2013 . Figure 2.10 (top) shows the repartition of users per country. Since January 2012, VIP has, in average, 2,000 page views and 150 unique visits per month among which about 16% of these visits are performed by newcomers[8]. According to EGI, VIP's robot certificate has the greatest number of registered

[8]Reported by Google Analytics.

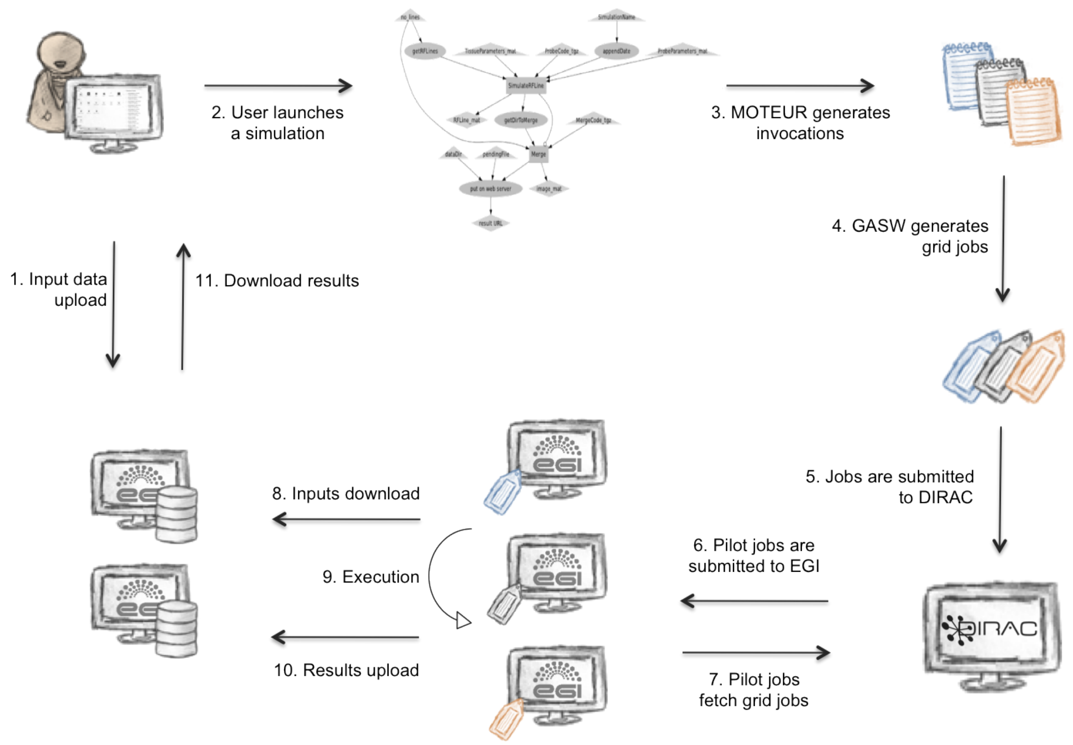

2. User launches a simulation
3. MOTEUR generates invocations
4. GASW generates grid jobs
1. Input data upload
11. Download results
5. Jobs are submitted to DIRAC
8. Inputs download
6. Pilot jobs are submitted to EGI
9. Execution
10. Results upload
7. Pilot jobs fetch grid jobs
DIRAC
egi

Figure 2.9: Workflow execution flow in VIP.

users in Europe[9]. Since January 2011, 11,514 workflow executions were launched through the platform that represents in average a monthly CPU consumption about 18.95 years and a yearly cumulative of 379 years. In August 2012, VIP started to use France-Grilles' DIRAC national instance. Figure 2.10 (bottom) shows VIP CPU consumption since then. This level of activity demonstrates VIP usability, availability and reliability.

2.4 Challenges and limitations

VIP has no *a-priori* model of the execution time of their applications because (*i*) task costs depend on input data with no explicit model, and (*ii*) characteristics of the available resources, in particular network and RAM, depend on background load. Modeling application execution time in these conditions requires cumbersome experiments which cannot be conducted for every new application in the platform. As a consequence, such science-gateways platforms operate in *non-clairvoyant* conditions, where little is known about executions before they actually happen. Such platforms also run in *online* conditions, i.e., users may launch or cancel applications at any time and resources may appear or disappear at any time too.

Resource heterogeneity of production grids, such as EGI, raises workflow execution issues: input and output data transfers may fail because of network glitches or limited site intercommunication; application executions may fail because of corrupted executable files, missing

[9]https://wiki.egi.eu/wiki/EGI_robot_certificate_users

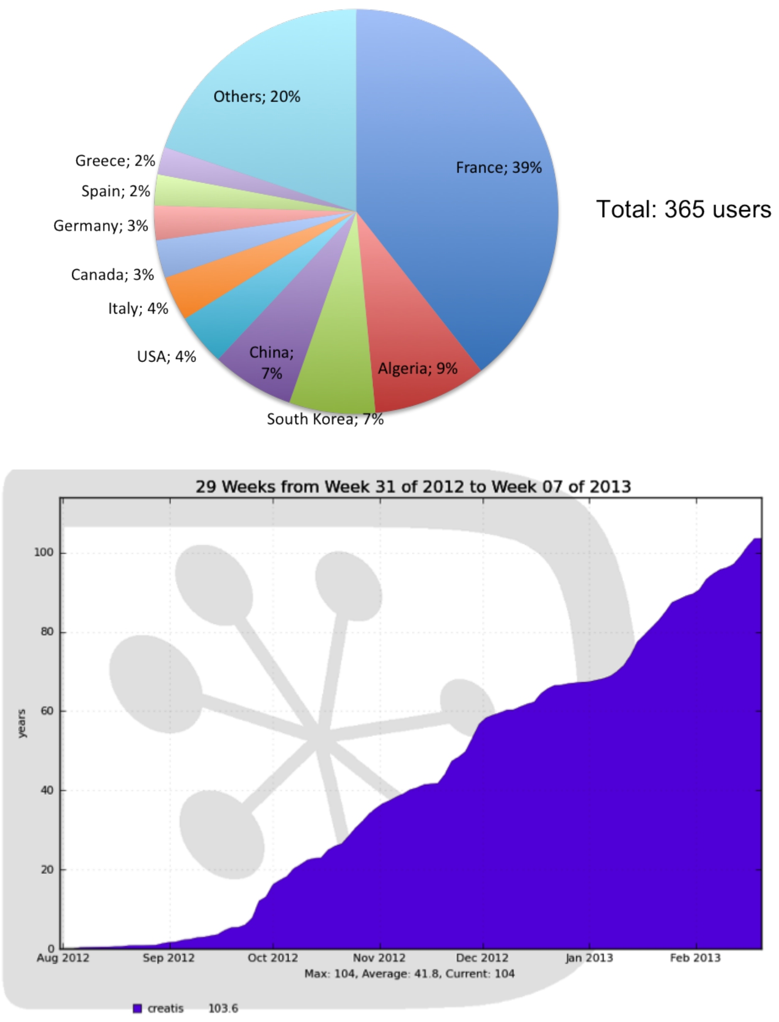


Figure 2.10: Repartition of users per country on VIP in February 2013 (top) and VIP consumption since August 2012 reported by France-Grilles (bottom).

dependencies, or incompatibility; application executions may slowdown because of resources with poorer performance. Therefore, methods have to be designed to handle that. We address this issue in Chapters 4 and 5.

In platforms where the communication overhead and queueing time are high, the performance of short tasks may slowdown the workflow execution. Task grouping techniques (Chapter 6) could be used to group tasks with shared input data to save data transfers time and queueing time.

Performance of small workflow executions is a common issue on platforms where resources are shared among several users and communities. Although pilot jobs are used to reduce latency, tasks still have to queue from 1 minute to 1 hour. A few dedicated resources could be used to address this problem, but this approach falls in a scheduling problem of task prioritization studied in Chapter 7.

2.5 Conclusion

This chapter provided a complete overview of VIP architecture for workflow execution presenting employed methods and techniques. Users only need a valid email address to have access to the set of applications available through a web portal. A robot certificate operates grid operations on behalf of the user. Applications are described as workflows and enacted by the MOTEUR workflow engine. Resource provisioning and task scheduling is provided by DIRAC using so-called "pilot jobs". Tasks are executed on the biomed virtual organization of the European Grid Infrastructure (EGI). The platform currently has 441 registered users who consumed 379 years of CPU time since January 2011.

We also introduced issues and limitations related to the execution infrastructure and to the adoption of robot certificates. Manual interventions can be seen as a possible solution, but it is expensive and affordable only for small production systems. Therefore, the development of automated methods is unavoidable to target these issues on science-gateways, such as VIP.

To develop such methods, fine-grained information about application executions is required. This level of information is not found in common workload archives obtained at the infrastructure-level, such as the Parallel Workload Archive[10], the Grid Workload Archive [Iosup et al., 2008], and the Grid Observatory [Germain-Renaud et al., 2011]. In the next chapter we present a workload archive [Ferreira da Silva and Glatard, 2013] acquired at science-gateway level, and we show its added value on several case studies related to user accounting, pilot jobs, fine-grained task analysis, bag of tasks, and workflows. The workload archive provides the historical information for our self-healing process for autonomous detection and handling of operational incidents in workflow execution (Chapter 4).

[10]www.cs.huji.ac.il/labs/parallel/workload/logs.html

Chapter 3

A science-gateway workload archive

Contents

Archives of distributed workloads acquired at the infrastructure level reputably lack information about users and application-level middleware, while such fine-grained information is fundamental for the development of self-healing methods. Science-gateways provide consistent access points to the infrastructure, and therefore are an interesting information source to cope with this issue. In this chapter, we describe a workload archive acquired at the science-gateway level, and we show its added value on several case studies related to user accounting, pilot jobs, fine-grained task analysis, bag of tasks, and workflows. Results show that science-gateway workload archives can detect workload wrapped in pilot jobs, improve user identification, give information on distributions of data transfer times, make bag-of-task detection accurate, and retrieve characteristics of workflow executions. Some limits are also identified.

Les archives de traces d'exécution acquises au niveau des infrastructures manquent d'information sur les utilisateurs et les applications. Ce niveau fin d'information est fondamental pour le développement de méthodes d'auto-guérison. Les « science-gateways » fournissent des points d'accès compatibles avec l'infrastructure, et sont donc une source d'information intéressante pour faire face à ce problème. Dans ce chapitre, nous décrivons une archive de traces d'exécution acquise auprès du science-gateway, et nous montrons sa valeur ajoutée sur plusieurs études de cas : le suivi de l'activité des utilisateurs, les tâches pilotes, l'analyse détaillée des tâches, les tâches indépendentes (« bag of tasks ») et workflows. Les résultats montrent que l'archive de traces d'exécution peut détecter la charge de travail encapsulé dans des tâches pilotes, améliorer l'identification des utilisateurs, donner des informations sur les distributions de temps de transfert de données, assurer la détection précise de tâches indépendentes, et récupérer les caractéristiques des exécutions de workflow. Certaines limites sont également identifiées.

3.1 Introduction

Grid workload archives [Iosup et al., 2008, Iosup and Epema, 2011, Kondo et al., 2010, Germain-Renaud et al., 2011, Ostermann et al., 2008] are widely used for research on distributed systems, to validate assumptions, to model computational activity [Christodoulopoulos et al., 2008, Medernach, 2005], and to evaluate methods in simulation or in experimental conditions. Available workload archives are acquired at the infrastructure level, by computing sites or by central monitoring and bookkeeping services. Scientific gateways users, such as VIP (Chapter 2), access the infrastructure through stacks of application-level middleware such as a workflow engine, an application wrapper, a pilot-job system, and a web portal. As a result, workload archives lack critical information about dependencies among tasks, about task sub-steps, about artifacts introduced by application-level scheduling, and about users. Methods have been proposed to recover this information. For instance, [Iosup et al., 2007] detects bags of tasks as tasks submitted by a single user in a given time interval. In other cases, information can hardly be recovered: [Iosup and Epema, 2011] reports that there is currently no study of a pilot-job workload, and workflow studies such as [Ostermann et al., 2008] are mostly limited to test runs conducted by developers.

Science-gateways, however, gathers rich information about workload patterns. They can provide fine-grained information on user accounting, pilot jobs, bag of tasks, and workflows. This chapter describes a science-gateway workload archive [Ferreira da Silva and Glatard, 2013], and illustrates its added value w.r.t. archives acquired at the infrastructure level. In the second part of this manuscript, this workload archive will be used to provide historical information for our self-healing process for autonomous detection and handling of operational incidents in workflow executions.

The workload archive model is presented in Section 3.2 and used in 5 case studies in Section 3.3: Section 3.3.1 studies pilot jobs, Section 3.3.2 compares user accounting to data acquired by the infrastructure, Section 3.3.3 performs fine-grained task analysis, Section 3.3.4 evaluates the accuracy of bag of task detection [Iosup et al., 2007] from infrastructure-level traces, and Section 3.3.5 analyzes workflows in production. Section 3.4 concludes the chapter.

3.2 A Science-Gateway Workload Archive

Science gateways usually involve a subset or all the entities of VIP's architecture shown on Figure. 2.9 (Chapter 2). This science gateway model totally applies to e-bioinfra, and partly to the P-Grade portal, the Science-Gateway framework in [Ardizzone et al., 2011], medigrid-DE, and CBRAIN.

Our science-gateway archive model adopts the schema on Figure. 3.1. `Task` contains information such as final status, exit code, timestamps of internal steps, application and workflow activity name. Each task is associated to a `Pilot Job`. `Workflow Execution` gathers all the activities and tasks of a workflow execution, `Site` connects pilots and tasks to a grid site, and `File` provides the list of files associated to a task and workflow execution. In this work we focus on `Task`, `Workflow Execution` and `Pilot Job`.

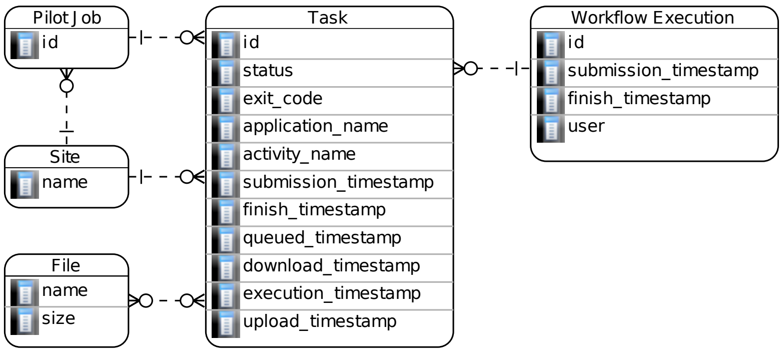


Figure 3.1: Science-gateway archive model.

The science-gateway archive is extracted from VIP. `Task`, `Site` and `Workflow Execution` information are acquired from databases populated by the MOTEUR workflow engine at runtime. `File` and `Pilot Job` information are extracted from the parsing of task standard output and error files.

Studies presented in the following Sections are based on the workload of the VIP from January 2011 to April 2012. It consists of 2,941 workflow executions, 112 users, 339,545 pilot jobs, 680,988 tasks where 338,989 are completed tasks, 138,480 error tasks, 105,488 aborted tasks, 15,576 aborted task replicas, 48,293 stalled tasks and 34,162 submitted or queued tasks;

task average waiting time is about 36 minutes. Stalled tasks are tasks which lost communication with the pilot manager, e.g. because they were killed by computing sites due to quota violation. Tasks ran on the biomed virtual organization of the European Grid Infrastructure (EGI). Traces used in this work are available to the community in the Grid Observatory[1].

3.3 Case studies

3.3.1 Pilot jobs

As mentioned in Chapter 2, pilot jobs are increasingly used to improve scheduling and reliability on production grids [Luckow et al., 2012, Tsaregorodtsev et al., 2009, Thain et al., 2005]. This type of workload, however, is difficult to analyze from infrastructure traces as a single pilot can wrap several tasks, which remains unknown to the infrastructure. In our case, pilots are discarded after 5 task executions, if the remaining walltime allowed on the site cannot be obtained, if they are idle for more than 10 minutes, or if one of their tasks fails. Pilots can execute any task submitted by the science gateway, regardless of the workflow execution and user.

Figure. 3.2 shows the number of tasks and users per pilot in the archive. The figure shows only 453,547 out of 646,826 executed tasks, i.e. 70% of the complete task set. This amount corresponds to tasks where a standard output containing the pilot id could be retrieved. Most pilots (83%) execute only 1 task due to walltime limits or other discards. These 83% execute 282,331 tasks, which represents 62% of the considered tasks. Workload acquired at the infrastructure level would usually assimilate pilot jobs to tasks. Our data shows that this hypothesis is only true for 62% of the tasks. The distribution of users per pilots has a similar decrease: 95% of the pilots execute tasks of a single user.

3.3.2 Accounting

On a production platform like EGI, accounting data consists of the list of active users and their number of submitted jobs, consumed CPU time, and wall-clock time. Here, we compare data provided by the infrastructure-level accounting services of EGI[2] to data obtained from the science-gateway archive.

Figure. 3.3 compares the number of users reported by EGI and the scientific gateway. It shows a dramatic discrepancy between the two sources of information, explained by the use of a robot certificate in the gateway. Robot certificates are regular X.509 user certificates that are used for all grid operations performed by a science gateway, namely data transfers and task submission. From an EGI point of view, all VIP users are accounted as a single user

[1]http://www.grid-observatory.org
[2]http://accounting.egi.eu

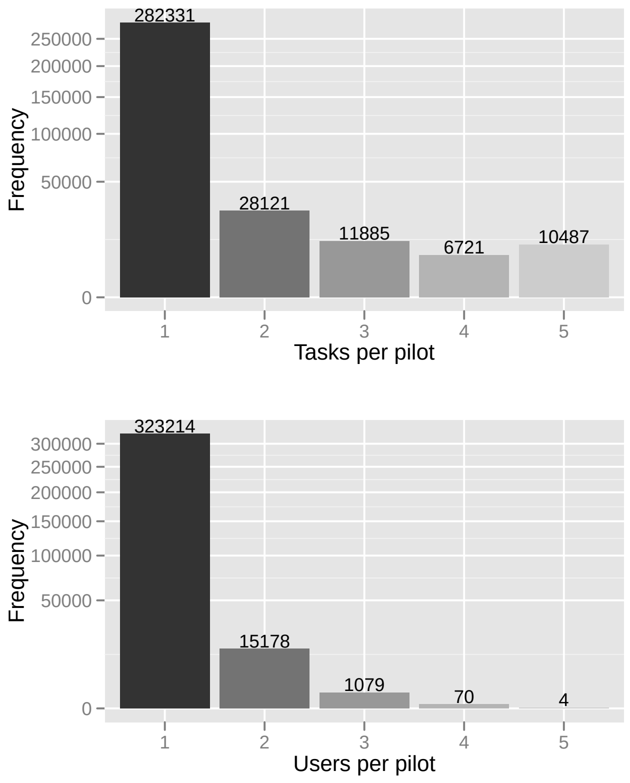


Figure 3.2: Histogram of tasks per pilot (top) and users per pilot (bottom).

regardless of their real identity. EGI reports more than one user for months 12, 13, 15 and 16 due to updates of the VIP certificate. The adoption of robot certificates totally discards the accounting of user names at the infrastructure level. Studies such as presented on Figure 17 in [Ilijasic and Saitta, 2009] or on Figure 1 in [Iosup and Epema, 2011], cannot be considered reliable in this context. Robot certificates are not an exception: a survey available online[3] shows that 80 of such certificates are known on EGI. By avoiding the need for users to request personal certificates, they simplify the access to the infrastructure to a point that their very large adoption in science gateways seems unavoidable.

Figure 3.4 compares the number of submitted jobs, consumed CPU time, and consumed wall-clock time obtained by the EGI infrastructure and by VIP. The number of jobs reported by EGI is almost twice as important as in VIP. This huge discrepancy is explained by the fact that many pilot jobs do not register to the pilot system due to some technical issues, or do not execute any task due to the absence of workload, or execute tasks for which no standard output containing the pilot id could be retrieved. These pilots cannot be identified from the task logs. While this highlights serious potential improvements in the pilot manager, it also reveals that a significant fraction of the workload measured by EGI does not come from applications but

[3]https://wiki.egi.eu/wiki/EGI_robot_certificate_users

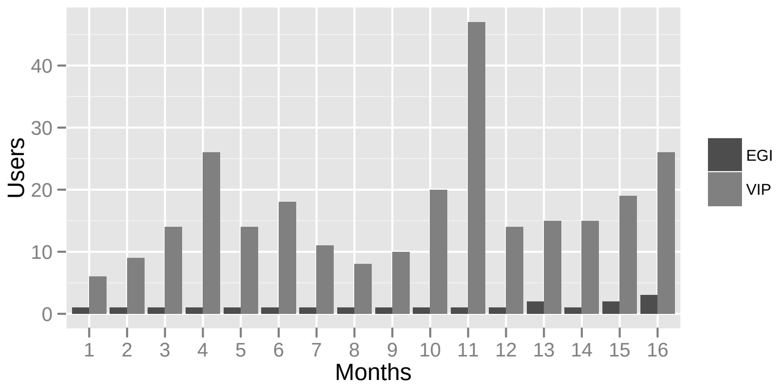


Figure 3.3: Number of reported EGI and VIP users.

from artifacts introduced by pilot managers. This should be taken into account when conducting studies on application-level schedulers from workload acquired at the infrastructure level.

About 60 walltime years are missing from the science gateway archive, compared to the infrastructure. This is due to the pilot setup time (a few minutes per pilot), and to the computing time of lost tasks, for which no standard output containing monitoring data could be retrieved. Tasks are lost (a.k.a stalled) in case of technical issues such as network interruption or deliberate kill from sites due to quota violation. For instance, Table 3.1 shows the consumed CPU time repartition among tasks of a workflow execution. Stalled tasks represent 4% of the total number of tasks and consume about 38% of the total CPU time.

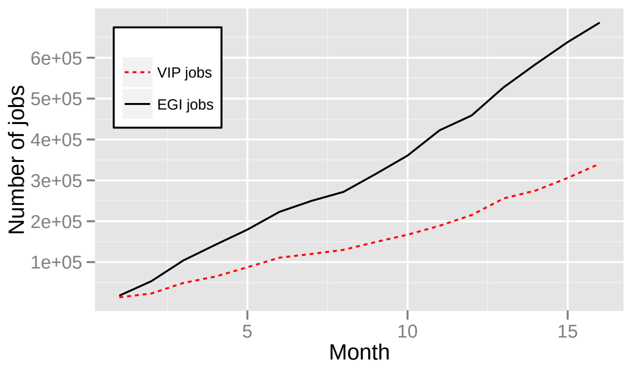
Task status	Number of tasks	Consumed CPU time (s)
Completed	122	55,147
Stalled	5	36,414
Error	4	4,905

Table 3.1: Consumed CPU time repartition among completed, errors, and stalled tasks of a workflow execution.

3.3.3 Task analysis

Traces acquired at the science-gateway level provide fine-grained information about tasks, which is usually not possible at the infrastructure level. Figure 3.5 shows the distributions of download, upload and execution times for successfully completed tasks. Distributions show a substantial amount of very long steps.

Error causes can also be investigated from science-gateway archives. Figure 3.6 (left) shows the occurrence of 6 task-level errors. These error codes are application-specific and not accessible to infrastructure level archives, see e.g. [Lingrand et al., 2010] (Table 3) and [Kondo et al., 2010].



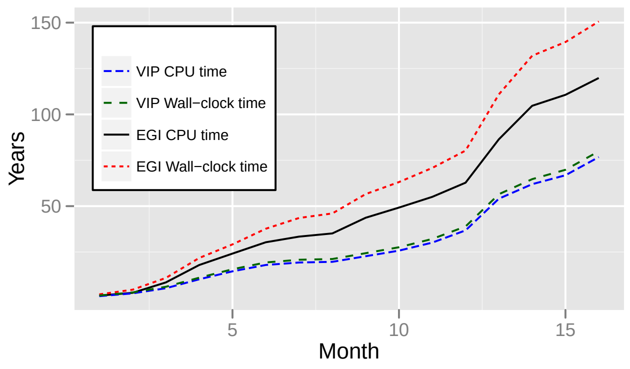


Figure 3.4: Number of submitted pilot jobs (top), and consumed CPU and wall-clock time (bottom) by the infrastructure (EGI) and the science gateway (VIP).

A common strategy to cope with recoverable errors is to replicate tasks [Casanova, 2006], which is usually not known to the infrastructure. Figure 3.6 (right) shows the occurrence of task replication in the science-gateway archive.

3.3.4 Bag of tasks

In this section, we evaluate the accuracy of the method presented in [Iosup et al., 2007] to detect bag of tasks (BoT). This method considers that two tasks successively submitted by a user belong to the same BoT if the time interval between their submission times is lower or equal to a time Δ. The value of Δ is set to $120s$ as described in [Iosup et al., 2007]. Figure 3.7 presents the impact of Δ on BoT sizes (a.k.a. batch sizes) for $\Delta = 10s, 30s, 60s$ and $120s$.

Figure 3.8 presents the comparison of BoT characteristics obtained from the described method for $\Delta = 120s$ and from VIP. BoTs in VIP were extracted as the tasks generated by the same activity in a workflow execution. Thus, they can be considered as ground truth and are named `Real Non-Batch` for single-task BoTs and `Real Batch` for others. Analogously, we name `Non-Batch` and `Batch` BoTs determined by the method. `Batch` has about 90% of its BoT sizes ranging from 2 to 10 while these batches represent about 50% of `Real Batch`. This discrepancy has a direct impact on the BoT duration (makespan), inter-arrival time and

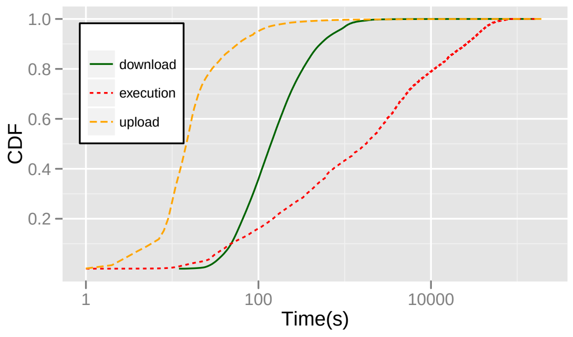


Figure 3.5: Different steps in task life.

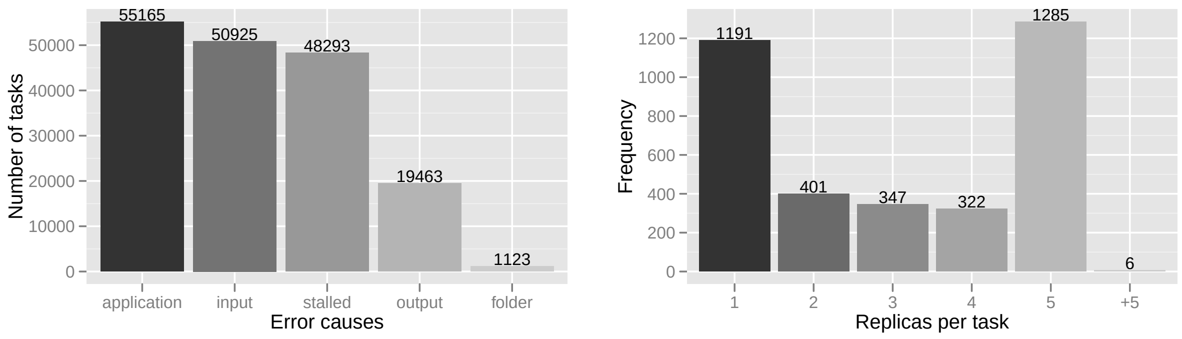


Figure 3.6: Task error causes (left) and number of replicas per task (right).

consumed CPU time. The duration of `Non-Batch` are overestimated up to 400%, inter-arrival times for both `Batch` and `Non-Batch` are underestimated by about 30% in almost all intervals, and consumed CPU times are underestimated of 25% for `Non-Batch` and of about 20% for `Batch`. This data shows that detecting bag of tasks based on infrastructure-level traces is very inaccurate. Such inaccuracy may have important consequences on works based on such detection, e.g. [Brasileiro et al., 2011].

3.3.5 Workflows

Few works study the characterization of grid workflow executions. In [Ostermann et al., 2008], the authors present the characterization of 2 workloads that are mostly test runs conducted by developers. To the best of our knowledge, there is no work on the characterization of grid workflows in production.

Figure 3.9 presents characteristics of the workflow executions extracted from our science-gateway archive. They could be used to build workload generators for the evaluation of scheduling algorithms. Let N be the number of tasks in a workflow execution; we redefine the 3 classes presented in [Ostermann et al., 2008] to `small` for $N \leq 100$, `medium` for $100 < N \leq 500$ and

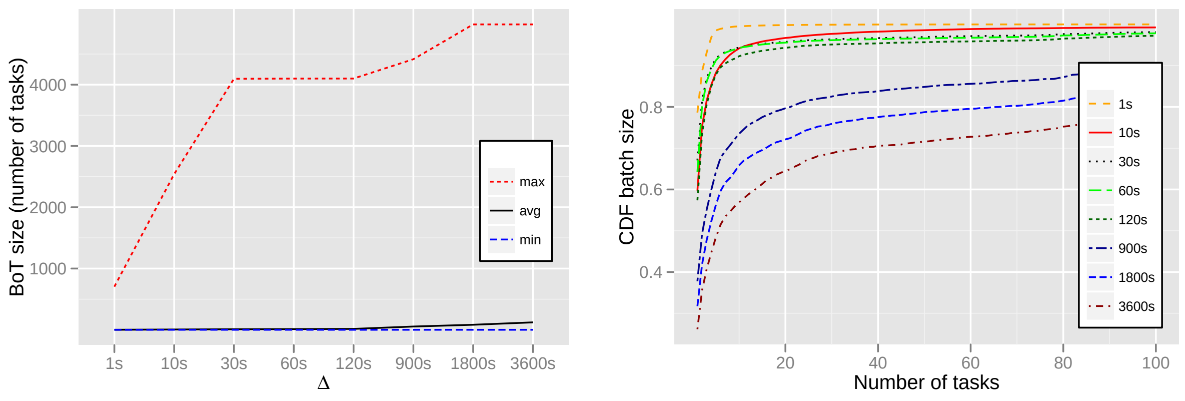


Figure 3.7: Impact of parameter Δ on BoT sizes: minimum, maximum and average values (*left*); and its distribution (*right*).

large for $N > 500$. From Figure 3.9 (top left), we observe that the workload is composed by 52%, 31% and 17% of small, medium and large executions respectively. In Figure 3.9 (top right), 90% of small, 66% of medium and 54% of large executions have a makespan lower than 14 hours. Figure 3.9 (bottom left) shows that speed-up increases with the size of the workflow, which is commonly observed. Critical path lengths are mostly up to 2 levels for small and large executions and up to 3 for medium executions (bottom right of Figure 3.9).

3.4 Conclusion

We presented a science-gateway model of workload archive containing detailed information about users, pilot jobs, task sub-steps, bag of tasks and workflow executions. We illustrated the added value of science-gateway workloads compared to infrastructure-level traces using information collected by the Virtual Imaging Platform in 2011/2012, which consist of 2,941 workflow executions, 339,545 pilot jobs, 680,988 tasks and 112 users that consumed about 76 CPU years.

Several conclusions demonstrate the added-value of a science-gateway approach to workload archives. First, it can exactly identify tasks and users, while infrastructure-level traces cannot identify 38% of the tasks due to their bundling in pilot jobs, and cannot properly identify users when robot certificates are used. Infrastructure archives are also hampered by additional workload artifacts coming from pilot-job schedulers, which can be distinguished from application workload using science-gateway archives. More detailed information about tasks is also available from science-gateway traces, such as distributions of download, upload and execution times, and information about replication. Besides, the detection of bag of tasks from infrastructure traces is inaccurate, while a science-gateway contains ground truth. Finally, we reported a few parameters on workflow executions, which could not be extracted from infrastructure-level

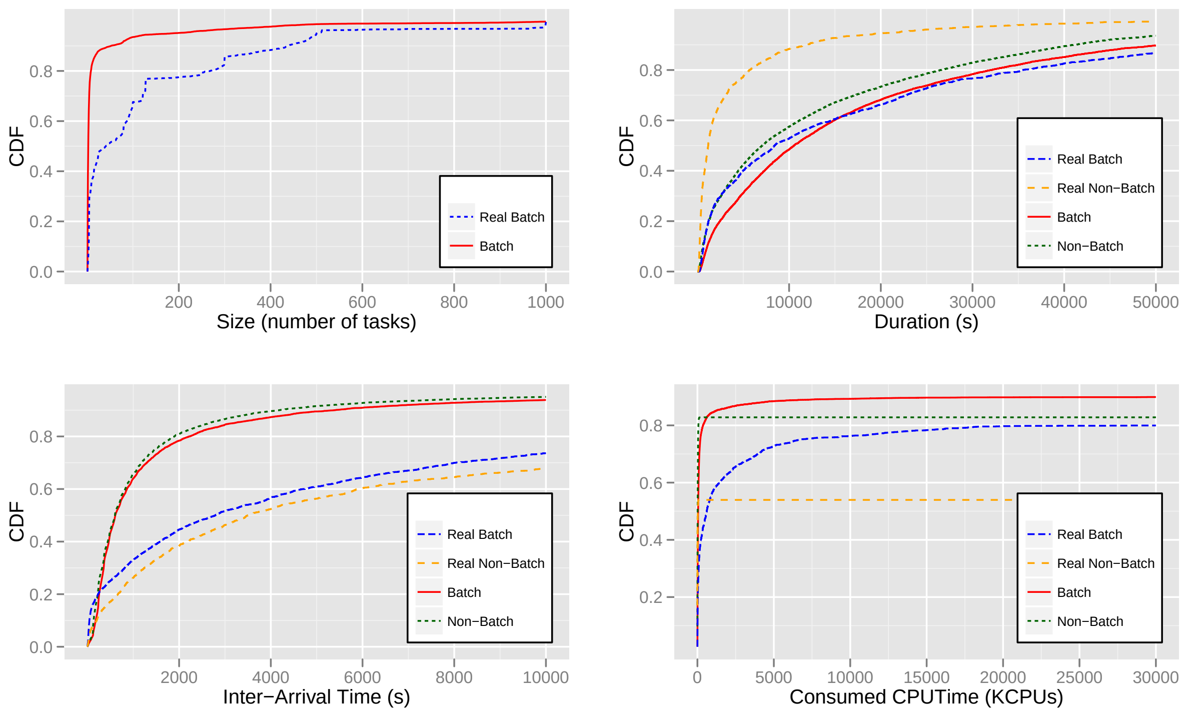


Figure 3.8: CDFs of characteristics of batched and non-batched submissions: *BoT sizes*, *duration per BoT*, *inter-arrival time* and *consumed CPU time* ($\Delta = 120s$).

traces. Limits of science-gateway workloads still exist. In particular, it is very common that a significant fraction of lost tasks do not report complete information.

Traces acquired by the Virtual Imaging Platform will be regularly made available to the community in the Grid Observatory. We hope that other science-gateway providers could also start publishing their traces so that computer-science studies can better investigate production conditions. In the second part of this thesis, the science-gateway workload archive is used to feed our self-healing algorithms with historical information. It can also be used to elaborate benchmarks, or to simulate applications and algorithms targeting production systems.

Depending on the interest or application, more information could be extracted from the science-gateway traces. For instance, information about file access pattern, about the number and location of computing sites used per workflow or bag-of-task execution, and about task resubmission is available in the archive.

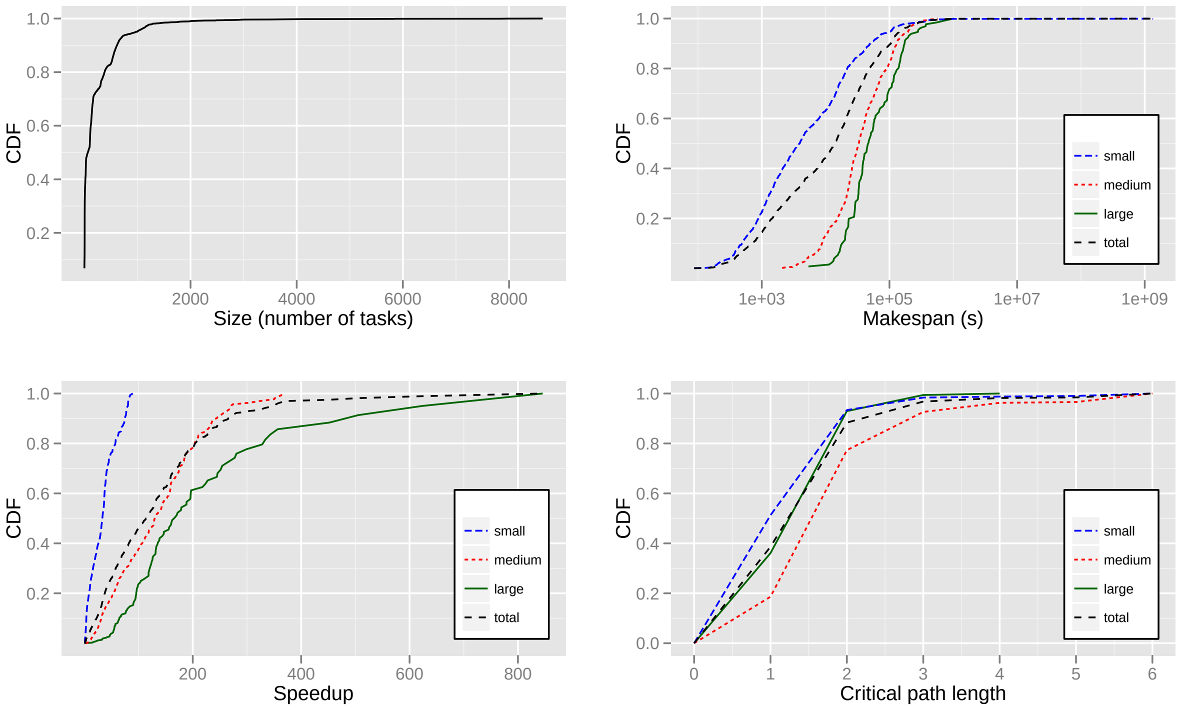


Figure 3.9: Characteristics of workflow executions: number of tasks (*top left*), CPU time and makespan (*top right*), speedup (*bottom left*) and critical path length (*bottom right*).

Part II

Self-healing of workflow executions on grids

Chapter 4

A self-healing process for workflow executions on grids

Contents

The management of science gateways such as VIP requires important human intervention to handle operational incidents. This chapter presents a self-healing process for workflow executions. The process quantifies incident degrees from metrics that can be computed online. These metrics make little assumptions on the application or resource characteristics. From their degree, incidents are classified in levels and associated to sets of healing actions. The healing process is parametrized on real application traces acquired in production on the European Grid Infrastructure. We present a first example of this process on 7 basic incidents related to task errors. Experimental results obtained in the Virtual Imaging Platform show that the proposed method properly detects and handles recoverable and unrecoverable errors. The next chapters exploit this process on more complex problems.

La gestion des « scientific gateways » nécessite une intervention humaine importante pour traiter les incidents opérationnels. Ce chapitre présente un processus d'auto-administration pour les exécutions de workflow. Le processus quantifie le degré de gravité des incidents à partir de paramètres qui peuvent être calculés en ligne. Les métriques utilisées emploient peu de caractéristiques d'application ou de ressource. À partir de leur degré de gravité, les incidents sont classés afin de leur appliquer les actions de guérison appropriées. Le processus de guérison est paramétré à partir de traces d'applications réelles acquises sur EGI. Nous présentons un premier exemple de ce processus sur sept incidents. Les résultats expérimentaux obtenus dans la plate-forme VIP montrent que la méthode proposée détecte et gère les erreurs récupérables et irrécupérables de manière adaptée. Les chapitres suivants exploitent ce processus sur des problèmes plus complexes.

4.1 Introduction

Science-gateways, such as VIP (see Chapter 2), provide access to important amounts of resources transparently. However, their large scale and the number of middleware systems involved lead to many system errors and faults. Easy-to-use interfaces provided by these gateways exacerbate the need for properly solving operational incidents encountered on grid computing infrastructures since end users expect high reliability and performance with no extra monitoring or parametrization from their side. In practice, such services are often backed by substantial support staff who monitors running experiments by performing simple yet crucial actions such as rescheduling tasks, restarting services, killing misbehaving runs or replicating data files to reliable storage facilities. Fair QoS can then be delivered, yet with important human intervention. For instance, VIP administration demands an important amount of time and accurate expertise on workflow execution and grid computing. Figure 4.1 shows a diagram of all instances involved in a workflow execution based on the VIP's architecture (see Chapter 2). All these instances are prone to errors from the task execution level, such as application execution failures, or unavailability of files, up to the web portal level, such as unscheduled downtimes.

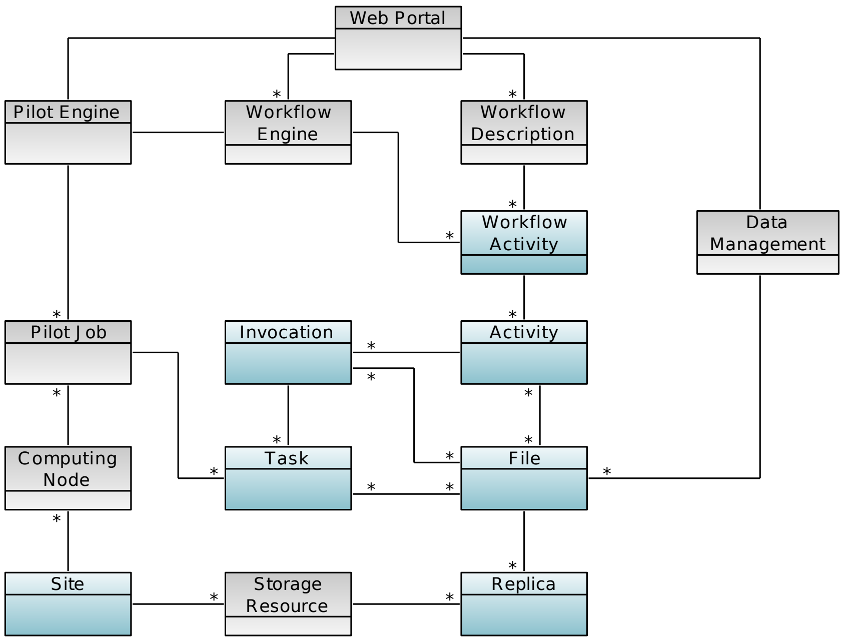


Figure 4.1: Instances involved in a workflow execution.

Automating such operations is challenging for two reasons. First, the problem is *online* by nature because no reliable user activity prediction can be assumed, and new workloads may arrive at any time. Therefore, the considered metrics, decisions and actions have to remain simple and to yield results while the application is still executing. Second, it is *non-clairvoyant* due to the lack of information about applications and resources in production conditions. Computing resources are usually dynamically provisioned from heterogeneous clusters, clouds or desktop grids without any reliable estimate of their availability and characteristics. Models of application execution times are hardly available either, in particular on heterogeneous computing resources. To the best of our knowledge, there is no such framework that address both conditions, online and non-clairvoyant.

In this chapter, we propose a general healing process for workflow executions. Instances are modeled as Fuzzy Finite State Machines (FuSM) [Malik et al., 1994] where state degrees of membership are determined by an external healing process. Degrees of membership are computed from metrics assuming that incidents have outlier performance, e.g. a site or a particular invocation behaves differently than the others. Based on incident degrees, the healing process identifies incident levels using thresholds determined from platform history. A specific set of actions is then selected from association rules among incident levels.

In this manuscript, we consider seven of the instances depicted in Figure 4.1: `Workflow`

`Activity`, `Activity`, `Invocation`, `Task`, `File`, `Replica`, and `Site`. In this chapter, we illustrate the healing process on seven incident cases related to input and output data transfer errors, and application execution errors. These incidents involve all instances, except `Workflow Activity`. In the next chapters, we instantiate the healing process to late task executions and task granularity incidents involving the `Activity` instance, and unfairness among workflow executions incident involving `Workflow Activity` instance.

Our general self-healing process is described in Section 4.2: we show how metrics used to quantify incident degrees are determined, and how we characterize incident levels. In Section 4.3 we present a first example of this process on 7 basic incidents related to task errors. Section 4.4 shows improvements to mode detection, and Section 4.5 concludes the chapter.

4.2 General healing process

An instance is modeled as a FuSM as shown on Figure 4.2. The instance is initialized in `Initializing` where it prepares the execution (e.g., invocation submission for activities). `Running` is a state where no particular issue is detected; no action is taken and the instance is assumed to behave normally. `Completed` (resp. `Failed`) is a terminal state used when the instance successfully completes it execution (resp. the execution finishes with errors). These 4 states are crisp (not fuzzy) and exclusive. Their degree can only be 0 or 1 and if 1 then all other states have a degree 0. The other states are fuzzy states corresponding to detected incidents.

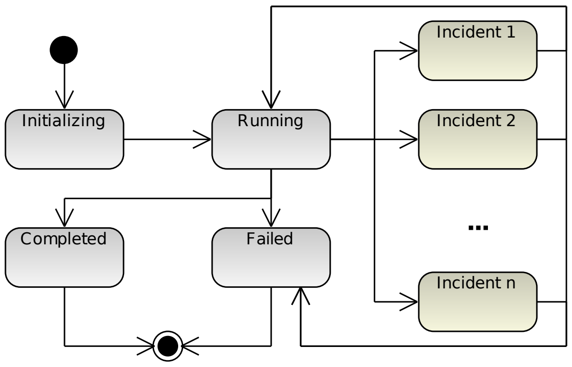


Figure 4.2: Fuzzy Finite State Machine (FuSM) representing a general instance.

The healing process sets the degree of FuSM states from incident detection metrics. Then, it determines actions to address the incidents. If no action is required then the process waits until an event occurs (e.g., task status change) or a timeout is reached. The process is described formally in the next paragraphs.

Let $I = \{x_i, i = 1, \ldots, n\}$ be the set of possible incidents and $\eta = (\eta_1, \ldots, \eta_n) \in [0, 1]^n$ their degrees in the FuSM. Incident x_i can occur at m_i different levels $\{x_{i,j}, j = 1, \ldots, m_i\}$ delimited by thresholds values $\tau_i = \{\tau_{i,j}, 1, \ldots, m_i\}$. The level of incident i is determined by j such that

$\tau_{i,j} \leq \eta_i < \tau_{i,j+1}$. A set of actions $a_i(j)$ is available to address $x_{i,j}$:

$$\begin{aligned} a_i : [1, m_i] &\to \wp(A) \\ j &\mapsto a_i(j) \end{aligned}$$

where A is the set of possible actions taken by the healing process and $\wp(A)$ is the power set of A.

In addition to the incidents themselves, incident co-occurrences are taken into account. Association rules [Agrawal et al., 1993] are used to identify relations between levels of different incidents. Association rules to $x_{i,j}$ are defined as $R_{i,j} = \{r_{i,j}^{u,v} = (x_{u,v}, x_{i,j}, \rho_{i,j}^{u,v})\}$. Rule $r_{i,j}^{u,v}$ means that when $x_{u,v}$ happens then $x_{i,j}$ also happens with confidence $\rho_{i,j}^{u,v} \in [0, 1]$. The confidence of a rule is an estimate of probability $P(x_{i,j}|x_{u,v})$. For the sake of completeness, $r_{i,j}^{i,j} \in R_{i,j}$ and $\rho_{i,j}^{i,j} = 1$. We also define $R = \bigcup_{i\in[\![1,n]\!], j\in[\![1,m_i]\!]} R_{i,j}$. The inference made by an association rule does not necessarily imply causality. Instead, it quantifies co-occurrence between the rule's terms [Tan et al., 2005].

Algorithm 1 presents the algorithm used at each iteration of the healing process. Incident degrees are determined based on metrics and incident levels j are obtained from historical data as explained in the next section. A roulette wheel selection [De Jong, 1975] based on η is performed to select $x_{i,j}$ the incident level of interest for the iteration. In a roulette wheel selection, incident x_i is selected with a probability p_i proportional to its degree: $p(x_i) = \eta_i \int_{j=1}^{n} \eta_j$. A potential cause $x_{u,v}$ for incident $x_{i,j}$ is then selected from another roulette wheel selection on the association rules $r_{i,j}^{u,v}$, where x_u is at level v. Rule $r_{i,j}^{u,v}$ is weighted $\eta_u \times \rho_{i,j}^{u,v}$ in this second roulette selection. Only first-order causes are considered here but the approach could be extended to include more recursion levels. Note that $r_{i,j}^{i,j}$ participates in this selection so that a first-order cause is not systematically chosen. Finally, actions in $a_u(v)$ are performed.

Algorithm 1 One iteration of the healing process.

```
1: input: history of $\eta$
2: Output: set of actions $a$
3: wait for event or timeout
4: determine incident degrees $\eta \in [0, 1]$ based on metrics
5: determine incident levels $j$ such that $\tau_{i,j} \leq \eta_i < \tau_{i,j+1}$
6: select incident $x_i$ by roulette wheel selection based on $\eta$
7: select rule $r_{u,v} = (x_{u,v}, x_{i,j}, \rho_{i,j}^{u,v}) \in R_{i,j}$ by roulette wheel selection based on $\eta_u \times \rho_{i,j}^{u,v}$, where $x_u$ is at level $v$
8: $a = a_u(v)$
9: perform actions in $a$
```

Table 4.1 illustrates this mechanism on an example case where only 3 incidents are considered, and Figure 4.3 shows it as a MAPE-K loop.

Incidents degrees η_i are determined by metrics identified by human operators (step `02` on Algorithm 1). These metrics must be computable on online conditions. In addition, we con-

Step `02` and `03`: incident degrees and levels are determined:

x_i: incident	Degree η_i	Level j
x_1	0.8	2
x_2	0.1	1
x_3	0.4	1

Step `04`: $x_{1,2}$ is selected with probability $\frac{0.8}{0.8+0.4+0.1}$.

Step `05`: association rules $r_{1,2}^{2,1}$, $r_{1,2}^{3,1}$ and $r_{1,2}^{1,2}$ are considered:

Rule	Confidence
$r_{1,2}^{2,1}$: $x_{2,1} \rightarrow x_{1,2}$	0.8
$r_{1,2}^{3,1}$: $x_{3,1} \rightarrow x_{1,2}$	0.2
$r_{1,2}^{1,2}$: $x_{1,2} \rightarrow x_{1,2}$	1

$r_{1,2}^{2,1}$ is chosen with probability $\frac{0.8\times0.4}{0.8\times0.4+0.2\times0.1+0.8\times1}$.

Step `06`: actions in $a_2(1)$ are performed.

Table 4.1: Example case.

straint η_i to be in $[0,1]$, so that any new metric could be added provided it can be quantified online and ranges from 0 to 1.

Incident degrees are quantified in discrete incident levels so that different sets of actions can be used to address different levels of the incident. Thresholding consists in clustering platform configurations into groups. We determine τ_i, the threshold value of an incident degree x_i, from execution traces, for which different thresholding approached can be used. For instance, we could consider that $x\%$ of the platform configurations are inappropriate while the rest are acceptable. The choice of x, however, would be arbitrary. Instead, we inspect the modes of the distribution of η_i to determine a threshold. Thresholds τ_i are determined from visual mode clustering. The number m_i of incident levels associated to incident i is set as the number of modes in the observed distribution of η_i. Incidents levels and thresholds are determined offline; thus they do not create any overhead on the workflow execution. Figure 4.4 shows an example where, for $\tau_i > 0.6$ the configuration is inappropriate, and for $\tau_i \leq 0.6$ the configuration is acceptable.

4.3 Illustration on task errors

This section presents a first example of the healing process on workflow activity incidents related to input and output data transfer errors, and application execution errors. First, we define metrics used to determine each incident degree; then, incident levels and association rules are defined from historical information and associated to action sets. Finally, experiments are conducted to evaluate the healing process in production conditions. We consider seven

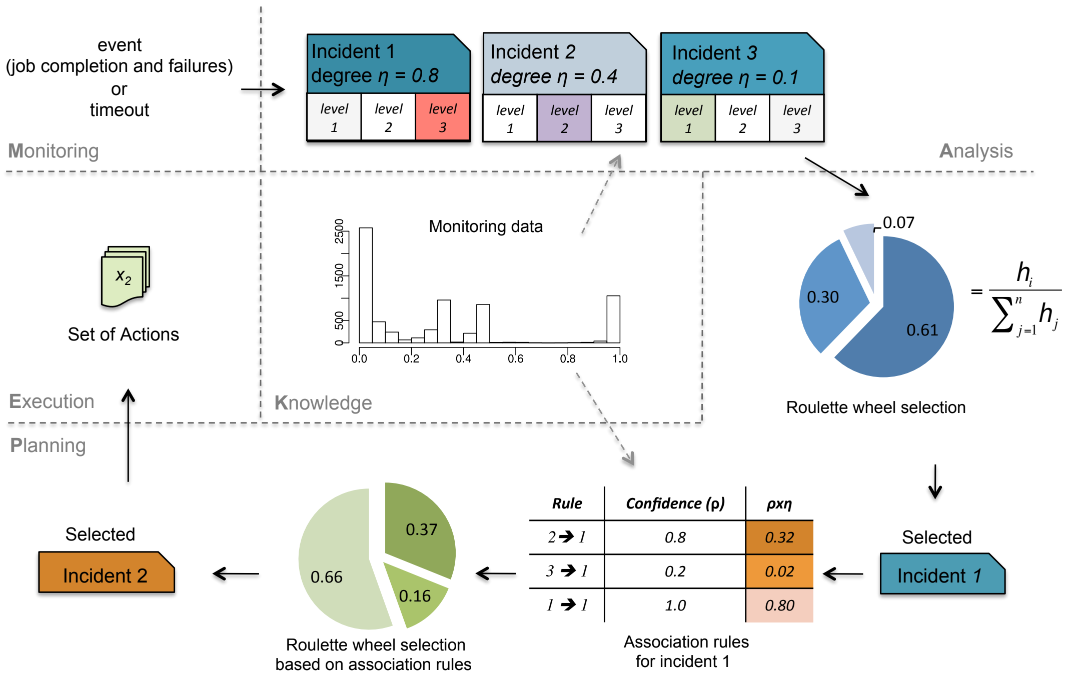


Figure 4.3: Example case showed as a MAPE-K loop.

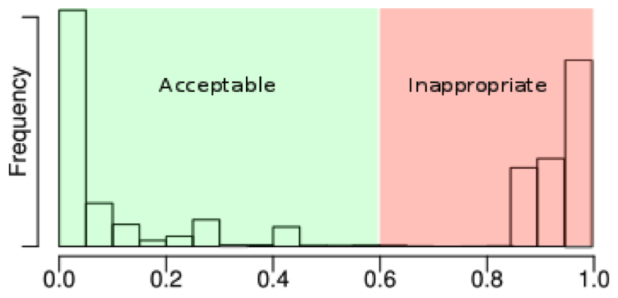


Figure 4.4: Example of incident levels for a threshold $\tau_i = 0.6$.

incidents: input data unavailability and non-existence, output data unavailability, application execution error, and site misconfiguration for input and output data, and application execution.

4.3.1 Incident degree

Input data unavailable. This happens when a file is registered in the file catalog but the storage resource(s) is(are) unavailable or unreachable. The incident degree η_{iu} in this state is determined from the input transfer failure rate due to data unavailability. Transfers of completed, failed, and running invocations are considered.

Input data does not exist. This happens when an incorrect data path was specified, the file was removed by mistake or the file catalog is unavailable or unreachable. Again, the incident degree η_{ie} is directly determined by the input transfer failure rate due to non-existent data. Non-existent file is distinguished from file unavailability using ad-hoc parsing of standard error files. Transfers of completed, failed, and running invocations are considered.

Site misconfigured for input data. This incident happens when computing sites of the infrastructure (see Section 2.2.5 in Chapter 2) have extreme input data transfer failure rate. The incident degree η_{is} is measured as follows:

$$\eta_{\text{is}} = \max(\phi_1, \phi_2, \dots, \phi_k) - \text{median}(\phi_1, \phi_2, \dots, \phi_k)$$

where ϕ_i denotes the input transfer failure ratio (including both input data unavailable and input data does not exist) on site i, and k is the number of white-listed sites used by the activity. The difference between the maximum rate and the median ensures that the incident degree has high values only when some sites are misconfigured. This metric is correlated but not redundant with the two previous ones. If some input data file is not available due to site-independent issues with the storage system, then η_{iu} will grow but η_{is} will remain low because all sites fail identically. On the contrary, η_{is} may grow while η_{iu} and η_{ie} remain low.

Output data unavailable. Output data can also be unavailable. Unavailability happens due to three main reasons: the user did not specify the output path correctly, the application did not produce the expected data, or the file catalog or storage resource are unavailable or unreachable. The incident degree η_{ou} is determined by the output transfer failure rate. Transfers of completed, failed and running invocations are considered.

Site misconfigured for output data. The incident degree η_{os} in this incident is determined as follows:

$$\eta_{\text{os}} = \max(\psi_1, \psi_2, \dots, \psi_k) - \text{median}(\psi_1, \psi_2, \dots, \psi_k)$$

where ψ_i denotes the output transfer failure ratio on site i, and k is the number of white-listed sites used by the activity.

Application error. Applications can fail due to a variety of reasons among which: the application executable is corrupted, dependencies are missing, or the executable is not compatible with the execution host (see Section 3.3.3 in Chapter 3). The incident degree η_{a} in this state is measured by the task failure rate due to application errors. Completed, failed, and running tasks are considered.

Site misconfigured for application. The incident degree η_{as} in this state is measured as follows:

$$\eta_{\text{as}} = \max(\alpha_1, \alpha_2, \dots, \alpha_k) - \text{median}(\alpha_1, \alpha_2, \dots, \alpha_k)$$

where α_i denotes the task failure rate due to application errors on site i, and k is the number of white-listed sites used by the activity.

4.3.2 Incident levels

We replayed the events found in the science-gateway workload archive (Chapter 3) to compute incident degree values after each event. Figure 4.5 displays histograms of computed incident degrees. For readability purposes, only $\eta_i \neq 0$ values are represented. Most of the histograms appear multi-modal, which confirms that incident degrees are quantified. Level numbers and threshold values τ are set from visual mode detection in these histograms and reported on Table 4.2 with associated actions.

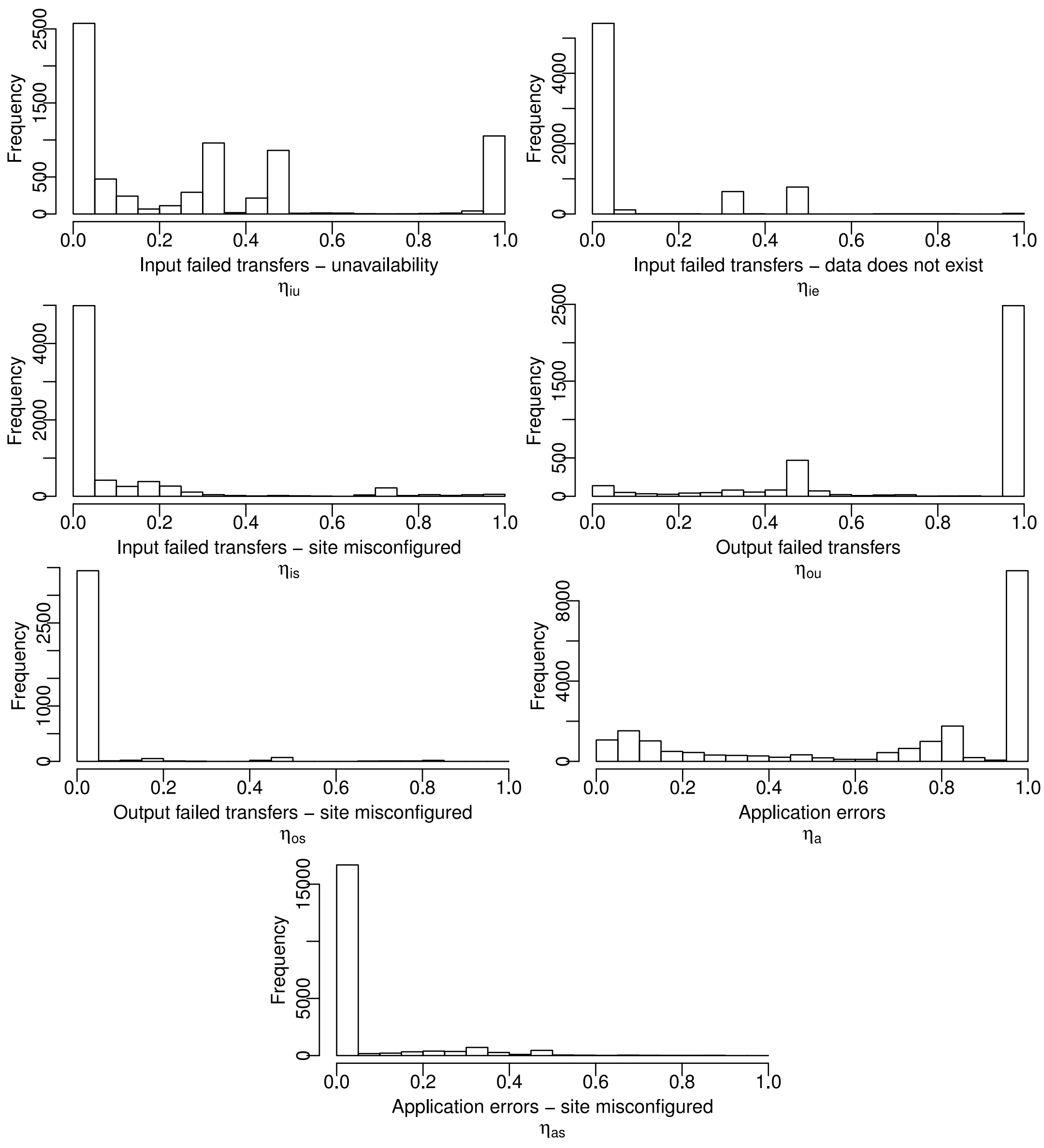


Figure 4.5: Histograms of incident degrees sampled in bins of 5%.

Incidents at level 1 are considered harmless for the execution and they do not trigger any action. Other levels can lead to radical (completely stop the activity or blacklist a site) or

Incident (x_i)	Number of incident	Level 1		Level 2		Level 3	
	levels (m_i)	$\tau_{i,1}$	actions	$\tau_{i,2}$	actions	$\tau_{i,3}$	actions
x_1: input data unavailable	3	0	$\emptyset$	0.2	replicate input files	0.8	stop activity
x_2: input data does not exist	2	0	$\emptyset$	0.8	stop activity		
x_3: site misconfigured for input data	3	0	$\emptyset$	0.3	replicate files on sites reachable from problematic site	0.65	blacklist site
x_4: output data unavailable	2	0	$\emptyset$	0.8	stop activity		
x_5: site misconfigured for output data	2	0	$\emptyset$	0.1	blacklist site		
x_6: application error	2	0	$\emptyset$	0.5	stop activity		
x_7: site misconfigured for application	2	0	$\emptyset$	0.1	blacklist site		

Table 4.2: Incident levels and actions.

intermediate actions (file replication).

4.3.3 Association Rules

Association rules are computed based on the frequency of occurrences of two incident levels in the training dataset. The confidence $\rho_{i,j}^{u,v}$ of a rule $x_{u,v} \Rightarrow x_{i,j}$ measures the probability that an incident level $x_{i,j}$ happens when $x_{u,v}$ occurs. Table 4.3 shows rule samples extracted from the workload archive and ordered by decreasing confidence value. The set of rules leading to input data unavailable ($x_{1,2}$, rules 1 and 2) and site misconfigured for input data ($x_{3,3}$, rule 3) incidents shows that they are partially dependent on other "cause" incidents, which is considered by the self-healing process.

At the bottom of the table we find rules with null confidence. These are consistent with common-sense interpretation of the incident dependencies. For instance, no site-specific issue occurs when input data is unavailable ($x_{3,3} \Rightarrow x_{1,3}$, rule 217), or does not exist ($x_{3,3} \Rightarrow x_{2,2}$, rule 218), or vice-versa (rules 212 and 214).

4.3.4 Actions

Three actions are performed by the self-healing process: file replication, site blacklisting and activity stop. The first two are described below.

File Replication. File replication is implemented differently depending on the incident. In case of input data unavailability, a file is replicated to a storage resource selected randomly. In case a site is misconfigured, replication to the site local storage resource is first attempted. This aims at circumventing inter-domain connectivity issues. If there is no local storage available or the replication process fails, then a second attempt is performed to a storage resource successfully accessed by other tasks executed on the same site. Otherwise, a storage resource is randomly selected. Algorithm 2 shows this process.

#	Association rule	$\rho_{i,j}^{u,v}$
1	$x_{1,2} \Rightarrow x_{7,2}$	0.1700
2	$x_{1,2} \Rightarrow x_{6,2}$	0.1538
3	$x_{3,3} \Rightarrow x_{1,2}$	0.1538
4	$x_{5,2} \Rightarrow x_{1,3}$	0.1250
5	$x_{1,3} \Rightarrow x_{7,2}$	0.1228
6	$x_{5,2} \Rightarrow x_{1,2}$	0.0625
...	...	...
127	$x_{1,2} \Rightarrow x_{3,3}$	0.0454
128	$x_{1,2} \Rightarrow x_{3,2}$	0.0116
129	$x_{1,2} \Rightarrow x_{2,2}$	0.0037
...	...	...
211	$x_{1,3} \Rightarrow x_{3,2}$	0.0000
212	$x_{1,3} \Rightarrow x_{3,3}$	0.0000
213	$x_{2,2} \Rightarrow x_{3,2}$	0.0000
214	$x_{2,2} \Rightarrow x_{3,3}$	0.0000
215	$x_{3,2} \Rightarrow x_{1,3}$	0.0000
216	$x_{3,2} \Rightarrow x_{2,2}$	0.0000
217	$x_{3,3} \Rightarrow x_{1,3}$	0.0000
218	$x_{3,3} \Rightarrow x_{2,2}$	0.0000

Table 4.3: Confidence of rules between incident levels.

Site blacklisting. Problematic sites are only temporarily blacklisted during a time interval set from exponential back-off. The site is first blacklisted for 1 minute only and then put back on the white list. In case it is detected misconfigured again, then the blacklist duration is increased to 2 minutes, then to 4 minutes, 16 minutes, etc.

4.3.5 Experiments

The healing process is implemented in the Virtual Imaging Platform (see description in Chapter 2) and deployed in production. The experiments presented hereafter, conducted for two real workflow activities, evaluate the ability of the healing process to (i) properly detect and handle recoverable errors (*Experiment 1*) and (ii) quickly identify and report critical issues (*Experiment 2*).

Experiment conditions. Experiment 1 aims at testing that recoverable errors are properly detected and handled. This experiment uses a correct execution where all the input files exist and the application is supposed to run properly and produce the expected results. Five repetitions

Algorithm 2 Site misconfigured: replication process for one file.

```
1: input: File $f$, set of storage resources $S$, set of completed tasks on the same site $T$
2: replicate $f$ to local storage resource $j$
3: if replication not successful then
4:     select storage $s_i \in S$ where $t \in T$ could access $s_i$, $i \neq j$
5:     replicate $f$ to $s_i$
6:     if replication not successful then
7:         select randomly $s_r \in S$, $r \neq i$
8:         replicate $f$ to $s_r$
9:     end if
10: end if
```

are performed for each workflow activity.

Experiment 2 aims at testing that unrecoverable errors are quickly identified and the execution is stopped. Unrecoverable errors are intentionally injected in 3 different runs: in run `non-existent inputs`, non-existent file paths are used for all the invocations; in `application-error`, all the file paths exist but input files are corrupted; and in `non-existent output`, input files are correct but the application does not produce the expected results.

Two workflow activities are considered for each experiment: `FIELD-II/pasa` and `Mean-Shift/hs3` (see Appendix A for description). Files are replicated on two storage sites for both activities. For each experiment, a workflow execution using our method (`Self-Healing`) is compared to a control execution (`No-Healing`). Executions are launched on the biomed VO of the EGI, in production conditions. `Self-Healing` and `No-Healing` are both launched simultaneously to ensure similar grid conditions. The DIRAC scheduler is configured to equally distribute resources among executions.

The FuSM and healing process are implemented in the MOTEUR workflow engine. The timeout value in the healing process is computed dynamically as the median of the task inter-completion delays in the current execution. Task replication is performed by resubmitting running tasks to DIRAC. MOTEUR is configured to resubmit failed tasks up to 5 times in all runs of both experiments. We use DIRAC v5r12p9 and MOTEUR 0.9.19.

Results and discussion. Experiment 1: Table 4.4 shows occurrences of incident levels and associated actions for the 5 repetitions. All recoverable incidents were observed, except site misconfigured for output data ($x_{5,2}$). The healing process successfully detects and handles all recoverable errors.

Experiment 2: Figure 4.6 shows the makespan of `FIELD-II/pasa` and `Mean-Shift/hs3` for the 3 runs where unrecoverable errors are introduced. `No-Healing` was manually stopped after 7 hours to avoid flooding the infrastructure with faulty tasks. In all cases, `Self-Healing` is able to detect the issue and stop the execution far before `No-Healing`. It confirms that the

Activity	Incident level	Occurrence	Actions
FIELD-II/pasa	$x_{1,2}$	32	replicate input files
	$x_{7,2}$	12	blacklist site
Mean-Shift/hs3	$x_{1,2}$	23	replicate input files
	$x_{3,2}$	16	replicate files on sites
	$x_{3,3}$	6	blacklist site
	$x_{7,2}$	8	blacklist site

Table 4.4: Experiment 1: occurrences of incident levels (cumulative values for 5 repetitions).

healing process is able to identify unrecoverable errors and stop the execution accordingly. As shown on Table 4.5, the number of submitted fault tasks is significantly reduced, which has benefits both to the infrastructure and to the gateway.

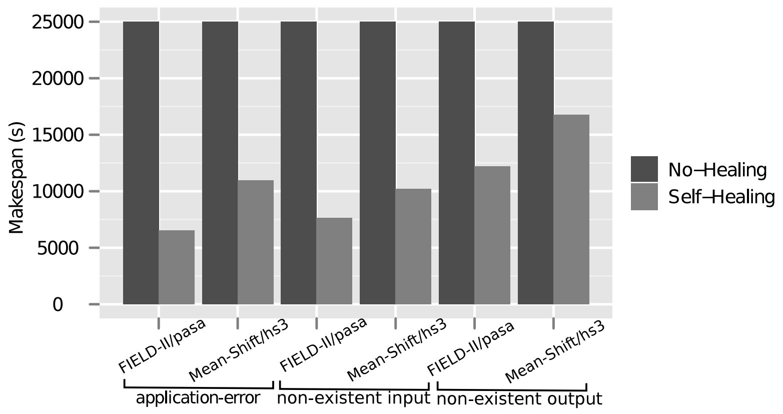


Figure 4.6: Experiment 2: makespan of `FIELD-II/pasa` and `Mean-Shift/hs3` for 3 different runs.

Although the healing process detects and stops the execution of unrecoverable workflow executions earlier than control executions, it still takes some time to perform the action. As we are in online and non-clairvoyant conditions, we should wait for the completion of a few tasks to be able to draw an execution model. Then, actions can be taken to cope with errors.

4.4 Improvements to mode detection

In Section 4.3.2 we determined incident levels numbers and thresholds from visual mode detection in the histograms showed in Figure 4.5. We assume that modes in the observed distribution of η_i are well separated. Otherwise, the metric used to determine η_i does not properly quantifies an incident. Two possible ways to improve mode detection are (*i*) automated detection

Run		Number of tasks	
		Self-Healing	No-Healing
application-error	FIELD-II/pasa	196	732
	Mean-Shift/hs3	249	1500
non-existent input	FIELD-II/pasa	293	732
	Mean-Shift/hs3	417	1500
non-existent output	FIELD-II/pasa	287	732
	Mean-Shift/hs3	364	1500

Table 4.5: Number of submitted faulty tasks.

(e.g. with K-Means [MacQueen, 1967] or Mean-Shift [Comaniciu and Meer, 2002]), and (*ii*) periodical update from execution history.

Figure 4.7 shows histograms of computed incidents degrees clustered with K-Means. Incident levels thresholds are defined from the highest value of a group (except for the last group), and are showed in Table 4.6. For each incident, k clusters are set according to the number of incident levels m_i from Table 4.2. Most threshold values are similar to the ones determined by visual mode detection, except for incidents x_2 (input data does not exist) and x_3 (site misconfigured for output data). This divergence of values is related to the scarce appearance of these incidents on the platform, i.e., the amount of incident degree values is insufficient to determine a mode.

Incident (x_i)	Number of incident levels (m_i)	Level 1 $\tau_{i,1}$	Level 2 $\tau_{i,2}$	Level 3 $\tau_{i,3}$
x_1: input data unavailable	3	0	0.2	0.75
x_2: input data does not exist	2	0	0.3	
x_3: site misconfigured for input data	3	0	0.1	0.5
x_4: output data unavailable	2	0	0.7	
x_5: site misconfigured for output data	2	0	0.25	
x_6: application error	2	0	0.55	
x_7: site misconfigured for application	2	0	0.15	

Table 4.6: Incident levels determined with K-Means.

From the histogram of the input data does not exist incident degree (η_{ie}), three clusters could be identified by visual detection: $\tau_{2,1} = 0$, $\tau_{2,2} = 0.3$, and $\tau_{2,3} = 0.8$. However, the common-sense interpretation of the incident lead us to two configurations: recoverable and unrecoverable. For incident degrees values lower than 0.8 the incident is considered harmless and no action is triggered. Otherwise, the incident is considered unrecoverable and the execution should be stopped.

K-Means clustering could be envisaged as an alternative to automate mode detection when an incident happens exhaustively on the platform, so that modes can be well identified. Other-

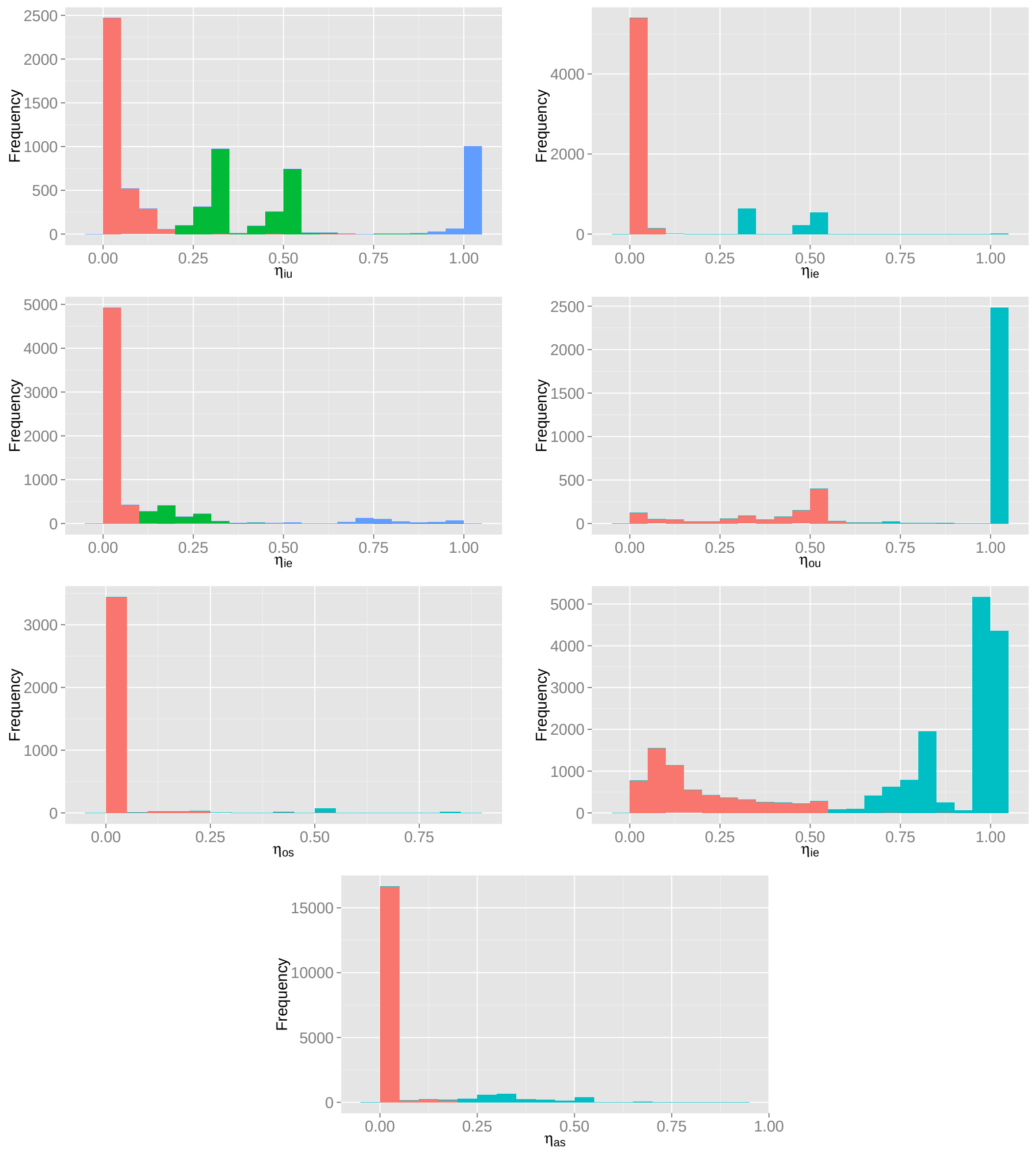


Figure 4.7: Histograms of incident degrees clustered with K-Means sampled in bins of 5%.

wise, visual mode detection still is the most suitable.

Similarities between incident distributions could also be studied to determine relations of dependency between threshold values and applications.

4.5 Conclusion

We presented a simple, yet practical method for autonomous detection and handling of operational incidents on workflow executions. No strong assumption is made on the task duration or resource characteristics and incident degrees are measured with metrics that can be computed online. We made the hypothesis that incident degrees were quantified into distinct levels, which we verified using real traces collected from the Virtual Imaging Platform. Incident levels are associated offline to action sets. Action sets are selected based on the degree of their associated incident level and on confidence of association rules determined from execution history.

We showed an application of the healing process on seven simple incidents related to input and output data transfer errors, and application execution errors. The strategy was implemented in the MOTEUR workflow engine and deployed on the European Grid Infrastructure with the DIRAC resource manager. Results show that our healing mechanism properly detects and handles recoverable and unrecoverable errors, which significantly reduces the execution makespan. The time spent to acquire information about the workflow execution impacts on the makespan of unrecoverable executions, delaying their cancellation.

Mode detection could be automated, for instance, by using K-Means clustering. However, this technique can only be applied when modes are well defined. Otherwise, visual mode detection is still recommended.

In the next chapters of this part, we use our self-healing mechanism to address two activity-level incidents: the long tail effect issue (Chapter 5), and the task granularity problem (Chapter 6); and an incident at platform level: unfairness among workflow executions (Chapter 7).

Chapter 5

Handling blocked activities

Contents

The long-tail effect is a common frustration for users who have to wait to retrieve the last pieces of their computation. This issue happens due to execution on slow machines, poor network connection, or communication issues, and leads to substantial speed-up reductions. In this chapter, we propose a new algorithm to handle the long-tail effect and to control task replication. The healing process is parametrized on real application traces acquired in production on the European Grid Infrastructure. Experimental results obtained in the Virtual Imaging Platform show that the proposed method speeds up execution up to a factor of 4.5, and consumes up to 35% less resource time than a control execution.

Les retards de tâches sont une frustration courante pour les utilisateurs qui doivent attendre de récupérer les derniers morceaux de leur calcul. Ce problème se produit en raison de machines lentes, d'une mauvaise connexion au réseau, ou de problèmes de communication, et conduit à d'importantes ralentissements. Dans ce chapitre, nous proposons un nouvel algorithme pour gérer les retards de tâches et pour contrôler la réplication des tâches. Le processus de auto-administration est paramétré sur les traces d'applications réelles acquises sur EGI. Les résultats expérimentaux obtenus sur la plateforme VIP montrent que la méthode proposée réduit le temps de calcul d'un facteur pouvant atteindre 4,5, et consomme jusqu'à 35% de moins de ressources que l'exécution témoin.

5.1 Introduction

The long-tail effect [Cirne et al., 2007] is a common frustration for users who have to wait to retrieve the last few pieces of their computations (a.k.a. late tasks). Tasks may be delayed, for instance, due to execution on a slow machine, low network throughput or just loss of contact. Science-gateways operators may be able to address this problem by rescheduling these tasks, but detection is very time consuming and still approximate. Instead, actions should be triggered automatically when late tasks are detected in an activity.

In this chapter, we present a self-healing process that quantifies the incident degree of blocked workflow activities from metrics measuring the long-tail effect. We propose two methods to determine the incident degree. The first, named `Slope Contraction`, identifies blocked activities by detecting decreases of the derivative along time of the number of completed tasks, and the second, named `Median Estimation`, identifies blocked activities as the ones whose tasks are performing worse than the median of already completed tasks. In both cases, tasks are assumed of identical costs. This assumption considers that the variation of task durations of correct executions due to resource heterogeneity is negligible compared to the variation when an incident happens.

Algorithm 3 describes our activity blocked control process. The methods are implemented in VIP, and evaluated with different applications, in production conditions, on the European Grid Infrastructure.

The first method to detect blocked activities (`Slope Contraction`) is presented in Section 5.2. Section 5.3 presents the second method to cope with blocked activities and the task replication control process (`Median Estimation`). Section 5.4 concludes the chapter.

Algorithm 3 Main loop for activity blocked control

```
1: input: $m$ workflow executions
2: while there is an active workflow do
3:     wait for timeout or task status change in any workflow
4:     determine blocked degree $\eta_b$
5:     if $\eta_b > \tau_b$ then
6:         replicate late tasks
7:     end if
8: end while
```

5.2 Slope Contraction

5.2.1 Incident degree and levels

Activity blocked degree η_b. This incident is detected online from the number $n(t)$ of completed tasks at time t (see Figure 5.1). At time t, we compute the slope $a(t)$ of the regression line of $\{(t_i, n(t_i)), t_i \leq t\}$. If the iteration is triggered by a timeout instead of an event (see `step 01` on Algorithm 1 in Chapter 4), then $(t, n(t) + 1)$ is added to the regression set. This is meant to ensure that long-running tasks can be handled before they complete. We then define the incident degree η_b from the contraction rate of the linear regression slope:

$$\eta_b = 1 - \frac{a(t)}{a_{\max}(t)}$$

where $a_{\max}(t)$ is the maximal value of $a(t)$ in $[0, t]$. $t = 0$ is the time when the activity is started, i.e., all the tasks are initialized. Note that the maximum degree $\eta_b = 1$ is reached when the activity is completely blocked ($\lim_{t\to\infty} a(t) = 0$). On the other hand, $\eta_b = 0$ is reached when $a(t) = a_{\max}(t)$.

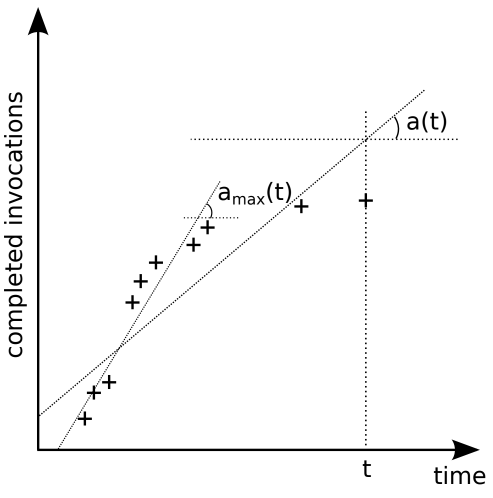


Figure 5.1: Detection of blocked activity.

Threshold value τ_b. The threshold value for η_b separates configurations where the activity has acceptable performance ($\eta_b \leq \tau_b$) from configurations where the activity is blocked ($\eta_b > \tau_b$). We determine τ_b from observed distributions of η_b. The blocked degree η_b was computed after each event found in the workload archive (Chapter 3), and as shown in Figure 5.2. The histogram appears bimodal, which indicates that η_b separates platform configurations in two distinct groups; we choose $\tau_b = 0.6$. We assume that values in the lowest mode correspond to acceptable performance, and values in the highest mode correspond to low performance. Thus, for $\eta_b > 0.6$, task replication will be triggered. Tasks are replicated if there are no queued replicas.

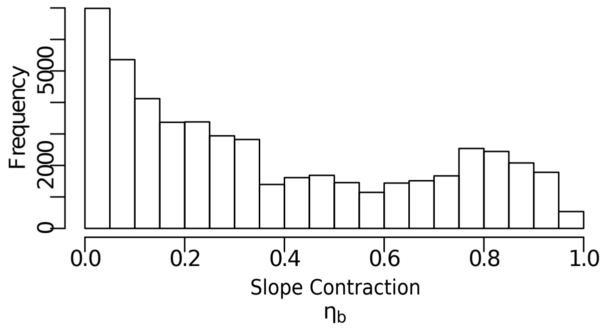


Figure 5.2: Histogram of activity blocked degree sampled in bins of 0.05.

5.2.2 Experiments and results

The experiment presented hereafter evaluate the ability of the `Slope Contraction` method to improve workflow makespan when tasks are late.

Experiment conditions. The healing control process was implemented as a plugin of the MOTEUR workflow engine, receiving notifications about task status changes and task phase durations. Task replication is performed by resubmitting running tasks to DIRAC. To avoid concurrency issues in the writing of output files, a simple mechanism based on file renaming is implemented. To limit infrastructure overload, running tasks are replicated up to 5 times. MOTEUR is configured to resubmit failed tasks up to 5 times in all runs. We use DIRAC v5r12p9 and MOTEUR 0.9.19.

The experiment aims at testing that blocked activities are properly detected and handled; the other incidents are ignored. This experiment uses a correct execution where the application is supposed to run properly and produce the expected results. Five repetitions are performed for each workflow activity. Two workflow activities are considered for the experiment: `FIELD-II/pasa` and `Mean-Shift/hs3` (see Appendix A for description). Table 5.1 shows their main characteristics.

Workflow activity		#Tasks	CPU time	Input	Output
FIELD-II	(data-intensive)	122	few seconds to 15 minutes	~208 MB	~40 KB
Mean-Shift	(CPU-intensive)	250	few minutes to 1 hour	~182 MB	~1 KB

Table 5.1: Workflow activity characteristics.

A workflow execution using our method (`Self-Healing`) is compared to a control execution (`No-Healing`). Executions are launched on the biomed VO of the EGI, in production conditions. `Self-Healing` and `No-Healing` are both launched simultaneously to ensure similar grid conditions. The DIRAC scheduler is configured to equally distribute resources among executions.

Task replication may waste resources, i.e., resources are consumed by a set of tasks that compute the same operations. Cirne et al. [Cirne et al., 2007] define waste as the ratio between the number of cycles consumed by replicas unused by the application and the total of cycles necessary to execute the application. This metric does not fit our context because it cannot provide an effective estimation of the amount of resource wasted by self-healing executions when compared to the control ones. Here, resource waste is assessed by the amount of resource time consumed by `Self-Healing` executions related to the amount of resource time consumed by control executions. We use the `waste coefficient` (w), defined as follows:

$$w = \frac{\sum_{i=1}^{n} h_i + \sum_{j=1}^{m} r_j}{\sum_{i=1}^{n} c_i} - 1$$

where h_i and c_i are the resource time consumed (CPU time + data transfers time) by n completed tasks for `Self-Healing` and `No-Healing` executions respectively, and r_i is the resource time consumed by m unused replicas. Note that task replication usually leads to $h_i \leq c_i$. If $w > 0$, `Self-Healing` wastes resources compared to the control. If $w < 0$, `Self-Healing` consumes less resources than the control, which can happen when faster resources are selected.

Results and discussion. Figure 5.3 presents the makespan of `FIELD-II/pasa` and `Mean-Shift/hs3` for the 5 repetitions. The makespan was considerably reduced in all repetitions of both activities. Speed-up values yielded by `Self-Healing` ranged from 2.6 to 4 for `FIELD-II/pasa` and from 1.3 to 2.6 for `Mean-Shift/hs3`.

Figure 5.4 present a cumulative density function (CDF) of the number of completed tasks for `FIELD-II/pasa` (top) and `Mean-Shift/hs3` (bottom). In most cases completion curves of both `Self-Healing` and `No-Healing` executions are similar up to 95%. This confirms that both executions had similar grid conditions. In some cases (e.g. `Repetition 2` for `FIELD-II` in Figure 5.4) the `Self-Healing` has lower performance than `No-Healing` but it is compensated by the long-tail effect produced by the latter.

Tables 5.2 show the waste coefficient value for `FIELD-II/pasa` (top) and `Mean-Shift/hs3` (bottom). The `Self-Healing` process consumed up to 33% more re-

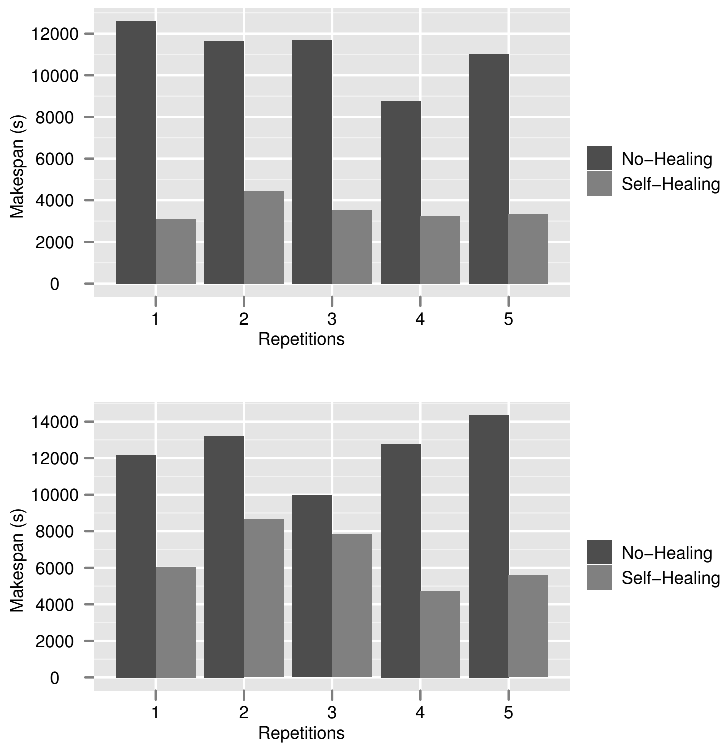


Figure 5.3: Execution makespan for `FIELD-II/pasa` (top) and `Mean-Shift/hs3` (bottom) by using the `Slope Contraction` method.

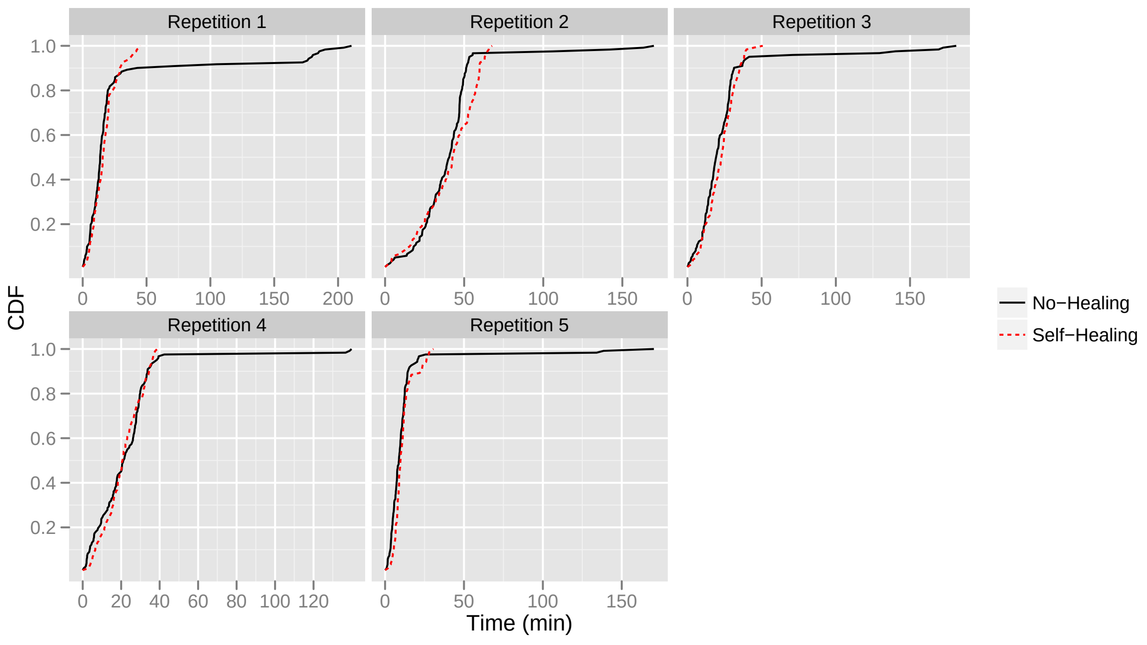
sources for `FIELD-II/pasa` and 75% for `Mean-Shift/hs3` compared to a control execution. Even if this mechanism properly detects blocked states, it wastes the resources of badly performing tasks that will be overlapped by replicas. This waste of resource is related to (*i*) a late detection of the blocked state and (*ii*) a greedy replication process. For instance, the poor performance of a task submitted at the beginning of the execution can be masked by the good performance of the others. In the next section, we present a novel metric to determine blocked activities based on the median estimation of task execution times, which considers both early detection and waste.

5.3 Median Estimation

5.3.1 Incident degree and levels

Activity blocked degree η_b. We define the incident degree η_b of an activity from the max of the performance coefficients p_i of its n tasks, which relate the task phase durations (`setup`, `inputs download`, `application execution` and `outputs upload`) to their medians:

$$\eta_\text{b} = 2.\max\left\{p_i = p(t_i, \tilde{t}) = \frac{t_i}{\tilde{t} + t_i}, i \in [1, n]\right\} - 1 \tag{5.1}$$



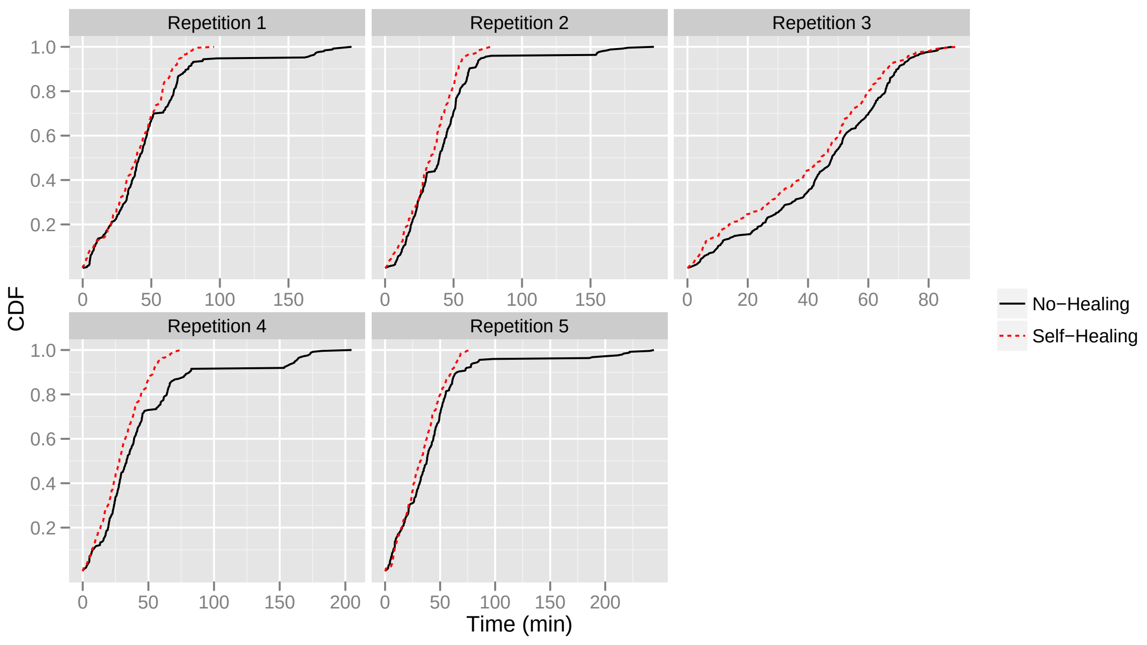


Figure 5.4: CDF of the number of completed tasks for `FIELD-II/pasa` (top) and `Mean-Shift/hs3` (bottom) repetitions by using the `Slope Contraction` method.

Repetition	h	r	c	w
1	39,035s	28,653s	55,244s	0.22
2	37,035s	5,191s	50,829s	−0.17
3	28,454s	9,594s	28,594s	0.33
4	21,021s	13,764s	27,586s	0.26
5	37,494s	13,438s	42,019s	0.21

Repetition	h	r	c	w
1	84,499s	74,917s	95,319s	0.67
2	121,496s	88,963s	129,250s	0.63
3	81,745s	16,418s	88,032s	0.11
4	98,235s	146,016s	141,292s	0.73
5	103,867s	81,614s	105,783s	0.75

Table 5.2: Waste coefficient values for `FIELD-II/pasa` (top) and `Mean-Shift/hs3` (bottom) by using the `Slope Contraction` method.

where $t_i = t_{i\_setup} + t_{i\_input} + t_{i\_exec} + t_{i\_output}$ is the estimated duration of task i and $\tilde{t} = \tilde{t}_{setup} + \tilde{t}_{input} + \tilde{t}_{exec} + \tilde{t}_{output}$ is the sum of the median durations of tasks 1 to n. Note that $\max\{p_i, i \in [1, n]\} \in [0.5, 1]$ so that $\eta_b \in [0, 1]$. Moreover, $\lim_{t_i \to +\infty} p_i = 1$ and $\max\{p_i, i \in [1, n]\} = 0.5$ when all the tasks behave like the median. When less than 2 tasks are completed, medians remain undefined and the control process is inactive.

The estimated duration t_i of a task is computed phase by phase, as follows: (*i*) for completed task phases, the actual consumed resource time is used; (*ii*) for ongoing task phases, the maximum value between the current consumed resource time and the median consumed time is taken; and (*iii*) for unstarted task phases, the time slot is filled by the median value. Figure 5.5 illustrates the estimation process of a task where the actual durations are used for the two first completed phases (42s for `setup` and 300s for `inputs download`), the `application execution` phase uses the maximum value between the current value of 20s and the median value of 400s, and the last phase (`outputs upload`) is filled by the median value of 15s, as it is not started yet.

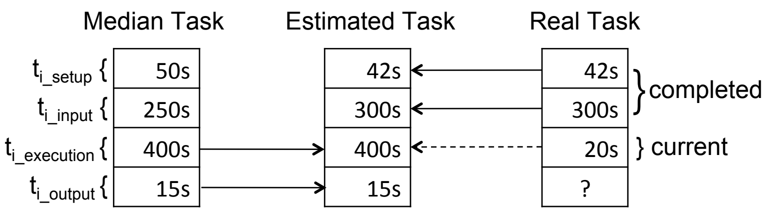


Figure 5.5: Task estimation based on median values.

Threshold value τ_b. Figure 5.6 shows the distribution of η_b. Since the modes are not clearly separable visually, we used K-Means to determine the threshold value $\tau_b = 0.35$. We assume that values in the lowest mode correspond to acceptable performance, and values in the highest mode correspond to low performance. Thus, for $\eta_b > 0.35$ task replication will be triggered.

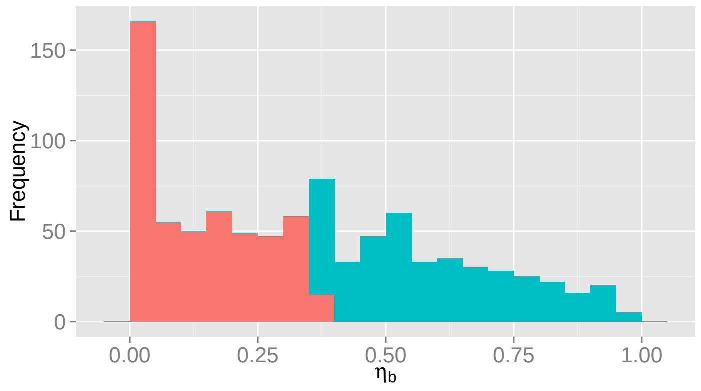


Figure 5.6: Histogram of activity blocked degree sampled in bins of 0.05.

Task replication. Blocked activities are addressed by task replication. To limit resource waste, the replication process for a particular task is controlled by two mechanisms. First, a task is not replicated if a replica is already queued. Second, if replica j has better performance than replica r (i.e. $p(t_r, t_j) > \tau_b$, see equation 5.1) and replica j is in a more advanced phase than replica r, then replica r is aborted. Algorithm 4 presents the algorithm of the replication process. It is applied to all tasks with $p_i > \tau_b$, as defined on equation 5.1.

Algorithm 4 Replication process for one task.

```
 1: input: Set of replicas $R$ of a task $i$
 2: rep = true
 3: for $r \in R$ do
 4:     for $j \in R, j \neq r$ do
 5:         if $p(t_r, t_j) > \tau$ and $j$ is a step further than $r$ then
 6:             abort $r$
 7:         end if
 8:     end for
 9:     if $r$ is started and $p(t_r, \tilde{t}) \leq \tau$ then
10:         rep = false
11:     else if $r$ is queued then
12:         rep = false
13:     end if
14: end for
15: if rep == true then
16:     replicate $r$
17: end if
```

5.3.2 Experiments and results

The experiment presented hereafter evaluate the ability of the activity blocked control process to improve workflow makespan without wasting resources in case of tasks are late.

Experiment conditions. The experiment is performed in the exact same conditions as the experiment described in Section 5.2.2.

Results and discussion. Figure 5.7 shows the makespan of `FIELD-II/pasa` (top) and `Mean-Shift/hs3` (bottom) for the 5 repetitions. The makespan values are comparable to the results presented in Figure 5.3. The makespan was considerably reduced in all repetitions of both activities. Speed-up values yielded by `Self-Healing` ranged from 1.7 to 4.5 for `FIELD-II/pasa` and from 1.5 to 3.2 for `Mean-Shift/hs3`.

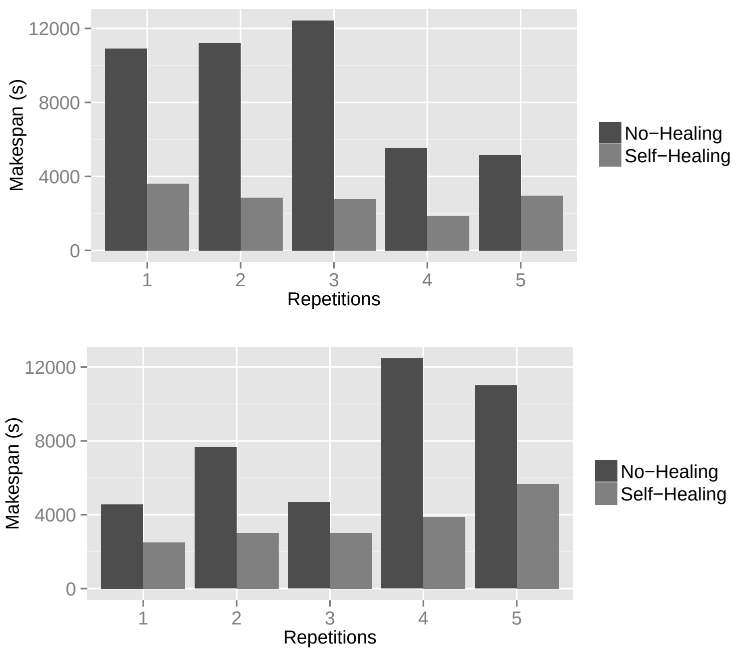


Figure 5.7: Execution makespan for `FIELD-II/pasa` (top) and `Mean-Shift/hs3` (bottom) by using the `Median Estimation` method.

Analogously to Figure 5.4, Figure 5.8 present the CDF of the number of completed tasks for both applications. Again, curve similarities up to 95% indicate similar grid conditions.

Table 5.3 shows the waste coefficient values for the 5 repetitions for `FIELD-II/pasa` (top) and `Mean-Shift/hs3` (bottom). The `Self-Healing` process reduces resource consumption up to 35% when compared to the control execution. This happens because replication increases the probability to select a faster resource. The total number of replicated tasks for all repetitions is 292 for `FIELD-II/pasa` (i.e. 0.48 task replication per task in average) and 712 for `Mean-Shift/hs3` (i.e. 0.57 task replication per task in average).

Repetition	h	r	c	w
1	41, 338s	23, 823s	71, 853s	−0.09
2	37, 190s	28, 251s	66, 435s	−0.01
3	40, 209s	25, 068s	68, 792s	−0.05
4	39.009s	32, 973s	78, 723s	−0.08
5	38, 847s	37, 393s	78, 988s	−0.03

Repetition	h	r	c	w
1	97, 875s	17, 799s	116, 853s	−0.01
2	85, 100s	19, 086s	161, 801s	−0.35
3	98, 736s	25, 162s	125, 615s	−0.01
4	107, 071s	62, 746s	204, 456s	−0.17
5	126, 344s	2, 195s	131, 446s	−0.02

Table 5.3: Waste coefficient values for `FIELD-II/pasa` (top) and `Mean-Shift/hs3` (bottom) by using the `Median Estimation` method.

Experiment results show that `Slope Contraction` and `Median Estimation` methods properly detect blocked activities and speed up the execution. Although task replication is controlled (see Algorithm 4), `Median Estimation` performs similar to `Slope Contraction`, which indicates that blocked tasks are detected earlier.

5.4 Conclusion

In this chapter, we presented two methods to cope with the activity blocked incident: `Slope Contraction` and `Median Estimation`. The first one identifies blocked activities by detecting decreases of the derivative along time of the number of completed tasks, and the second identifies blocked activities as the ones whose tasks are performing worse than the median of already completed tasks. No strong assumption is made on the task duration or resource characteristics. Experimental results show that both methods properly detect blocked activities and speed up workflow executions up to a factor of 4.5.

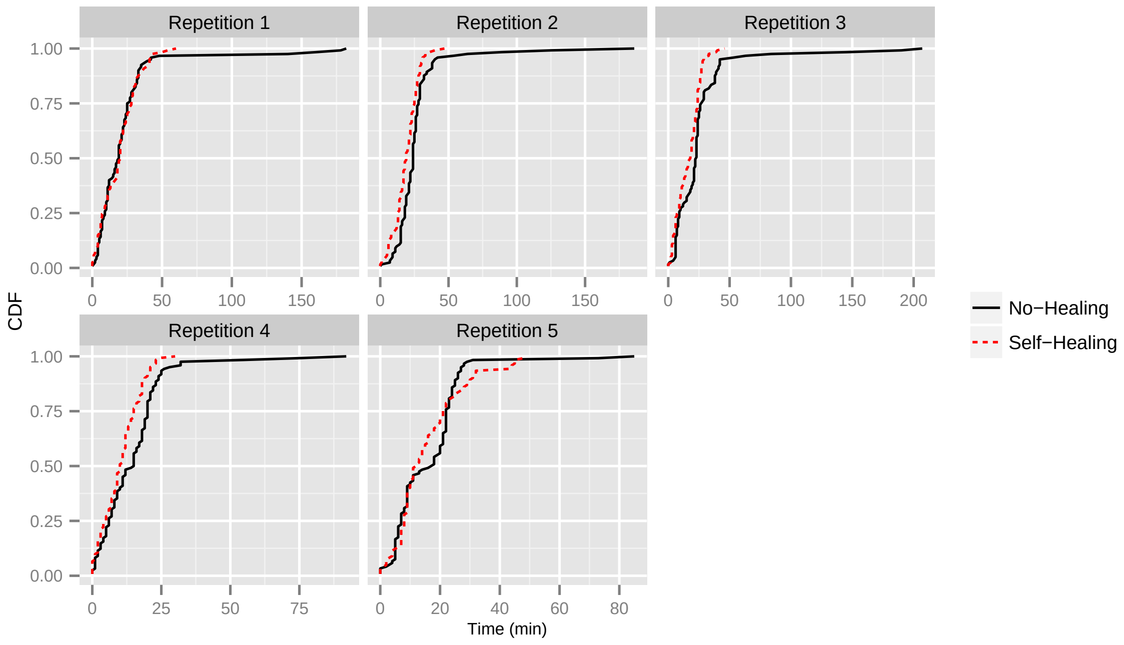
We also defined a waste metric to compute the amount of resource time consumed by self-healing executions compared to control executions. Results show that `Median Estimation` consumes up to 35% less resources than a control execution, while `Slope Contraction` consumes up to 75% more resources.

We believe that results of this chapter are the first ones presented to control blocked activities in such conditions which are often met in production platforms.

The `Median Estimation` method is currently used in production by VIP. It is implemented as a plugin for GASW[1]. From August 2012 to February 2013 , more than 3400 workflow executions benefited from the activity blocked control process.

[1]http://vip.creatis.insa-lyon.fr:9002/projects/gasw-healing-plugin

In the next chapter, we present a healing process to control the granularity of workflow activities. Our method groups tasks when the fineness degree of the application becomes higher than a threshold. In addition, a de-grouping mechanism is triggered when new resources arrive.



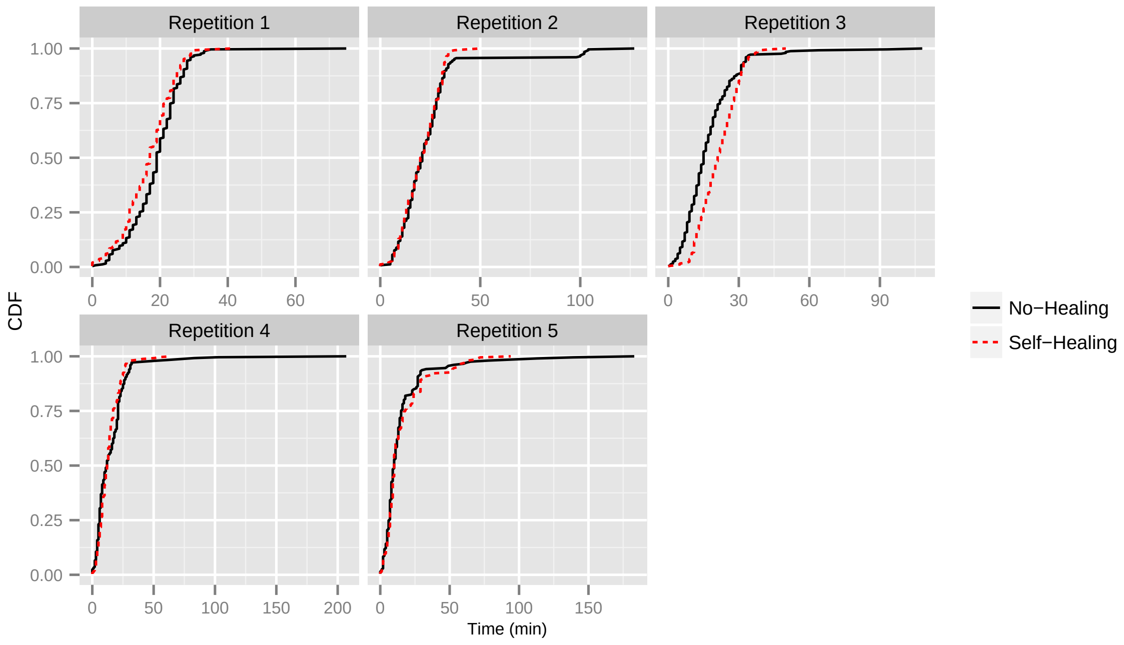


Figure 5.8: CDF of the number of completed tasks for `FIELD-II/pasa` (top) and `Mean-Shift/hs3` (bottom) repetitions by using the `Median Estimation` method.

Chapter 6

Optimizing task granularity

Contents

Controlling the granularity of workflow activities executed on grids is required to reduce the impact of task queuing and data transfer time. Most existing granularity control approaches assume extensive knowledge about the applications and resources (e.g. task duration on each resource), and that both the workload and available resources do not change over time. We propose a granularity control algorithm for platforms where such clairvoyant and offline conditions are not realistic. Our method groups tasks when the fineness degree of the application, which takes into account the ratio of shared data and the queuing/round-trip time ratio, becomes higher than a threshold determined from execution traces. The algorithm also de-groups task groups when new resources arrive. The application's behavior is constantly monitored so that the characteristics useful for the optimization are progressively discovered. Experimental results, obtained with 3 workflow activities deployed on EGI, show that (i) the grouping process yields speed-ups of about 2.5 when the amount of available resources is constant and that (ii) the use of de-grouping yields speed-ups of 2 when resources progressively appear.

Le contrôle de la granularité des activités de workflows exécutés sur grille est nécessaire pour réduire l'impact des temps d'attente et de transferts de données. La plupart des approches existantes de contrôle de granularité supposent une connaissance approfondie sur les applications et les ressources (par exemple, durée de la tâche sur chaque ressource), et que la charge de travail et les ressources disponibles ne changent pas au fil du temps. Nous proposons un algorithme de contrôle de granularité pour les plate-formes où les conditions de clairvoyance et « offline » ne sont pas réalistes. Notre méthode regroupe les tâches lorsque le degré de finesse de l'application, qui prend en compte la proportion de données partagées et le ratio temps d'attente / temps d'attente plus temps d'exécution, devient supérieur à un seuil déterminé à partir des traces d'exécution. L'algorithme dégroupe également des groupes de tâches lorsque de nouvelles ressources arrivent. Le comportement de l'application est suivi en permanence de telle sorte que les caractéristiques utiles pour l'optimisation sont progressivement découvertes. Les résultats expérimentaux, obtenus avec 3 activités de workflow déployés sur EGI, montrent que (i) le processus de regroupement accélère l'exécution d'un facteur environ 2,5 lorsque la quantité de ressources disponibles est constante et que (ii) l'utilisation dégroupage accélère l'exécution d'un facteur 2 lorsque les ressources apparaissent progressivement.

6.1 Introduction

The low performance of *lightweight* (a.k.a. *fine-grained*) tasks is a common problem on widely distributed platforms where the communication overhead and queuing time are high, such as grid systems. To address this issue, fine-grained tasks are commonly grouped into *coarse-grained* tasks [Muthuvelu et al., 2013, Singh et al., 2008, Muthuvelu et al., 2005, Ng et al., 2006, Ang et al., 2009], which reduces the cost of data transfers when grouped tasks share input data [Muthuvelu et al., 2013] and saves queuing time when resources are limited [Singh et al., 2008]. However, task grouping also limits parallelism and therefore should be used sparingly.

We consider such a granularity problem in a science-gateway executing workflows on a grid. We propose an algorithm to optimize the granularity of workflow activities on non-clairvoyant online grid platforms. Our algorithm progressively discovers the characteristics of the running applications to compute a metric quantifying the fineness degree of a task group. This fineness metric includes measured task queuing times, and median-based estimations of task running times and transfer time of shared input data. Tasks are grouped when the fineness metric goes beyond a threshold learned from platform traces. In addition, a de-grouping mechanism is triggered when parallelism losses are detected, i.e. when the number of queued tasks is lower than the number of running tasks. The method is implemented in VIP, and evaluated with different applications, in production conditions, on the European Grid Infrastructure (EGI).

To the best of our knowledge, this algorithm is the first example of task granularity control in a non-clairvoyant online context. The next Section details the granularity control process, Section 6.3 reports experiments and results, and the chapter closes with a discussion and conclusions.

6.2 Task Granularity Control Process

Algorithm 5 describes our task granularity control composed of two processes: (*i*) fineness control groups too fine task groups for which the fineness degree η_f is greater than threshold τ_f, and (*ii*) coarseness control de-groups too coarse task groups for which the coarseness degree η_c is greater than threshold τ_c. This section describes how η_f, η_c, τ_f and τ_c are computed, and details the grouping and de-grouping algorithms.

Algorithm 5 Main loop for granularity control

```
 1: input: $n$ waiting tasks
 2: create $n$ 1-task groups $T_i$
 3: while there is an active task group do
 4:     wait for timeout or task status change
 5:     determine fineness degree $\eta_f$
 6:     if $\eta_f > \tau_f$ then
 7:         group task groups using Algorithm 6
 8:     end if
 9:     determine coarseness degree $\eta_c$
10:     if $\eta_c > \tau_c$ then
11:         degroup coarsest task groups
12:     end if
13: end while
```

6.2.1 Fineness control

Fineness degree η_f. Let n be the number of waiting tasks in a workflow activity, and m the number of task groups. Tasks of an activity are assumed independent, but with similar costs (bag of tasks). Initially, 1 group is created for each task ($n = m$). T_i is the set of tasks in group i, and n_i is the number of tasks in T_i. Groups are a partition of the set of waiting tasks: $T_i \bigcap_{i \neq j} T_j = \emptyset$ and $\sum_{i=1}^{m} n_i = n$. The activity fineness degree η_f is the maximum of all group fineness degrees f_i:

$$\eta_f = \max_{i \in [1,m]} (f_i). \tag{6.1}$$

All η_f are in [0,1], and high fineness degrees indicate fine granularities. We use a max operator in this equation to ensure that *any* task group with a too fine granularity will be detected. The fineness degree f_i of group i is defined as:

$$f_i = d_i \cdot r_i, \tag{6.2}$$

where d_i is the ratio between the transfer time of the input data shared among all tasks in the activity, and the total execution time of the group:

$$d_i = \frac{\tilde{t}_{\_shared}}{\tilde{t}_{\_shared} + n_i(\tilde{t} - \tilde{t}_{\_shared})},$$

where $\tilde{t}_{\_shared}$ is the median transfer time of the input data shared among all tasks in the activity, and $\tilde{t}$ is the sum of its median task phase durations corresponding to application setup, input data transfer, application execution and output data transfer: $\tilde{t} = \tilde{t}_{\_setup} + \tilde{t}_{\_input} + \tilde{t}_{\_exec} + \tilde{t}_{\_output}$. Median values $\tilde{t}_{\_shared}$ and $\tilde{t}$ are computed from values measured on completed tasks (see Section 5.3 in Chapter 5). When less than 2 tasks are completed, medians remain undefined and the control process is inactive. This online estimation makes our process non-clairvoyant with respect to the task duration which is progressively estimated as the workflow activity runs. Yet, it assumes that all tasks in an activity have similar costs.

In equation 6.2, r_i is the ratio between the max of the task queuing times q_i in the group, and the total round-trip time (queuing+execution) of the group:

$$r_i = \frac{\max_{j\in[1,n_i]} q_j}{\max_{j\in[1,n_i]} q_j + \tilde{t}_{\_shared} + n_i(\tilde{t} - \tilde{t}_{\_shared})}$$

Group queuing time is the max of all task queuing times in the group; group execution time is the time to transfer shared input data plus the time to execute all task phases in the group except for the transfers of shared input data. Note that d_i, r_i, and therefore f_i and η_f are in $[0, 1]$. η_f tends to 0 when there is little shared input data among the activity tasks or when the task queuing times are low compared to the execution times; in both cases, grouping tasks is indeed useless. Conversely, η_f tends to 1 when the transfer time of shared input data becomes high, and the queuing time is high compared to the execution time; grouping is needed in this case.

Threshold value τ_f. The threshold value for η_f separates configurations where the activity's fineness is acceptable ($\eta_f \leq \tau_f$) from configurations where the activity is too fine ($\eta_f > \tau_f$). We determine τ_f from execution traces, inspecting the modes of the distribution of η_f. Values of η_f in the highest mode of the distribution, i.e. which are clearly separated from the others, will be considered too fine.

The fineness degree η_f was computed after each event found in the workload archive (Chapter 3). Figure 6.1 shows the histogram of these values. The histogram appears bimodal, which indicates that η_f separates platform configurations in two distinct groups. We assume that these groups correspond to "acceptable fineness" (lowest mode) and "too fine granularity" (highest mode), and thus we choose $\tau_f = 0.55$. For $\eta_f \geq 0.55$, task grouping will therefore be triggered.

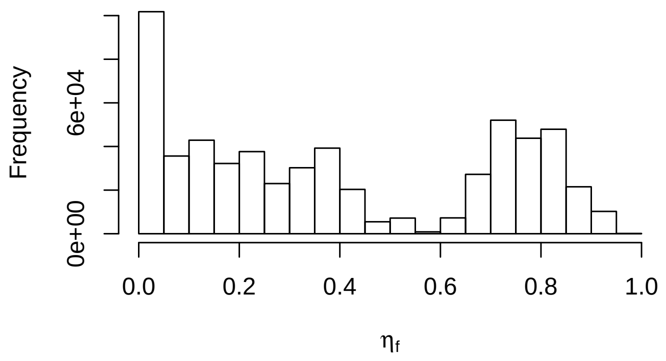


Figure 6.1: Histogram of fineness incident degree sampled in bins of 0.05.

Task grouping. We assume that running tasks cannot be pre-empted, i.e. only waiting tasks can be grouped. Algorithm 6 describes our task grouping. Groups with $f_i > \tau_f$ are grouped pairwise until $\eta_f \leq \tau_f$ or until the amount of waiting groups Q is smaller or equal to the amount of running groups R. Although η_f ignores scattering (Equation 6.1 uses a max), the algorithm considers it by grouping tasks in all groups where $f_i > \tau_f$. Ordering groups by decreasing f_i values tends to equally distribute tasks among groups. The grouping process stops when $Q \leq R$ to avoid parallelism loss. This condition also avoids conflicts with the de-grouping process described in the next sub-section.

Algorithm 6 Task grouping

```
1:  input: $f_1$ to $f_m$ //group fineness degrees, sorted in decreasing order
2:  input: $Q$, $R$ // number of queued and running task groups
3:  for $i = 1$ to $m - 1$ do
4:      $j = i + 1$
5:      while $f_i > \tau_f$ and $Q > R$ and $j \leq m$ do
6:          if $f_j > \tau_f$ then
7:              Group all tasks of $T_j$ into $T_i$
8:              Recalculate $f_i$ using Equation 6.2
9:              $Q = Q - 1$
10:         end if
11:         $j = j + 1$
12:     end while
13:     $i = j$
14: end for
15: Delete all empty task groups
```

6.2.2 Coarseness control

Condition $Q > R$ used in Algorithm 6 ensures that all resources will be exploited *if the number of available resources is stationary.* In case the number of available resources decreases, the fineness control process may further reduce the number of groups. However, if the number of available resources increases, task groups may need to be de-grouped to maximize resource exploitation. This de-grouping is implemented by our coarseness control process.

The coarseness control process monitors the value of η_c defined as:

$$\eta_c = \frac{R}{Q+R}. \tag{6.3}$$

The threshold value τ_c is set to 0.5 so that $\eta_c > \tau_c \Leftrightarrow Q < R$.

When an activity is considered too coarse, its groups are ordered by increasing values of η_f and the first groups (i.e. the coarsest ones) are split until $\eta_c < \tau_c$. Note that de-grouping increases the number of queued tasks, therefore tends to reduce η_c. Table 6.1 illustrates the method on a simple example.

In this example, let's consider a workflow composed of one activity with 10 tasks initially split in 10 groups, and assume that task input data are shared among all tasks (i.e. $\tilde{t}_{\_shared} = \tilde{t}_{\_input}$).
Let $\tilde{t} = 10$ and $\tilde{t}_{\_shared} = 7$ (in arbitrary time units) obtained from two completed task groups.
At time t, we assume $R = 2$ and $Q = 6$ with the following values for waiting task groups:

i	$\max_{j\in[1,n_i]} q_j$	d_i	r_i	f_i
5	50	0.70	0.83	0.58
6	48	0.70	0.82	0.58
7	45	0.70	0.81	0.57
8	43	0.70	0.81	0.57
9	41	0.70	0.80	0.56
10	40	0.70	0.80	0.56

Eq. 6.1 gives $\eta_f = 0.58$. As $\eta_f > \tau_f = 0.55$ and $Q > R$, the activity is considered too fine and task grouping is triggered. Groups with $f_i > \tau_f$ are grouped pairwise until $\eta_f \leq \tau_f$ or $Q \leq R$:

i	$\max_{j\in[1,n_i]} q_j$	d_i	r_i	f_i
11 [5,6]	50	0.53	0.79	0.42
12 [7,8]	45	0.53	0.77	0.41
13 [9,10]	41	0.53	0.76	0.40

Groups 5 and 6, 7 and 8, and 9 and 10 are grouped into groups 11, 12, and 13.

Let's consider that at time $t' > t$, group 11 starts running, thus $Q = 2 < R = 3$.
Eq. 6.3 gives $\eta_c = 0.6$. As $\eta_c > \tau_c = 0.5$, the activity is consider too coarse and task de-grouping is triggered. Then, group 13 is de-grouped to reduce η_c.

Table 6.1: Example

6.3 Experiments and Results

The experiments presented hereafter evaluate, in a production environment, the fineness control process under stationary load, and the interest of controlling coarseness under non-stationary load.

6.3.1 Experiment Conditions

The granularity control process was implemented as a plugin of the MOTEUR workflow manager, receiving notifications about task status changes and task phase durations. The plugin then uses this data to group and de-group tasks according to Algorithm 5, where the timeout value is set to 2 minutes.

The target computing platform for these experiments is the biomed VO of EGI used by VIP (see Section 2.2.5 in Chapter 2). To ensure resource limitation without flooding the production system, experiments are performed only on 3 sites of different countries. Tasks generated by MOTEUR are submitted to EGI using the DIRAC scheduler. As no online task modification is possible in DIRAC, we implemented task grouping by canceling queued tasks and submitting grouped tasks as a new task.

Three workflow activities, implementing different kinds of medical image simulation, are used in the experiments: `SimuBloch`, `FIELD-II`, and `PET-Sorteo/emission` (see description in Appendix A). Table 6.2 shows their main characteristics.

Workflow activity		#Tasks	CPU time	Input	Output	$\tilde{t}_{\_shared}/\tilde{t}$
SimuBloch	(data-intensive)	25	few seconds	~15 MB	< 5 MB	~0.9
FIELD-II	(data-intensive)	122	few seconds to 15 minutes	~208 MB	~40 KB	[0.4,0.6]
PET-Sorteo	(CPU-intensive)	80	~10 minutes	~20 MB	~50 MB	[0.5,0.8]

Table 6.2: Workflow activity characteristics.

Two sets of experiments are conducted, under different load patterns. Experiment 1 evaluates the fineness control process only under stationary load. It consists of separated executions of `SimuBloch`, `FIELD-II`, and `PET-Sorteo/emission`. A workflow activity using our task grouping mechanism (`Fineness`) is compared to a control activity (`No-Granularity`). Resource contention on the 3 execution sites is maintained high and constant so that no degrouping is required.

Experiment 2 evaluates the interest of using the de-grouping control process under non-stationary load. It uses activity `FIELD-II`. An execution using both fineness and coarseness control (`Fineness-Coarseness`) is compared to an execution without coarseness control (`Fineness`) and to a control execution (`No-Granularity`). Executions are started under resource contention, but the contention is progressively reduced during the experiment. This is done by submitting a heavy workflow before the experiment starts, and canceling it when half of the control tasks are completed.

For both experiments, workflow activities executions are launched simultaneously to ensure similar grid conditions. For each grouped task resubmitted in the `Fineness` or `Fineness-Coarseness` executions, a task in the `No-Granularity` is resubmitted too to ensure equal race conditions for resource allocation. Five repetitions of each experiment are performed, along a time period of 4 weeks to cover different grid conditions. We use MOTEUR 0.9.21, configured to resubmit failed tasks up to 5 times, and with the task replication mechanism (`Median Estimation`) described in Chapter 5 activated. We use the DIRAC v6r6p2 instance provided by France-Grilles (see Section 2.2.4 in Chapter 2). Results could not be compared to other grouping/de-grouping methods due to the lack of non-clairvoyant, online method available in the literature.

6.3.2 Results and Discussion

Experiment 1. Figure 6.2 shows the makespan of `SimuBloch`, `FIELD-II`, and `PET-Sorteo/emission` executions. `Fineness` yields a significant makespan reduction for all repetitions. Table 6.3 shows the makespan (M) values and the number of task groups. The task grouping mechanism is not able to group all `SimuBloch` tasks in a single group because 2 tasks must be completed for the process to have enough information about the application (i.e. $\tilde{t}_{\_shared}$ and $\tilde{t}$ can be computed). This is a constraint of our non-clairvoyant conditions, where task durations cannot be determined in advance. `FIELD-II` tasks are initially not grouped, but as the queuing time becomes important, tasks are considered too fine and grouped. `PET-Sorteo/emission` is an intermediary case where only a few tasks are grouped. Results show that the task grouping mechanism speeds up `SimuBloch` and `FIELD-II` executions up to a factor of 2.6, and `PET-Sorteo/emission` executions up to a factor of 2.5.

Experiment 2. Figure 6.3 shows the makespan (top) and evolution of task groups (bottom). Makespan values are reported in Table 6.4. In the first three repetitions, resources emerge progressively during workflow executions. `Fineness` and `Fineness-Coarseness` speed up executions up to a factor of 1.5 and 2.1. Since `Fineness` does not benefit of newly arrived resources, it has a lower speed up compared to `No-Granularity` due to parallelism loss. In the two last repetitions, the de-grouping process in `Fineness-Coarseness` allows to reach similar performance than `No-Granularity`, while `Fineness` is penalized by its lack of adaptation: a slowdown of 20% is observed compared to `No-Granularity`.

Our task granularity control process works best under high resource contention, when the amount of available resources is stable or decreases over time (Experiment 1). Coarseness control can cope with soft increases in the number of available resources (Experiment 2), but fast variations remain difficult to handle. In the worst-case scenario, tasks are first grouped due

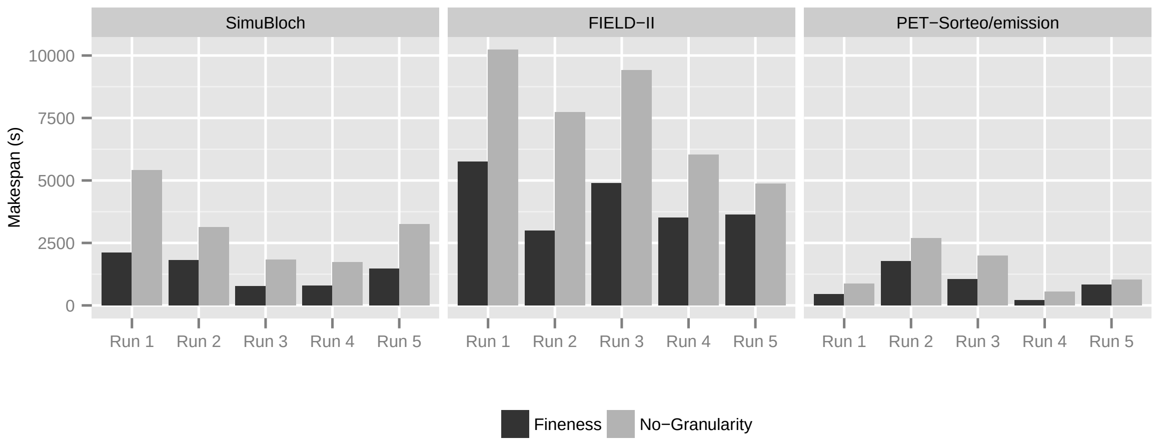


Figure 6.2: Experiment 1: makespan for `Fineness` and `No-Granularity` executions for the 3 workflow activities under stationary load.

		SimuBloch		FIELD-II		PET-Sorteo	
		M (s)	Groups	*M* (s)	Groups	*M* (s)	Groups
1	No-Granularity	5421	25	10230	122	873	80
	Fineness	2118	3	5749	80	451	57
2	No-Granularity	3138	25	7734	122	2695	80
	Fineness	1803	3	2982	75	1766	40
3	No-Granularity	1831	25	9407	122	1983	80
	Fineness	780	4	4894	73	1047	53
4	No-Granularity	1737	25	6026	122	552	80
	Fineness	797	6	3507	61	218	64
5	No-Granularity	3257	25	4865	122	1033	80
	Fineness	1468	4	3641	91	831	71

Table 6.3: Experiment 1: makespan (*M*) and number of task groups for `SimuBloch`, `FIELD-II` and `PET-Sorteo/emission` executions for the 5 repetitions.

to resource limitation, and resources suddenly appear once all task groups are already running. In this case the de-grouping algorithm has no group to handle, and granularity control penalizes the execution. Task pre-emption should be added to the method to address this scenario.

6.4 Conclusion

We presented a method to optimize task granularity in distributed workflows in an online and non-clairvoyant environment. We defined a metric η_f for online determination of task fineness based on queue waiting time and estimated data transfer time of shared input data. For high η_f values, tasks are considered too fine and task grouping is triggered. Queued tasks are grouped pairwise as long as the number of queued tasks is greater than the number of running tasks and

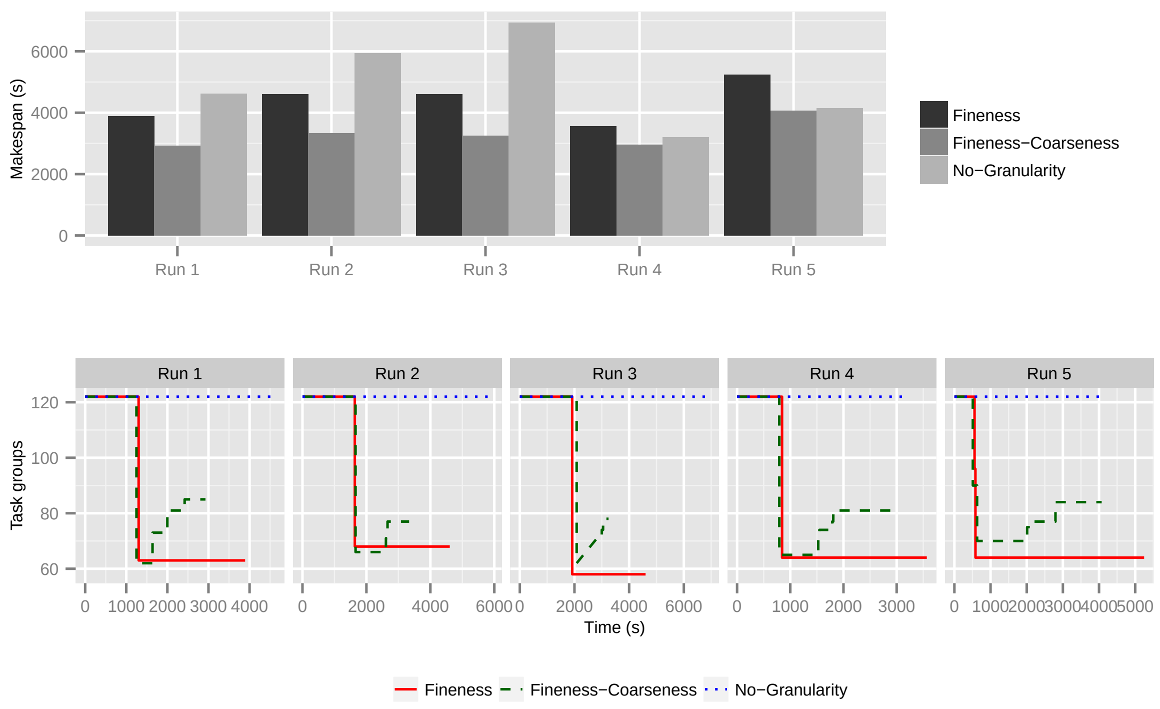


Figure 6.3: Experiment 2: makespan (top) and evolution of task groups (bottom) for `FIELD-II` executions under non-stationary load (resources arrive during the experiment).

	Run 1		Run 2		Run 3		Run 4		Run 5	
	M (s)	$\bar{q}$ (s)	M (s)	$\bar{q}$ (s)	M (s)	$\bar{q}$ (s)	M (s)	$\bar{q}$ (s)	M (s)	$\bar{q}$ (s)
No-Granularity	4617	2111	5934	2765	6940	3855	3199	1863	4147	2295
Fineness	3892	2036	4607	2090	4602	2631	3567	1928	5247	2326
Fineness-Coarseness	2927	1708	3335	1829	3247	2091	2952	1586	4073	2197

Table 6.4: Experiment 2: makespan (M) and average queuing time ($\bar{q}$) for `FIELD-II` workflow execution for the 5 repetitions.

η_f is considered too fine. We also define a metric η_c for online determination of task coarseness based on the ratio of the number of queued tasks related to the number of running tasks. This metric aims at maximizing resource exploitation by de-grouping tasks groups when the number of available resources increases.

The task granularity control strategy was implemented in the MOTEUR workflow engine and deployed on EGI with the DIRAC resource manager. We tested it on three applications extracted from the Virtual Imaging Platform (Chapter 2). Two experiments were conducted, to evaluate the fineness control process only under stationary load and the fineness and coarseness control process under non-stationary load. Results showed that under stationary load, our fineness control process significantly reduces the makespan of all applications. Under non-stationary load, task grouping is penalized by its lack of adaptation, but our de-grouping algorithm corrects it in case variations in the number of available resources are not too fast.

So far we have handled incidents up to the activity level. In the next chapter, we present a healing process to cope with an incident at the platform level: unfairness among workflow executions.

Chapter 7

Controlling fairness among workflow executions

Contents

Fairly allocating distributed computing resources among workflow executions is critical to multi-user platforms. However, this problem remains mostly studied in clairvoyant and offline conditions, where task durations on resources are known, or the workload and available resources do not vary along time. We consider a non-clairvoyant, online fairness problem where the platform workload, task costs and resource characteristics are unknown and not stationary. We propose a fairness control loop which assigns task priorities based on the fraction of pending work in the workflows. Workflow characteristics and performance on the target resources are estimated progressively, as information becomes available during the execution. Our method is implemented and evaluated on 4 different applications executed in production conditions on the European Grid Infrastructure. Results show that our technique reduces slowdown variability by a factor of 3 to 7 compared to first-come-first-served.

Répartir équitablement les ressources de calcul entre les exécutions de workflow est essentiel pour des plate-formes multi-utilisateurs. Toutefois, ce problème reste exclusivement étudié dans des conditions de clairvoyance et « offline » , où la durée des tâches sur les ressources sont connues, et oú les ressources disponibles et la charge de travail ne varie pas au cours du temps. Nous considérons un problème d'équité non-clairvoyant et « online » où les caractéristiques de charge de travail de la plate-forme, les caractéristiques de ressources, et les coûts de tâches sont inconnus et non stationnaires. Nous proposons une boucle de contrôle de l'équité qui assigne les priorités des tâches en fonction de la proportion de calcul en cour d'exécution dans leur workflow. Les caractéristiques et les performances des workflows sur les ressources cibles sont estimés progressivement, au fin et à mesure que l'information devient disponible. Notre méthode est mise en œuvre et évaluée sur 4 différentes applications exécutées dans des conditions de production sur EGI. Les résultats montrent que notre technique permet de réduire la variabilité du ralentissement d'un factor de 3 à 7 par rapport au FCFS.

7.1 Introduction

The problem of fairly allocating computing resources to application workflows rapidly arises on shared computing platforms such as grids or clouds. It must be addressed whenever the demand for resources is higher than the offer, that is, when some workflows are slowed down by concurrent executions. In some cases, unfairness makes the platform totally unusable, for instance when very short executions are launched concurrently with longer ones. We define fairness as in [N'Takpe and Suter, 2009, Zhao and Sakellariou, 2006, Casanova et al., 2010], i.e. as the variability in a set of workflows of the *slowdown* $\frac{M_{multi}}{M_{own}}$, where M_{multi} is the makespan when concurrent executions are present, and M_{own} is the makespan without concurrent executions.

We consider a science-gateway where users can, at any time, launch application workflows that will compete for computing resources. Our two main assumptions are that the problem is *online* and *non-clairvoyant*. We also assume a limited control on the scheduler, i.e. that only task priorities can be changed to influence scheduling.

In this chapter, we propose an algorithm to control fairness in these conditions. Based on a progressive discovery of applications' characteristics on the infrastructure, our method dynamically estimates the fraction of pending work for each workflow. Task priorities are then adjusted to harmonize this fraction among the active workflows. This way, resources are allocated to application workflows relatively to their amount of work to compute. The method is implemented in VIP, and evaluated with different workflows, in production conditions, on the EGI. We use the slowdown as a *post-mortem* metric, to evaluate our method once execution times are known.

The next section details our fairness control process, and section 7.3 presents experiments and results. Section 7.4 concludes the chapter.

7.2 Fairness control process

Workflows consist of linked activities spawning tasks for which the executable and input data are known, but the computational cost and produced data volume are not. Algorithm 7 summarizes our fairness control process. Fairness is controlled by allocating resources to workflows according to their fraction of pending work. It is done by reprioritising tasks in workflows where the unfairness degree η_u is greater than a threshold τ_u. This section describes how η_u and τ_u are computed, and details the re-prioritization algorithm.

Algorithm 7 Main loop for fairness control

```
1: input: $m$ workflow executions
2: while there is an active workflow do
3:     wait for timeout or task status change in any workflow
4:     determine unfairness degree $\eta_u$
5:     if $\eta_u > \tau_u$ then
6:         re-prioritize tasks using Algorithm 8
7:     end if
8: end while
```

Unfairness degree η_u. Let m be the number of workflows with an active activity; a workflow activity is active if it has at least one waiting (queued) or running task. The unfairness degree η_u is the maximum difference between the fractions of pending work:

$$\eta_u = W_{\max} - W_{\min}, \tag{7.1}$$

with $W_{\min} = \min\{W_i, i \in [1, m]\}$ and $W_{\max} = \max\{W_i, i \in [1, m]\}$. All W_i are in $[0, 1]$. For $\eta_u = 0$, we consider that resources are fairly distributed among all workflows; otherwise, some workflows consume more resources than they should. The fraction of pending work W_i of a workflow $i \in [1, m]$ is defined from the fraction of pending work $w_{i,j}$ of its n_i active activities:

$$W_i = \max_{j \in [1, n_i]} (w_{i,j}) \tag{7.2}$$

All $w_{i,j}$ are between 0 and 1. A high $w_{i,j}$ value indicates that the activity has a lot of pending work compared to the others. We define $w_{i,j}$ as:

$$w_{i,j} = \frac{Q_{i,j}}{Q_{i,j} + R_{i,j} P_{i,j}} \cdot T_{i,j}, \tag{7.3}$$

where $Q_{i,j}$ is the number of waiting tasks in the activity, $R_{i,j}$ is the number of running tasks in the activity, $P_{i,j}$ is the performance of the activity, and $T_{i,j}$ is its relative observed duration. $T_{i,j}$

is defined as the ratio between the median duration $\tilde{t}_{i,j}$ of the completed tasks in activity j and the maximum median task duration among all active activities of all running workflows:

$$T_{i,j} = \frac{\tilde{t}_{i,j}}{\max_{v\in[1,m],w\in[1,n_i^*]}(\tilde{t}_{v,w})} \tag{7.4}$$

Tasks of an activity all consist of the following successive phases: `setup`, `inputs download`, `application execution` and `outputs upload`; $\tilde{t}_{i,j}$ is computed as $\tilde{t}_{i,j} = \tilde{t}_{i,j}^{setup} + \tilde{t}_{i,j}^{input} + \tilde{t}_{i,j}^{exec} + \tilde{t}_{i,j}^{output}$. Medians are progressively estimated as tasks complete. At the beginning of the execution, $T_{i,j}$ is initialized to 1 and all medians are undefined; when two tasks of activity j complete, $\tilde{t}_{i,j}$ is updated and $T_{i,j}$ is computed with equation 7.4. In this equation, the max operator is computed only on $n_i^* \leq n_i$ activities with at least 2 completed tasks, i.e. for which $\tilde{t}_{i,j}$ can be determined. We are aware that using the median may be inaccurate. However, without a model of the applications' execution time, we must rely on observed task durations. Using the whole time distribution (or at least its few first moments) may be more accurate but it would complexify the method.

In Equation 7.3, the performance $P_{i,j}$ of an activity varies between 0 and 1. A low $P_{i,j}$ indicates that resources allocated to the activity have bad performance for the activity; in this case, the contribution of running tasks is reduced and $w_{i,j}$ increases. Conversely, a high $P_{i,j}$ increases the contribution of running tasks, therefore decreases $w_{i,j}$. For an activity j with k_j active tasks, we define $P_{i,j}$ as:

$$P_{i,j} = 2 \cdot \left(1 - \max_{u\in[1,k_j]}\left\{\frac{t_u}{\tilde{t}_{i,j} + t_u}\right\}\right), \tag{7.5}$$

where $t_u = t_u^{setup} + t_u^{input} + t_u^{exec} + t_u^{output}$ is the sum of the estimated durations of task u's phases. Estimated task phase durations are computed as the max between the current elapsed time in the task phase (0 if the task phase has not started) and the median duration of the task phase (see Section 5.3 in Chapter 5). $P_{i,j}$ is initialized to 1, and updated using Equation 7.5 only when at least 2 tasks of activity j are completed. Note that computing $P_{i,j}$ is equivalent to computing the complement of the activity blocked degree by median estimation $1 - \eta_b$ for activity j of workflow i (see Section 5.3 in Chapter 5).

If all tasks perform as the median, i.e. $t_u = \tilde{t}_{i,j}$, then $\max_{u\in[1,k_j]}\left\{\frac{t_u}{\tilde{t}_{i,j}+t_u}\right\} = 0.5$ and $P_{i,j} = 1$. Conversely, if a task in the activity is much longer than the median, i.e. $t_u \gg \tilde{t}_{i,j}$, then $\max_{u\in[1,k_j]}\left\{\frac{t_u}{\tilde{t}_{i,j}+t_u}\right\} \approx 1$ and $P_{i,j} \approx 0$. This definition of $P_{i,j}$, considers that bad performance results in a few tasks blocking the activity. Indeed, we assume that the scheduler does not deliberately favor any activity and that performance discrepancies are manifested by a few "unlucky" tasks slowed down by bad resources. Performance, in this case, has a relative definition: depending on the activity profile, it can correspond to CPU, RAM, network bandwidth, latency, or a combination of those. We admit that this definition of $P_{i,j}$ is a bit rough. However, under our non-clairvoyance assumption, estimating resource performance for the activity more accurately is hardly possible because (*i*) we have no model of the application, therefore task durations cannot be predicted from CPU, RAM or network characteristics, and (*ii*) network characteristics and even available RAM are shared among concurrent tasks running on the infrastructure, which makes them hardly measurable.

Thresholding unfairness: τ_u. Task prioritization is triggered when the unfairness degree is considered critical, i.e $\eta_u > \tau_u$. We inspect the modes of the distribution of η_u to determine a threshold with a practical justification: values of η_u in the highest mode of the distribution, i.e. which are clearly separated from the others, will be considered unfair. The distribution of η_u is measured from traces collected from VIP (Chapter 3).

Figure 7.1 shows the histogram of these values, where only $\eta_u \neq 0$ values are represented. This histogram is clearly bi-modal, which is a good property since it reduces the influence of τ_u. From this histogram, we choose $\tau_u = 0.2$. For $\eta_u > 0.2$, task prioritization is triggered.

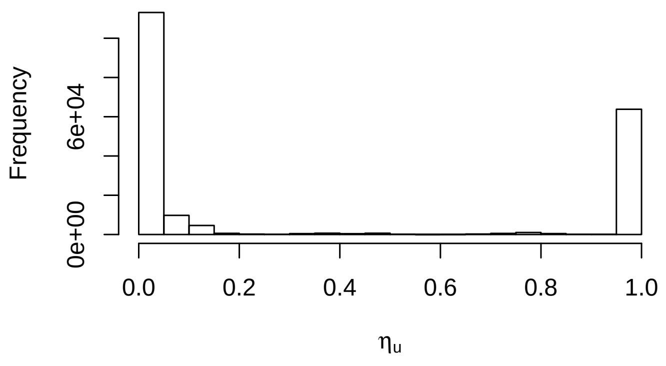


Figure 7.1: Histogram of the unfairness degree η_u sampled in bins of 0.05.

Task prioritization. Task priority is an integer initialized to 1. The action taken to cope with unfairness is to increase the priority of $\Delta_{i,j}$ waiting tasks for all activities j of workflow i where $w_{i,j} - W_{\min} > \tau_u$. Running tasks cannot be pre-empted. $\Delta_{i,j}$ is determined so that $\tilde{w}_{i,j} = W_{min} + \tau_u$, where $\tilde{w}_{i,j}$ is the estimated value of $w_{i,j}$ after $\Delta_{i,j}$ tasks are prioritized. We approximate $\tilde{w}_{i,j}$ as:

$$\tilde{w}_{i,j} = \frac{Q_{i,j} - \Delta_{i,j}}{Q_{i,j} + R_{i,j} P_{i,j}} \hat{T}_{i,j},$$

which assumes that $\Delta_{i,j}$ tasks will move from status queued to running, and that the performance of new resources will be maximal. It gives:

$$\Delta_{i,j} = Q_{i,j} - \left\lfloor \frac{(\tau_u + W_{\min})(Q_{i,j} + R_{i,j} P_{i,j})}{T_{i,j}} \right\rfloor, \tag{7.6}$$

where $\lfloor\rfloor$ rounds a decimal down to the nearest integer value.

Algorithm 8 describes our task re-prioritization. *maxPriority* is the maximal priority value in all workflows. The priority of $\Delta_{i,j}$ waiting tasks is set to *maxPriority+1* in all activities j of workflows i where $w_{i,j} - W_{min} > \tau_u$. Note that this algorithm takes into account scatter among W_i although η_u ignores it (see Equation 7.1). Indeed, tasks are re-prioritized in *any* workflow i for which $W_i - W_{\min} > \tau_u$.

Algorithm 8 Task re-prioritization

```
1:  input: $W_1$ to $W_m$ //fractions of pending works
2:  maxPriority = max task priority in all workflows
3:  for i=1 to m do
4:      if $W_i - W_{\min} > \tau_u$ then
5:          for j=1 to $a_i$ do
6:              //$a_i$ is the number of active activities in workflow $i$
7:              if $w_{i,j} - W_{\min} > \tau_u$ then
8:                  Compute $\Delta_{i,j}$ from equation 7.6
9:                  for p=1 to $\Delta_{i,j}$ do
10:                     if ∃ waiting task q in activity $j$ with priority ≤ maxPriority then
11:                         $q$.priority = maxPriority + 1
12:                     end if
13:                 end for
14:             end if
15:         end for
16:     end if
17: end for
```

The method also accommodates online conditions. If a new workflow i is submitted, then $R_{i,j} = 0$ for all its activities and $\hat{T}_{i,j}$ is initialized to 1. This leads to $W_{max} = W_i = 1$, which increases η_u. If η_u goes beyond τ_u, then $\Delta_{i,j}$ tasks of activity j of workflow i have their priorities increased to restore fairness. Similarly, if new resources arrive, then $R_{i,j}$ increase and η_u is updated accordingly. Table 7.1 illustrates the method on a simple example.

7.3 Experiments and results

Experiments are performed on a production grid platform to ensure realistic conditions. Evaluating fairness in production by measuring the slowdown is not straightforward because M_{own} (see definition in the Introduction) cannot be directly measured. As described in Section 7.3.1, we estimate the slowdown from task durations, but this estimation may be challenged. Thus, Experiment 1 evaluates our method on a set of identical workflows, where the variability of the measured makespan can be used as a fairness metric. In Experiment 2, we add a very short workflow to this set of identical workflow, which was one of the configurations motivating this

Let's consider two identical workflows composed of one activity with 6 tasks, and assume the following values at time t:

i	$Q_{i,1}$	$R_{i,1}$	$\tilde{t}_{i,1}$	$P_{i,1}$	$T_{i,1}$	$W_i = w_{i,1}$
1	1	3	10	0.9	1.0	0.27
2	6	0	-	1.0	1.0	1.00

Values unknown at time t are noted '-'. Workflow 1 has 2 completed and 3 running tasks with the following phase durations (in arbitrary time units):

u	t_u^{setup}	t_u^{input}	t_u^{exec}	t_u^{output}	t_u
1	2	2	4	1	9
2	1	2	3	2	8
3	2	3	5	-	-
4	2	2	-	-	-
5	1	-	-	-	-

, We have $\tilde{t}_{1,1}^{setup} = 2$, $\tilde{t}_{1,1}^{input} = 2$, $\tilde{t}_{1,1}^{exec} = 4$ and $\tilde{t}_{1,1}^{output} = 2$. Therefore, $\tilde{t}_{1,1} = 10$.

The configuration is clearly unfair since workflow 2 has 0 started tasks.
Eq. 7.1 gives $\eta_u = 0.73$. As $\eta_u > \tau_u = 0.2$, the platform is considered unfair and task re-prioritization is triggered.

$\Delta_{2,1}$ tasks from workflow 2 should be prioritized. According to Eq. 7.6:
$\Delta_{2,1} = Q_{2,1} - \left\lfloor \frac{(\tau_u+W_1)(Q_{2,1}+R_{2,1}P_{2,1})}{T_{2,1}} \right\rfloor = 6 - \left\lfloor \frac{(0.2+0.27)(6+0 \cdot 1.0)}{1.0} \right\rfloor = 4$

At time $t' > t$:

i	$Q_{i,1}$	$R_{i,1}$	$\tilde{t}_{i,1}$	$P_{i,1}$	$T_{i,1}$	$W_i = w_{i,1}$
1	1	3	10	0.8	1.0	0.29
2	2	4	-	1.0	1.0	0.33

.

Now, $\eta_u = 0.04 < \tau_u$. The platform is considered fair and no action is performed.

Table 7.1: Example

study. Finally, Experiment 3 considers the more general case of 4 different workflows with heterogeneous durations.

7.3.1 Experiment conditions

Fairness control was implemented as a MOTEUR plugin receiving notifications about task and workflow status changes. Each workflow plugin forwards task status changes and $\tilde{t}_{i,j}$ values to a service centralizing information about all the active workflows. This service then re-prioritizes tasks according to Algorithms 7 and 8. The timeout value used in Algorithm 7 is set to 3 minutes. As no online task modification is possible in DIRAC, we implemented task prioritization by canceling and resubmitting queued tasks to DIRAC with new priorities.

The computing platform for these experiments is the biomed VO (see Section 2.2.5 in Chapter 2). To ensure resource limitation without flooding the production system, experiments are performed only on 3 sites of different countries (France, Spain and Netherlands). Four real medical simulation workflows are considered: `GATE`, `SimuBloch`, `FIELD-II`, and `PET-Sorteo` (see description in Appendix A). Table 7.2 shows their main characteristics.

Workflow		#Tasks	CPU time	Input	Output
GATE	(CPU-intensive)	100	few minutes to one hour	~115 MB	~40 MB
SimuBloch	(data-intensive)	25	few seconds	~15 MB	< 5 MB
FIELD-II	(data-intensive)	122	few seconds to 15 minutes	~208 MB	~40 KB
PET-Sorteo	(CPU-intensive)	1→80→1→80→1→1	~10 minutes	~20 MB	~50 MB

Table 7.2: Workflow characteristics (→ indicate task dependencies).

Three experiments are conducted. Experiment 1 tests whether unfairness among *identical workflows* is properly addressed. It consists of three `GATE` workflows sequentially submitted. Experiment 2 tests if the performance of *very short workflow executions* is improved by the fairness mechanism. Its workflow set has three `GATE` workflows launched sequentially, followed by a `SimuBloch` workflow. Experiment 3 tests whether unfairness among *different workflows* is detected and properly handled. Its workflow set consists of a `GATE`, a `FIELD-II`, a `PET-Sorteo` and a `SimuBloch` workflow launched sequentially.

For each experiment, a workflow set using our fairness mechanism (`Fairness – F`) is compared to a control workflow set (`No-Fairness – NF`). No method from the literature could be included in the comparison because, as mentioned in Section 1.2.4 (Chapter 1), they are either non-clairvoyant or offline. `Fairness` and `No-Fairness` are launched simultaneously to ensure similar grid conditions. For each task priority increase in the `Fairness` workflow set, a task in the `No-Fairness` workflow set task queue is also prioritized to ensure equal race conditions for resource allocation. Four repetitions of each experiment are done, along a time period of four weeks to cover different grid conditions. We use MOTEUR 0.9.21, configured to resubmit failed tasks up to 5 times, and with the task replication mechanism described in Section 5.3 (Chapter 5) activated. We use the DIRAC v6r5p1 instance provided by France-Grilles (Section 2.2.4 in Chapter 2), with a first-come, first-served policy imposed by submitting workflows with decreasing priority values.

Two different fairness metrics are used. The unfairness μ is the area under the curve η_u during the execution:

$$\mu = \sum_{i=2}^{M} \eta_u(t_i) \cdot (t_i - t_{i-1}),$$

where M is the number of time samples until the makespan. This metric measures if the fairness process can indeed minimize its own criterion η_u. In addition, the slowdown s of a completed workflow execution is measured as:

$$s = \frac{M_{multi}}{M_{own}}$$

where M_{multi} is the makespan observed on the shared platform, and M_{own} is the estimated makespan if it was executed alone on the platform. In our conditions, M_{own} is estimated as:

$$M_{own} = \max_{p \in \Omega} \sum_{u \in p} t_u,$$

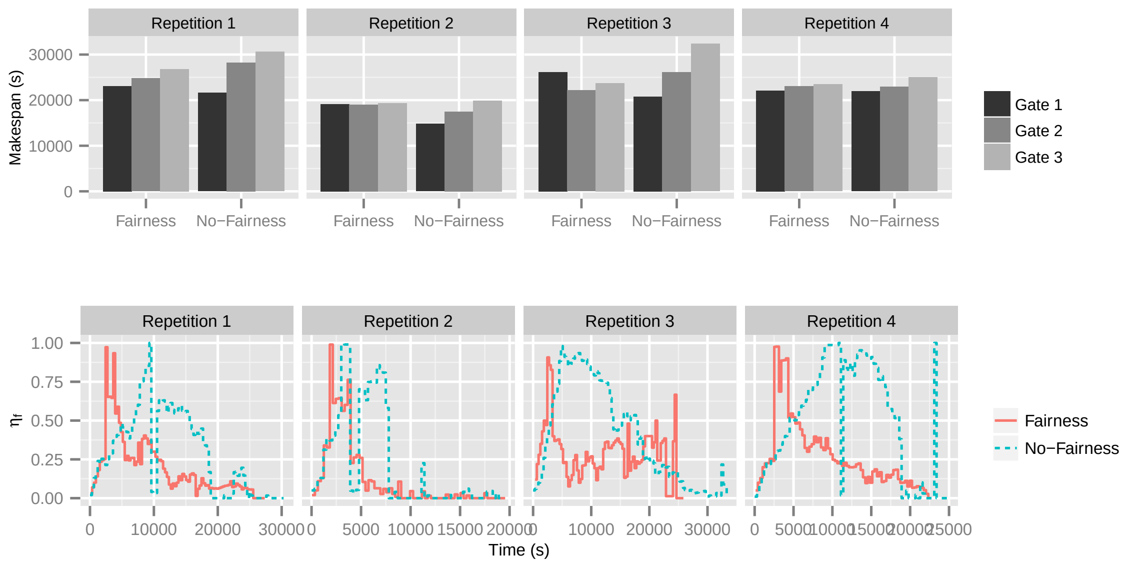
where Ω is the set of task paths in the workflow, and t_u is the measured duration of task u. This assumes that concurrent executions only impact task waiting time. For instance, network congestion or changes in performance distribution resulting from concurrent executions are ignored. We use σ_s, the standard deviation of the slowdown to quantify unfairness. In Experiment 1, the standard deviation of the makespan (σ_m) is also used.

7.3.2 Results and discussion

Experiment 1 (*identical workflows*). Figure 7.2 shows the makespan, unfairness degree η_u, makespan standard deviation σ_m, slowdown standard deviation σ_s and unfairness μ for the 4 repetitions. The difference among makespans and unfairness degree values are significantly reduced in all repetitions of Fairness. Both Fairness and No-Fairness behave similarly until η_u reaches the threshold value $\tau_u = 0.2$. Unfairness is then detected and the mechanism triggers task prioritization. Paradoxically, the first effect of task prioritization is a slight increase of η_u. Indeed, $P_{i,j}$ and $\hat{T}_{i,j}$, that are initialized to 1, start changing earlier in Fairness than in No-Fairness due to the availability of task duration values to compute $\tilde{t}_{i,j}$. Note that η_u reaches similar maximal values in both cases, but reaches them faster in Fairness. The fairness mechanism then manages to decrease η_u back under 0.2 much faster than it happens in No-Fairness when tasks progressively complete. Finally, slight increases of η_u are sporadically observed towards the end of the execution. This is due to task replications performed by MOTEUR (see Section 5.3 in Chapter 5): when new tasks are created, the fraction of pending work work W increases, which has an effect on η_u. Quantitatively, the fairness mechanism reduces σ_m up to a factor of 15, σ_s up to a factor of 7, and μ by about 2.

Experiment 2 (*very short execution*). Figure 7.3 shows the makespan, unfairness degree η_u, unfairness μ and slowdown standard deviation. In all cases, the makespan of the very short SimuBloch executions is significantly reduced for Fairness. The evolution of η_u is coherent with Experiment 1: a common initialization phase followed by an anticipated growth and decrease for Fairness. Fairness reduces σ_s up to a factor of 5.9 and unfairness up to a factor of 1.9.

Table 7.3 shows the execution makespan (m), average wait time ($\bar{w}$) and slowdown (s) values for the SimuBloch execution launched after the 3 GATE. As it is a non-clairvoyant scenario



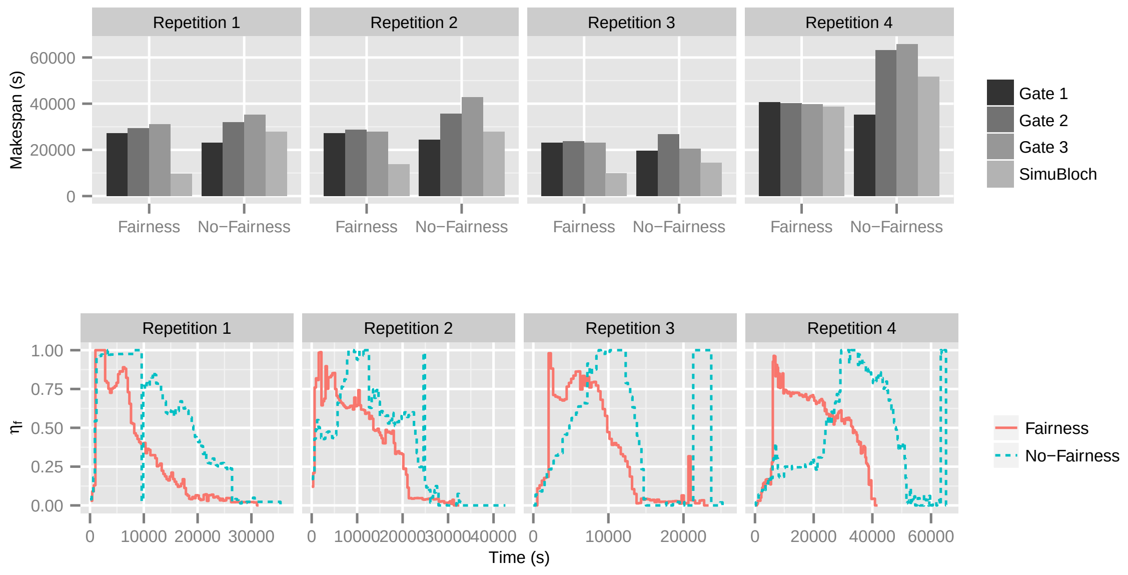
Repetition 1				Repetition 2				Repetition 3				Repetition 4			
	$\sigma_m(s)$	σ_s	$\mu(s)$		$\sigma_m(s)$	σ_s	$\mu(s)$		$\sigma_m(s)$	σ_s	$\mu(s)$		$\sigma_m(s)$	σ_s	$\mu(s)$
`NF`	4666	1.03	8758	`NF`	2541	0.50	4154	`NF`	5791	2.10	13392	`NF`	1567	0.87	12283
`F`	1884	0.40	5292	`F`	167	0.07	2367	`F`	2007	0.84	7243	`F`	706	0.24	6070

Figure 7.2: Experiment 1 (identical workflows). Top: comparison of the makespans; middle: unfairness degree η_u; bottom: makespan standard deviation σ_m, slowdown standard deviation σ_s and unfairness μ.

where no information about task execution time and future task submission is known, the fairness mechanism is not able to give higher priorities to `SimuBloch` tasks in advance. Despite that, the fairness mechanism speeds up `SimuBloch` executions up to a factor of 2.9, reduces task average wait time up to factor of 4.4 and reduces slowdown up to a factor of 5.9.

Experiment 3 (*different workflows*). Figure 7.4 shows slowdown, unfairness degree, unfairness μ and slowdown standard deviation σ_s for the 4 repetitions. `Fairness` slows down `GATE` while it speeds up all other workflows. This is because `GATE` is the longest and the first to be submitted; in `No-Fairness`, it is favored by resource allocation to the detriment of other workflows. The evolution of η_u is similar to Experiments 1 and 2. σ_s is reduced up to a factor of 3.8 and unfairness up to a factor of 1.9.

In all 3 experiments, fairness optimization takes time to begin because the method needs to acquire information about the applications which are totally unknown when a workflow is launched. We could think of reducing the time of this information-collecting phase, e.g. by designing initialization strategies maximizing information discovery, but it could not be totally removed. Currently, the method works best for applications with a lot of short tasks because the first few tasks can be used for initialization, and optimization can be exploited for the remaining



	Repetition 1		Repetition 2		Repetition 3		Repetition 4	
	σ_s	$\mu(s)$	σ_s	$\mu(s)$	σ_s	$\mu(s)$	σ_s	$\mu(s)$
`NF`	94.88	17269	100.05	16048	87.93	11331	213.60	28190
`F`	15.95	9085	42.94	12543	57.62	7721	76.69	21355

Figure 7.3: Experiment 2 (very short execution). Top: comparison of the makespans; middle: unfairness degree η_u; bottom: unfairness μ and slowdown standard deviation.

tasks. The worst-case scenario is a configuration where the number of available resources stays constant and equal to the number of tasks in the first submitted workflow: in this case, no action could be taken until the first workflow completes, and the method would not do better than first-come-first-served. Pre-emption of running tasks should be considered to address that.

7.4 Conclusion

We presented a method to address unfairness among workflow executions in an online and non-clairvoyant environment. We defined a novel metric η_u quantifying unfairness based on the fraction of pending work in a workflow. It compares workflow activities based on their ratio of queuing tasks, their relative durations, and the performance of resources where tasks are running. Performance is defined from the variability of task duration in the activity: good performance is assumed to lead to homogeneous task durations. To separate fair configurations from unfair ones, a threshold on η_u was determined from platform traces. Unfair configurations are handled by increasing the priority of pending tasks in the least performing workflows. This is done by estimating the number of running tasks that these workflows should have to bring η_u under the threshold value.

The method was implemented in the MOTEUR workflow engine and deployed on EGI

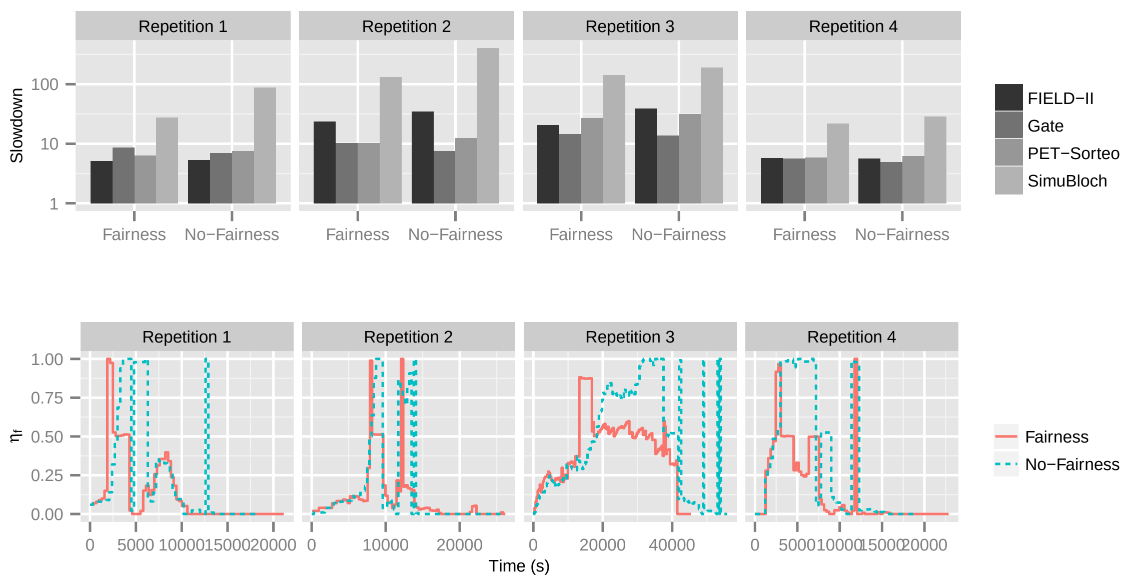
Run	Type	m (secs)	$\bar{w}$ (secs)	s
1	No-Fairness	27854	18983	196.15
	Fairness	9531	4313	38.43
2	No-Fairness	27784	19105	210.48
	Fairness	13761	10538	94.25
3	No-Fairness	14432	13579	182.68
	Fairness	9902	8145	122.25
4	No-Fairness	51664	47591	445.38
	Fairness	38630	27795	165.79

Table 7.3: Experiment 2: `SimuBloch`'s makespan, average wait time and slowdown.

with the DIRAC resource manager. We tested it on four applications extracted from VIP. Three experiments were conducted, to evaluate the capability of the method to improve fairness (*i*) on identical workflows, (*ii*) on workflow sets containing a very short execution and (*iii*) on different workflows. In all cases, results showed that our method can very significantly reduce the standard deviation of the slowdown, and the average value of our metric η_u.

The study presented in this chapter is a step in our attempt to control computing platforms where very little is known about applications and resources, and where situations change over time. We believe that results of this chapter are the first ones presented to control fairness in such conditions which are often met in production platforms. Future work could include task pre-emption in the method, and evaluate more accurately the influence of the relative task duration ($T_{i,j}$) and of the performance factor ($P_{i,j}$).

In the next chapter, we present general conclusions and highlight perspectives of this thesis.



	Repetition 1		Repetition 2		Repetition 3		Repetition 4	
	σ_s	μ (s)	σ_s	μ (s)	σ_s	μ (s)	σ_s	μ (s)
`NF`	40.29	4443	192.58	4004	81.81	25173	11.61	6613
`F`	10.52	2689	58.83	2653	60.56	18537	8.10	3303

Figure 7.4: Experiment 3 (different workflows). Top: comparison of the slowdown; middle: unfairness degree η_u; bottom: unfairness μ and slowdown standard deviation.

Part III

Conclusions

Chapter 8

Conclusion and perspectives

Contents

8.1 Contributions summary

In this manuscript, we addressed the autonomic management of workflow executions on science gateways in an online and non-clairvoyant environment. In the first part of this thesis, we introduced the design of a science gateway and its main components, and then we presented a workload archive that provides fine-grained information about application executions. In the second part, we introduced our general self-healing mechanism, based on the MAPE-K loop, to cope with operational incidents of workflow executions. Then, we presented the application of our self-healing method to seven simple incidents related to input and output data transfers errors, and application execution errors. We also show the application of our method to handle late task executions, task granularities, and unfairness among workflow executions. These contributions are summarized hereafter:

Chapter 2: A science-gateway for workflow executions on grids. In this chapter, we introduced the Virtual Imaging Platform (VIP), an openly-accessible online science-gateway for medical imaging simulation, that provides access to distributed computing and storage resources. The chapter provided a complete overview of VIP architecture for workflow execution presenting employed methods and techniques. The platform currently has 441 registered users from 49 countries who consumed 379 years of CPU time since January 2011. We also introduced issues and limitations related to the execution infrastructure and to the adoption of robot certificates. Manual interventions can still deliver fair quality of service, but it is expensive and affordable only for small production systems. VIP was the platform used in this manuscript for the development and evaluation of our self-healing mechanism and its applications.

Chapter 3: A science-gateway workload archive. In this chapter, we presented a science-gateway model of workload archive containing fine-grained information about users, pilot jobs, task sub-steps, bag of tasks, and workflow executions. This level of information is not found in common workload archives obtained at the infrastructure-level. The workload archive provides the historical information for our self-healing process. It is composed by traces collected from the Virtual Imaging Platform in 2011/2012, which consist of 2,941 workflow executions, 339,545 pilot jobs, 680,988 tasks, and 112 users that consumed about 76 CPU years. Traces are available to the community in the Grid Observatory. Results show that science-gateway workload archives can detect workload wrapped in pilot jobs, improve user identification, give information on distributions of data transfer times, make bag-of-task detection accurate, and retrieve characteristics of workflow executions. Some limits are also identified.

Chapter 4: A self-healing process for workflow executions. In this chapter, we presented our method for autonomous detection and handling of operational incidents on workflow executions. No strong assumption is made on the task duration or resource characteristics and incident degrees are measured with metrics that can be computed online. We made the hypothesis that incident degrees were quantified into distinct levels, and incident levels are associated online to action sets. Action sets are selected based on the degree of their associated incident level and on confidence of association rules determined from execution history. This method is the basis for the incidents addressed in the remaining chapters. We also presented the application of the self-healing method to seven simple incidents related to input and output transfer errors, and application execution errors. Experimental results obtained in VIP show that the proposed method properly detects and handles recoverable and unrecoverable errors.

Chapter 5: Handling blocked activities. In this chapter, we presented two methods to cope with the long-tail effect incident. The first one identifies blocked activities by detecting decreases of the derivative along time of the number of completed tasks, and the second identifies blocked activities as the ones whose tasks are performing worse than the median of already completed tasks. Experimental results show that both methods properly detect blocked activities, speed up workflow executions up to a factor of 4.5, and reduce resource consumption up to 35% and 75%, respectively. The second method is currently used in production by VIP, and already healed more than 1,400 workflow executions since August 2012.

Chapter 6: Optimizing task granularity. In this chapter, we presented a method to optimize task granularity in distributed workflow executions. We defined a metric for online determination of task fineness based on queue waiting time and estimated data transfer time of shared input data, as well as a metric for online determination of task coarseness based on the ratio of the number of queued tasks related to the number of running tasks. On one hand, task grouping is triggered when resources are scarce and tasks are considered too fine. On the other hand, task ungrouping is triggered when the number of available resources increases. Experimental results, obtained with 3 workflow activities deployed on EGI, show that the grouping process yields speed-ups of about 2.5 when the amount of available resources is constant and that the use of de-grouping yields speed-ups of 2 when resources progressively appear.

Chapter 7: Controlling fairness among workflow executions. In this chapter, we presented a method to address unfairness among workflow executions when the number of available resources is scarce. We defined a metric to quantify unfairness based on the ratio of queuing tasks, their relative durations, and the performance of resources where tasks are running. Performance is defined from the variability of task duration in the activity: good performance is assumed to lead to homogeneous task durations. Results show that our technique reduces slowdown variability by a factor of 3 to 7 compared to first-come-first-served, and show that our method quickly detects unfairness, and performs actions to fairly distribute the load among the available resources.

8.2 Concluding remarks and future directions

The self-healing method proposed in this thesis demonstrated its effectiveness to handle operational incidents on workflow executions. The use of a MAPE-K loop is fundamental to achieve a fair quality of service by using control loops that constantly perform online monitoring, analysis, and execution of a set of curative actions.

Although we showed the application of the self-healing method in a medical imaging science-gateway using a grid infrastructure, the method is general enough to be used by other platforms and infrastructures. For instance, in [Ferreira da Silva et al., 2013d] we use a similar approach to estimate workflow task needs such as runtime, disk usage, and memory consumption, using a different workflow engine (Pegasus WMS [Deelman et al., 2005]) from the one used in this manuscript, a workflow from the astronomy field (Montage [Berriman et al., 2004]), and a cloud infrastructure.

Some limitations are also identified. For instance, the method needs to acquire information about the applications which are completely unknown when a workflow is launched. In Chapter 5, this limitation delays the decision to replicate a task; at least two tasks should be finished to estimate the median durations of each phase. The same delay is observed in Chapter 6, where tasks are grouped once an estimation of the duration is available. In Chapter 7, the relative observed duration parameter also depends on task duration estimations, thus the metric does not consider this parameter while the estimations are not available. One approach to circumvent this issue, could be to initialize such estimations according to observed distributions of these values, adjusting the estimations along the workflow execution.

In Chapter 4, we also presented the main instances involved in a workflow execution of a science-gateway. In this manuscript, we considered seven of these instances showed in Figure 4.1. Our self-healing method can be easily extended to handle other incidents from different instances of a science-gateway, provided that they can be quantified online by a metric ranging from 0 to 1, and they have an action set to handle the incident. For instance, a possible extension would be a method to address data management operational issues, such as data placement, availability, and transfer. A control loop method could be used to constantly monitor data availability and the efficiency of transfer operations, to take decisions of whether to replicate a file, or schedule a task to a specific resource.

In the remainder of this section, we present particular perspectives identified along the development of this thesis. Some perspectives presented hereafter come from limitations of our proposed methods, and some come from novel research directions inferred by the results presented in this manuscript.

Mode detection automation. As presented in Chapter 4, incident degrees are quantified in discrete incident levels separated by a threshold. We determine the threshold value of an incident degree by examining execution traces. In this manuscript, threshold are either determined by visual mode clustering, or by using K-Means. In Section 4.4 of Chapter 4, we showed that visual mode detection gives similar threshold values when compared to K-Means, and it gives better classification when an incident does not happen exhaustively. However, visual mode detection is affordable only when addressing a small number of incidents, and the historical information is static. In an autonomic system, the historical information may be dynamic, so that threshold values vary. Therefore, research directions arise in developing automated techniques to automatically detect variation on threshold values, and update the autonomic system accordingly. Besides, the technique should be computable online. For instance, a method to automatic detect unimodal and bimodal thresholds on histograms of image segmentation was proposed in [Ng, 2006]. The idea of the method is to select a threshold value that has a small probability of occurrence, and also maximize the between group variance. This method could be extended to detect multi-modal histograms.

Time-windowed historical information. Operational incidents addressed in this manuscript were identified along the 16 months of the workflow executions trace presented in Chapter 3. However, incidents may be time related, i.e. errors may be restricted to a specific time span. For instance, a site may have low performance because of temporary network glitches; a software update may introduce failures in the system, but once fixed the incident does not happen again. Besides, user's behavior may change, which can shift the system to different states.

Determining the size of the historical window to be used is a challenging research subject. One approach could be to use machine learning techniques to learn user behaviors and resource characteristics, and derive models, so that the size of the window could be automatically determined. Nonetheless, this solution is usually cumbersome and not applicable in production [Kearns, 1990]. Another approach could be to analyze the histogram of incidents, attempting to detect the non-ocurrence of incidents.

Optimization of the incident selection method. In Section 4.2 of Chapter 4, we showed that our self-healing method uses successive roulette wheel selections based on the degree of an incident to select the one to be handled. We adopted this strategy due to its simplicity and efficiency to be computed. However, there is no mechanism to prevent an incident to be successively selected, thus the mechanism can be stalled trying to handle an incident while the cause may be another incident. Markov chain with memory appears as an alternative to roulette wheel selection. A Markov chain is a mathematical system that undergoes transitions from one state to another, between a finite or countable number of possible states. In a Markov chain with memory, predictions for future state transitions are dependent of past state transitions. In this model, incidents are modeled as states, and transitions represent incident degrees.

Sensitivity analysis of parameters. Our metrics defined in Chapters 6 and 7, are composed by several parameters, such as the median transfer time of shared input data and task queueing time in the fineness control (Section 6.2.1), and the relative task duration and the performance factor in the fairness control (Section 7.2). A sensitivity analysis would evaluate the influence of such parameters on the metrics.

Workflow workload archive. Although the science-gateway workload archive model, presented in Chapter 3, gathers enough fine-grained information for the incidents addressed in the chapters of the Part II of this manuscript, it still does not embrace all characteristics inherent to a workflow execution. For instance, task and data dependencies are not modeled, as well as the workflow structure itself. A repository of workflow workloads would be beneficial for both infrastructure providers and researchers: infrastructure providers can use such information to develop workload and user behavior models, while researchers may use to evaluate scheduling algorithms, and develop task and workflow execution models for autonomic management.

Part IV

APPENDIX

Appendix A

Applications description

In this appendix we describe the applications used in the experiments included in this manuscript. Applications were extracted from the Virtual Imaging Platform (see Chapter 2).

A.1 FIELD-II

FIELD-II [Jensen and Svendsen, 1992] is a program for simulating ultrasound transducer fields and ultrasound imaging using linear acoustics. It consists of 3 activities: ExecutionField, Merge, and ReconstructSectorial (Figure A.1). ExecutionField has 122 tasks which simulate the radio-frequency (RF) lines involved in the simulation. It is a data-intensive activity where invocations use from a few seconds to some 15 minutes of CPU time on resources of the biomed VO; it transfers 208 MB of input data and outputs about 40 KB of data; the median transfer time of the input data shared among all tasks in the activity ranges from 40% to 60% of the execution time. Once all lines are simulated, an RF matrix is assembled by the Merge activity, and the final image is reconstructed by activity ReconstructSectorial.

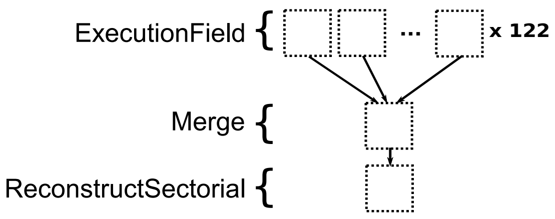


Figure A.1: FIELD-II workflow.

Experiments conducted in Chapters 4, 5, and 6 only consider the execution of the ExecutionField activity, because the Merge and ReconstructSectorial activities consist of a single task each, then incidents cannot be measured. Experiments performed in Chapter 7 consider the whole workflow execution.

A.2 Mean-Shift

`Mean-Shift` [Comaniciu and Meer, 2002] is an image processing technique used to implement filtering, clustering, and segmentation in a *d*-dimensional space. It has 250 CPU-intensive tasks of an image filtering application (Figure A.2). Task CPU time ranges from a few minutes up to one hour; input data size is about 182 MB and output is less than 1 KB.

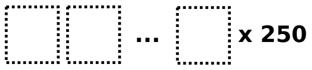


Figure A.2: Mean-Shift workflow.

A.3 SimuBloch

The simulator `SimuBloch` [Cao et al., 2012] is made for a fast simulation of MRIs based on Bloch equation. It is a very short activity made of 25 concurrent tasks (Figure A.3); task CPU time is of a few seconds; input data size is about 15 MB and output is less than 5 MB; the median transfer time of the input data shared among all tasks in the activity is about 90% of the execution time.

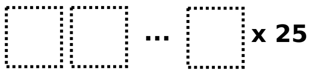


Figure A.3: SimuBloch workflow.

A.4 PET-Sorteo

`PET-Sorteo` [Reilhac et al., 2005] is a Monte Carlo-based simulation platform designed to generate realistic PET (positron emission tomography) images. It is implemented as a doubled-diamond-shaped workflow (Figure A.4) where each fork pattern has 80 concurrent tasks. Tasks from the activity `Singles` consume about 10 minutes of CPU time, while `Emission` tasks are of about 2 CPU minutes; input data size is about 20 MB and output is about 50 MB; the median transfer time of the input data shared among all tasks in the activity `Emission` ranges from 50% to 80% of the execution time.

Experiments conducted in Chapter 6 only consider the execution of the `Emission` activity, because it is the only activity where the granularity of tasks are too fine. Experiments performed in Chapter 7 consider the whole workflow execution.

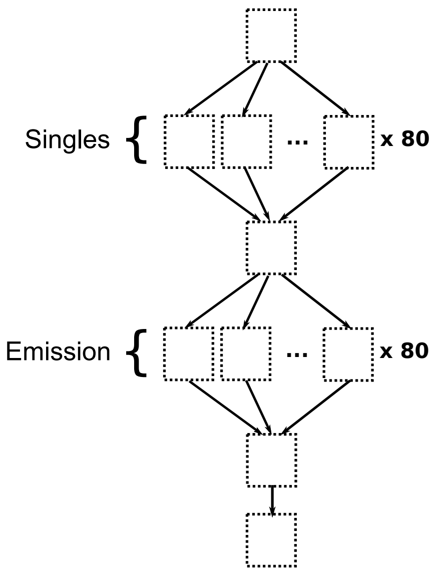


Figure A.4: PET-Sorteo workflow.

A.5 GATE

`GATE` [Jan et al., 2011] is a Geant4-based open-source software to perform nuclear medicine simulations, especially for TEP and SPECT imaging, as well for radiation therapy. It consists of 100 CPU-intensive tasks ranging from a few minutes up to one hour (Figure A.5). Each task transfers about 115 MB of input data and outputs 40 MB of data. GATE is available in VIP through the GateLab system [Camarasu-Pop et al., 2013].

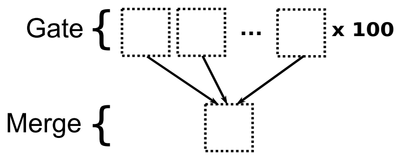


Figure A.5: GATE workflow.

Experiments conducted in Chapter 7 does not consider the execution of the `Merge` activity, because it is a very long data-intensive activity which leads to significant execution errors. Cope with these errors are out of the scope of this thesis, and they are addressed in works such as [Camarasu-Pop et al., 2013].

Bibliography

[fis, 2013] (2013). Paul Fishwick: Introduction to computer simulations. http://www.cise.ufl.edu/~fishwick/introsim. Accessed: 07/16/2013.

[Agrawal et al., 1993] Agrawal, R., Imielinski, T., and Swami, A. (1993). Mining Association Rules between Sets of Items in Large Databases. pages 207–216.

[Altintas et al., 2004] Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., and Mock, S. (2004). Kepler: an extensible system for design and execution of scientific workflows. In *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*, pages 423–424.

[Anderson, 2004] Anderson, D. P. (2004). Boinc: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, GRID '04, pages 4–10, Washington, DC, USA. IEEE Computer Society.

[Ang et al., 2009] Ang, T., Ng, W., Ling, T., Por, L., and Liew, C. (2009). A bandwidth-aware job grouping-based scheduling on grid environment. *Information Technology Journal*, 8:372–377.

[Arabnejad and Barbosa, 2012] Arabnejad, H. and Barbosa, J. (2012). Fairness resource sharing for dynamic workflow scheduling on heterogeneous systems. In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pages 633 –639.

[Ardizzone et al., 2011] Ardizzone, V., Barbera, R., Calanducci, A., Fargetta, M., Ingrà, E., La Rocca, G., Monforte, S., Pistagna, F., Rotondo, R., and Scardaci, D. (2011). A european framework to build science gateways: architecture and use cases. In *2011 TeraGrid Conference: Extreme Digital Discovery*, pages 43:1–43:2, New York. ACM.

[Balderrama et al., 2012] Balderrama, J., Huu, T., and Montagnat, J. (2012). Scalable and resilient workflow executions on production distributed computing infrastructures. In *Parallel and Distributed Computing (ISPDC), 2012 11th International Symposium on*, pages 119–126.

[Barbera et al., 2011] Barbera, R., Brasileiro, F., Bruno, R., Ciuffo, L., and Scardaci, D. (2011). Supporting e-science applications on e-infrastructures: Some use cases from latin america. In *Grid Computing*, Computer Communications and Networks, pages 33–55.

[Barbera et al., 2009] Barbera, R., Donvito, G., Falzone, A., La Rocca, G., Milanesi, L., Maggi, G., and Vicario, S. (2009). The genius grid portal and robot certificates: a new tool for e-science. *BMC Bioinformatics*, 10(Suppl 6):S21.

[Basumallik et al., 2007] Basumallik, A., Zhao, L., Song, C. X., Sriver, R. L., and Huber, M. (2007). A community climate system modeling portal for the teragrid. In *Proceedings of the TeraGrid 2007 Conference*.

[Bell et al., 2003] Bell, W. H., Cameron, D. G., Carvajal-Schiaffino, R., Millar, A. P., Stockinger, K., and Zini, F. (2003). Evaluation of an economy-based file replication strategy for a data grid. In *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*, page 661.

[Ben-Yehuda et al., 2012] Ben-Yehuda, O., Schuster, A., Sharov, A., Silberstein, M., and Iosup, A. (2012). Expert: Pareto-efficient task replication on grids and a cloud. In *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 167–178.

[Berriman et al., 2004] Berriman, G. B., Deelman, E., Good, J. C., Jacob, J. C., Katz, D. S., Kesselman, C., Laity, A. C., Prince, T. A., Singh, G., and Su, M.-H. (2004). Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand. 5493:221–232.

[Brasileiro et al., 2011] Brasileiro, F., Gaudencio, M., Silva, R., Duarte, A., Carvalho, D., Scardaci, D., Ciuffo, L., Mayo, R., Hoeger, H., Stanton, M., Ramos, R., Barbera, R., Marechal, B., and Gavillet, P. (2011). Using a simple prioritisation mechanism to effectively interoperate service and opportunistic grids in the eela-2 e-infrastructure. *Journal of Grid Computing*, 9:241–257.

[Caan et al., 2012] Caan, M., Shahand, S., Vos, F., van Kampen, A., and Olabarriaga, S. (2012). Evolution of grid-based services for diffusion tensor image analysis. *Future Generation Computer Systems*, 28(8):1194 – 1204. Including Special sections SS: Trusting Software Behavior and SS: Economics of Computing Services.

[Camarasu-Pop et al., 2011] Camarasu-Pop, S., Glatard, T., Benoit-Cattin, H., and Sarrut, D. (2011). *Enabling Grids for GATE Monte-Carlo Radiation Therapy Simulations with the GATE-Lab*.

[Camarasu-Pop et al., 2013] Camarasu-Pop, S., Glatard, T., Ferreira da Silva, R., Gueth, P., Sarrut, D., and Benoit-Cattin, H. (2013). Monte carlo simulation on heterogeneous distributed systems: A computing framework with parallel merging and checkpointing strategies. *Future Generation Computer Systems*, 29(3):728 – 738. Special Section: Recent Developments in High Performance Computing and Security.

[Cao et al., 2012] Cao, F., Commowick, O., Bannier, E., FerrÃl', J.-C., Edan, G., and Barillot, C. (2012). Mri estimation of t1 relaxation time using a constrained optimization algorithm. In Yap, P.-T., Liu, T., Shen, D., Westin, C.-F., and Shen, L., editors, *Multimodal Brain Image Analysis*, volume 7509 of *Lecture Notes in Computer Science*, pages 203–214. Springer Berlin Heidelberg.

[Cappello et al., 2005] Cappello, F., Le Mahec, G., Dayde, M., Desprez, F., Jegou, Y., Primet, P., Jeannot, E., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Quetier, B., and Richard, O. (2005). Grid'5000: a large scale and highly reconfigurable grid experimental testbed. In *Grid Computing, 2005. The 6th IEEE/ACM International Workshop on*, pages 8 pp.–.

[Caron et al., 2002] Caron, E., Desprez, F., Lombard, F., Nicod, J.-M., Philippe, L., Quinson, M., and Suter, F. (2002). A scalable approach to network enabled servers (research note). In *Proceedings of the 8th International Euro-Par Conference on Parallel Processing*, Euro-Par '02, pages 907–910, London, UK, UK. Springer-Verlag.

[Casanova, 2006] Casanova, H. (2006). On the harmfulness of redundant batch requests. *International Symposium on High-Performance Distributed Computing*, 0:255–266.

[Casanova et al., 2010] Casanova, H., Desprez, F., and Suter, F. (2010). On cluster resource allocation for multiple parallel task graphs. *J. of Par. and Dist. Computing*, 70(12):1193 – 1203.

[Chen et al., 2013] Chen, W., Ferreira da Silva, R. Deelman, E., and Sakellariou, R. (2013). Balanced task clustering in scientific workflows. In *e-Science 2013 9th IEEE International Conference on*, page to appear.

[Christodoulopoulos et al., 2008] Christodoulopoulos, K., Gkamas, V., and Varvarigos, E. (2008). Statistical analysis and modeling of jobs in a grid environment. *Journal of Grid Computing*, 6:77–101.

[Cirne et al., 2006] Cirne, W., Brasileiro, F., Andrade, N., Costa, L., Andrade, A., Novaes, R., and Mowbray, M. (2006). Labs of the world, unite!!! *Journal of Grid Computing*, 4(3):225–246.

[Cirne et al., 2007] Cirne, W., Brasileiro, F., Paranhos, D., Goes, L., and Voorsluys, W. (2007). On the Efficacy, Efficiency and Emergent Behavior of Task Replication in Large Distributed Systems. *Parallel Computing*, 33:213–234.

[Comaniciu and Meer, 2002] Comaniciu, D. and Meer, P. (2002). Mean Shift: A Robust Approach Toward Feature Space Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619.

[De Craene et al., 2013] De Craene, M., Marchesseau, S., Heyde, B., Gao, H., Alessandrini, M., Bernard, O., Piella, G., Porras, A., Saloux, E., Tautz, L., Hennemuth, A., Prakosa, A., Liebgott, H., Somphone, O., Allain, P., Ebeid, S., Delingette, H., Sermesant, M., and D'hooge, J. (2013). 3d strain assessment in ultrasound (straus): A synthetic comparison of five tracking methodologies. *Medical Imaging, IEEE Transactions on*, PP(99):1–1.

[De Jong, 1975] De Jong, K. A. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, Ann Arbor, MI, USA. AAI7609381.

[Deelman et al., 2005] Deelman, E., Singh, G., hui Su, M., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G. B., Good, J., Laity, A., Jacob, J. C., and Katz, D. S. (2005). Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming Journal*, 13(3):219–237.

[Elghirani et al., 2008] Elghirani, A. H., Subrata, R., and Zomaya, A. Y. (2008). A proactive non-cooperative game-theoretic framework for data replication in data grids. In *8th IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, page 433.

[Ellert et al., 2007] Ellert, M., Grønager, M., Konstantinov, A., Kónya, B., Lindemann, J., Livenson, I., Nielsen, J. L., Niinimäki, M., Smirnova, O., and Wäänänen, A. (2007). Advanced resource connector middleware for lightweight computational grids. *Future Gener. Comput. Syst.*, 23(2):219–240.

[Erwin, 2002] Erwin, D. W. (2002). Unicore—a grid computing environment. *Concurrency and Computation: Practice and Experience*, 14(13-15):1395–1410.

[Fahringer et al., 2005a] Fahringer, T., Prodan, R., Duan, R., Nerieri, F., Podlipnig, S., Qin, J., Siddiqui, M., Truong, H.-L., Villazon, A., and Wieczorek, M. (2005a). Askalon: A grid application development and computing environment. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, GRID '05, pages 122–131, Washington, DC, USA. IEEE Computer Society.

[Fahringer et al., 2005b] Fahringer, T., Qin, J., and Hainzer, S. (2005b). Specification of grid workflow applications with agwl: an abstract grid workflow language. In *Cluster Computing* *and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, volume 2, pages 676–685 Vol. 2.

[Farkas and Kacsuk, 2011] Farkas, Z. and Kacsuk, P. (2011). P-grade portal: A generic workflow system to support user communities. *Future Generation Computer Systems*, 27(5):454 – 465.

[Ferreira da Silva et al., 2011] Ferreira da Silva, R., Camarasu-Pop, S., Grenier, B., Hamar, V., Manset, D., Montagnat, J., Revillard, J., Balderrama, J. R., Tsaregorodtsev, A., and Glatard, T. (2011). Multi-Infrastructure Workflow Execution for Medical Simulation in the Virtual Imaging Platform. In *HealthGrid 2011*, Bristol, UK.

[Ferreira da Silva and Glatard, 2013] Ferreira da Silva, R. and Glatard, T. (2013). A science-gateway workload archive to study pilot jobs, user activity, bag of tasks, task sub-steps, and workflow executions. In Caragiannis, I., Alexander, M., Badia, R., Cannataro, M., Costan, A., Danelutto, M., Desprez, F., Krammer, B., Sahuquillo, J., Scott, S., and Weidendorfer, J., editors, *Euro-Par 2012: Parallel Processing Workshops (CGWS-2012)*, volume 7640 of *Lecture Notes in Computer Science*, pages 79–88. Springer Berlin Heidelberg.

[Ferreira da Silva et al., 2012] Ferreira da Silva, R., Glatard, T., and Desprez, F. (2012). Self-healing of operational workflow incidents on distributed computing infrastructures. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 318 –325.

[Ferreira da Silva et al., 2013a] Ferreira da Silva, R., Glatard, T., and Desprez, F. (2013a). On-line, non-clairvoyant optimization of workflow activity granularity on grids. In Wolf, F., Mohr, B., and Mey, D., editors, *Euro-Par 2013 Parallel Processing*, volume 8097 of *Lecture Notes in Computer Science*, pages 255–266. Springer Berlin Heidelberg.

[Ferreira da Silva et al., 2013b] Ferreira da Silva, R., Glatard, T., and Desprez, F. (2013b). Self-healing of workflow activity incidents on distributed computing infrastructures. *Future Generation Computer Systems*. In press.

[Ferreira da Silva et al., 2013c] Ferreira da Silva, R., Glatard, T., and Desprez, F. (2013c). Workflow fairness control on online and non-clairvoyant distributed computing platforms. In Wolf, F., Mohr, B., and Mey, D., editors, *Euro-Par 2013 Parallel Processing*, volume 8097 of *Lecture Notes in Computer Science*, pages 102–113. Springer Berlin Heidelberg.

[Ferreira da Silva et al., 2013d] Ferreira da Silva, R., Juve, G., Deelman, E., Glatard, T., and Desprez, F. (2013d). Toward fine-grained online task needs estimation in scientific workflows. In *The 8th Workshop on Workflows in Support of Large-Scale Science (WORKS'13)*. Submitted.

[Forestier et al., 2011] Forestier, G., Marion, A., Benoit-Cattin, H., Clarysse, P., Friboulet, D., Glatard, T., Hugonnard, P., Lartizien, C., Liebgott, H., Tabary, J., and Gibaud, B. (2011). Sharing object models for multi-modality medical image simulation: A semantic approach. In *Computer-Based Medical Systems (CBMS), 2011 24th International Symposium on*, pages 1 –6.

[Foster, 2005] Foster, I. (2005). Globus toolkit version 4: software for service-oriented systems. In *Proceedings of the 2005 IFIP international conference on Network and Parallel Computing*, NPC'05, pages 2–13, Berlin, Heidelberg. Springer-Verlag.

[Foster et al., 2002] Foster, I., Kesselman, C., Nick, J. M., and Tuecke, S. (2002). The physiology of the grid: An open grid services architecture for distributed systems integration. In *Open Service Infrastructure WG*, pages 210–232.

[Foster et al., 2001] Foster, I., Kesselman, C., and Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222.

[Frisoni et al., 2011] Frisoni, G., Redolfi, A., Manset, D., Rousseau, M., Toga, A., and Evans, A. (2011). Virtual imaging laboratories for marker discovery in neurodegenerative diseases. *Nature reviews. Neurology*, 7(8):429–438.

[Gagliardi et al., 2005] Gagliardi, F., Jones, B., Grey, F., Bégin, M.-E., and Heikkurinen, M. (2005). Building an infrastructure for scientific grid computing: status and goals of the egee project. *Phil. Trans. R. Soc. A 15*, 363(1833):1729–1742.

[Germain-Renaud et al., 2011] Germain-Renaud, C., Cady, A., Gauron, P., Jouvin, M., Loomis, C., Martyniak, J., Nauroy, J., Philippon, G., and Sebag, M. (2011). The grid observatory. *IEEE International Symposium on Cluster Computing and the Grid*, pages 114–123.

[Gesing and van Hemert, 2011] Gesing, S. and van Hemert, J., editors (2011). *Concurrency and Computation: Practice and Experience, Special Issue on International Workshop on Portals for Life-Sciences 2009*, volume 23.

[Glatard et al., 2013] Glatard, T., Lartizien, C., Gibaud, B., Ferreira da Silva, R., Forestier, G., Cervenansky, F., Alessandrini, M., Benoit-Cattin, H., Bernard, O., Camarasu-Pop, S., Cerezo, N., Clarysse, P., Gaignard, A., Hugonnard, P., Liebgott, H., Marache, S., Marion, A., Montagnat, J., Tabary, J., and Friboulet, D. (2013). A virtual imaging platform for multi-modality medical image simulation. *Medical Imaging, IEEE Transactions on*, 32(1):110 –118.

[Glatard et al., 2008] Glatard, T., Montagnat, J., Lingrand, D., and Pennec, X. (2008). Flexible and Efficient Workflow Deployment of Data-Intensive Applications on Grids with MO-

TEUR. *International Journal of High Performance Computing Applications (IJHPCA)*, 22(3):347–360.

[Goldchleger et al., 2004] Goldchleger, A., Kon, F., Goldman, A., Finger, M., and Bezerra, G. C. (2004). Integrade: object-oriented grid middleware leveraging the idle computing power of desktop machines. *Concurrency and Computation: Practice and Experience*, 16(5):449–459.

[Grevillot et al., 2012] Grevillot, L., Bertrand, D., Dessy, F., Freud, N., and Sarrut, D. (2012). Gate as a geant4-based monte carlo platform for the evaluation of proton pencil beam scanning treatment plans. *Physics in Medicine and Biology*, 57(13):4223.

[Hirales-Carbajal et al., 2012] Hirales-Carbajal, A., Tchernykh, A., Yahyapour, R., González-García, J. L., Röblitz, T., and Ramírez-Alcaraz, J. M. (2012). Multiple workflow scheduling strategies with user run time estimates on a grid. *Journal of Grid Computing*, 10:325–346.

[Hsu et al., 2011] Hsu, C.-C., Huang, K.-C., and Wang, F.-J. (2011). Online scheduling of workflow applications in grid environments. *Future Generation Computer Systems*, 27(6):860 – 870.

[Huedo et al., 2010] Huedo, E., Montero, R. S., and Llorente, I. (2010). Grid architecture from a metascheduling perspective. *Computer*, 43(7):51–56.

[Ilijasic and Saitta, 2009] Ilijasic, L. and Saitta, L. (2009). Characterization of a Computational Grid as a Complex System. In *Grid Meets Autonomic Computing(GMAC'09)*, pages 9–18.

[Iosup and Epema, 2011] Iosup, A. and Epema, D. (2011). Grid computing workloads: bags of tasks, workflows, pilots, and others. *Internet Computing, IEEE*, 15(2):19 –26.

[Iosup et al., 2007] Iosup, A., Jan, M., Sonmez, O., and Epema, D. (2007). The characteristics and performance of groups of jobs in grids. In *Euro-Par*, pages 382–393.

[Iosup et al., 2008] Iosup, A., Li, H., Jan, M., Anoep, S., Dumitrescu, C., Wolters, L., and Epema, D. H. J. (2008). The grid workloads archive. *Future Gener. Comput. Syst.*, 24(7):672–686.

[Jan et al., 2011] Jan, S., Benoit, D., Becheva, E., Carlier, T., Cassol, F., Descourt, P., Frisson, T., Grevillot, L., Guigues, L., Maigne, L., Morel, C., Perrot, Y., Rehfeld, N., Sarrut, D., Schaart, D., Stute, S., Pietrzyk, U., Visvikis, D., Zahra, N., and Buvat, I. (2011). Gate v6: a major enhancement of the gate simulation platform enabling modelling of ct and radiotherapy. *Phys. in Med. and Biol.*, 56(4):881–901.

[Jensen and Svendsen, 1992] Jensen, J. and Svendsen, N. (1992). Calculation of pressure fields from arbitrarily shaped, apodized, and excited ultrasound transducers. *Ultrasonics, Ferroelectrics and Frequency Control, IEEE Transactions on*, 39(2):262 –267.

[Kacsuk, 2011] Kacsuk, P. (2011). P-grade portal family for grid infrastructures. *Concurr. Comput. : Pract. Exper.*, 23(3):235–245.

[Kacsuk et al., 2012] Kacsuk, P., Farkas, Z., Kozlovszky, M., Hermann, G., Balasko, A., Karoczkai, K., and Marton, I. (2012). Ws-pgrade/guse generic dci gateway framework for a large variety of user communities. *Journal of Grid Computing*, 10:601–630.

[Kandaswamy et al., 2008] Kandaswamy, G., Mandal, A., and Reed, D. (2008). Fault tolerance and recovery of scientific workflows on computational grids. In *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on*, pages 777–782.

[Kearns, 1990] Kearns, M. J. (1990). *The Computational Complexity of Machine Learning*. MIT Press.

[Kephart and Chess, 2003] Kephart, J. and Chess, D. (2003). The vision of autonomic computing. *Computer*, 36(1):41 – 50.

[Kondo et al., 2010] Kondo, D., Javadi, B., Iosup, A., and Epema, D. (2010). The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems. In *CCGrid 2010*, pages 398 –407.

[Korkhov et al., 2009] Korkhov, V. V., Moscicki, J. T., and Krzhizhanovskaya, V. V. (2009). Dynamic workload balancing of parallel applications with user-level scheduling on the grid. *Future Generation Computer Systems*, 25(1):28 – 34.

[Krauter et al., 2002] Krauter, K., Buyya, R., and Maheswaran, M. (2002). A taxonomy and survey of grid resource management systems for distributed computing. *Software Practice and Experience*, 32(2):135–164.

[Leporq et al., 2013] Leporq, B., Camarasu-Pop, S., Davila-Serrano, E. E., Pilleul, F., and Beuf, O. (2013). Enabling 3d-liver perfusion mapping from mr-dce imaging using distributed computing. *Journal of Medical Engineering*, 2013:7.

[Lingrand et al., 2010] Lingrand, D., Montagnat, J., Martyniak, J., and Colling, D. (2010). Optimization of jobs submission on the EGEE production grid: modeling faults using workload. *Journal of Grid Computing (JOGC) Special issue on EGEE*, 8(2):305–321.

[Litke et al., 2007] Litke, A., Skoutas, D., Tserpes, K., and Varvarigou, T. (2007). Efficient task replication and management for adaptive fault tolerance in mobile grid environments. *Future Generation Computer Systems*, 23(2):163 – 178.

[Liu and Liao, 2009] Liu, Q. and Liao, Y. (2009). Grouping-based fine-grained job scheduling in grid computing. In *ETCS '09*, volume 1, pages 556 –559.

[Luckow et al., 2010] Luckow, A., Lacinski, L., and Jha, S. (2010). Saga bigjob: An extensible and interoperable pilot-job abstraction for distributed applications and systems. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 135–144.

[Luckow et al., 2012] Luckow, A., Santcroos, M., Weidner, O., Merzky, A., Maddineni, S., and Jha, S. (2012). Towards a common model for pilot-jobs. In *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*, HPDC '12, pages 123–124, New York, NY, USA. ACM.

[Ma et al., 2013] Ma, J., Liu, W., and Glatard, T. (2013). A classification of file placement and replication methods on grids. *Future Generation Computer Systems*, 29(6):1395 – 1406. Including Special sections: High Performance Computing in the Cloud Resource Discovery Mechanisms for P2P Systems.

[MacQueen, 1967] MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, Berkeley, CA.

[Malik et al., 1994] Malik, D., Mordeson, J. N., and Sen, M. (1994). On Subsystems of a Fuzzy Finite State Machine. *Fuzzy Sets and Systems*, 68(1):83 – 92.

[Marion et al., 2011] Marion, A., Forestier, G., Benoit-Cattin, H., Camarasu-Pop, S., Clarysse, P., da Silva, R., Gibaud, B., Glatard, T., Hugonnard, P., Lartizien, C., Liebgott, H., Specovius, S., Tabary, J., Valette, S., and Friboulet, D. (2011). Multi-modality medical image simulation of biological models with the virtual imaging platform (vip). In *Computer-Based Medical Systems (CBMS), 2011 24th International Symposium on*, pages 1 –6.

[Medernach, 2005] Medernach, E. (2005). Workload analysis of a cluster in a grid environment. In *Job Scheduling Strategies for Parallel Processing*, pages 36–61.

[Missier et al., 2010] Missier, P., Soiland-Reyes, S., Owen, S., Tan, W., Nenadic, A., Dunlop, I., Williams, A., Oinn, T., and Goble, C. (2010). Taverna, reloaded. In Gertz, M., Hey, T., and Ludaescher, B., editors, *SSDBM 2010*, Heidelberg, Germany.

[Montagnat et al., 2010] Montagnat, J., Glatard, T., Reimert, D., Maheshwari, K., Caron, E., and Desprez, F. (2010). Workflow-based comparison of two distributed computing infrastructures. In *Workflows in Support of Large-Scale Science (WORKS), 2010 5th Workshop on*, pages 1–10.

[Montagnat et al., 2009] Montagnat, J., Isnard, B., Glatard, T., Maheshwari, K., and Fornarino, M. B. (2009). A data-driven workflow language for grids based on array programming principles. In *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*, WORKS '09, pages 7:1–7:10, New York, NY, USA. ACM.

[Muthuvelu et al., 2010] Muthuvelu, N., Chai, I., Chikkannan, E., and Buyya, R. (2010). Online task granularity adaptation for dynamic grid applications. In Hsu, C.-H., Yang, L., Park, J., and Yeo, S.-S., editors, *Algorithms and Architectures for Parallel Processing*, volume 6081 of *Lecture Notes in Computer Science*, pages 266–277. Springer Berlin Heidelberg.

[Muthuvelu et al., 2008] Muthuvelu, N., Chai, I., and Eswaran, C. (2008). An adaptive and parameterized job grouping algorithm for scheduling grid jobs. In *Advanced Communication Technology, 2008. ICACT 2008. 10th International Conference on*, volume 2, pages 975 – 980.

[Muthuvelu et al., 2005] Muthuvelu, N., Liu, J., Soe, N. L., Venugopal, S., Sulistio, A., and Buyya, R. (2005). A dynamic job grouping-based scheduling for deploying applications with fine-grained tasks on global grids. In *Proceedings of the 2005 Australasian workshop on Grid computing and e-research - Volume 44*, ACSW Frontiers '05, pages 41–48, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

[Muthuvelu et al., 2013] Muthuvelu, N., Vecchiola, C., Chai, I., Chikkannan, E., and Buyya, R. (2013). Task granularity policies for deploying bag-of-task applications on global grids. *Future Generation Computer Systems*, 29(1):170 – 181. Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures.

[Ng, 2006] Ng, H.-F. (2006). Automatic thresholding for defect detection. *Pattern Recognition Letters*, 27(14):1644 – 1649.

[Ng et al., 2006] Ng, W. K., Ang, T. F., Ling, T. C., and Liew, C. S. (2006). Scheduling framework for bandwidth-aware job grouping-based scheduling in grid computing. *Malaysian Journal of Computer Science*, 19.

[Nilsson et al., 2011] Nilsson, P., Caballero, J., De, K., Maeno, T., Stradling, A., Wenaus, T., and the Atlas Collaboration (2011). The atlas panda pilot in operation. *Journal of Physics: Conference Series*, 331(6):062040.

[N'Takpe and Suter, 2009] N'Takpe, T. and Suter, F. (2009). Concurrent scheduling of parallel task graphs on multi-clusters using constrained resource allocations. IPDPS '09, pages 1–8.

[Oinn et al., 2006] Oinn, T., Greenwood, M., Addis, M., Alpdemir, M. N., Ferris, J., Glover, K., Goble, C., Goderis, A., Hull, D., Marvin, D., Li, P., Lord, P., Pocock, M. R., Senger, M., Stevens, R., Wipat, A., and Wroe, C. (2006). Taverna: lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience*, 18(10):1067–1100.

[Ostermann et al., 2008] Ostermann, S., Prodan, R., Fahringer, T., Iosup, R., and Epema, D. (2008). On the characteristics of grid workflows. In *CoreGRID Symposium - Euro-Par 2008*.

[Pandey et al., 2009] Pandey, S., Voorsluys, W., Rahman, M., Buyya, R., Dobson, J. E., and Chiu, K. (2009). A grid workflow environment for brain imaging analysis on distributed systems. *Concurr. Comput. : Pract. Exper.*, 21(16):2118–2139.

[Plankensteiner et al., 2011] Plankensteiner, K., Montagnat, J., and Prodan, R. (2011). Iwir: a language enabling portability across grid workflow systems. In *Proceedings of the 6th workshop on Workflows in support of large-scale science*, WORKS '11, pages 97–106, New York, NY, USA. ACM.

[Plankensteiner et al., 2009] Plankensteiner, K., Prodan, R., and Fahringer, T. (2009). A new fault tolerance heuristic for scientific workflows in highly distributed environments based on resubmission impact. In *e-Science, 2009. e-Science '09. Fifth IEEE International Conference on*, pages 313–320.

[Prakosa et al., 2013] Prakosa, A., Sermesant, M., Delingette, H., Marchesseau, S., Saloux, E., Allain, P., Villain, N., and Ayache, N. (2013). Generation of synthetic but visually realistic time series of cardiac images combining a biophysical model and clinical images. *Medical Imaging, IEEE Transactions on*, 32(1):99–109.

[Ramakrishnan et al., 2009] Ramakrishnan, L., Koelbel, C., Kee, Y.-S., Wolski, R., Nurmi, D., Gannon, D., Obertelli, G., YarKhan, A., Mandal, A., Huang, T., Thyagaraja, K., and Zagorodnov, D. (2009). Vgrads: enabling e-science workflows on grids and clouds with fault tolerance. In *High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on*, pages 1–12.

[Rehn et al., 2006] Rehn, J., Barrass, T., Bonacorsi, D., Hernandez, J., Semeniouk, I., Tuura, L., and Wu, Y. (2006). Phedex high-throughput data transfer management system. In *Computing in High Energy Physics, CHEP'2006*.

[Reilhac et al., 2005] Reilhac, A., Batan, G., Michel, C., Grova, C., Tohka, J., Collins, D., Costes, N., and Evans, A. (2005). Pet-sorteo: validation and development of database of simulated pet volumes. *Nuclear Science, IEEE Transactions on*, 52(5):1321 – 1328.

[Rogers et al., 2013] Rogers, D., Harvey, I., Huu, T., Evans, K., Glatard, T., Kallel, I., Taylor, I., Montagnat, J., Jones, A., and Harrison, A. (2013). Bundle and pool architecture for multi-language, robust, scalable workflow executions. *Journal of Grid Computing*, pages 1–24.

[Rojas Balderrama et al., 2010] Rojas Balderrama, J., Montagnat, J., and Lingrand, D. (2010). jGASW: A Service-Oriented Framework Supporting High Throughput Computing and Non-functional Concerns. In *IEEE International Conference on Web Services*, ICWS 2010, Miami (FL), USA. IEEE Computer Society.

[Rojas Balderrama et al., 2011] Rojas Balderrama, J., Truong Huu, T., and Montagnat, J. (2011). A comprehensive framework for scientific applications execution on distributed computing infrastructures. In *Rencontres Scientifiques France Grilles 2011*, Lyon, FR.

[Romanus et al., 2012] Romanus, M., Mantha, P. K., McKenzie, M., Bishop, T. C., Gallichio, E., Merzky, A., El Khamra, Y., and Jha, S. (2012). The anatomy of successful ecss projects: lessons of supporting high-throughput high-performance ensembles on xsede. In *Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment: Bridging from the eXtreme to the campus and beyond*, XSEDE '12, pages 46:1–46:9, New York, NY, USA. ACM.

[Sabin et al., 2004] Sabin, G., Kochhar, G., and Sadayappan, P. (2004). Job fairness in non-preemptive job scheduling. In *Proceedings of the 2004 International Conference on Parallel Processing*, ICPP '04, pages 186–194.

[Shahand et al., 2012] Shahand, S., Santcroos, M., Kampen, A., and Olabarriaga, S. (2012). A grid-enabled gateway for biomedical data analysis. *Journal of Grid Computing*, 10:725–742.

[Singh et al., 2008] Singh, G., Su, M.-H., Vahi, K., Deelman, E., Berriman, B., Good, J., Katz, D. S., and Mehta, G. (2008). Workflow task clustering for best effort systems with pegasus. In *Proceedings of the 15th ACM Mardi Gras conference: From lightweight mash-ups to lambda grids: Understanding the spectrum of distributed computing requirements, applications, tools, infrastructures, interoperability, and the incremental adoption of key capabilities*, MG '08, pages 9:1–9:8, New York, NY, USA. ACM.

[Sommerfeld and Richter, 2011] Sommerfeld, D. and Richter, H. (2011). Efficient Grid Workflow Scheduling Using a Two-Tier Approach. In *Proceedings of HealthGrid 2011*, Bristol, UK.

[Soni et al., 2010] Soni, V. K., Sharma, R., and Mishra, M. K. (2010). Grouping-based job scheduling model in grid computing. *World Academy of Science, Engineering and Technology*, 41:781–784.

[Tan et al., 2005] Tan, P.-N., Steinbach, M., and Kumar, V. (2005). *Introduction to Data Mining*. Addison-Wesley.

[Taylor et al., 2007] Taylor, I., Shields, M., Wang, I., and Harrison, A. (2007). The triana workflow environment: Architecture and applications. In Taylor, I., Deelman, E., Gannon, D., and Shields, M., editors, *Workflows for e-Science*, pages 320–339. Springer London.

[Thain et al., 2003] Thain, D., Tannenbaum, T., and Livny, M. (2003). *Condor and the Grid*, pages 299–335. John Wiley & Sons, Ltd.

[Thain et al., 2005] Thain, D., Tannenbaum, T., and Livny, M. (2005). Distributed computing in practice: the condor experience. *Concurrency and Computation: Practice and Experience*, 17(2-4):323–356.

[Tsaregorodtsev et al., 2009] Tsaregorodtsev, A., Brook, N., Ramo, A. C., Charpentier, P., Closier, J., Cowan, G., Diaz, R. G., Lanciotti, E., Mathe, Z., Nandakumar, R., Paterson, S., Romanovsky, V., Santinelli, R., Sapunov, M., Smith, A. C., Miguelez, M. S., and Zhelezov, A. (2009). DIRAC3. The New Generation of the LHCb Grid Software. *Journal of Physics: Conference Series*, 219(6):062029.

[van der Aalst and ter Hofstede, 2005] van der Aalst, W. and ter Hofstede, A. (2005). Yawl: yet another workflow language. *Information Systems*, 30(4):245 – 275.

[von Laszewski et al., 2010] von Laszewski, G., Fox, G. C., Wang, F., Younge, A. J., Kulshrestha, A., Pike, G. G., Smith, W., Voeckler, J., Figueiredo, R. J., Fortes, J., Keahey, K., and Delman, E. (2010). Design of the futuregrid experiment management framework. In *GCE2010 at SC10*, New Orleans. IEEE, IEEE.

[Wang et al., 2012a] Wang, L., Camarasu-Pop, S., Glatard, T., Zhu, Y.-M., and Magnin, I. E. (2012a). Diffusion mri simulation with the virtual imaging platform. In *Journées Scientifiques mésocentres et France Grilles 2012*, Paris, FR.

[Wang et al., 2012b] Wang, L., Zhu, Y., Li, H., Liu, Y., and Magnin, I. (2012b). Multiscale modeling and simulation of the cardiac fiber architecture for dmri. *Biomedical Engineering, IEEE Transactions on*, 59(1):16–19.

[Wieczorek et al., 2008] Wieczorek, M., Hoheisel, A., and Prodan, R. (2008). Taxonomies of the multi-criteria grid workflow scheduling problem. In *Grid Middleware and Services*, pages 237–264. Springer US.

[Yu and Buyya, 2005] Yu, J. and Buyya, R. (2005). A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3(3-4):171–200.

[Zhang et al., 2004] Zhang, X., Zagorodnov, D., Hiltunen, M., Marzullo, K., and Schlichting, R. (2004). Fault-tolerant grid services using primary-backup: feasibility and performance. In *Cluster Computing, 2004 IEEE International Conference on*, pages 105 – 114.

[Zhang et al., 2009] Zhang, Y., Mandal, A., Koelbel, C., and Cooper, K. (2009). Combined fault tolerance and scheduling techniques for workflow applications on computational grids. In *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, pages 244–251.

[Zhao and Sakellariou, 2006] Zhao, H. and Sakellariou, R. (2006). Scheduling multiple DAGs onto heterogeneous systems. IPDPS'06, pages 159–159.

[Zhao et al., 2011] Zhao, X., Hover, J., Wlodek, T., Wenaus, T., Frey, J., Tannenbaum, T., and Livny, M. (2011). Panda pilot submission using condor-g: Experience and improvements. *Journal of Physics: Conference Series*, 331(7):072069.

[Zomaya and Chan, 2004] Zomaya, A. and Chan, G. (2004). Efficient clustering for parallel tasks execution in distributed systems. In *18th IPDPS*, pages 167–174.

FOLIO ADMINISTRATIF

THESE SOUTENUE DEVANT L'INSTITUT NATIONAL DES SCIENCES APPLIQUEES DE LYON

NOM : FERREIRA DA SILVA DATE de SOUTENANCE : 29 Novembre 2013
(avec précision du nom de jeune fille, le cas échéant)

Prénoms : Rafael

TITRE :
A science-gateway for workflow executions:
online and non-clairvoyant self-healing of workflow executions on grids

NATURE : Doctorat Numéro d'ordre : AAAAISALXXXX

Ecole doctorale : École Doctorale InfoMaths (ED 512)

Spécialité : Informatique

RESUME :
Les science-gateways, telles que la Plate-forme d'Imagerie Virtuelle (VIP), permettent l'accès à un grand nombre de ressources de calcul et de stockage de manière transparente. Cependant, la quantité d'informations et de couches intergicielles utilisées créent beaucoup d'échecs et d'erreurs de système. Dans la pratique, ce sont souvent les administrateurs du système qui contrôlent le déroulement des expériences en réalisant des manipulations simples mais cruciales, comme par exemple replanifier une tâche, redémarrer un service, supprimer une exécution défaillante, ou copier des données dans des unités de stockages fiables. De cette manière, la qualité de service fournie est correcte mais demande une intervention humaine importante.
Automatiser ces opérations constitue un défi pour deux raisons. Premièrement, la charge de la plate-forme est en ligne, c'est-à-dire que de nouvelles exécutions peuvent se présenter à tout moment. Aucune prédiction sur l'activité des utilisateurs n'est donc possible. De fait, les modèles, décisions et actions considérés doivent rester simples et produire des résultats pendant l'exécution de l'application. Deuxièmement, la plate-forme est non-clairvoyante à cause du manque d'information concernant les applications et ressources en production. Les ressources de calcul sont d'ordinaire fournies dynamiquement par des grappes hétérogènes, des clouds ou des grilles de volontaires, sans estimation fiable de leur disponibilité ou de leur caractéristiques. Les temps d'exécution des applications sont difficilement estimables également, en particulier dans le cas de ressources de calculs hétérogènes.
Dans ce manuscrit, nous proposons un mécanisme d'auto-guérison pour la détection autonome et traitement des incidents opérationnels dans les exécutions des chaînes de traitement. Les objets considérés sont modélisés comme des automates finis à états flous (FuSM) où le degré de pertinence d'un incident est déterminé par un processus externe de guérison. Les modèles utilisés pour déterminer le degré de pertinence reposent sur l'hypothèse que les erreurs, par exemple un site ou une invocation se comportant différemment des autres, sont rares. Le mécanisme d'auto-guérison détermine le seuil de gravité des erreurs à partir de l'historique de la plate-forme. Un ensemble d'actions spécifiques est alors sélectionné par règle d'association en fonction du niveau d'erreur.

MOTS-CLES : Détection autonome et traitement des erreurs, exécutions des chaînes de traitement, systèmes distribués en production.

Laboratoire (s) de recherche : Laboratoire CREATIS – CNRS UMR 5220 – INSERM U1044

Directeur de thèse: Frédéric DESPREZ, Tristan GLATARD

Président de jury :

Composition du jury :
Eric RUTTEN, Frédéric DESPREZ, Tristan GLATARD, Silvia D. OLABARRIAGA, Johan MONTAGNAT, Hugues BENNOIT-CATTIN, Martin QUINSON