MAD resource allocation

Allocation de ressources en Domaines Administratifs Multiples

---

Xavier Gréhant
Isabelle Demeure

2010D013

Avril 2010

Département Informatique et Réseaux
Groupe S3 : Systèmes, Logiciels, Services

Telecom ParisTech

MAD Resource Allocation

Allocation de ressources en Domaines Administratifs Multiples

Xavier Gréhant[1] [2] Isabelle Demeure[1]

[1] with Institut Telecom, Telecom ParisTech, CNRS UMR 5141, LTCI
[2] with CERN openlab and supported by HP Labs

Abstract

Computing grids span multiple administrative domains. As a consequence, the objectives of participants to resource allocation differ. Resource users are interested in processing performance, while resource providers are concerned with energy consumption and obstruction to maintenance and internal use. In addition, strategies differ even to optimize the same metric. On the basis of a new model for resource allocation, this paper proposes a few hypotheses under which the different objectives can be solved independently. They translate into a new architectural design, Symmetric Mapping.

Les grilles de calcul regroupent des domaines administratifs multiples. Dans ces environnements, les participants à l'allocation de ressource ont des objectifs différents. Les utilisateurs de ressources sont généralement intéressés par la rapidité d'exécution tandis que les fournisseurs de ressources ont le souci de la consommation d'énergie, de la liberté d'effectuer les opérations nécessaires de maintenance des serveurs, et de la liberté d'utiliser les ressources en interne aux moments opportuns. En outre plusieurs stratégies coexistent pour optimiser la même métrique. Sur la base d'un nouveau modèle de l'allocation de ressources, nous proposons des hypothèses qui guarantissent que différents objectifs peuvent être atteints indépendamment. Ces hypothèses se traduisent en un nouveau pattern de conception d'architectures, Symmetric Mapping.

Contents

Chapter 1

Introduction

Computing grids aggregate resources from multiple institutions that support scientific projects of common interest. The placement of a task on a server involves a resource user and a resource provider. User and provider are bound in a contract, explicit or tacit, that motivates their co-operation. However, since in general they report to different institutions, they have diverging goals.

- Users are generally interested in computing speed. Computing speed has different definitions for a single large scale application and for a collaboration of individuals who constantly execute new tasks. User concerns also include adaptation of software configuration, optimization of computing throughput, compliance with task priorities, minimization of resource oversubscription if subscription is limited [CGV07], and prevention of resource undersubscription.
- Providers favor minimization of resource supply costs, i.e. minimization of server activity and power consumption [CIL+07, JB07], optimization of cooling efficiency [BF07], minimization of obstruction to servers maintenance, to local use and local policies.

These objectives may conflict in the resource allocation process. Initially based on voluntary collaborations, grids have not focussed on mitigating conflicts. As a result, compromises in performance and flexibility are made on both sides. We propose to start with the separation of concerns. In this aim, we introduce an architectural pattern, Symmetric Mapping.

Design patterns were originally introduced for object oriented programming by the *Gang of Four* in OOPSLA meeting [GHJV95]. They identify best practices to solve recurrent software design problems. They are defined in terms of relationships between objects that compose the software. Similarly, in distributed systems, architectural patterns define component structures to solve recurrent architectural problems [BMR+96b]. Famous examples include *Model-view-controller* that isolates application logic from interface [Gre07], and *Peer-to-peer* that decentralizes control and resources to all elements in a system, making them functionally equivalent [CvR05].

Symmetric Mapping has his foundations in a new model of resource allocation. The model is an analogy with the Multiple Administrative Domains (MAD) model developed for the study of fault tolerance in distributed systems [AAC+05]. In a MAD distributed system, participants' behaviors are not controlled. Instead, the system builds on assumptions. Namely, a participant is rational or not, altruist or egoistic, or byzantine (entirely unpredictable). In our case, we assume that participants are selfish and rational, and the objective is to satisfy each of them independently. To the best of our knowledge, our formalism is the first to use MADs principles for resource allocation. Symmetric Mapping was initially described in [GD09].

Section 2 details previous work. Section 3 identifies several participants objectives. Section 4 introduces Symmetric Mapping. Section 5 simulate its implementation on synthetic examples. The rest of the paper presents the model. Section 6 introduces the fundamentals. Section 7 formalizes the problem. Section 8 formally establishes separation of responsibilities as a solution.

Chapter 2

Previous work

In this paper, we propose a new architectural pattern with its foundations in a new formal model. Relevant previous work includes the identification of existing architectural patterns in the same area, and the existing formal models that underly their emergence. This discussion is largely based on a historical perspective on resource allocation approaches in grids [GDJ08]. This section synthesizes the observed patterns and the underlying models.

Models based on queuing theory and game theory are considered, as well as models that formalize the notion of resource containment. Existing architectural patterns are organized around Metascheduling, Late Binding, and economic patterns.

2.1 Queuing models

The study of grid resource allocation is traditionally based on queuing theory [CK88, All90, MAS+99, BBC+04, Van08]. So are grid systems in production today [GDJ08]. The model we introduce in this paper departs from queuing theory and allows to consider resources that do not use queues, and events that occur outside of queues, including co-allocation and live-migration. This fine-grained, dynamic view is necessary to capture the level at which users and providers incentives collide or settle.

Matchmaking or resource brokering systems use requirements and preferences from both sides [FTF+02, CFK04, CIL+07]. The design obtained in this paper builds on comparable requirements and preferences from decision-taking participants. It can use matchmaking or brokering to implement some of its elements. Section 2 gives detailed comparisons with related designs.

In systems not primarily designed with participants autonomy in mind, services are added in the attempt to centrally handle every possible constraint participants might find important [CFK04]. Instead, we separate responsibilities between participants, so that they deal with their own concerns, as opposed to having them managed by a third party.

2.2 Economic models

As opposed to queuing systems, economic models have received increasing attention recently. They study pricing strategies that achieve user objectives such as performance or fairness with income guarantees for providers [HWZ05, BMB+08]. The problem we address is relevant when the contract between users and providers does not entirely determine the resource allocation process. This is mostly the case in grids where high level contracts bind together large organizations. Such contracts may or may not use monetary compensation.

Game theory includes the study of possible strategies and resulting equilibria in *games* with competing participants. It has yield various protocols in distributed computer systems [CS00, JZ09]. In this area, Multiple Administrative Domains (MADs) were introduced to design fault tolerant systems. Participants' behaviors are given, and the goal is to design the *game* so that the equilibrium coincides with the objective. MADs principles have been used to design cooperative backup services, peer-to-peer data streaming, Internet and wireless routing [AAC+05]. Our formalism is the first to use MADs principles to design grid resource allocation architectures.

2.3 Containment in other models

Containment can be used to separate concerns. Various systems introduce containment with virtual machines to separate users from providers [KFFZ05, RMX05, RIG+06, GPJ+07]. They evaluate the benefits of virtual machines in addressing the problems that users may want a different software configurations than the native software of the host, and providers may want to move execution environment while in use. Our approach yields a new definition of containment, of which virtual machines are potential implementations, as well as a new definition of responsibilities.

Abstract State Machines (ASMs) are formal tools that can be understood as generalizations of turing machines [Gur03]. They are used to specify complex systems at every level of abstraction and generate tests. High level ASMs have been proposed as a basis for the specification of grid systems [NS06]. For generality, the authors introduced generic entities called *abstract resources*, *user mapping* and *resource mapping*. ASMs start by modeling a design. By contrast we start by modeling the specific problem of reconciling the participants diverging objectives. The solution we obtain yields a definition of *containers* consistent with *abstract resources*, that divide resource allocation in the same two parts, one carried out by users and the other by providers. It is consistent with their model and provides a semantic refinement of their components. However, we do not express it in ASM terms.

2.4 Metascheduling Pattern

The *Master-Slave* pattern specifies that a master process allocates tasks to slave processes [BMR+96a]. In addition, the *Broker* pattern defines a class that acts on behalf of requesters and hides the details of its action [BMR+96a]. Inspired from these two design patterns, *Metascheduling* was an early architectural pattern for grid resource allocation [Wei98]. It specifies that tasks are submitted to a global meta-scheduler, which in turn submits to local schedulers. This is the pattern used in most grid systems until recently [GDJ08].

The mainstream infrastructure behind LCG, the Large Hadron Collider Computing Grid, is an implementation of Metascheduling [BBB+05]. LCG is a collaboration of academia and scientific communities that aggregates resources for use primarily by CERN experiments. Its architecture is described in details in .

In LCG, a task comes with requirements in terms of the software that must be present on the execution environment. Computing resources are not guaranteed. Their type and amount varies according to site policies, their servers, and the load of other tasks possibly collocated on the same servers.

The central component is the *workload management system*, which is not under the user's nor the provider's control. It selects a cluster that satisfies the task requirements, on a site that accepts contracts with the VO[1] involved, based on access-right policies [And04].

Although the workload management system does not pervade a grid site, it sets strong terms on site resource management. In practice, the only freedom left to the resource provider is the choice of a local batch system such as Condor, LSF[2], PBS[3] or SGE[4] [BHK+00, IGF05, Hum06]. The middleware provides interfaces to supported batch systems. The only effort to minimize allocation cost is done by having the batch system statically assign a new container to the least loaded server on the cluster, that is, the server with shortest task queue. Providers have difficulties to perform maintenance operations to their resources because running tasks are directly bound to resources. For critical security upgrades, user tasks are abruptly discontinued.

The allocation of tasks is a static assignment. Tasks and allocated resources remain bound until the expiration of one of the two. Resources are divided into *server slots* that typically consists in three CPUs, without data isolation, and assigned by the provider's batch system.

Other grids such as EGEE[5], NorduGrid, BaltiGrid, Naregi and OSG[6] have a similar design [Ave07].

Virtual Workspaces isolates user resource in a Metascheduling implementation, the Globus Toolkit [KFFZ05]. Virtual Workspaces provides a Web-Service

[1] Virtual Organization, identified as a single user.
[2] Load Sharing Facility
[3] Portable Batch System
[4] Sun Grid Engine
[5] Enabling Grids for E-Science
[6] Open Science Grid

interface to deploy and configure Xen virtual machines (VMs) [BDF+03]. The targeted users and providers are the same as in scientific grids. Instead of a server slot, a user is given a *workspace*, i.e. a VM.

The VM deployment interface is presented to Globus middleware. VM management is assigned to the user or a third party. This is comparable with Condor's recent *VM Universe*, which lets the user define the deployment of a VM on a remote host.

The Broker pattern specifies that resources are hidden to the user and task management is delegated. This is not compatible with Symmetric Mapping.

2.5 Pull Mechanism and Late Binding Patterns

Master-Slave was later referred to as *Push Mechanism* by contrast with *Pull Mechanism* where idle resources request tasks [SBP03]. Pull Mechanism makes allocation resilient to broken resources and removes queues in front of end resources.

The *Late Binding* pattern allows the introduction of Pull Mechanism on a system designed with Push Mechanism. In Late Binding, monitors *pushed* to an end resource check the resource status before *pulling* actual tasks [BHL+06]. The pattern was first implemented by Condor Glide-In in 2001 [TTL02]. Since 2003, major grids are moving towards Late Binding [GDJ08].

AliEn is an example of Late Binding. AliEn is the Analysis Environment for ALICE, a virtual organization (VO). AliEn uses its own scheduling system on LCG infrastructure [SBP03].

As a consequence from using LCG, members of ALICE rely on the sites' best effort. However, tasks are processed immediately when submitted because a task is pulled when relevant resources are avaiable. AliEn gives more flexibility to the ALICE collaboration for task mapping. A *job agent* monitors every resource and triggers task selections from ALICE *job queue*. This mechanism is designed both to cope with lack of guarantees in LCG SLAs, and to prioritize tasks.

AliEn initiated the emergence of a proper front mapping via Late Binding. Most major VOs are now implementing a similar system: CDF with GlideCAF, ATLAS with Cronus and Panda, LHCb with DIRAC, CMS with Glidein-WMS and other independent large-scale applications may use DIANE [GDJ08].

Symmetric mapping decomposes the allocation in two parts and Pull Mechanism is a possible design for both parts. Late Binding separates resource subscription from task allocation. A system that implements Late Binding implements Symmetric Mapping if the underlying Metascheduling implementation does not constrain or obstruct the provider.

2.6 Market Based Control and P2P Matching Patterns

Market-Based Control simulates markets to share resources efficiently among competing users [Cle96, LB06].

For example Tycoon balances the load across and inside servers according to user payment [Lai05]. A contract involves the provider with the least expensive offer. The price of a provider's resources depends on the load. The provider does not intervene in the allocation. Commercial services are analogous. Amazon EC2[7] is a web-service to sell Amazon's resources on demand. The only provider is Amazon. In both examples, users have direct access to virtual machines. By contrast with Market-Based Control, Symmetric Mapping also supports the cases where the contract between user and provider does not entirely determine the allocation.

Peer-to-peer Matching is another alternative for contract mapping. In the absence of a market, participants match and negotiate contacts in a collaborative manner. For example, a Condor flock is the assembly of computer centers that borrow hardware resource from one another when needed [ELvD$^+$96]. A component called the *Gateway* is hosted by each participant. If a user cannot assign tasks to its own resources the Gateway examines its pairs. An important part of Condor is designed specifically for matchmaking and negotiation. The *Matchmaking* mechanism is provided through the use of *ClassAds* [Ram00]. Users and providers define themselves with relevant attribute and write their requirements with regular expressions on the other participant's attributes.

[7] Elastic Compute Cloud, amazon.com

Chapter 3

Objectives

This section examines typical user and provider concerns. We identify a **value** function in each case. This function quantifies a benefit that a given participant can expect from the allocation. We discuss the ability of a participant to predict or measure **value**, and optimize it with a schedule. The following concerns are defined : *makespan, sum of weighted flows, energy consumption* and *obstruction.*

3.1 Minimum makespan

The makespan is the time between submission of the first task and termination of the last. A user who wants all tasks to finish as soon as possible wants to minimize the makespan. This is the case if the user launches a single embarrassingly parallel application.

We write $m(a)$ the makespan of allocation a. For a user concerned with makespan, $\mathbf{value}(a) = -m(a)$ is a valid value function.

With identical tasks and identical resources, the exclusive availability of a processing unit (whatever this means: server, CPU, core or hardware thread) for a given time period is a valid container. A simple online greedy algorithm finds the optimal schedule with constant complexity. It buffers tasks as they come. Whenever a processing unit is free, it assigns a waiting task to run until termination.

In conventional scheduling, a task $t \in Tasks$ has a *length* l_t, a processing unit $r \in Resources$ has a *performance* p_r, and the execution time of t on r is l_t/p_r. The resulting problem is the Multiprocessor Scheduling Problem. No tractable algorithm finds the exact optimal schedule. However, achieving near-optimality with known error is tractable [AMZ03, MKK$^+$05, LSV06].

Without loss of generality a task $t \in Tasks$ has a number of instructions $s(t) \in \mathbb{N}$. An allocation $a \in A$ yields an instruction rate $\rho(t, a, \tau)$ for task t at time τ. The instruction rate is the number of instructions per second. The

makespan $m(a)$ is written

$$m(a) = \max_{t \in Tasks} \min\{\tau' \in Time | \int_0^{\tau'} \rho(t, a, \tau) d\tau \geq s(t)\}$$

The estimation of $\rho(t, a, \tau)$ is reputedly a difficult problem. Precise analysis of computing performance, that takes into account the affinity between tasks and resources, is in its inception. It is addressed in areas of real time systems and heterogeneous computing [XZQ00, KL01, SOBS04, PRV08].

3.2 Minimum sum of weighted flows

A task flow is the time between task submission and termination. A user $u \in X$ who wants every task to finish as soon as possible wants to minimize a sum of weighted flows [LSV06]. This is the case when the user is actually a collaboration of individuals who launch batch computations over time.

For $t \in Tasks$ submitted at time τ_t, if $F(t, a)$ is its flow given allocation $a \in A$:

$$F(t, a) = \min\{\tau' \in Time | \int_{\tau_t}^{\tau_t + \tau'} \rho(t, a, \tau) d\tau \geq s(t)\}$$

If w_t the priority of task t, the sum of weighted flows is written

$$w(a) = \sum_{t \in Tasks} w_t F(t, a)$$

$\mathbf{value_u}(a) = -w(a)$ is a valid value function for user u.

Sum of weighted flows minimization suffers the same problems of tractability and modeling as makespan minimization.

3.3 Minimum energy consumption

The operation of a computing resource affects its power consumption. Power-manageable processors and devices exhibit different states which limit at different levels the range of their operation and their power consumption. State transitions require some time and consume energy, too. Power management policies determine the conditions of state transitions in an attempt to minimize the energy consumption of a computing resource under workload constraints [BR04, RRT+08]. We write $\mu(r, a, \tau)$ the resulting power consumed by $r \in Resources$ at time τ given the allocation $a \in A$ and the power management policy.

Part of the consumed power dissipates in heat. Cooling devices are operated according to the temperature in their zone of influence. The operation of a cooling device also consumes power. Thermal management policies determine the level of operation of each cooling device in an attempt to maintain acceptable temperatures with minimum energy consumption [BF07, TGV08]. We write

$\mu(d, a, \tau)$ the resulting power consumed by cooling device $d \in Devices$ at time τ. We write $Z_d \subset Resources$ the resources that have thermal transfers with d.

A provider $p \in X$ concerned with energy consumption wants to minimize $e(a)$, or maximize $\mathbf{value_p}(a) = -e(a)$, where

$$e(a) = \int_0^{+\infty} \left(\sum_{r \in Resources} \mu(r, a, \tau) + \sum_{d \in Devices} \mu(d, a, \tau) \right) d\tau$$

$\mu(d, a, \tau)$ depends on $\mu(r, a, \tau')$ $\forall \tau' \leq \tau$ and for all resource r in the zone of influence of d.

$\mu(d, a, \tau)$ is difficult to know because thermal transfers are involved. In a simple representation, thermal transfers are considered instantaneous. Under this hypothesis, the power consumed by a cooling device is a function of the power consumed in its zone of influence at the same time. If this function is replaced with its linear approximation, $\exists \alpha_d$, $\forall a \in A$, $\forall \tau \in Time$

$$\mu(d, a, \tau) = \alpha_d \sum_{r \in Z_d} \mu(r, a, \tau)$$

With this simplification,

$$e(a) = \int_0^{+\infty} \sum_{r \in Resources} \mu(r, a, \tau) \left(1 + \sum_{d \in Devices | r \in Z_d} \alpha_d \right) d\tau$$

In the simplest formulation, a resource r is a server. Its states are *up* or *sleep*. On up mode, r consumes power $\mu_u(r)$. A resource r has a power efficiency $\eta(r)$ where

$$\eta(r) = \mu_u(r) \left(1 + \sum_{d \in Devices | r \in Z_d} \alpha_d \right)$$

In the simplest greedy algorithm, tasks are taken in order of starting time. The server r with highest $\eta(r)$ which is not already busy is assigned the next task. A server which is not assigned a task for a sufficiently long time is put in sleep mode.

A provider that applies this algorithm would think that any workload with a specific lifetime is a valid container. In fact this is not accurate because the energy consumption of a server does not only depend on its run/sleep status. It depends on the workload applied to its processors and each of its devices. However, a provider who already carries out power management of its resources and thermal management of its cooling devices will probably want to carry out energy-aware resource scheduling because it is the last piece of control to get full responsibility of energy expenses.

3.4 Minimum obstruction

We call *obstruction* the event in which an external user blocks resources that the provider wants to use internally or access for maintenance. It is not always

possible for the provider to pre-empt resources or foresee when resources will be needed internally. The cost of obstruction can be quantified based on the eagerness of provider p to free some resource at a given time:

$$\mathbf{eager_p} : \begin{cases} Resources \times Time \longrightarrow \mathbb{R}^+ \\ (r, \tau) \longmapsto \mathbf{eager_p}(r, \tau) \end{cases}$$

$\mathbf{eager_p}(r, \tau) = 0$ if the provider does not need r at τ. Otherwise, $\mathbf{eager_p}(r, \tau) > 0$ and $\mathbf{eager_p}(r, \tau)$ indicates the relative benefit from monopolizing r at τ. The measure of eagerness is entirely up to the provider.

A provider p who wants to minimize obstruction wants to maximize the following value function.

$$\mathbf{value_p}(a) = \sum_{r \in Resources} \int_0^{+\infty} \mathbf{eager_p}(r, \tau)(1 - \delta_a(r, \tau))d\tau$$

Where $\delta_a(r, \tau) = 1$ if $\exists t \in Tasks | a(t, r, \tau) = true$, 0 otherwise.

Very often $\mathbf{eager_p}(r, \tau)$ is not known before τ, and the algorithm to maximize $\mathbf{value_p}(a)$ is an online algorithm.

Chapter 4

The Symmetric Mapping pattern

In order to separate the concerns of resource providers and resource users, we define an architectural pattern: Symmetric Mapping.

4.1 Overview

Symmetric Mapping is intended to design architectures that perform resource allocation in a way that satisfies both resource providers and resource users when their interests differ.

The allocation of tasks to resource determines the satisfaction of both providers and users. In general, their incentives conflict. For example the most power-efficient server is not always the fastest. An allocation directed by a decision system under user control can result in high resource supply costs and an allocation directed by a decision system under provider control can result in low user-perceived resource value. Instead of compromising with them, Symmetric Mapping builds on these differences from the system design.

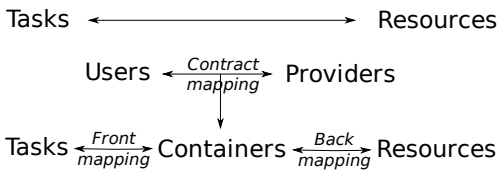


Figure 4.1: Mapping decomposition.

The principle of Symmetric Mapping is to divide resource allocation in three parts: **contract mapping**, **front mapping** and **back mapping** (fig 4.1). *Contract mapping* consists in the match between users and providers and the issue of containers that specify elementary resource transactions and subdivide

higher level contracts. Mapping of a task to a resource always contributes to fulfilling a contract. It always involves a container. Symmetric Mapping specifies that providers map their endorsed containers to their physical resources (*back mapping*) and users map their tasks to their subscribed containers (*front mapping*).

4.2 Definition

The Open Group Architecture Framework[1], a standardization consortium, identifies a terminology that can be used to define architectural patterns. According to the consortium, a pattern is defined with a *name*, an *intent*, *preconditions*, *forces* that play a role towards the intent, the *solution*, *postconditions* and *rationale*.

- Name: Symmetric Mapping
- Intent: Separate the concerns of providers and users and dispatch their responsibilities accordingly, so that their actions do not conflict and yield to the independent optimization of their respective concerns.
- Preconditions: Participants are administratively independent. Users need to run tasks and providers propose resources for tasks to run. They are bound by implicit or explicit contracts which justify their exchange. They are supposed to have rational and selfish behaviors. They have some idea of their objective and some knowledge on how to reach it.
- Forces: The greater the amount and heterogeneity of resources, tasks, and participants objectives, the better the opportunity to optimize the objectives [KSS+07].
- Solution:
  1. Insert intermediate entities between tasks and resources. These entities are called **containers**. A container holds attributes of a resource transaction. In conjunction with the actual resources involved in the transaction, it determines a cost for the provider. In conjunction with the tasks involved in the transaction, it determines a revenue for the user.
  2. Decompose the mapping of tasks to resources into the mapping of tasks to containers and the mapping of resources to containers.
- Postconditions: Users and providers match and issue containers. This is the first part of the allocation called **contract mapping**. Providers map resources to containers. This is another part of the allocation called **back mapping**. Users map tasks to containers. This is the third part of the allocation called **front mapping**.

[1] opengroup.org

- Rationale: If given appropriate responsibility, a participant will reach his own objective better than anyone else on his behalf.

4.3 Practical perspective

4.3.1 Participants

Formally, *(a)* If A and B are two sets, $\mathcal{P}(A)$ is the set of all possible subsets of A. $B \in \mathcal{P}(A)$ is equivalent to $B \subset A$. *(b)* A partition of A is a set of disjoint subsets of A that cover A in its entirety, i.e. C is a partition of A *iff* $C \in \mathcal{P}(\mathcal{P}(A))$ and $\forall a \in A, \exists! B \in C | a \in B$.

We define $Demand \in \mathcal{P}(\mathcal{P}(Tasks))$ a partition of $Tasks$ and $Supply \in \mathcal{P}(\mathcal{P}(Resources))$ a partition of $Resources$:

$$Demand = \{T \in \mathcal{P}(Tasks) | \exists x \in X, T = Tasks_x\}$$
$$Supply = \{R \in \mathcal{P}(Resources) | \exists x \in X, T = Resource_x\}$$

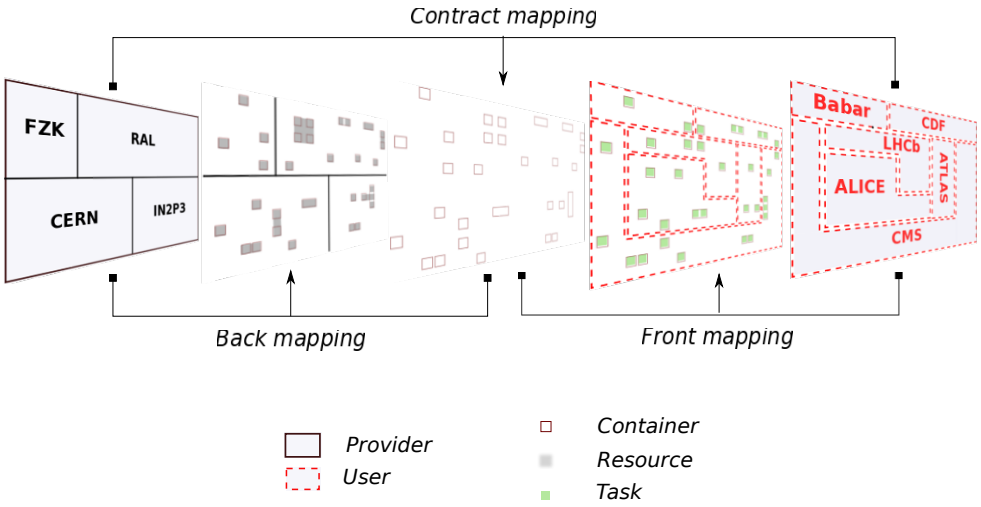


Figure 4.2: Participants, tasks, resources and containers.

Figure 4.2 features participants in a large particle physics grid. Resource providers are represented on the left hand side plane. The names correspond to actual institutions involved in the analysis of high energy physics data: CERN (Switzerland) is called the *Tier0*. It is where data is generated and stored in the first place. FZK (Germany), IN2P3 (France) and RAL (Great Britain) are example *Tier1*'s, where data is replicated [Rob06]. Tier0 and Tier1's are computer centers where a large part of the analysis takes place. At the time of writing, CERN has 7192 CPUs, IN2P3 3356, FZK 8340, and RAL 4016.

Resource users are represented on the right hand side plane. These are instances of virtual organizations (VOs): Babar, CDF, LHCb, ALICE, ATLAS,

CMS. Each of them is a community of researchers whose data is generated by a particle physics detector. The VO names are detector names. Members of a VO analyze data and therefore generate tasks that run on the LCG. A VO is considered a single user because its members report to the same institution, have common objectives and the same applications.

In Contract mapping, users and providers define containers. In Back mapping, providers select resources to back their containers, and in Front mapping, users select tasks to use their containers. A container is backed by one or an assembly of resources from a unique provider. It supports one or several tasks from a unique user.

4.3.2 Containers

Resource exchange between two autonomous institutions is the result of a contract between them. Grid sites engage in long term support for chosen virtual organizations. Symmetric Mapping requires that the contract that sets the terms of this support is made explicit. In addition, these contracts must be divisible into non-redundant subcontracts called *containers*. Containers must not allow resource oversubscription or undersubscription, and must be described with enough precision to have a determined value as perceived by each participant.

Therefore containers are specific kinds of *Service Level Agreements* (SLAs) [CIL+07]. SLAs typically specify the type and amount of subscribed resources and their lifetime, as well as constraints on the workload. Sometimes SLAs can be defined directly in terms of Quality of Service guarantees, which directly determine perceived values [RCAM06].

If relevant to determine the expected perceived value, container descriptions may include specifications on dedicated memory and cache hierarchy, the number of dedicated cores, their frequency, their optimization logic, network bandwidth, disk space, bandwidth and latency. Software configuration also belongs to resource specifications, potentially including operating system flavor, compilers and interpreters and their versions, and available administrative utilities.

A large liquidity of tasks or resources helps better use or support a given container. It also helps from over- or undersubscribing.

The implementation of a container may or may not force participants into complying with its specifications. A Condor sandbox pins to one processor but does not restrict the use of memory, whereas platform-level virtual machines do. Both do not constrain the provider [TTL02, BDF+03]. At least each participant must be able to check for compliance with specifications. Trust is usually necessary and mechanisms like reputation facilitate it.

4.3.3 Contract mapping

The process by which users and providers agree on containers requires search and decisions supervised by a neutral third party or performed by a peer-to-peer

mechanism [RLS98]. It can be based on a market, real or simulated.

Contract mapping is the match between a user and a provider and the issuing of containers that bind them. Symmetric Mapping specifies that contract mapping is separated from the rest of the allocation. Contract mapping is a function:

$$Contract : \begin{cases} Demand \times Supply \longrightarrow \mathcal{P}(Containers) \\ U, P \longmapsto Contract(U, P) \end{cases}$$

We define the activity of a participant by its subscribed containers. For example if $u \in X$ is a user, let $U = Tasks_u$.

$$Activity_u = \bigcup_{P \in Supply} Contract(U, P)$$

4.3.4 Front and back mappings

Let $(u, p) \in X^2$ be a grid resource user and provider. They carry out front and back mappings defined as in section 7.1.

$$u : \begin{cases} Activity_u \longrightarrow p_u(A)|c \\ c \longmapsto \arg \max_{p_u \in \mathbf{p_u}(A)|c} \mathbf{value_{u\downarrow}} \left(\mathbf{p_u}^{-1}(\{p_u\})|c \right) \end{cases}$$

$$p : \begin{cases} Activity_p \longrightarrow p_p(A)|c \\ c \longmapsto \arg \max_{p_p \in \mathbf{p_p}(A)|c} \mathbf{value_{p\downarrow}} \left(\mathbf{p_p}^{-1}(\{p_p\})|c \right) \end{cases}$$

Containers leave adequate flexibility on both sides. A user independently schedules its tasks and a provider its resources.

Symmetric Mapping respects the possibility that users schedule their tasks according to their objective and their knowledge on how to achieve it. In addition, users can adapt to unplanned task behaviors and faulty resources by dynamic reallocation, or checkpointing and migration.

To the provider, a container is non obstructive. It clearly isolates user access and limits the provider's commitment. In addition, a container is loosely-coupled to the provider's resources. While containers constrain resource types, providers choose physical resources. Since commitments are known in advance, resource consolidation can be planned [PZU+07]. Since back mapping is dynamic, providers can freely administrate their resources, grant access to local users or for regular maintenance operations, and react to some light cases of resource faults.

4.4 Relevance of Symmetric Mapping

Whether Symmetric Mapping should be used instead of another design must be decided on a case by case basis. Relevant considerations include:

Autonomy. The more autonomous providers are from users, the more useful Symmetric Mapping is with regards to Metascheduling or Late Binding.

Expertise. Whether users allocate their tasks manually or use a decision system, the relevance of Symmetric Mapping or Late Binding with regards to Metascheduling depends on their ability to perform allocations that impacts their perceived value of the resources. Similarly, a provider who does not manage energy consumption or maintain servers will not have strong diverging requirements.

Liquidity. The more tasks and the more resources, the higher the gain from optimized allocation on each side.

Trust. Participants must trust a system that addresses their concerns on their behalf. Otherwise, the use of Symmetric Mapping is relevant.

Sensitivity to cost and value. If users can obtain enough resources at no charge, and if they do not make a difference between heterogeneous resources, they do not have the incentives that justify the use of Symmetric Mapping. The situation is similar if providers have fixed operating budgets and if they do not make other use of their resources.

Chapter 5

Accuracy and benefits

In reality, users and providers do not know the precise information and mechanisms that determine the value of their objective functions. In addition, accurate optimization might be intractable. These limitations may weigh against an architecture that gives participants the responsibility of their objectives. The following simulation suggests that even with approximate knowledge and basic algorithms, participants benefit from the separation of responsibilities that we propose.

We consider a user interested in minimum makespan and a provider interested in minimum obstruction.

5.1 Resources and tasks

For simplicity, a resource is a server, and two tasks do not run together on a server. $\forall t \in Tasks$, $\forall r \in Resources$, $\exists \rho(t,r) \in \mathbb{R}^+$, $\forall \tau \in Time$,

$$\rho(t,a,\tau) = \begin{cases} \rho(t,r) & \text{if } a(t,r,\tau) = true \\ 0 & \text{otherwise} \end{cases}$$

The throughput, $\rho(t,r)$ is the number of instructions per second.

In order to reflect correlations between throughputs on the same server, we write:

$$\rho(t,r) = \rho_{c/s}(r)\rho_{i/c}(t,r)$$

$\rho_{c/s}(r)$ is cycle rate of r, i.e. the number of hardware threads times the frequency. $\rho_{c/s}(r)$ reflects the maximum absolute performance of r. $\rho_{i/c}(t,r)$ is the number of instructions of t per cycle of r. $\rho_{i/c}(t,r)$ reflects the relative width of the bottleneck, or affinity between r and t.

In the simulation, we generate a random cycle rate $\rho_{c/s}(r)$ for each resource r, a random task size $s(t)$ in number of instructions for each task t, and a random affinity $\rho_{i/c}(t,r)$ for each couple (t,r).

In practice, participants have an approximate understanding of the mechanisms that determine performance. To account for it, we suppose that the participants do not know $\rho_{c/s}(r)$, $\rho_{i/c}(t,r)$ and $s(t)$. Instead, they have a notion of task length l_t and server performance p_r obtained by observation. l_t is the observed execution time of task t in average. The user who owns t knows l_t.

$$l_t = \frac{1}{|Resources|} \sum_{r \in Resources} \frac{s(t)}{\rho(t,r)}$$

p_r is the observed average ratio between a task length and its actual execution time on the server. The provider who owns r knows p_r.

$$p_r = \frac{1}{|Tasks|} \sum_{t \in Tasks} \frac{l_t \rho(t,r)}{s(t)}$$

To simulate obstruction to the provider, for every server r we pick random time periods $T_r \subset Time$ such that $\mathbf{eager_p}(r,\tau) = 1$ if $\tau \in T_r$ and 0 otherwise, and such that $\mathbb{P}(\tau \in T_r)$ is specified in the input of the simulation, for example 10%.

5.2 Algorithms

For a user u and a provider p, the contract $C_{u,p}$ says: *Starting at t_0, u obtains the monopoly on up to N_p servers of p as long as u uses these servers to process any of the N_u specified tasks.*

We compare makespan and obstruction in the following cases.

1. A third party controls the allocation, independently from user and provider objectives, and with less information on resources and tasks.
2. The user controls the allocation, in a rational and selfish manner, with less dynamic control as the provider would have.
3. The provider controls the allocation, in a rational and selfish manner.
4. Provider and user control their respective schedules on containers compatible with their contract, and such that a schedule determines the perceived value.
5. These are compared to a theoretically attainable measure that uses exact values of performance, affinity and task size.

The allocation carried out by a third party is implemented as a static random assignment of the tasks to N_p random servers.

With full control, user u picks each task t in order of decreasing l_t and assigns it to a server of minimum cumulated lengths $\sum_{t' \in T_r} l_{t'}$. $T_r \subset Tasks$ is the set of tasks assigned to server r.

With full control, p starts with the same random allocation as a third party. When willing to preempt a busy server, p moves its tasks to a random free server if there is one.

A valid set of containers has N_p containers. Each of them says: *Starting at t_0, p provides u with the possibility to compute one of the N_u specified tasks with a performance p_c that does not vary so much that it affects the value perceived by the user.* We consider that p_c must remain within the minimum standard deviation of performance σ.

$$\sigma = \min_{r \in Resources} \sqrt{\frac{1}{|Tasks|} \sum_{t \in Tasks} \left(\frac{l_t \rho(t,r)}{s(t)} - p_r \right)^2}$$

Given these containers, user u picks each task t in order of decreasing l_t and assigns it to a container of minimum predicted load $(\sum_{t' \in T_c} l_{t'})/p_c$. $T_c \subset Tasks$ is the set of tasks assigned to container c.

For $r \in Resources$, we write its neighborhood $\mathcal{N}_r$.

$$\mathcal{N}_r = \{r' \in Servers | r' \neq r \text{ and } |p_{r'} - p_r| \leq \sigma\}$$

p can move containers inside the same neighborhood.

Theoretical attainable values are measured from the following allocation. Initially, each task t is picked in order of decreasing $s(t)$ and assigned to a container of minimum size $\sum_{t' \in T_c} s_{t'}$. Whenever a task t starts or the provider wants to preempt a server r, the corresponding container is moved to the free server of best throughput $\rho(t,r)$.

5.3 Results

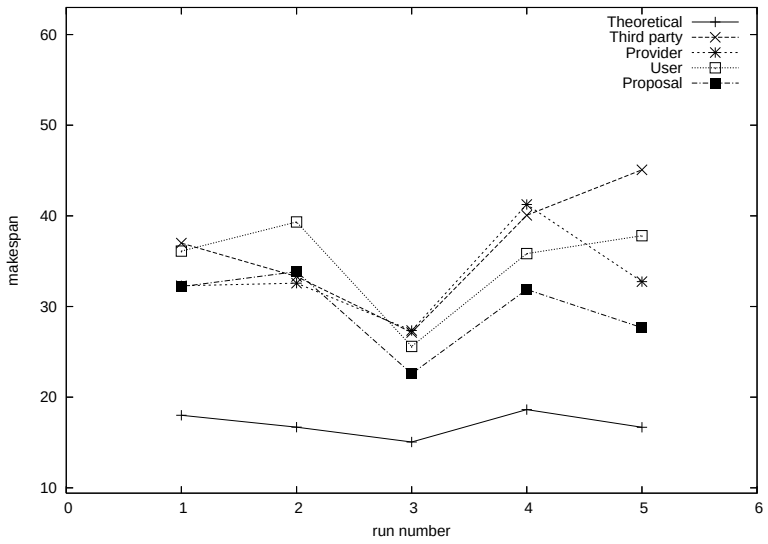
Simulations are written in Python using test-driven development. The code is released under Artistic License 2 and available on a public repository[1].

Each figure shows the makespan and obstruction on five runs. Each run corresponds to a set of tasks and resources, and a number of containers. On all figures, all five runs have the same number of tasks, resources, containers, statistical distribution of task sizes, cycle rates, instructions per cycle.

Eagerness is identically generated in all cases. On every server, periods of availability ($\mathbf{eager_p}(r,\tau) = 0$) follow a normal distribution of average 10 hours and standard deviation 5 hours. This is a reasonable period, e.g. night time, during which a server is available without interruption to external use. Periods of providers potential occupation ($\mathbf{eager_p}(r,\tau) = 1$) follow a normal distribution of 5 hours in average and 2 hours in standard deviation, which is the time typically needed for server maintenance or interactive use.

Figures 5.1, 5.2 and 5.3 are taken with heterogeneous tasks. $s(t)$ is uniform from $7,200$ to $18,000$ billion instructions. This corresponds to two to five hours

[1] code.google.com/p/symmetric-mapping



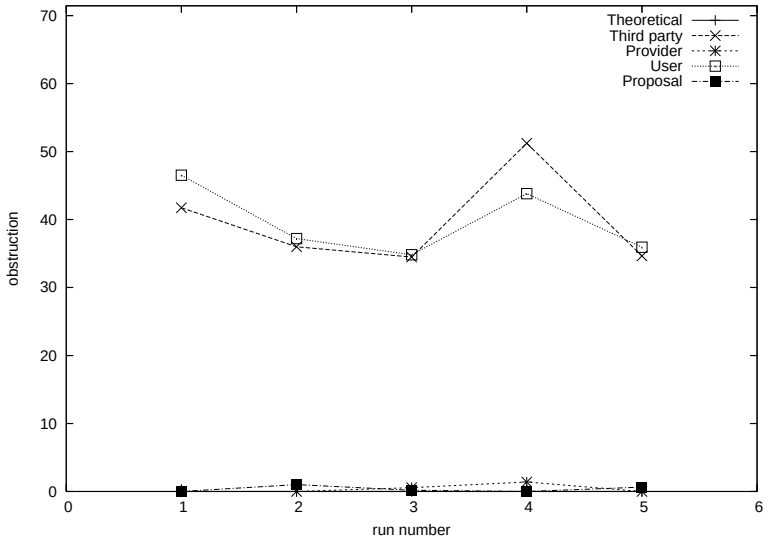
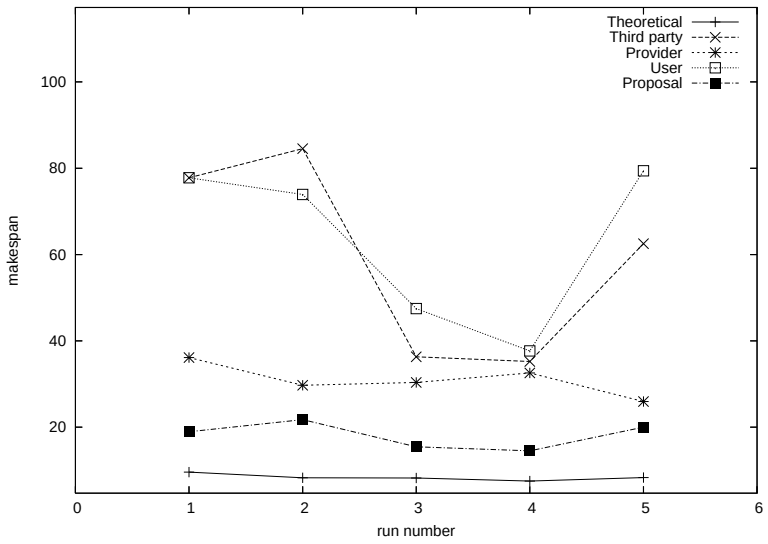


Figure 5.1: Small liquidity



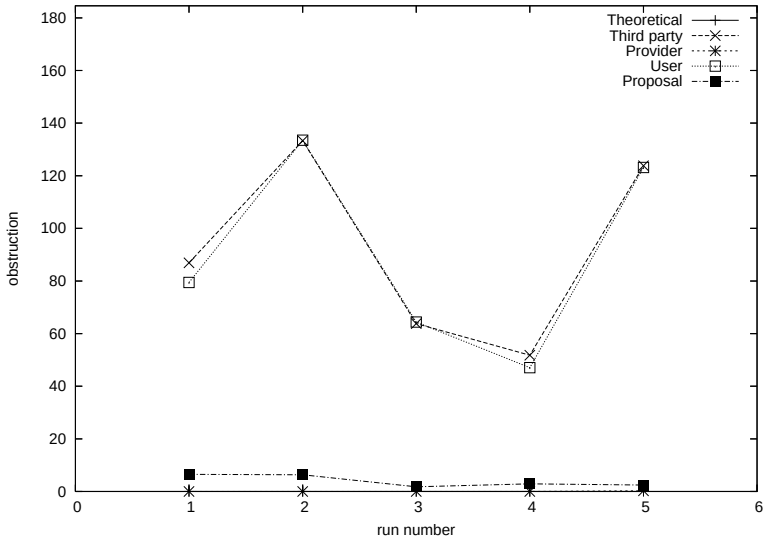
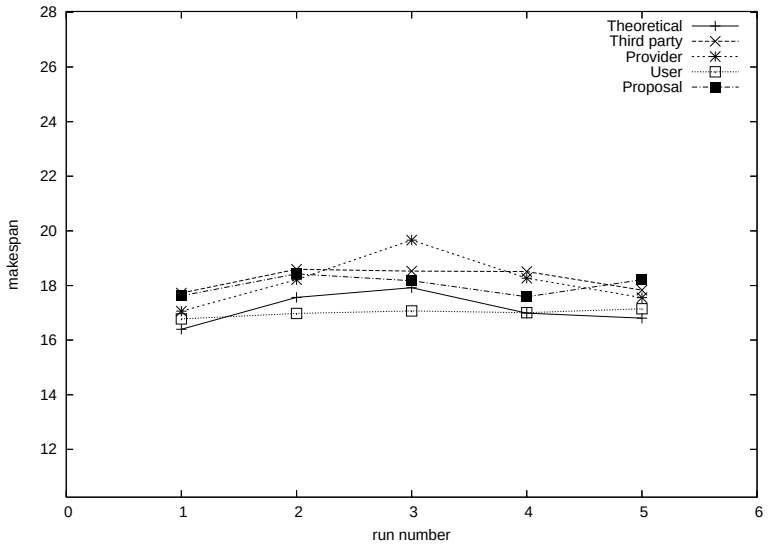


Figure 5.2: Large liquidity



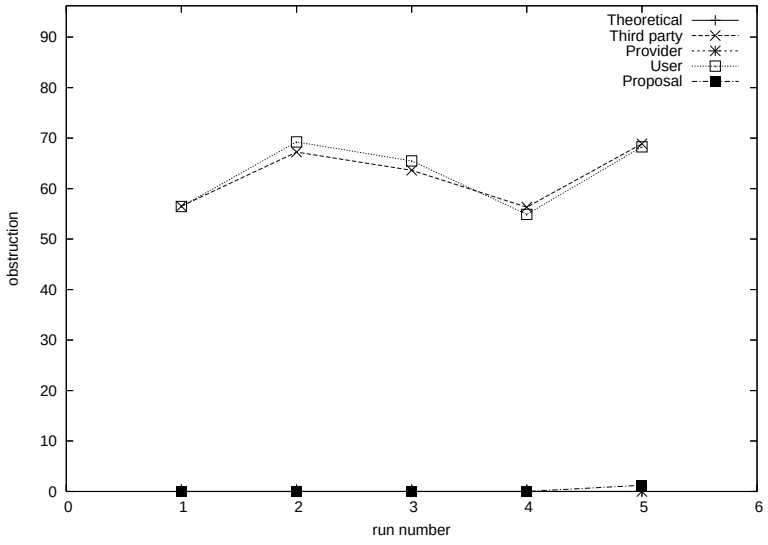
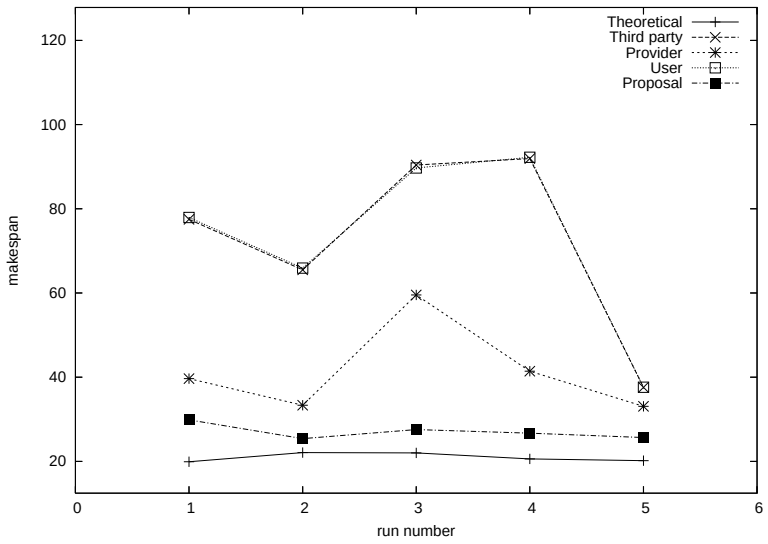


Figure 5.3: Identical resources, large liquidity



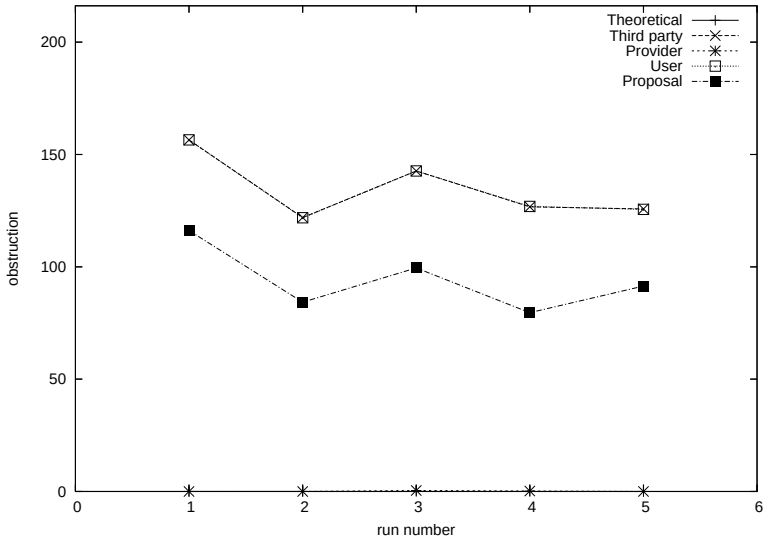


Figure 5.4: Identical tasks, large liquidity

on a one giga-ops system, which is common to observe for a single submission on a CPU.

Figures 5.1, 5.2 and 5.4 are taken with heterogeneous resources. $\rho_{c/s}(r)$ is uniform from 1 to 6 billion cycles per second. $\rho_{i/c}(t, r)$ is uniform from 0.5 to 0.75 instructions per cycle. This can be observed with performance monitoring software on commodity servers.

Figure 5.3 is taken with homogeneous resources. Server performance $\rho_{c/s}(r)$ is constant $= 8$ and $\rho_{i/c}(t, r)$ is uniform from 0.5 to 0.55. $\rho_{i/c}(t, r)$ denotes the affinity between task t and server r. Since the processor is always the same, the task is the only factor of variations and the variation is lower than with heterogeneous resources.

Figure 5.4 is taken with homogeneous tasks. $s(t)$ is constant $= 18,000$ billion instructions.

Figure 5.1 is taken with 100 tasks, 20 servers and 5 containers, and the others figures with 300 tasks, 60 servers and 15 containers.

In all cases, the provider obtains a better obstruction than the user or the third party. The user obtains a better makespan on homogeneous resources only. In other cases user efforts are not evidently better than random because task length has little correlation with actual task execution time. The user advantage in the case of homogeneous resources is marginal. It suggests that the gain from *load balancing* in homogeneous computing is marginal compared to the gain from *mapping* in heterogeneous computing, taking into account affinities between tasks and resources.

Data marked *Proposal* corresponds to the use of containers and the dispatch of responsibilities between user and provider. The obstruction is similar to the obstruction obtained by a provider, except with homogeneous tasks. The makespan is the best, except on homogeneous resources, where user makespan is the best. In cases where tasks or resources are homogeneous, σ is small and the number of servers in the neighborhood is limited.

The difference with theoretical values illustrates the effect of an approximate performance model. These experiments suggest that even when the underlying models of the participants are approximate, it is valuable to appropriately divide the allocation and dispatch responsibilities. Benefits increase with heterogeneity and liquidity.

Chapter 6

The model

This section introduces a new model of resource allocation. This model provides the foundations of Symmetric Mapping. Its novelty is to consider dynamic scheduling under constraints.

Section 6.1 defines an allocation as the association of a resource and a task at a given time, specified with a boolean function. Section 6.2 splits an allocation into two functions, one that specifies how resources are scheduled, and one that specifies how tasks are scheduled. Section 6.3 introduces a specifications as a predicate that a given allocation must satisfy. Section 6.4 defines the value of an allocation as perceived by a participant.

6.1 Allocations

An allocation is an application that takes a task, a resource and a time, and returns $true$ if the task is allocated to the resource at that time, $false$ otherwise.

$$a : Tasks \times Resources \times Time \longrightarrow \{true, false\}$$

We write $\mathcal{F}(E, G)$ the set of applications from set E to set G. Let A be the set of allocations.

$$A = \mathcal{F}(Tasks \times Resources \times Time, \{true, false\})$$

We call X the set of participants. If $u \in X$ is a resource user that owns $Tasks_u \subset Tasks$ and $p \in X$ is a resource provider that owns $Resources_p \subset Resources$, we define $A_{u,p}$ the set of allocations that involve u and p.

$$A_{u,p=}\mathcal{F}(Tasks_u \times Resources_p \times Time, \{true, false\})$$

We define the null allocation $\emptyset \in A$ such that $\forall(t, r, \tau) \in Tasks \times Resources \times Time$:

$$\emptyset(t, r, \tau) = false$$

We define an operation on allocations, that takes two allocations and returns the allocation that is the combination of the two. We call it their *merger*. Operator $\vee$ between booleans is logical *or*. For all $(a_1, a_2) \in A^2$, the merger of a_1 and a_2 is $a_1 \vee a_2$ such that $\forall (t, r, \tau) \in Tasks \times Resources \times Time$:

$$(a_1 \vee a_2)(t, r, \tau) = a_1(t, r, \tau) \vee a_2(t, r, \tau)$$

The merger of two sets of allocations is the set of all possible mergers with one allocation from each set. If E is a set, we write $\mathcal{P}(E)$ the set of subsets of E. For two sets of allocations $(A_1, A_2) \in \mathcal{P}(A)^2$, we define their merger $A_1 \vee A_2$.

$$A_1 \vee A_2 = \{a \in A | \exists (a_1, a_2) \in A_1 \times A_2, a = a_1 \vee a_2\}$$

6.2 Schedules

In this section we divide allocations in parts. Each part is called a *schedule*. A schedule involves either tasks or resources but not both. A resource schedule tells if a given resource is involved at a given time, and a task schedule tells if a given task is involved at a given time.

A resource schedule returns *true* for a couple (r, τ) where resource r is assigned at time τ; and a task schedule returns *true* for a couple (t, τ) where task t is assigned at time τ.

The goal is to isolate every part of the allocation that interests a single participant. In this section we introduce a notation that allows to represent participants as mathematical objects. In fact, a participant is represented with an function of allocations, or a *projector* that, given an allocation, returns the part of the allocation of interest to the participant.

Let u be a user. We define its projector $\mathbf{p_u}$

$$\mathbf{p_u} : \begin{cases} A \longrightarrow \mathbf{p_u}(A) \\ a \longmapsto \mathbf{p_u} \circ a \end{cases}$$

such that:

$$\mathbf{p_u}(A) = \mathcal{F}(Tasks_u \times Time, \{true, false\})$$

and $\forall a \in A, \forall (t, \tau) \in Tasks_u \times Time$,

$$\mathbf{p_u} \circ a(t, \tau) = \begin{cases} true & \text{if } \exists r \in Resources | a(t, r, \tau) \\ false & \text{otherwise} \end{cases}$$

$\forall a \in A$, $\mathbf{p_u} \circ a$ is the task schedule of allocation a.

Let p be a provider. We define its projector $\mathbf{p_p}$.

$$\mathbf{p_p} : \begin{cases} A \longrightarrow \mathbf{p_p}(A) \\ a \longmapsto \mathbf{p_p} \circ a \end{cases}$$

Such that:

$$\mathbf{p_p}(A) = \mathcal{F}(Resource_p \times Time, \{true, false\})$$

and $\forall a \in A, \forall (r, \tau) \in Resources_p \times Time$,

$$\mathbf{p_p} \circ a(r, \tau) = \begin{cases} true & \text{if } \exists t \in Tasks | a(t, r, \tau) \\ false & \text{otherwise} \end{cases}$$

$\forall a \in A$, $\mathbf{p_p} \circ a$ is the resource schedule of allocation a.

Projectors allow to extract the part of the allocation of interest to a user or a provider. It is also possible to reconstruct an allocation from two projections. The following shows how a schedule of tasks and a schedule of resources together define an allocation.

Let u be a user and p a provider. $\forall a_u \in \mathbf{p_u}(A), \forall a_p \in \mathbf{p_p}(A)$, we define[1] $a_u \wedge a_p$ the expansion of a_u and a_p such that $\forall (t, r, \tau) \in Tasks_u \times Resources_p \times Time$,

$$(a_u \wedge a_p)(t, r, \tau) = a_u(t, \tau) \wedge a_p(r, \tau)$$

An allocation is the expansion of a resource schedule and a task schedule. If a resource is assigned at a given time in the allocation, it is assigned at the same time in the corresponding resource schedule. If a task is assigned at a given time in the allocation, it is assigned at the same time in the corresponding task schedule.

6.3 Specifications

Specifications represent "anything that can be said about an allocation". A specification is a predicate. A specification applies to an allocation if the specification and the allocation are said to *match*. The **match** function returns *true* when applied to them.

In fact we introduce specifications to represent the contracts between users and providers that justify the existence of allocations. The allocation matches the specification that represents the contract between involved users and providers.

We call *Specs* the set of specifications. We introduce a function **match** that says if an allocation is compatible with a specification.

$$\mathbf{match} : A \times Specs \longrightarrow \{true, false\}$$

A specification can be split into constraints on the schedule of each participant. $\forall s \in Specs, \forall x \in X$, $\exists s_x \in Specs$ such that $\forall a \in A$

$$\mathbf{match}(a, s) \Leftrightarrow \bigwedge_{x \in X} \mathbf{match}(\mathbf{p_x}(a), s_x)$$

[1] Operator $\wedge$ between booleans is logical *and*.

If A' is a set of allocations, for readability we define $A'|s$ the subset of A' in which all allocations satisfy s. $\forall A' \in \mathcal{P}(A), \forall s \in Specs$:

$$A'|s = \{a \in A' | \mathbf{match}(a, s)\}$$

We introduce the notion of compatibility of a specification with a set of specifications. A specification can be decomposed into a *compatible* set of specifications.

Let $s \in Specs$, $S \in \mathcal{P}(Specs)$. We say that S is compatible with s, and we write $S \vDash s$ if and only if

$$\bigvee_{s' \in S} A|s' = A|s$$

This formula says that by choosing an allocation for each specification of the compatible set, such that the allocation satisfies the specification, and by merging all these allocations, we obtain an allocation that satisfies the initial specification.

6.4 Value

In this section we define the *value* of an allocation. It is the function illustrated in section 5. It quantifies the outcome of an allocation as perceived by a participant, in terms of how well the allocation reaches the participant's objective and how beneficial it is for the participant.

Let x be a participant. x is a user or a provider. We write $\mathbf{value_x}$ the value of an allocation for participant x, i.e. how much x gains from the allocation.

$$\mathbf{value_x} : A \longrightarrow \mathbb{R}$$

The value of the null allocation is null: $\forall x \in \{u, v\}$,

$$\mathbf{value_x}(\emptyset) = 0$$

The inclusion-exclusion principle applies: $\forall x \in \{u, v\}$, $\forall (a_1, a_2) \in A^2$,

$$\begin{aligned}\mathbf{value_x}(a_1 \vee a_2) = \mathbf{value_x}(a_1) + \mathbf{value_x}(a_2) \\ - \mathbf{value_x}(a_1 \wedge a_2)\end{aligned}$$

If x is a user, $\mathbf{value_x}$ is generally positive because a user gains in having their tasks processed. If x is a provider, $\mathbf{value_x}$ is generally negative because processing tasks generates a cost.

We write $\mathbf{value_{x\downarrow}}$ the guaranteed value of a set of allocations, knowing one allocation of the set will be effective.

$$\mathbf{value_{x\downarrow}} : \begin{cases} \mathcal{P}(A) \longrightarrow \mathbb{R} \\ \quad A' \longmapsto \min\limits_{A'} \mathbf{value_x} \end{cases}$$

The guaranteed value is the minimum value on the set of allocations that contain the effective allocation.

Chapter 7

MAD resource allocation problem

This section formalizes the hypotheses and the objective of grid resource allocation. The formulation is an analogy with the MAD[1] model for fault tolerance in distributed systems. We propose reasonable hypotheses on the behavior of autonomous participants in resource allocation. The objective is to independently optimize the value as perceived by every participant.

7.1 Hypotheses

1. Participants are selfish and rational.
2. A user can only schedule its own tasks and a provider can only schedule its own resources.
3. Participant $x \in \{u, p\}$ is bound in a contract $S_{u,p}$.

This summarizes as follows. $\forall S \in \mathcal{P}(Specs)$ compatible with $S_{u,p}$ ($S \vDash S_{u,p}$ as defined in section 6.3), $\forall s \in S$, $x \in \{u, p\}$ chooses the schedule:

$$x(s) = \arg \max_{p_x \in \mathbf{p_x}(A)|s} \mathbf{value_{x\downarrow}} \left({\mathbf{p_x}}^{-1}(\{p_x\})|s \right)$$

It means that a participant chooses a schedule that satisfies her constraints, and that guarantees the best value as she perceives it, provided specifications are satisfied.

This hypothesis is a statement of responsibility rather than capability. The efforts of a participant will be based on her perception of what is or is not valuable, whether her perception is correct or not. However, participants must have enough confidence in their own assessments in order to find it worthwhile to perform their own scheduling.

[1] Multiple Administrative Domains

$S_{u,p}$ is normally negotiated to make sure that some requirements are fulfilled. If there is a minimum acceptable value $vmin_x$ for each participant $x \in \{u, p\}$

$$\forall a \in A | S_{u,p}, \mathbf{value_x}(a) \geq vmin_x$$

7.2 Objective

We want to find an allocation $\bar{a}$ that independently maximizes all perceived values among allocations that satisfy the contracts. For all participant x bound in contract S_x,

$$\bar{a} = \arg\max_{A|S_x} \mathbf{value_x}$$

Chapter 8

A solution to the MAD problem for grids

We propose a solution to the problem of multiple administrative domains in resource allocation. This solution relies on the existence of containers. A container determines the perceived value of the part of the allocation that it contains.

8.1 Containers

Our proposal relies on the existence of a set of specifications that follows certain properties. We call $Containers$ this set. $Containers \subset Specs$. The following hypotheses define a notion of containment that allows to solve the MAD problem for grids.

1. A container forbids over- or under-subscription.

   $\forall c \in Containers$, $\exists T_c \subset Time$, $\forall a \in A|c$,

   $$\forall \tau \in T_c,\ \exists (t,r) \in Tasks \times Resources, a(t,r,\tau)$$
   $$\forall \tau \in Time \backslash T_c,\ \forall (t,r) \in Tasks \times Resources, \neg a(t,r,\tau)$$

   T_c is called the container lifetime.

2. Among allocations that satisfy a container, schedules determine the perceived value.

   $\forall x \in \{u,p\}$, $\forall c \in Containers$, $\forall (a_1,a_2) \in (A_{u,p}|c)^2$,

   $$\mathbf{p_x}(a_1) = \mathbf{p_x}(a_2) \Rightarrow \mathbf{value_x}(a_1) = \mathbf{value_x}(a_2)$$

3. For every specification, there is a compatible non redundant set of containers. $\forall s \in Specs$, $\exists C \in \mathcal{P}(Containers)$ such that

   $$A|s = \bigvee_{c \in C} A|c$$

and

$$\forall (c_1, c_2) \in C^2 | c_1 \neq c_2,$$
$$\forall (a_1, a_2) \in A | c_1 \times A | c_2, a_1 \wedge a_2 = \emptyset$$

8.2 Protocol

If $Containers$ exists, it is possible to independently optimize the value perceived by each participant. This is done by giving participants the possibility to pick a set of non redundant containers $C_{u,p} \in \mathcal{P}(Containers)$ compatible with their contract: $C_{u,p} \vDash S_{u,p}$. From property 3 of $Containers$, $C_{u,p}$ exists.

In the following, we show that this protocol yields a unique allocation, which satisfies the contract and the optimality objective.

8.3 Correctness

Theorem 1. *The protocol yields a unique allocation.*

$$\bar{a} = \bigvee_{c \in C_{u,p}} u(c) \wedge p(c)$$

Proof. Let $c \in C$, $H = \mathbf{p_u}^{-1}(\{u(c)\}) \cap \mathbf{p_p}^{-1}(\{p(c)\})$. We will prove that $u(c) \wedge p(c) = \bigvee_{a \in H} a$.

Part 1 We show that $u(c) \wedge p(c) \in H$.

First, we show that $u(c) \wedge p(c) \in \mathbf{p_u}^{-1}(\{u(c)\})$, i.e. $\mathbf{p_u}(u(c) \wedge p(c)) = u(c)$. Let $(t, \tau) \in Tasks \times Time$.

Case $\mathbf{p_u}(u(c) \wedge p(c))(t, \tau) = true$. $\exists r \in Resources$ such that $(u(c) \wedge p(c))(t, r, \tau) = true$. *A fortiori*, $u(c)(t, \tau) = true$.

Case $\mathbf{p_u}(u(c) \wedge p(c))(t, \tau) = false$; *ad absurdum*. Suppose $u(c)(t, \tau) = true$. Necessarily, $\forall r' \in Resources$, $p(c)(r', \tau) = false$, i.e. $\exists a \in A_{u,p} | c$, $\forall (r', t') \in Resources \times Tasks$, $a(t', r', \tau) = false$. It means that $\tau \notin T_c$, and therefore $u(c)(t, \tau) = false$.

Second, the proof of $u(c) \wedge p(c) \in \mathbf{p_p}^{-1}(\{p(c)\})$ is analogous.

Part 2 We show that $\forall a \in H, a \vee (u(c) \wedge p(c)) = u(c) \wedge p(c)$. Let $a \in H$, $(t, r, \tau) \in Tasks \times Resources \times Time$.

Case $(a \vee (u(c) \wedge p(c)))(t, r, \tau) = true$; *ad absurdum*. Suppose $(u(c) \wedge p(c))(t, r, \tau) = false$. It yields $a(t, r, \tau) = true$. It follows $(\mathbf{p_u} \circ a)(t, \tau) = true$. Since $a \in \mathbf{p_u}^{-1}(\{u(c)\})$, $\mathbf{p_u} \circ a = u(c)$. Therefore $u(c)(t, \tau) = true$. Similarly, $(\mathbf{p_p} \circ a)(r, \tau) = true$ and therefore $u(p)(r, \tau) = true$. Finally, $(u(c) \wedge p(c))(t, r, \tau) = true$.

Case $(a \vee (u(c) \wedge p(c)))(t, r, \tau) = false$. It follows $a(t, r, \tau) = false$ and $(u(c) \wedge p(c))(t, r, \tau) = false$.

Part 3 $u(c) \wedge p(c) \in A|c$ because

$$\mathbf{p_u}(u(c) \wedge p(c)) = u(c) \in \mathbf{p_u}(A)|c_u$$
$$\mathbf{p_p}(u(c) \wedge p(c)) = u(c) \in \mathbf{p_p}(A)|c_p$$

□

Theorem 2. *The obtained allocation satisfies a set of specifications compatible with the contract that binds the participants.*

$$\bar{a} \in A|C_{u,p} \text{ and } C_{u,p} \vDash S_{u,p}$$

Proof.

$$\bar{a} = \bigvee_{c \in C_{u,p}} u(c) \wedge p(c) \in \bigvee_{c \in C_{u,p}} A|c = A|C_{u,p}$$

□

Theorem 3. *The obtained allocation is optimal.*

$$\forall x \in \{u, p\}, \bar{a} = \arg\max_{A|S_{u,p}} \mathbf{value_x}$$

Proof. Let $c \in C_{u,v}$, $x \in \{u, v\}$. From hypothesis,

$$x(c) = \arg\max_{p_x \in \mathbf{p_x}(A)|c_x} \mathbf{value_{x\downarrow}}\left(\mathbf{p_x}^{-1}(\{p_x\})|c\right)$$

$$\mathbf{value_{x\downarrow}}\left(\mathbf{p_x}^{-1}(\{x(c)\})|c\right) = \max_{p_x \in \mathbf{p_x}(A)|c_x} \mathbf{value_{x\downarrow}}\left(\mathbf{p_x}^{-1}(\{p_x\})|c\right)$$

Let $p_x \in \mathbf{p_x}(A)|c_x$,

$$\mathbf{value_{x\downarrow}}\left(\mathbf{p_x}^{-1}(\{p_x\})|c\right) \leq \mathbf{value_{x\downarrow}}\left(\mathbf{p_x}^{-1}(\{x(c)\})|c\right)$$

$$\min_{a_2 \in \mathbf{p_x}^{-1}(\{p_x\})|c} \mathbf{value_x}(a_2) \leq \min_{a_1 \in \mathbf{p_x}^{-1}(\{x(c)\})|c} \mathbf{value_x}(a_1)$$

$\forall a_1 \in \mathbf{p_x}^{-1}(\{x(c)\})|c$, $\exists a_2 \in \mathbf{p_x}^{-1}(\{p_x\})|c$,

$$\mathbf{value_x}(a_2) \leq \mathbf{value_x}(a_1)$$

From containers property 2, $\forall a \in \mathbf{p_x}^{-1}(\{p_x\})|c$,

$$\mathbf{value_x}(a) = \mathbf{value_x}(a_2)$$

Therefore, $\forall a_1 \in \mathbf{p_x}^{-1}(\{x(c)\})|c$, $\forall a \in \mathbf{p_x}^{-1}(\{p_x\})|c$,

$$\mathbf{value_x}(a) \leq \mathbf{value_x}(a_1)$$

It simplifies as $\forall a_1 \in \mathbf{p_x}^{-1}(\{x(c)\})|c$, $\forall a \in A|c$,

$$\mathbf{value_x}(a) \leq \mathbf{value_x}(a_1)$$

Since $u(c) \wedge p(c) \in \mathbf{p_x}^{-1}(\{x(c)\})|c$,

$$\mathbf{value_x}(a) \leq \mathbf{value_x}(u(c) \wedge p(c))$$

It means that:

$$\mathbf{value_x}(u(c) \wedge p(c)) = \max_{A|c} \mathbf{value_x}$$

Since $\forall (c_1, c_2) \in C^2_{u,v} | c_1 \neq c_2$, $u(c_1) \wedge p(c_1) \in A|c_1$ and $u(c_2) \wedge p(c_2) \in A|c_2$,

$$(u(c_1) \wedge p(c_1)) \wedge (u(c_2) \wedge p(c_2)) = \emptyset$$

Therefore, $\forall x \in \{u, v\}$,

$$\begin{aligned} &\mathbf{value_x}((u(c_1) \wedge p(c_1)) \vee (u(c_2) \wedge p(c_2))) \\ &\quad = \mathbf{value_x}(u(c_1) \wedge p(c_1)) + \mathbf{value_x}(u(c_2) \wedge p(c_2)) \end{aligned}$$

It yields:

$$\mathbf{value_x}\left(\bigvee_{c \in C_{u,v}} u(c) \wedge p(c)\right) = \sum_{c \in C_{u,v}} \mathbf{value_x}(u(c) \wedge p(c))$$

Finally, let $x \in \{u, p\}$

$$\begin{aligned} &\mathbf{value_x}\left(\bigvee_{c \in C_{u,v}} u(c) \wedge p(c)\right) \\ &\quad = \sum_{c \in C_{u,v}} \mathbf{value_x}(u(c) \wedge p(c)) \\ &\quad = \sum_{c \in C_{u,v}} \max_{A|c} \mathbf{value_x} \\ &\quad = \sum_{c \in C_{u,v}} \mathbf{value_x}\left(\arg\max_{A|c} \mathbf{value_x}\right) \\ &\quad = \mathbf{value_x}\left(\bigvee_{c \in C_{u,v}} \arg\max_{A|c} \mathbf{value_x}\right) \\ &\quad = \mathbf{value_x}\left(\arg\max_{A|S_{u,v}} \mathbf{value_x}\right) \\ &\quad = \max_{A|S_{u,v}} \mathbf{value_x} \end{aligned}$$

□

Chapter 9

Conclusion

The fact that grids span multiple administrative domains is commonly acknowledged as their distinctive feature among other distributed computing systems. However, prior to this work, the diverging objectives of the participants to grid resource allocation were not taken into account in the architectural design.

A new model for grid resource allocation permits to apply the MAD principles and formalize the problem. It yields a definition of containment that allows to separate the concerns of the participants, and independently optimize their diverging objectives. Specifically, the contracts that bind resource users and providers are decomposed into *containers*. Containers split the allocation into task schedules and resource schedules. A schedule carried out by a participant determines her perceived value of the container.

The outcome of this model is translated in terms of a new architectural design pattern, Symmetric Mapping. Symmetric Mapping separates the concerns of resource users and providers. As a side effect, Symmetric mapping allows to carry out the mapping of tasks to heterogeneous resources according to affinities between tasks and resources, instead of the traditional load balancing on homogeneous resources.

For modeling and tractability issues, participants can only approximately optimize their objective functions. Still, experiments suggest that the proposed separation of concerns yields better outcome than other distributions of responsibilities, and that the gain from traditional load balancing on homogeneous resources can be marginal compared to the gain from dynamically mapping heterogeneous tasks and resources.

Bibliography

[AAC+05] Amitanand S. Aiyer, Lorenzo Alvisi, Allen Clement, Mike Dahlin, Jean-Philippe Martin, and Carl Porth. Bar fault tolerance for cooperative services. *SIGOPS Oper. Syst. Rev.*, 39(5):45–58, 2005.

[All90] Arnold O. Allen. *Probability, statistics, and queueing theory with computer science applications*. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[AMZ03] Gagan Aggarwal, Rajeev Motwani, and An Zhu. The load rebalancing problem. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 258–265, New York, NY, USA, 2003. ACM Press.

[And04] P. Andreetto. Practical approaches to grid workload and resource management in the egee project. In *CHEP '04: Proceedings of the Conference on Computing in High Energy and Nuclear Physics*, volume 2, pages 899–902, Interlaken, Switzerland, 09 2004.

[Ave07] Paul Avery. Open science grid: Building and sustaining general cyberinfrastructure using a collaborative approach. In *Cyberinfrastructure for Collaboration and Innovation*, number CSD5052, June 2007.

[BBB+05] I. Bird, K. Bos, N. Brook, D. Duellmann, C. Eck, I. Fisk, D. Foster, B. Gibbard, C. Grandi, F. Grey, J. Harvey, A. Heiss, F. Hemmer, S. Jarp, R. Jones, D. Kelsey, J. Knobloch, M. Lamanna, H. Marten, P. Mato Vila, F. Ould-Saada, B. Panzer-Steindel, L. Perini, L. Robertson, Y. Schutz, U. Schwickerath, J. Shiers, and T. Wenaus. Lcg technical design report. Technical report, CERN, 06 2005.

[BBC+04] Cyril Banino, Olivier Beaumont, Larry Carter, Jeanne Ferrante, Arnaud Legrand, and Yves Robert. Scheduling strategies for master-slave tasking on heterogeneous processor platforms. *IEEE Transactions on Parallel and Distributed Systems*, PDS-15(4):319–330, April 2004.

[BDF+03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[BF07] Cullen Bash and George Forman. Cool job allocation: Measuring the power savings of placing jobs at cooling-efficient locations in the data center. Technical Report HPL-2007-62, HP Labs, Palo Alto, August 2007.

[BHK+00] Brett Bode, David M. Halstead, Ricky Kendall, Zhou Lei, and David Jackson. The portable batch scheduler and the maui scheduler on linux clusters. In *Proceedings of the 4th Annual Showcase & Conference (LINUX-00)*, pages 217–224, Berkeley, CA, October 10–14 2000. The USENIX Association.

[BHL+06] Stefano Belforte, Shih-Chieh Hsu, Elliot Lipeles, Matthew Norman, Frank Wu thwein, Donatella Lucchesi, Subir Sarkar, and Igor Sfiligoi. Glidecaf: A late binding approach to the grid. In *Proceedings of CHEP'06*, 2006.

[BMB+08] Xin Bai, Dan C. Marinescu, Ladislau Bölöni, Howard Jay Siegel, Rose A. Daley, and I-Jeng Wang. A macroeconomic model for resource allocation in large-scale distributed systems. *J. Parallel Distrib. Comput.*, 68(2):182–199, 2008.

[BMR+96a] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-oriented software architecture: a system of patterns*. John Wiley & Sons, Inc., 1996.

[BMR+96b] Grank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture: a system of patterns*, volume 1. John Wiley and Sons, 1996.

[BR04] Ricardo Bianchini and Ram Rajamony. Power and energy management for server systems. *Computer*, 37(11):68–74, 2004.

[CFK04] Karl Czaijkowski, Ian Foster, and Carl Kesselman. *The Grid 2. Resource and Service Management*, chapter 18, pages 259–283. Morgan Kaufman, 2nd edition, 2004.

[CGV07] Ludmila Cherkasova, Diwaker Gupta, and Amin Vahdat. When virtual is harder than real: Resource allocation challenges in virtual machine based it environments. Technical Report 20070220, HPL, 02 2007.

[CIL+07] Yuan Chen, Subu Iyer, Xue Liu, Dejan Milojicic, and Akhil Sahai. Sla decomposition: Translating service level objectives to system

level thresholds. Technical report, Hewlett-Packard Laboratories, 01 2007.

[CK88] Thomas L. Casavant and Jon G Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEESE*, 14(2), February 1988.

[Cle96] Scott H. Clearwater, editor. *Market-based control: a paradigm for distributed resource allocation*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1996.

[CS00] Xinjie Chang and Krishnappa R. Subramanian. A cooperative game theory approach to resource allocation in wireless atm networks. In *NETWORKING '00*, pages 969–978, London, UK, 2000. Springer-Verlag.

[CvR05] Miguel Castro and Robbert van Renesse. *Peer-to-Peer Systems IV: 4th International Workshop, IPTPS 2005, Revised Selected Papers*, volume 3640. Lecture Notes in Computer Science, Ithaca, NY, USA, February 2005.

[ELvD+96] D. H. J. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne. A worldwide flock of condors: load sharing among workstation clusters. *Future Gener. Comput. Syst.*, 12(1):53–65, 1996.

[FTF+02] James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steve Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5:237–246, 2002.

[GD09] Xavier Gréhant and Isabelle Demeure. Symmetric mapping: an architectural pattern for resource supply in grids and clouds. In *Proceedings of SMTPS'09*, Rome, May 2009. IEEE.

[GDJ08] Xavier Gréhant, Isabelle Demeure, and Sverre Jarp. Towards efficient resource allocation on scientific grids. Technical Report ISSN 0751-1345 ENST D, ENST, Paris, January 2008.

[GHJV95] Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[GPJ+07] Xavier Grehant, Olivier Pernet, Sverre Jarp, Isabelle Demeure, and Peter Toft. Xen management with smartfrog: on-demand supply of heterogeneous, synchronized execution environments. In *VHPC '07: Proceedings of the Workshop on Virtualization in High-Performance Cluster and Grid Computing*, LNCS. Springer, 08 2007.

[Gre07] Derek. Greer. Interactive application architecture patterns. Aspiring Craftsman, 2007.

[Gur03] Yuri Gurevich. *Abstract State Machines: An overview of the Project*, chapter 2, pages 6–13. Lecture Notes in Computer Science (LNCS). Springer Berlin / Heidelberg, Microsoft Research, One Microsoft Way, Redmond, WA 98052, 2003.

[Hum06] Michael Humphrey. Altair's PBS - altair's PBS professional update. In *SC*, page 28. ACM Press, 2006.

[HWZ05] Bernardo A. Huberman, Fang Wu, and Li Zhang. Ensuring trust in one time exchanges: solving the qos problem. *Netnomics*, 7(1):27–37, 2005.

[IGF05] Saeed Iqbal, Rinku Gupta, and Yung-Chin Fang. Job scheduling in hpc clusters. Technical report, Dell Power Solutions, 2005.

[JB07] Janet L. Wiener Jennifer Burge, Partha Ranganathan. Cost-aware scheduling for heterogeneous enterprise machines (cash'em). In *Proceedings of USENIX'07*, Santa Clara, CA, June 2007.

[JZ09] Qiu Jing and Zhou Zheng. Distributed resource allocation based on game theory in multi-cell ofdma systems. *International Journal of Wireless Information Networks*, 16:44–50, 2009.

[KFFZ05] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual workspaces: Achieving quality of service and quality of life in the grid. *Scientific Programming*, 13(4):265–275, 2005.

[KL01] Alexey Kalinov and Alexey Lastovetsky. Heterogeneous distribution of computations solving linear algebra problems on networks of heterogeneous computers. *Journal of Parallel and Distributed Computing*, 61(4):520 – 535, 2001.

[KSS+07] Jong-Kook Kim, Sameer Shivle, Howard Jay Siegel, Anthony A. Maciejewski, Tracy D. Braun, Myron Schneider, Sonja Tideman, Ramakrishna Chitta, Raheleh B. Dilmaghani, Rohit Joshi, Aditya Kaul, Ashish Sharma, Siddhartha Sripada, Praveen Vangari, and Siva Sankar Yellampalli. Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment. *J. Parallel Distrib. Comput.*, 67(2):154–169, 2007.

[Lai05] Kevin Lai. Markets are dead, long live markets. *SIGecom Exch.*, 5(4):1–10, 2005.

[LB06] Colin Low and Andrew Byde. Market-based approaches to utility computing. Technical report, HPL, 02 2006.

[LSV06] Arnaud Legrand, Alan Su, and Frederic Vivien. Minimizing the stretch when scheduling flows of biological requests. In *SPAA '06: Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 103–112, New York, NY, USA, 2006. ACM Press.

[MAS+99] Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hensgen, and Richard F. Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 59:107–131, 1999.

[MKK+05] Anirban Mandal, K. Kennedy, C. Koelbel, G. Marin, J. Mellor-Crummey, B. Liu, and L. Johnsson. Scheduling strategies for mapping application workflows onto the grid. In *HPDC-14. Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing*, pages 125–134, July 2005.

[NS06] Zsolt Nemeth and Vaidy Sunderam. Virtualization in grids: A semantical approach. In Jos C. Cunha and Omer F. Rana, editors, *Grid Computing: Software environments and Tools*, chapter 1, pages 1–18. Springer Verlag, January 2006.

[PRV08] Jean-François Pineau, Yves Robert, and Frédéric Vivien. The impact of heterogeneity on master-slave scheduling. *Parallel Comput.*, 34(3):158–176, 2008.

[PZU+07] Pradeep Padala, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, Kenneth Salem, and Kang G. Shin. Adaptive control of virtualized resources in utility computing environments. In *Proceedings of the EuroSys 2007*, pages 289–302. ACM Press, 03 2007.

[Ram00] Rajesh Raman. *Matchmaking frameworks for distributed resource management.* PhD thesis, University of Wisconsin at Madison, 2000. Supervisor-Miron Livny.

[RCAM06] Jerry Rolia, Ludmila Cherkasova, Martin Arlitt, and Vijay Machiraju. Supporting application qos in shared resource pools. Technical Report 20060118, HPL, 01 2006.

[RIG+06] Lavanya Ramakrishnan, David Irwin, Laura Grit, Aydan Yumerefendi, Adriana Iamnitchi, and Jeff Chase. Toward a doctrine of containment: Grid hosting with adaptive resource control. In *SC '06: Supercomputing, 2006. Proceedings of the ACM/IEEE SC 2006 Conference on Supercomputing*, number 0-7695-2700-0, pages 20–20, New York, NY, USA, 2006. Renaissance Computing Institute, IEEE Computer Society, ACM Press.

[RLS98] Rajesh Raman, Miron Livny, and Marvin Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, Chicago, IL, July 28-31 1998.

[RMX05] Paul Ruth, Phil McGachey, and Dongyan Xu. "viocluster: Virtualization for dynamic computational domains". In *Proceedings of the IEEE International Conference on Cluster Computing (Cluster'05)*, Boston, MA, September 2005.

[Rob06] L Robertson. Status of the lcg project. Technical Report CERN-RRB-2006-112, CERN, Geneva, Oct 2006.

[RRT+08] Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang, and Xiaoyun Zhu. No "power" struggles: coordinated multi-level power management for the data center. In *ASPLOS XIII: Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, pages 48–59, New York, NY, USA, 2008. ACM.

[SBP03] Pablo Saiz, Predrag Buncic, and Andreas J. Peters. Alien resource brokers. In *Proceedings of CHEP'03*, June 2003.

[SOBS04] Hongzhang Shan, Leonid Oliker, Rupak Biswas, and Warren Smith. Job scheduling in heterogeneous grid environment. In *ADCOM2004: International Conference on Advanced Computing and Communication.*, 2004.

[TGV08] Qinghui Tang, Sandeep Kumar S. Gupta, and Georgios Varsamopoulos. Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach. *IEEE Transactions on Parallel and Distributed Systems*, 19(11):1458–1472, 2008.

[TTL02] Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the grid. In *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., 2002.

[Van08] D. C. Vanderster. *Resource Allocation and Scheduling Strategies using Utility and the Knapsack Problem on Computational Grids*. PhD thesis, University of Victoria, 2008.

[Wei98] Jon B. Weissman. Metascheduling: A scheduling model for metacomputing systems. In *Proceedings of High Performance Distributed Computing (HPDC'98)*, pages 348–349, 1998.

[XZQ00] Xiao, Zhang, and Qu. Effective load sharing on heterogeneous networks of workstations. In *IPPS: 14th International Parallel Processing Symposium*, pages 431–438, Los Alamitos, May 1–5 2000. IEEE Computer Society Press.

Dépôt légal : 2010 – 2ème trimestre
Imprimé à Télécom ParisTech – Paris
ISSN 0751-1345 ENST D (Paris) (France 1983-9999)

Télécom ParisTech

Institut TELECOM - membre de ParisTech

46, rue Barrault - 75634 Paris Cedex 13 - Tél. + 33 (0)1 45 81 77 77 - www.telecom-paristech.frfr

Département INFRES