UNIVERSITA' DEGLI STUDI DI CATANIA
DIPARTIMENTO DI INGEGNERIA
INFORMATICA E DELLE TELECOMUNICAZIONI

DOTTORATO DI RICERCA IN INGEGNERIA
INFORMATICA E DELLE TELECOMUNICAZIONI

XXIV CICLO

SLA and Advance Reservation management in Wide-Area Distributed Systems

CANDIDATO: Ing. **Daniele Zito**

IL COORDINATORE
Prof. O. Mirabella

IL TUTOR
Prof. A. Di Stefano

The rare occurence of expected...
William Carlos William

Contents

Bibliography **201**

List of Figures

Chapter 1

Introduction

The terms *Software on demand*, *Software as commodity* or *Software as a service* (SaaS), represent the keywords of a new trend in the field of distributed computing where the main attention is directed towards the users and their needs, in order to compose a set of new market/research segments.
Within this trend, two set of technologies have emerged:

- **Grid technologies** that manage combination of computer resources from multiple administrative domains to reach a common goal. The grid can be thought of as a distributed system with noninteractive workloads that involve a large number of tasks. What distinguishes grid computing from conventional high performance computing systems such as cluster computing is that grids tend to be more loosely coupled, heterogeneous, and geographically dispersed. Although a grid can be dedicated to a specialized application, it is more common that a single grid will be used for a variety of different purposes. Grids are often constructed with the aid of general-purpose grid software libraries known as middleware;

- **Cloud technologies** that provides computation, software, data access, and storage services that do not require end-user knowledge of the physical location and configuration of the system that delivers the services. The term cloud computing refers to a new supplement, consumption, and delivery model for IT services based on Internet protocols, that typically involves provisioning of dynamically scalable and often virtualised resources.

Through them, web is becoming a marketplace where each user can find, use and compose services and applications offered, with different qualities, from different providers. From the users' point of view, this scenario allows them to have *what they want*, *when they need it*, with a specific *quality of service*, without having to care for the management or ownership of hardware and software resources. From the service provider's point of view, instead, the ability to offer all these functionalities implies the adoption of an effective strategy for services and resources management: the providers have to cope with new and complex challenges in order to guarantee the satisfaction of the user's requests.

The aims of this thesis are to investigates these challenges both in a wide area Grid based service marketplace and in a Cloud based multicore environments, proposing for each one an effective solution and discussing the obtained advantages.

1.1   Grid Issues

The grid computing paradigm, born in the mid 1990's to enable the sharing of computational and storage resources among academic and research institutions, has been, in recent years, deeply influenced by the SOA advent.

Today, the grids are opening towards new applications, embracing not only the e-science field, where it is born and developed, but also the business, financial and educational ones.

The grids, now, have to be able to supply resources and services in a flexible manner, exposing and offering them on-demand to different typologies of users, each one characterized by specific requirements. It is creating a competitive grid services marketplace where the user's satisfaction of the QoS requirements becomes the fundamental issue. In fact, this grid evolution offers to the users a great number of grid services easily accessed via web. It can choose and use those services that better satisfy these requirements, obtaining them by different grid providers based on the services performance and cost.

In this distributed scenario, characterized by the lack of a centralized coordinator, some challenges have to be faced in order to guarantee its correct functioning:

(i) each grid, i.e. each service provider, has to guarantee the service fruition respecting the user's QoS requirements.

(ii) the user has to rely on a robust and scalable algorithm to find services under QoS constraints among the different providers.

(iii) an effective strategy has to be designed to manage QoS agreement for composed services belonging to different providers.

One of the aims of this thesis are to investigates these challenges in a wide area P2P-Grid based service marketplace, proposing for each one an effective solution and discussing the obtained advantages.

1.2   Cloud Issues

Cloud computing is a natural evolution of the widespread adoption of virtualisation, service-oriented architecture, autonomic, and utility computing. Details are abstracted from end-users, who no longer have need for expertise in, or control over, the technology infrastructure textitin the cloud that supports them.

Virtualization, in particular, is a key enabling technology for cloud computing environments. The concept of cloud computing has captured the attention and imagination of organizations of all sizes because its service delivery model converts the power of virtualization into measurable business value by adding the provisioning and billing capabilities.

Today, computing virtualization is provided with Hypervisor technology to create virtual servers, network virtualization is provided through multiprotocol routers and switches and storage virtualization is provided through specialized appliances supporting NAS and SAN. New appliances are being rolled out for databases and storage transaction management.

Different virtualization platforms and orchestrators that integrate them are flooding the market. The costs of associated services are skyrocketing. Figure 1.1 shows various layers of management to provide application specific availability, reliability, performance, security and billing functions. The products and services that have evolved bottom up in the services stack from the server, network, storage or application and infrastructure software domains are expanding their reach into other domains to gain market share.

Various vendors play at various levels in each layer. The complexity of heterogeneity, multiple vendor solutions and orchestrators that provide integration has been overwhelming the service developers, operators and consumers.

The infrastructure makers, the service developers and the service operators are
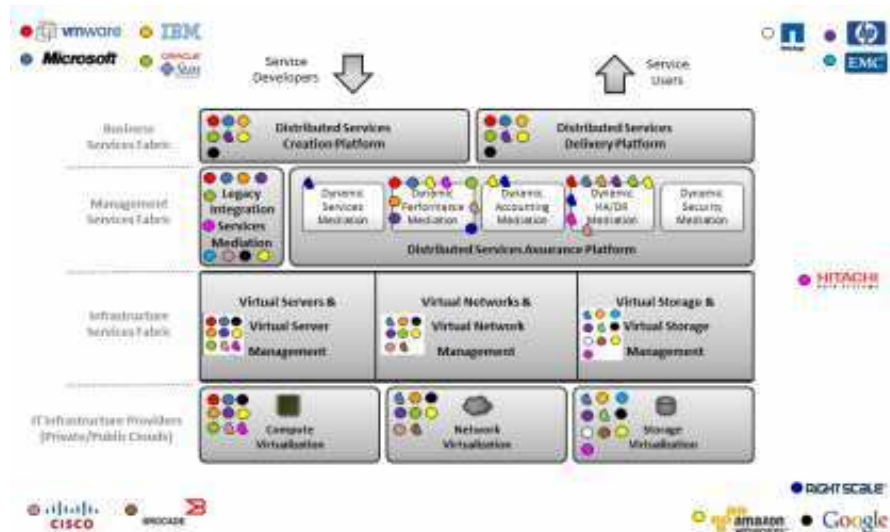
Figure 1.1: Current Cloud Market Landscape

striving to capture the big chunk of a large market share by racing to provide universal access to virtual computing services with telecom grade trust. There is a battle brewing between the Äppliances for High Performance Campänd the Öpen System Software and Services Approach for Everything Campöf the current IT infrastructure vendors. By this way, the current cloud solutions present similar issues:

1. Poor end-to-end distributed transaction reliability, availability, performance and security as recent episodes at Sony, Amazon, Google, and RSA [1–3] demonstrate.

2. The hardware upheaval caused by the new class of many-core processors that allow parallelism which cannot be fully exploited with current state of software innovation.
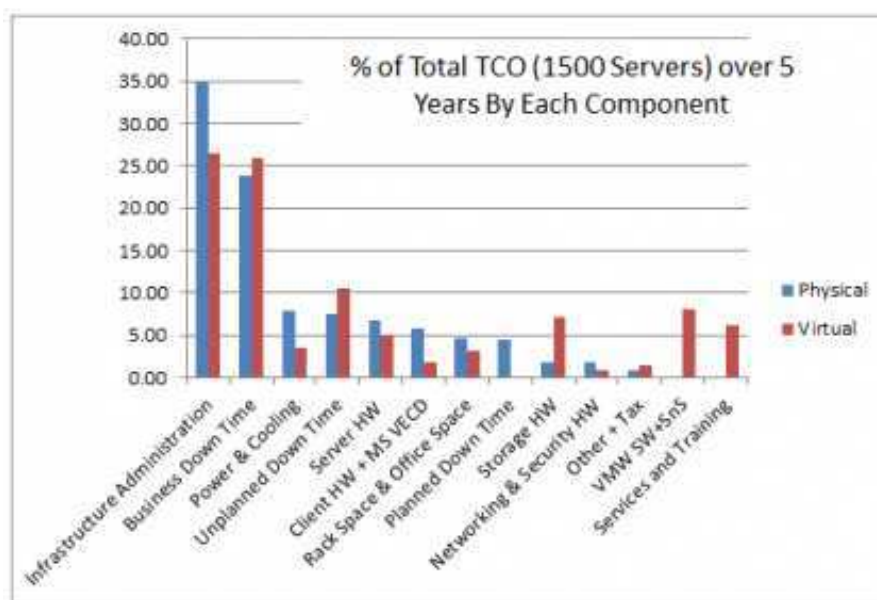
Figure 1.2: Five Year TCO of Virtualization According to a Vendor ROI Calculator

It is estimated that 60% to 70% of IT data center cost is in its operation and management with or without virtualization in spite of a 10X improvement in hardware, space and energy savings with the new class of servers available today [4]. Figure 1.2 shows percentage Total Cost of Ownership (TCO) (for a 1500 server data center) over five years by each component with and without virtualization. While virtualization introduces many benefits such as consolidation, real-time business continuity and elastic scaling of resources to meet wildly fluctuating workloads, it adds another layer of management systems in addition to current computing, network, storage and application management systems. 1.3 shows a reduction by 50% of the five-year TCO with virtualization. The Virtual Machine density of about 13 allows a great saving in hardware costs which is somewhat off-set by the new software, training and
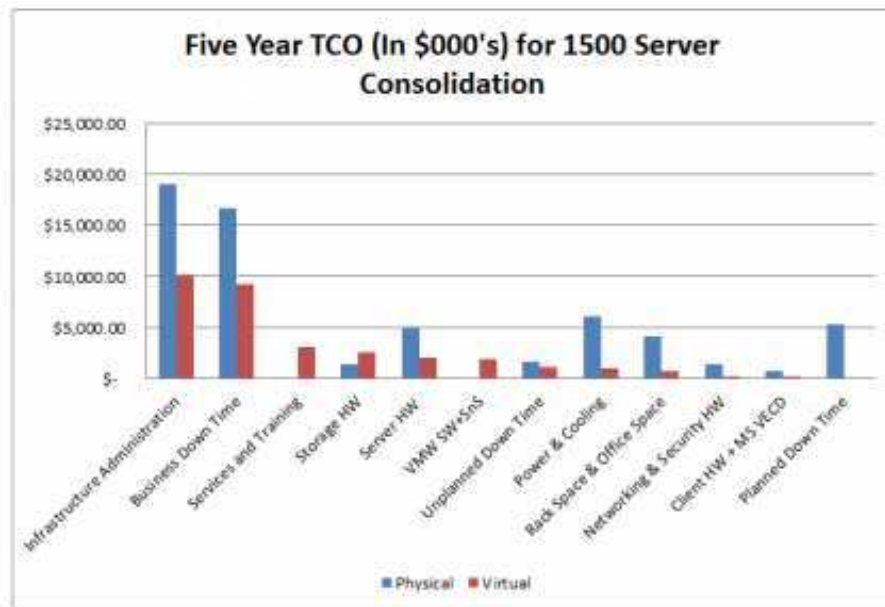
Figure 1.3: TCO over 5 Years with virtualization of 1500 servers using 13 VMs per Server

services costs of virtualization. In addition, there is the cost of new complexity in optimizing the 13 VMs within each server in order to match the resources (network bandwidth, storage capacity, IOPs and throughput) to application workload characteristics, business priorities and latency constraints. The cost per VM is estimated to be around $2500 with minor variation with the use of VMWare, Microsoft, Red Hat or Citrix solution. This is consistent with 2X improvement with a managed physical server cost of about $5000. In spite of vendor claims, there is not much difference between different Hypervisors just as there was no big difference between DB2, Oracle and Sybase in the 1980ś. They are all equally knowledge intensive and require expensive services solutions to maintain. Further improvements in TCO have to come

from new approaches that drastically reduce complexity of current layers of management systems. This thesis focuses on this issues, investigating a set of solutions that are able to increment the performances of a cloud both i) extending the existent solutions with a new pattern for the resources management and ii) proposing a new approach based on biological principles.

1.3 Basic concepts

In this section it will be given a brief description of the basic concepts in QoS management in Wide-area distributed systems..

1.3.1 QoS, SLA and resource reservation

Quality of Service (QoS) is a concept born in the networking field to indicate some aspects related to the communication quality such as transmission delay, number of packets loss or percentile of corrupted information.

With the advent of the SOA, the term QoS has assumed a more general meaning: today, the concept of quality is based on the perception that a user has of the service execution and it is related to the value of some (often called) *non-functional* parameters, such as e.g. service response time, reliability, availability and cost.

QoS represents an important aspect in a services marketplace, since it offers the user a choice of, on the basis of their expectations, the best one among different versions of a service.

In order to reach an agreement about functional and QoS parameters related to a service execution, the client and the provider have to negotiate the type and the value of this parameter. The result of this negotiation process is a specific document known as Service Level Agreement (SLA): it is used for describing

all the parameters characterising a service, both functional and QoS related, and the rules and conditions to properly use the service. In particular, the SLA specifies, for a proper subset of parameters, the corresponding values or range of values expected by the service requester and that the service provider is able to offer. Many implementations of an SLA are possible (WSLA, WS-agreement, SLAng), each one taking care of some specific characteristics of the context in which it is used. Generally, an SLA implementation should contain technical specifications and QoS related parameters. Technical specifications include:

- service name,

- service description, in terms of input/output parameters and availability,

- participants: provider, consumer and any *third party* needed to ensure a trusted service,

- service access mode, i.e. involved protocols and exchanged messages.

QoS parameters may include response time, throughput, availability, service cost, etc. Two important aspects have to be considered in the SLA management. The first item pertains to the opportunity of using, in the SLA management, standard structures that define the behaviour that the provider and the consumer have to adopt to reach a common target. These structures, called *SLA templates*, considerably simplify the agreement creation phase because they make transparent the issue related to "how to provide", focusing on "what is provided". Thus, the agreement process is reduced only to the negotiation of the parameters.

The second item, fundamental in the proposed SLA management system (see Chap.5), is the capability of establishing specific boundaries when recovery

activities have to be started due to an agreement violation. An SLA is a binding agreement: a provider, that is selected based on its promised performance, is bound to guarantee the proper execution of the service. If this does not happen due to the provider, the client must be refunded, based on both the service level and on the reason of the problem, in a partial or total way.

The exceptions are very important when they have to do with composed services because they allow to build recovery paths for the faults.

The provision of a QoS level with a certain degree of guarantee, implies the existence of a mechanisms able to set and monitor this guarantee, otherwise an SLA would be impossible to draw up. The service client, in fact, requires the overconfidence that its own requests are satisfied following the negotiated behaviour.

One of the more frequent situation in the Grid utilization is the request of allocation of resources to execute one or more services. In this context, the *best effort* strategy allows the users only to request a fixed amount of resources, but it does not provide any guarantee that the service will be executed in a given time range or that the service will start its execution at a fixed time.

For more users, this strategy is too restrictive: e.g for the execution of a particular class of services strongly dependent by time constraints or for the composition of a set of services characterized by time dependence among them. Similar questions pertain to resources different from the ones related to CPU allocation, like bandwidth. The execution of a particular service is strongly affected by the bandwidth between the node where they are executed the different jobs that compose it. In a purely *best effort* logic, the middleware cannot influence the provision of the service, because it has no tools to guarantee the bandwidth required for an optimal supply of the service. In order to guarantee the execution of the services respecting the user's expectations, the provider has to be able to reserve for it all the needed resources. The ad-

vance reservation, then, plays a crucial role in the provisionning of service with guaranteed QoS parameters. In the absence of mechanisms of advance reservation, in fact, the provider/resources broker can not provide any guarantee on the service provision/exectution. The adoption and the provisioning of a *quality of service(QoS)-based* management of resources in Grids (able to guarantee the respect of the agreements established between services consumers and providers) represents a fundamental requirement in order to obtain a flexible and dynamic services management.

Recognizing the centrality of the reservation in the actual research about Grid middleware, in the first chpater of this thesis, a new architectural pattern for the management of advance reservations in grid environments called Advance Resource Manager (ARM), is proposed.

1.3.2 From Grids to Services MarketPlace

Grid systems and Peer to Peer networks are the most commonly-used solutions to achieve the same goal: the sharing of resources and services in heterogeneous, dynamic, distributed environments. Many studies have proposed hybrid approaches that try to conjugate the advantages of the two models. This thesis proposes an architecture that integrates the P2P interaction model in Grid environments, so as to build an open cooperative model wherein Grid entities are composed in a decentralized way. In particular, the chapter 4 focuses on a QoS aware discovery algorithm for P2P Grid systems, analyzing protocols and explaining techniques used to improve its performance.

Today, Grid systems are applied in widely distributed systems where several administrative domains offer a huge amount of heterogeneous resources and services to meet pervasive and heavy requests with a number of different QoS and security constraints.

The standardization and the implementation of Grid systems are presently strongly structured in centralized or hierarchical architectures. This allows a single organization to adopt simple, even if rigid, strategies to guarantee the control of the owned resources and the maintenance of the designed security degrees. However, the use of a centralized approach compromises both scalability and reliability of the Virtual Organization [5] (a group of individuals or institutions belonging to different administrations who share the computational and storage resources of a Grid for a common goal), especially in terms of robustness and fault tolerance, reducing the effective exploitation of the resources and their availability issues.

Some studies [6], [7], [8] have explored the advantages of implementing Grid systems adopting Peer-To-Peer (P2P) models and techniques to manage the services even if critical concerns related to the decentralized control and the security issues remain.

This thesis proposes to combine Grid middleware facilities with the P2P approach, exploiting the capability of peer/CE to handle cooperation issues. One of the most critical issues that has to be taken into account when Grid and P2P systems are combined regards the resources/services discovery and services composition

1.3.3 Services MarketPlace Architecture

The term *P2P grid* architecture in this thesis indicates an overlay network where each peer is represented by a Computing Element (CE). Referring to the grid terminology [9], each CE is the master node of a computers cluster: it's aim is the distribution of tasks and data among other cluster machines, called Worker Nodes (WN). In a classical Grid organization (Fig. 1.4), the service requests are sent to CEs from a centralized Resource Broker (RB) that
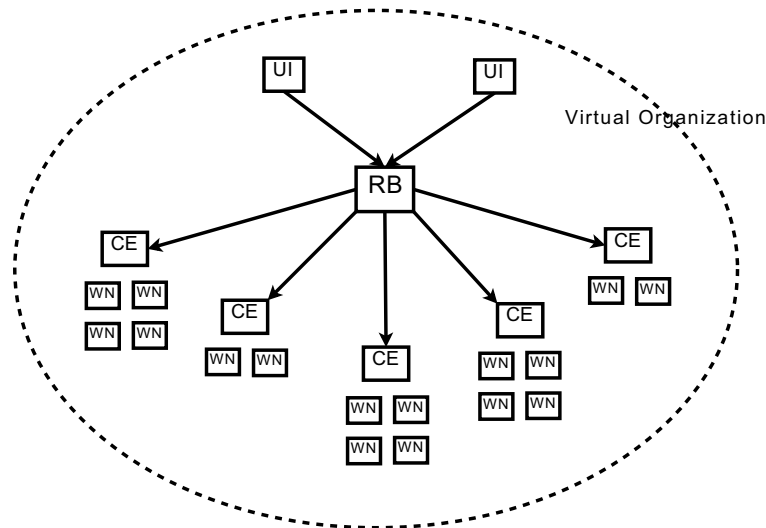
Figure 1.4: Grid

receives them from several sources, called User Interfaces (UI), that represent the user's access points to shared VO resources.

In the P2P schema here proposed, shown in fig.1.5, the concepts of centralized RB and UI are missing: the jobs arriving at each CE (peer) come from remote sources (e.g. a single user or another CE looking for services or resources not available in its own local environment) displaced over a wide area. This type of organization gives each peer a flexible way to interact with a great number of other peers: from the view point of a user (peer) that wants to exploit a generic service, the P2P grid environment can be considered as a *services marketplace* in which several providers offer the same services with different features in terms of availability, reliability, performance, cost and many other aspects related to non functional parameters of services.

In the proposed architecture, each CE holds an *ad hoc* QoS-aware services manager [10], that:

1. represents the access point to the services supplied by WNs registered on the peer;

2. handles (publishes and monitors) information about the set of services provided by the underlying WNs;

3. manages services execution;

Each service is characterized by a set of parameters used for QoS management. E.g. parameters like Operating System, processor type and computational power are typically associated with a job allocation service, while the type of memory, its free space and bandwidth available for data transfer are parameters associated with a distributed storage allocation service.

The services are classified according to QoS level [11] based on their features and their prices.

1.3.4   Cloud over multicore systems

The Cloud Computing paradigm decouples the hardware infrastructure and the virtual computing infrastructure, thus offering a grid of virtual computing, network and storage resources to implement a virtual grid services network, overcoming the limitation of present grids due to their rigid schema. For this reason cloud computing is, today, the most significant and diffused example of Services Oriented Architecture.

This is mainly due to the widespread adoption of virtualization technologies that, ensuring the isolation among the VMs, allow Cloud to manage a huge amount of software and hardware resources in a simple and extremely flexible way.

However, if not properly managed, the virtualization can adversely affect the
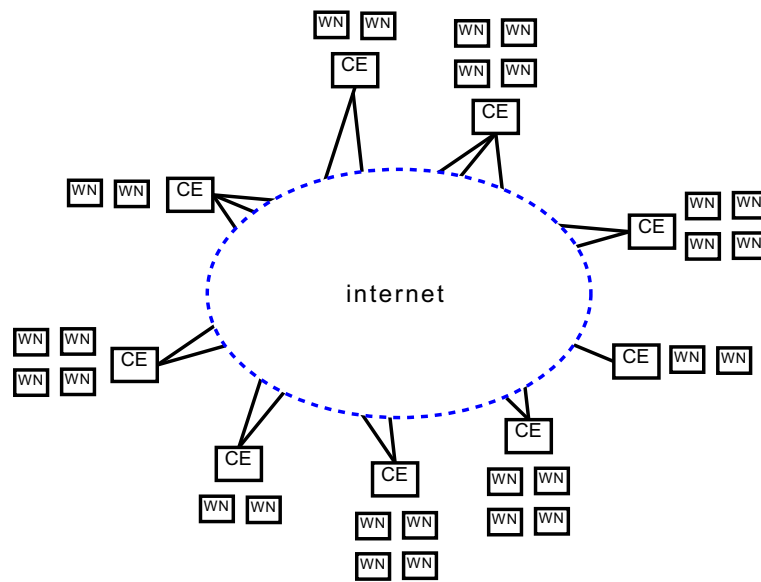
Figure 1.5: P2PGrid

underlying hardware performance: for instance, the current solution are not able to exploit, by default, all the functionalities offered by the multicore systems.

Moreover, these solutions do not guarantee the performance isolation among virtual machines and this makes impossible the creation of any form of QoS-guaranteed services management.

This requires the full control over the hardware resources: every QoS-aware request done by an user, containing parameters as execution time, waiting time, security and robustness, is translated into specific physical and logical constraints about the number and the types of resources managed at low layer.

For this reason I propose a Resources Management System (RMS) based on ARM, the behavioral pattern investigated for the grid environments (see 1.3.1. RMS is designed to provide, through VM allocation and advanced resource reservation, QoS-guaranteed services in a multicore-based cloud system.

However this solution mitigates, but does not solve the actual issues about the exploitation of the multicore computing resources.

More recently, David Patterson [4] asserts that taking real advantage of multi-core processors is limited by our inability to figure out how to make dependable parallel software that works efficiently as the number of cores increases. He concludes with a pessimistic note. Talking about dependable parallel software, he says: *Although Iḿ rooting for this outcome and many colleagues and I are working hard to realize it I have to admit that this third scenario is probably not the most likely one.* He cites many casualties from the past such as *Ardent, Convex, Encore, Floating Point Systems, Inmos, Kendall Square Research, MasPar, nCube, Sequent, Tandem, and Thinking Machines* as the most prominent names from a long list of long-gone parallel

hopefuls. He also cites the list of languages designed to support parallel processing and points out their inadequacy in making parallel programming easy or straight forward.

While the evolution of current software architectures starting from the operating system have evolved from a server centric architecture where the CPU resource is scarce and is shared to perform multiple tasks, the multi-core Processors promote a contra architecture where each CPU has multiple threads and multiple CPUs can be networked to perform tasks in parallel.

Todayś server-centric operating systems are more suited for operating on the resources in a virtual computer in a virtual grid rather than managing distributed resources that contribute to a virtual computer in the cloud. A distributed OS that assures Fault, Configuration, Accounting (of utilization), Performance and Security of a composition of distributed resources that constitute a virtual server is required to dynamically provision service levels such as the CPU, memory, bandwidth, storage capacity, IO and throughput on demand.

According to these considerations, in the last chpater of my thesis, I will revisit the design of distributed systems with a new non-von Neumann computing model (called Distributed Intelligent Managed Element (DIME) Network computing model) that integrates computational workflows with a parallel implementation of management workflows and provide dynamic real-time FCAPS management of distributed services to provide end-to-end service transaction management.

1.4   Contributions

This thesis proposes several novel research contributions in the field of QoS management, both in Grid and in P2P-Grid scenarios. In particular:

- The thesis proposes an innovative resource reservation pattern (called ARM) able to manage both immediate and advanced reservation. This reservation policy introduces two important new features. The first one regards the ability to allow the agreement renegotiation: the user can requests to modify the amount of time and resources at run time. The second feature regards the the ability to use a *resource-task* association instead of the classical *resource-user* one. The *resource-user*, adopted e.g. in LSF and in PBS, consists of reserving one or more resources for a user, where it can allocate a set of tasks. The use of this technique, however, does not allow a differentiation among the user's task: the agreement is taken for the whole set of tasks, making too complex the renegotiation mechanism for a single one. Instead, the choice to use the *resource-task* association makes the reservation management very flexible because it allows to discern the requirements and constraints of each task from the others and gives the ability to modify selectively the negotiated agreements.

- The thesis proposes also a gLite framework for the SLA Management (called SLAM). It takes into account the issues related with the creation, monitoring, modification, termination of SLAs in a grid environment, traslating it in specific directives for the resource exploitations. It is strictly related with ARM, beacuse the management of a service with a specific QoS operated by SLAM requires a set of *low level* mechanisms that allow the Computing Element to manage and monitor and

guarantee the resources allocation with required QoS.

- The thesis proposes an unstructured distributed services discovery protocol able to search services under QoS constraints. The ability to search services under QoS constraints is an important innovation in the field of discovery algorithms for distributed systems. It represents a fundamental value-added in a service market place scenario in that it allows not only to identify different providers for the desired services but also to discern among them the most suitable one to satisfy the user requirements.

- The thesis proposes two different strategies to handle QoS assurance for service composition both in grids and in P2P-Grid based services marketplace scenarios. In particular, both strategies face the issues related to services management when unexpected faults occur at runtime. The proposed strategies handle re-negotiation by finding other (better) services performing a runtime adaptation with the aim of avoiding degradation of the level of service that the user expects.

  The renegotiation and the runtime adaptation have been made possible by reaching the SLA, used to establish the agreement with each service provider, with an additional section, the *exception* section, which defines the penalties for the provider and the recovery activities that have to be started if an agreement violation happens.

- The thesis proposes the translation of the ARM pattern (previously defined for the grid environments) in the cloud environment, so as providing QoS-guaranteed services in a multicore-based cloud system, through VMs allocation and advanced resource reservation.

- Finally the thesis proposes new computing model called the Distributed

Intelligent Managed Element (DIME) network computing model to
overcome the actual limitations of cloud/multicore environments. This
model allows the encapsulation of each computing engine (a universal
Turing Machine) into a self-managed computing element that collabo-
rates with a network of other such elements to execute a workflow im-
plemented as a set of tasks, arranged or organized in a directed acyclic
graph (DAG).

1.5   Organization

The rest of the thesis is organized as follows.

Chapter 2 describes the ARM behavioral pattern. It focuses on the man-
agement of advance reservation and allocation of computing resources in a
gLite-based grid environment.

Chapter 3 proposes a web services based framework for SLA in grids. The
chapter explains the components of the framework and the main algorithms
used to guarantee QoS-based service in a gLite middleware.

Chapter 4 discusses the design of a QoS-aware services discovery algorithm
in the above mentioned P2P-Grid scenario. The chapter explains the dis-
tributed search strategy, all the involved messages and gives an overview of
the algorithm performance.

Chapter 5 analyses the issues related to the QoS management for services
composition. It discusses the meaning of composed QoS, provides the con-
cept of exception management in SLA management and proposes two tech-
niques to handle quality assurance in grids and in P2P-Grid scenarios.

Chapter 6 proposes a Resources Management System (RMS) based on ARM,
to provide, through VM allocation and advanced resource reservation, QoS-

guaranteed services in a multicore-based cloud system.

Chapter 7 discusses DIME, a network computing model able to create distributed computing clouds and execute distributed managed workflows with high degree of agility, availability, reliability, performance and security.

Finally, *Chapter 8* presents the thesis' conclusions.

1.6 Acknowledgments

Chapter 2

Immediate and Advanced Reservation

Currently, concepts as *execution time*, *costs*, *services security* and *availability* are becoming more and more important; the ability of supplying services that comply agreements related to QoS parameters [12] is now a basic requirement for a provider who wants to show better than from the other in a competitive scenario. This trend strongly influences also the grid middleware, that are changing quickly to follow these new concepts of services provisioning, although there are still many difficulties to face.

The strategy of executing all services requests with the same settings, without any distinction (neither related to users nor to applications), typical of the classical grids, is no longer able to provide sufficient guarantees on the enforcement of users requirements. Many Grid environments still operate only on a *best effort* basis, sharing the resources among users little or too strictly to distinguish the different needs of users.

In the emerging user centric service marketplace, the Grid middleware are
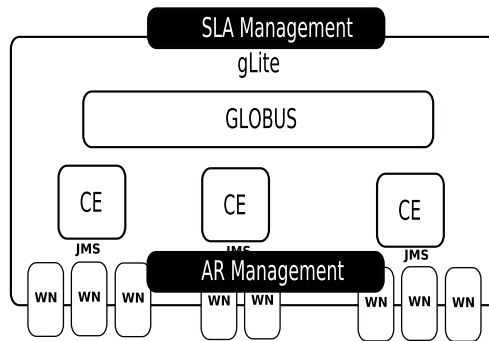
Figure 2.1: Two levels for the SLA management

called upon manage complex service requests specifying QoS parameters
(like maximum *execution* or *queue waiting* time) able to influence how they
have to execute.

This new way of booking service at delivery requires a deep re-design of the
software components responsible for managing heterogeneus and *changing in
time* user requests in a scalable and flexible way.

  My work poses exactly these goals: it focuses both on the modelling of a
QoS-aware job submission service in a Grid environment and on the proposal
of suitable protocols to manage different kinds of constraints on supplying
QoS guarantees.

To do this, I refer to the typical Grid architecture where a Resource Broker
(RB) (the service providers), forwards the user jobs to Computing Elements
(CEs), each responsible of managing all the resources of a cluster and allocat-
ing the job to the Worker Nodes (WNs) for running. Inside these architectures,
I identified two different SLA management levels (see fig. 2.1):

- an high level (on the Resource Broker) that receives and negotiates user
  QoS-aware requests, translates them in constraint for the underlying

resources and creates related SLA;

- a low level, within CEs, that imply a scheduling policy to identify the more suitable resources distribuition matching the user requirements and accordingly reserves them to guarantee the agreements.

These two level are strictly related: the management of a service with a specific QoS at high level requires a set of low level mechanisms that allow the CE to manage and monitor and guarantee the resources allocation with required QoS. The advance reservation, then, plays a crucial role in the provisionning of service with guaranteed QoS parameters. In the absence of mechanisms of advance reservation, in fact, the provider/RB can not provide any guarantee on the service provision/exectution. Recognizing the centrality of the reservation in the actual research about Grid middleware, here, a new architectural pattern for the management of AR, called Advance Resource Manager (ARM), is proposed. It guarantees the satisfaction of specific user requirements of a set of jobs submission [1] on the resources collected in a a gLite-based [13] grid environment. In other worlds, ARM is able to intercept the single reservation requests from users and to translate them in appropriate directives for the underlying Job Managemen Systems - JMS (responsible of job allocation on the WNs), keeping the status of CE in an information repository and monitoring the positive completion of all operations involved in the reservation.

ARM is able to provide reservation even if it does not foresee by the JMS [2].

At present, ARMis implemented within one of our research activities inside the PI2S2 project [14]. It is integrated with the two technologies used within the project PI2S2 to build the grid: the middleware gLite and the job manage-

---

[1]even only one job

[2]Notice that no hypothesis is made on the features supplied by the JMS

ment system LSF. This choice does not affect the extendibility of the compo-
nent realized, because the design of the pattern is general enough to be easily
extended to other technologies(like Globus, Unicore, PBS, etc.).

In this chapter, I propose also a set of solutions to mitigate the drop of per-
fomances related to the introduction of AR in a grid environment. Tipically,
in fact, reserving a set of resources, decrementing resources availability, de-
creases the average utilization of a CE and increases the waiting times of
queued applications that require simple allocation (see 2.4 for more informa-
tion).

I tested the opportunity to use a modification of the *classical* backfill algo-
rithms [15] to improve the performance indexes of the CE and led simulations
to conferm their hypotesis.

2.1   Some notes on SLA and QoS

2.1.1   QoS profile, best effort and QoS-guaranteed jobs

A *QoS profile* defines a set of parameters and values characterizing the per-
formance expectations about service execution. It can be associated with a
service request and comprises a given set of QoS parameters. According to
many studies about QoS, in a Grid it'possibile to distinguish two types of QoS
attributes:

1. those based on the quantitative characteristics of the Grid infrastructure.
   They refer to aspects such as network latency, CPU performance, or
   storage capacity (e.g. for the networks, quantitative parameters are:
   delay, delay jitter, throughput, packet-loss rate, etc.). It's difficult to
   measure these measures objectively.

2. those based on the qualitative charactericstics. They refer to aspects such as service reliability and user satisfaction.

We can distinguish *best effort jobs*, the jobs that require the simple execution of a service without defining no QoS profiles; while, conversely, we define *QoS-guaranteed jobs* the jobs that require a service with a specific QoS profile.

In this contribution, we restrict our resource model to CPU resources and we consider initially only services that exploit these typology of resources (tipically submission and allocation services).

In order to introduce diversification related to CPU performance, we introduce different service level for mulitple classes of CPUs with respect to their performances (in terms of GFlops/s). Each level is charaterized by a performance ratio which express the performance bonus of using a CPU from a particular level compared to a CPU from lowest level. Obviously, CPU belonging to the same type but to a different level are substitutable.

Each consumer has a queue of CPU-bound computational jobs that need to be executed and for which resources must be acquired form providers through participation in the market. The dispatch of a job to the CPU is effected by the ARM. Every job has a nominal running time, i.e. the time it takes to finish the job on a reference CPU.

2.1.2 SLA

A Software Level Agreement (SLA) is a formal contract stated between a client and a service provider, containing all the information about services exploitation (as *protocol adopted*, *messages exchanged*, *third-party entities involved* and *penalties in case of agreement breach*). Among these, great importance have the list and the values of both functional and non functional

(QoS) parameters negotiated between the two parties.

For a *job submission* service, the one considered in this chapter, an SLA can contain several aspects related to the temporal deadlines (as *maximum time* spent for job execution, *maximum queue waiting time* before job allocation or *maximum time* needed to gather the job results); to the job dependability (as *replication, checkpointable jobs*) or to the job security (*results' cryptography*).

As said in the introduction, the high level capability to provide (i.e. create, monitor and maintain) SLA-based and QoS-aware services implies that the provider is able to manage, at low level, its own resources in order to guarantee the fulfilment of the requirements stated in the SLA. This means that a services provider has to translate all the QoS parameters negotiated with the consumer in specific constraints on underlying resources. So e.g. if an user submits a job requiring a queue waiting time as short as possible, the provider has to map its execution on the node having the lowest number of jobs in its own execution queue. If, instead, an user submits a job requiring a specific delivery time for a given execution time foreseen, the provider has to forward the job toward a node able to satisfy this requirement reserving an adequate free slot time. If the user requires that job has to be replicated in two different nodes and that the results have to be encrypted, the provider has to find two nodes which not only have to be free for the specified time range, but also have to contain the needed software libraries for data encryption.

2.2 Current implementations of allocation and AR in the Job Management System

In a classical cluster administration, the key components that face the issues related with the schedulation, exploitation, and monitoring of the cluster's resources is called: Job Management System (JMS). It takes care on: i) the resource matching allocation policy: for each job, the JMS has to find the suitable resources able to satisfy the job constrains; ii) the job scheduling: for the execution of each job, the JMS has to select one or more resources among the suitable ones identified in the previous item; they are chosen according to various optimization strategies (for example, avoiding the overloading or the underutilization of some resources); iii) the job state monitoring: the JMS has to continuously collects information about the jobs life-cycle in order to supervise their correct and complete execution.

Obviously there are several implementations of JMS [16, 17], developed by severl software companies. Howevee, all these implemantation are featured by a common set of software components:

- ***User Server***: this component represents the JMS user interface. It allows users to submit their jobs and their related requirements (i.e. type and quantity of needed resources), to the JMS. Moreover, it answers to queries about jobs state and allow users to terminate or suspend their own jobs.

- ***Job Scheduler***: thid component distributes jobs among the available suitable resources according to specific rules related to system and users requirements.

- ***Resource Manager***: this component tipically consists of two main

components: Rosurce Monitor collecting information about the under-
lying resources; and Job Dispatcher allocating resources and starting
the jobs execution.

Usually, a basic JMS address only the allocation of the jobs on the available
resources, without supporting any advance reservation of resources; in order
to handle Advance Reservation, the architecture of JMS has to be enriched
with two new components, as shown in fig.2.2:

- ***AR Manager***: handles the *reservation life cycle*: it (i) checks if a spe-
  cific reservation is possible, (ii) provides an handler to access the reser-
  vation info, (iii) monitors reservation state, (iv) allocates the needed
  resources and (v) gathers the output of job execution.

- ***AR Mon***: monitors the current reservations and collects information on
  the resources still available for future reservations. Indeed, while an im-
  mediate or *just in time* allocation requires to know only the present state
  of the resources (i.e. a "snapshot" of the cluster's resources state), in
  case of *advanced reservation* it has to be able to foresee how the future
  state of resources will evolve, in order to determine where resources
  can be reserved.

2.2.1   Advanced Reservation Manager Pattern

All the above considerations led me to propose the architectural pattern
"*Advance Reservation Manager*" (ARM), according to the well known
schema briefly described in [18]. This paragraph will explain the objectives,
the software components involved in ARM and their interactions.
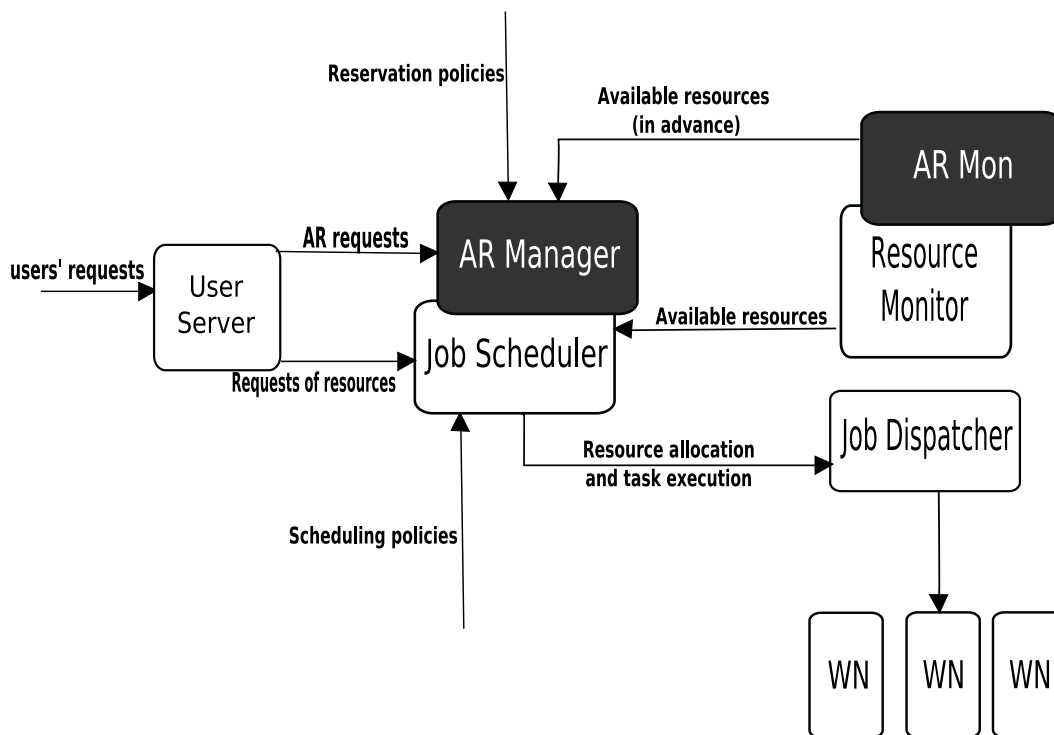
Figure 2.2: JMS Architecture

Pattern objectives

The objective of the proposed pattern is the provisioning of a robust, flexible and easy-to-adapt mechanism to manage both *immediate* and *advanced* resources reservations.

In particular, the pattern:

- provides a general support for different grid middlewares for resources allocation and advanced reservation.

- enforces QoS resources management taking into account requirements, constraints and, in general, all the parameters characterizing the reservation requests;

- exploits the functionalities of underlying Job Manager System (in this case the platform LSF/EGO) to perform respectively *job submission*, *job execution* and *resources state updating* on behalf of the reservation menager;

Pattern structure

The structure of the ARM pattern, shown in the Fig. 2.3 consists of five main elements: ARMServer, Table, TableManager, ARMController and Job Allocation Module (JAM).

- *ARMServer* is a event-driven proxy responsible for accepting the reservation requests coming from external clients/users (or software agents working on behalf of them), for parsing them and, finally, for forwarding valid requestes to the ARMController.

- ARMController is the *reservations coordinator*. It performs all the activities related to the resource reservation: it looks for the availability
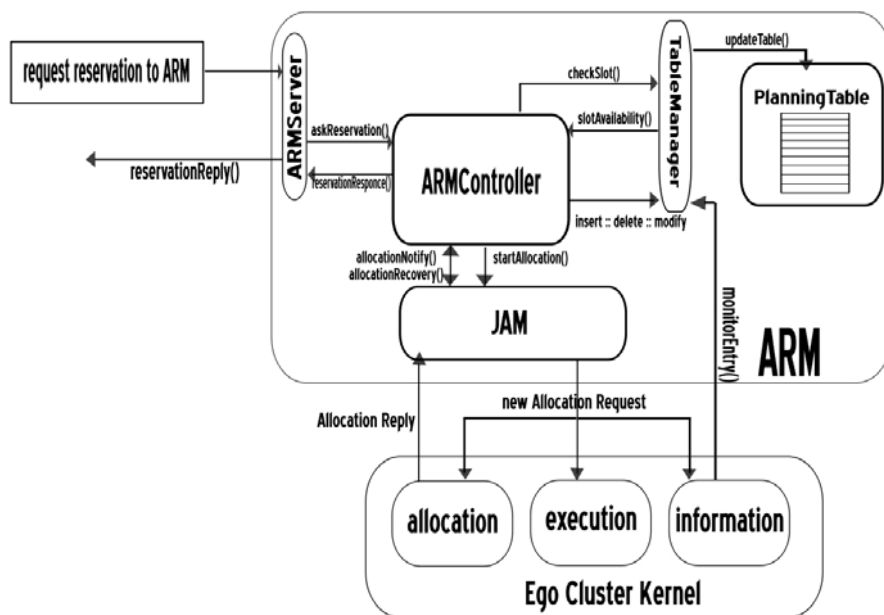
Figure 2.3: ARM architecture

of the resources, manages the reservation lifecycle, monitors the status of active reservations and finally coordinates the execution of best effort jobs in the slot times not reserved. ARMController also includes a local fault recovery engine that, in case of errors, tries to restore, where possible, the correct job execution.

- JAM (Job Allocation Manager) is the component responsable for allocating jobs. It consists of three main threads that exploit the functionalities offered by the underlying JMS (see 6.3.4 for more information).

- The Planning Table represents the information source of the reservation system. By means of it, ARM can obtain the information about the actual and future status of the resources held in a CE.

  In ARM the information are organized in a double hashed table (the reasons of this choice are expplainned in /refTime). The first hashtable indexes all the resources by means of their identificative (primary key) and the value of their QoS (secondary keys). Each item contains a second hashtable that holds the information about the state of the specific resource for the whole time horizon. This last uses the start time as primary key and the duration as secondary key.

  The items in the second hashtable are clusterized in epoches (see 2.2.3 for more information). The information held in the double hashed table are managed by two software components: a *listener* and a *daemon*. The *listener* monitors every change on the resource. Once an status changing has been revealed, the listener start a routine of updating of the proper item in the hashtable. The *daemon* periodically refreshes the items of hashtable. This means: i)updating the information on the state of any resource on the basis of listener's directives; ii) deleting all the items held in the second hashtable that refers on the actual time; iii)

keeping track of all the unused resources, so as to assign them to the best effort jobs; iv) for each resources, inspecting all the reservations related to the first epoch, to search potential candidate for the backfill (see section about backfill).

The Planning Table exposes their information through a proxy server, the TableManager , by means the ARM can communicate with Planning Table to require the addition, the removal, the modification of every table entry.

The TableManager represents the only access point to the information system.

Among these components, it is necessary to mention the local JMS: although it does not directly belong to the pattern structure, this component is very important for the pattern functioning because its provides the APIs for managing the underlying resources.

All the jobs requestes are described using JDL (Job Description Language).

Basing on the nomenclature of the model described in 2.2, ARMServer is the User Server, the TableManager is the AR Mon, JAM is the Resource Manager, while the AR Manager and the Job Scheduler are part of the ARMController.

Pattern functioning

The functioning of the ARM Pattern can be summarized as follows. Note that the numbers into round brackets refer actions in sequence diagram shown in Fig. 2.4.

ARMServer waits (act. 1) for reservation requests coming from external clients. When a request is received, ARMServer checks (act. 2) the JDL file, to verify whether reservation parameters are properly defined (e.g. if they be-
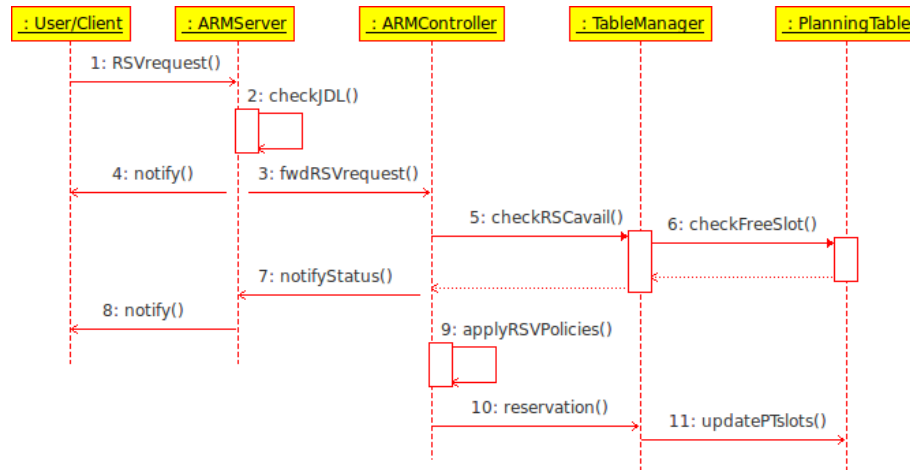
Figure 2.4: ARM: sequence diagram

long to a correct range of values) and whether the user has the authorization
to request the specified type and the given amount of needed resources. If the
checks succeed, ARMServer forwards (act. 3) the request to ARMController
and sends (act. 4) a reply (used to notify the beginning of reservation process)
to the client.

ARMController checks (act. 5) whether there are suitable resources able to
satisfy the user request, asking for free time slots to the Table. This inter-
action is, however, mediated by the TableManager (act. 6) that is the only
component authorized to access the Table information. The TableManager
returns this information to the ARMController which, basing on the defined
reservation policies, will take the better decision (act. 9). If the reservation
can be performed, ARMController asks (act. 10) TableManager to save the
reservation (act. 11) request for the chosen resource. Instead, if it is not pos-
sible, it notify to ARMServer the inability to perform reservation. At the same
time, ARMController forwards (act. 8), through ARMServer (act. 7) the pro-

cess result to client that has done the request.

When the start time of an advance reservation is reached, ARMController drives JAM to submits the job to the reserved resources. When the execution of jobs ends or the end time is reached, JAM warns ARMController.

If job ends before the estimated end time, the ARMController trough Table-Manager, updates the Planning Table entries freeing the reserved resources.

If the end time is reached before the ending of job execution, the ARMController has to decide whether (i) to extend reservation time (if possible), (ii) to suspend job or (iii)to kill it. This decision depends on resources management policies and on resources status.

The result of the whole reservation process is then notified to user, through ARMServer.

2.2.2   Selecting Applications for Termination

This section describes the techniques used to perform reservations assuming that running applications can be terminated and restarted at later time.

There are many alternatives to select which running applications that came from a queue should be terminated to allow a reservation to be satisfied. As described above, obviously, the application terminated must be chosen among the ones that have required a simple allocation, because we assume that an advance reservation gives the user the certain that its application will run on a fixed set of resource for a given time interval.

ARM orders running application from queue in a list based on some cost. The application are then terminated in increasing order of cost until enough nodes are available for the reservation to be satisfied, according to this formula:

$$N(\alpha T_e + \beta T_f)$$

where: α and β are constants, N is the number of nodes being used by the application, T_e is the amount of time the application has executed, T_f is the amount of time the ARM expects the application will continue to execute (this data is taken from the duration d indicated by the user when submit the request of allocation: $d = T_e + T_f$).

α represents the weight related to the termination of an application that as performed a large amount of work (that would be lost), while β represents the weight related to the manteinance for an application that still has a large amount of work to do.

It occurs vary the constants α and β to determine their optimal values. The best values to use for these constraints vary by the scheduling algorithm and on the basis of the parameter to optimize. If α is larger than β, indicates that the amount of work done thus far is the most important factor to consider when selecting applications to terminate. This choice takes care of the uncertainty with which we can determine T_f, and focuses on the T_e that, on the opposite, can be exactly calculated. But if a resource is blocked by an application that for any reason can continue own execution, a value of β larger than α could avoid the waste, lead the ARM to select the application blocked for the termination.

2.2.3   Pattern: features

In the last section, we have described the general architecture of ARM. To implement it, we have take into account several implmentative choices.
We described these in the next paragraphes.

The ownership of a reservation

In the proposed approach, the owner of the reservation is the single job (or group of jobs) rather then the user that submits the job, as it happens in the currently, mostly adopted solutions [16, 17].

This choice makes the reservation management more flexible compared to other solutions presently adopted: distinguishing the requirements and constraints related to each submitted job offers the opportunity to selectively modify the negotiated agreements with a finer grane (single job instead of overall user submission).

The most common alternative of associating the reservation with the user using a single SLA on his set of jobs implies that each modify on such SLA influences not only the running job involved in the user SLA, but the whole set of jobs belonging to the user.

Moreover, to enhance resources exploitation, the resources are released once the job is completed even when the reservation time is not elapsed. This does not happen when the reservation is associated with an user rather than with a job, indeed, in this case the user maintains the resource ownership even if its tasks are completed. This policy could imply a low resources utilization and, consequently, a worsening of the whole system performance. Instead, releasing the resource when the job execution is finished, makes possible to make get resource soon available for another reservation without waste of time.

Alteration of a reservation

Another foundamental feature of the proposed solution is the ability to modify an *active* reservation, i.e a mapping jobs-resources already in use,

supporting the renegotiation of previously established SLA: an user can modify same parameters of an established agreement only if the underlying JMS is able to modify the parameters of the reservation related with the SLA at any time.

The management of the knowledge of reservation

As described in sect. 2.2, the management of advance reservation requires the knowledge about the future usage of resources, making more complex the information schema used by CE.

The CE provider has to map the avalilability of resource on a time axis and then it has to estabilish how represent it and how manage the information on the future state of the CE. These imply some design solutions.

First of all, it has to be considered an observation time window, here defined *time horizont*, wherein a reservation can be made: a time horizon very large allows providers to accept advance reservation for a resource well in advance, but considerably increases both the seeking time and the dimension of the information source, the software component that tracks these information (tipically is a table); vice versa a small time horizon reduces the dimensions of information source and seeking time, but also reduces the chances to accept a reservation.

A simple example of "time horizont" implementation is given by a bitmap, where every bit represents the state of a resource in a specific time unit. The extension of time horizon strongly affects the sizing of bitmap. For each resource, the bitmap has to use an array of bits(one for each time unit taken in consideration). The size of this array of bits depend on the extension of the time horizon. If the time horizon grows, also the array of bits grows.

Guarateeing a very large time horizon means to use an high number of bit to represents the state of each single resource.

Notice that, adopting the bitmap, the dimension of the time horizon must be finite.

Besides, continuing to consider the example of bitmap, there is another parameter that affects the sizing of the bitmap: the extension of time slot that can be associated to a resource, i.e. the time grain.

A fine grain allows user to reserve a resource exactly for the time he needs, but increases the dimension of bitmap. Conversely a course grain reduces the space consumption, but increases the probability to waste resources (the user must reserve an amount of resources even if he doesn't use it in full.

In our pattern we try to face these issues adopting an approach based on a doubly hashed table.

First indexing derives from an hashing on the identifiers (primary key), and the QoS parameters (secondary key).

The second index on the same mapset derives from hashing the start time of the reservations that involve the resource (primary key) and the durations of such reservations slot (secondary key).

Each item of this hashtable contains information on both the actual and the future state of any single resource.

Using this representation we can consider an infinite time horizon: the size of hashtable depends only on the number of reservations active on CE. Our information source, in fact, has not one item for every time slot considered, as in the case of bitmap, but, on the contrary, it has one item for every reservation active. It is indipendent from both the extension of the time horizon and time grain.

This feature allows our pattern to accept reservation requests for any future slot time.

On the other hands, this feature introduces a strong dependence between the performances of the information source here taken in consideration and the amount of reservations for a resources. An high number of reservations on a specific resource increments the seeking time for the information related to that resource.

To face this issue, we have clusterized the information held in each item in epoches.

Each epoch refers on a fixed time interval and contains all the reservations whose start time fall in that time interval.

If a user needs information about the future state of a resource, it must not inspect the whole item related to that resource, but only the epoch containing the start time required, reducing significantly the seeking time.

Overtime issues

If the execution time of running service exceeds its reserved time, it would affect the normal execution of admitted but not yet started services or subservices.

It's impossible avoiding this situation because the end time of advance reservation is calculated by estimate, even using the approaches that use historical information to predict the run times of parallel application [**?**]. So, it cannot be avoided *de facto* that the real time (T_r) is shorter or longer than estimated time (T_e). We consider separately the two cases:

1. $T_r < T_e$: in this case, we waste the resource (then this leads to lower ratio of resource utilization), because the resource reserved in advance will result unused for the remaining time T_m ($T_m = T_r - T_e$). In order to avoid this, ARMController monitors all the reserved resources; if one

of those resources is unused (because the service either terminated its execution or because of faults) ARMController leads JAM to release that resurce and update opportunely the TableManager for allowing the reception of new jobs. This is possible because the reservation refers to that job and, once the jobs ends, the resources associated with it can be release.

2. $T_r > T_e$: in this case we will have a series of downstream services abnormal behaviour, a *fault chain*, like dominos. A solution would be to stop or migrate the active service to ensure the other services executed at the required time. Unfortunately, restarting or migrating need additional features not available in the LSF EGO: as a consequence, it is not possible doing "checkpoints" in a service. Then the action performed when the T_e exceeds the reserved quantum is the following: once the end time for a reservation expires, ARMController checks if the service still runs. If so, it estimates the pending requests of reservation and the availability of resource. There are two possibility:

   - if the availability exceeds the requests, ARMController leaves that the service continues its execution;

   - on the contrary, ARMController drives JAM to stop the execution of the reservation, giving the resource to the another service on the base of the scheduling policy. The referred service is tagged as *allocation* and is restarted when the resource becomes again available to accept an allocation.

   In both cases a penalty is charged to the user that own the reservation.

2.3   Tecnhologies exploited

2.3.1   Considerations about JAM and LSF

As said before, the JMS chosen for this work is the Platform LSF.

Although this JMS is a widely consolidated product in the grid environment, it lacks of mechanisms and tools to manage the advance reservation profile.

In order to overcome these limitations, the JAM (Job Allocation Manager) is able to interact directly with the LSF to obtain the AR profile described above.

JAM uses the Platform EGO to i) have a uniform vision of the resources available in the considered CE and ii) to submit a job maintaining the information about it's state.

JAM , as said before, is driven by the ARMController , the core of the whole ARM.

JAM is not able to distinguish between an allocation request and an advance reservation request in that it considers the allocation a *border line reservation* with start time set up on the current time.

The different management of both typologies of requests is demanded to the ARMController.

When the ARMController forwards the allocation requests to the JAM , the requests are queued in the specific resource queue.

The JAM is responsible for getting the allocation request from the resource queue and making an allocation request to Platform EGO.

The Cluster Kernel processes the requests and sends a reply to the JAM . These replies are held in the work queue. A thread of the JAM is responsible for getting the allocation reply from the work queue, adding the resource to the resource collection structure, and starting a container on the allocated host

slot.

This thread cycles through the queue until containers have been started on all allocated host slots.

JAM is also responsible to take the information about the status of the resources, properly querying to the underlying information system provided by Platform EGO Cluster Kernel.

2.3.2   gLite overview

gLite combines and refines some components developed in previous related projects (Condor [19], Globus [20], LCG [21], and VDT [22]). This middleware provides the user with a set of high level services (i) for scheduling and running computational jobs, (ii) for accessing and moving data, and (iii) for gathering information about the Grid infrastructure and the Grid applications. According to architecture standardization given by OGSA [23], gLite adopts the Web Services technologies and the Service Oriented Architecture in order to facilitate interoperability among the different components and to allow easier compliance with upcoming standards in this field.

For the proposal of this chapter, I will take into account the component that handles the jobs allocation issues in gLite: the Workload Manager Service (WMS).

The WMS accepts jobs coming from grid users and provides a set of tools (that will be referred to as WMS-UI from now on) by means of which users can exploit all the functionalities made available by the WMS itself. The architecture here proposed adds a new crucial feature to WMS: the opportunity to make an advance reservation for a resource, even if it could be easily integrated with other grid middlewares.

2.4   Reducing the impact of Advance Reservation: the proposed algorithm

The introduction of the reservation affects the performances of the resource scheduling.

An entity (user or job) requiring *best effort* services can not use the resources assigned to another entity B in the entire time interval they are reserved, even if B should finish its execution before the declared *end time*, beacuse, in according to the reservation' rules, the resources remain to the job B. Since statistically a job performing a reservation does not use the assigned resources for all the duration that has required, this rigid assignment could create a set of holes that imply an underutilization of the CE.

As previous said, in our model, the jobs are grouped in: *best effort jobs* (low priority) and *guaranteed jobs* (high priority). Each guaranteeed job has associated a reservation for a set of resources. The presence of guaranteed jobs in the schedulation queue, increments the queuetime for the remaining *best effort jobs*, because these last have a lower level of priririty.

These side effects can be reduced using suitable algorithms to handle the job schedulation, e.g. backfill.

2.4.1   Backfill

The duration of each reservation is calculated on the basis of a job profiling, and often it is overestimated to avoid that a reservation expires before a job completes its execution.

This means that a high percentage of guaranted jobs complete their activities before their reservation expire. This implies that resources assigned to a job are unused for all the time elapsing from the jobs completation time to the

end of reservation.

We refer to the WNs that are still reserved but have not a job that running on them, *idle WN*.

In 2.2.3, we have described the proposed model of advance reservation, explaining that in this approach the *owner* of the reservation is a single job (or group of jobs) rather then the user that has submitted it. This implies that once a job completes its execution, it releases all reserved resources (idle WNs) that can be used by other best effort jobs.

ARM uses backfill algorithm to reassign the idle WNs and to improve the performances of CE.

Generally, the Backfill [15] is an optimization strategy that allows to improve the exploitation of the resources available on a cluster of workstation executing jobs out of order.

This strategy assigns a priority value to each job, according to a predefined policy, and then re-orders the jobs into a highest priority first sorted list.

In this case, backfill operates basing on the job reservations information (*start time* and *duration*).

ARM knows: i) the idle WNs; ii) number and duration of the jobs that could start on those idle WNs, and which resources it will need at that time.

Basing on this information ARM determines what WNs are idle and how long they will remain idle till their next reservation. The association between a set of WNs and their idle times is defined as *backfill window*.

Once determined the backfill windows, ARM inspects the scheduling queue to determine which best effort jobs can be started within this backfill windows. This job is executed *out of order*, but this is possible because they do not delay the reserved job. By this way ARM improves the exploitation of resources, without affecting the privileges of guaranteed jobs.
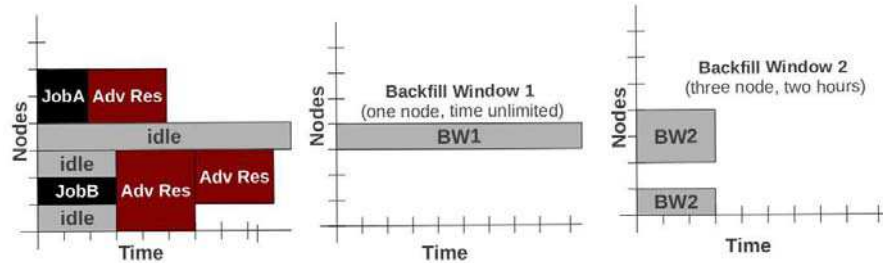
Figure 2.5: Backfill Windows

2.4.2   How our modified Backfill works

Fig. 2.5 shows a simple use case regarding two running jobs and a reservation
for a third job.

The present time is represented by the leftmost end of the box with the
future moving to the right. The gray rectangle represent currently idle nodes
which are eligible for backfill, i.e. the nodes where a resource is not already
interessed by a reservation. To determine backfill windows, ARM analyzes
the idle WNs essentially looking for largest node-time rectangles. In the case
represented in figure, it determines that there are two backfill windows. The
first window contains only one WN and has no time limit because this WN
is not blocked by any reservation. The second window, Window 2, consists
of 3 WNs which are available for two hours because some of the WNs are
blocked by a reservation. These backfill windows partially overlap yielding
larger windows and thus increasing backfill scheduling opportunities.

It is important to note that the choice of the number and the size of backfill
windows strongly affect the performance of the cluster. As shown in the
figure, the backfill windows that range mainly on the X axis (window 1)
privilege jobs that need a restricted set of resource, but for a long period of
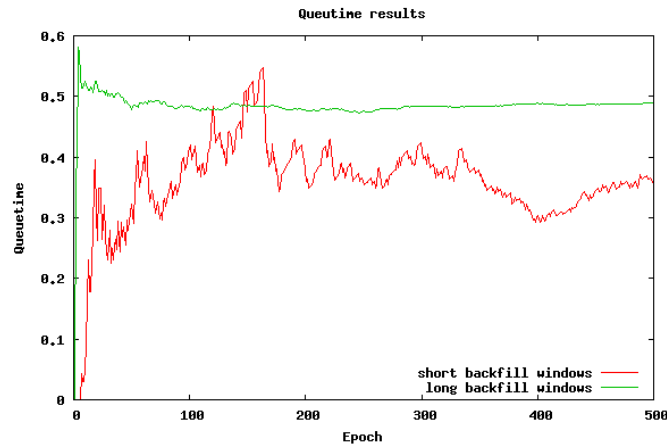
Figure 2.6: Comparation between long and short backfill windows: queuetime
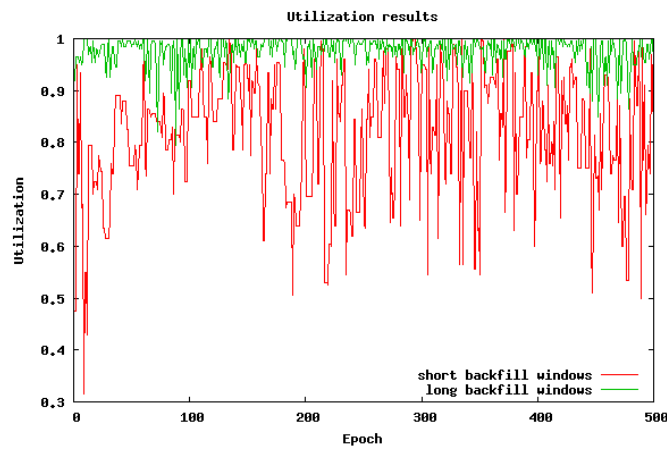
Figure 2.7: Comparation between long and short backfill windows: utilization of WNs

time; while, on the contrary, the backfill windows that range mainly on the Y axis (window 2 and 3) privilege jobs that required many resources for short period of time.

ARM selects the policy to adopt on the basis of an analisys of the allocation queue. It monitors this queue periodically. If there are many short jobs that are waiting for the allocation, it adopts *short* backfill windows (many resources for short time interval), while if there are many long jobs that are waiting for the allocation, it adopts long backfill windows (a restricted set of resources for long time interval).

Once the backfill windows have been determined, ARM begins to traverse them.

By default, these windows are traversed widest window first but this can be configured to allow a longest window first approach to be employed.

As each backfill window is evaluated we assigns it to a best effort job, extracting it from the schedulation queue.

The selection of job to start in the backfill window is operated on the basis of a predefined algorithm.

In our evalutations, we have considered three algorithms: first fit (selects the first job able to fill the backfill window), shortest job fit (selects the shortest job able to fill the backfill window), biggest job fit (selects the big job to fill the backfill job).

The choice of one of these algorithms and the choice of one of the policies previously described (for the determination of the size of backfill windows) represent the two main variables of the backfill proposed in this chapter.

Once the ARM selected a pair algorithm-policy (e.g best fit - short backfill windows), it is able to extract opportunely a best effort job from the allocation queue and run it ahead of the job that is waiting at the top of the queue.

2.5   Evaluation results

In order to better understand how the reservation affects the performance of
the job scheduling, I tested the behaviour of ARM on a CE handling a cluster
of 200 WNs.

Two different parameters have been considered for these simulations. The
first and most important parameters gives a measure of resources utilization
and it is expressed as the ratio between the number (average) of the resources
actually exploited and the ones available. The second, instead, takes into
account the average time spent in queue by each submitted job (average
queue waiting time). It is useful because gives indications about the influence
of reservation on the management of BF jobs. The simulation evaluates
several scheduling algorithms, chosen among the ones commonly used in the
considered scenario.

2.5.1   Some notes on the simulation approach

Our software is an event-driven discrete simulator that evolves through a suc-
cession of epochs.

Each epoch, that represents the "unit of time" chosen for simulation, lasts 10
minutes.

For each epoch, a set of jobs is submitted to the system: the number of jobs,
as well as their duration, type (i.e. reservation or best effort) and the needed
number of CPU for their execution are chosen following the considerations
done below:

1. **jobs' number**: the number of submitted jobs depends from the number
   of resources available on the CE. In particular, fixed the maximum value

of this number equals to the 1% of the available resources (from here $n_{res_{CE}}$), the number of jobs introduced in the considered CE is chosen statistically (uniform distribution) within the range $[1, \frac{1}{100} * n_{res_{CE}}]$.

2. **jobs' duration**: each submitted job has an own duration. This duration is related to the a priori established simulation time (i.e. number of chosen epoch, from here n_{EP}). Fixed the number of epochs (n_{EP}), the duration for each job is chosen statistically (uniform distribution) within the range $[1, \frac{1}{100} * n_{EP}]$.

3. **jobs' type**: the scheduler foresees two different type of jobs, "reserved" and "best effort" respectively. A request for a reserved job is submitted in the CE only if there are the needed quantity of available resources and time ranges to be executed: the scheduler does not influence this kind of jobs because their time and order execution are fixed "a priori" by the ARM thought an advanced resources reservation. A best effort request, instead, is submitted in the CE without any guarantees: it is executed under the guidelines established by the scheduling policy. In our simulations, the number of submitted jobs for each epoch is equally parted between the two different typologies (50% reserved, 50% best effort).

4. **number of needed CPUs**: each job needs a given number of CPUs to run. This number is chosen statistically for each job within the range $[1, \frac{25}{100} * n_{CE}]$ (uniform distribution).

2.5.2   Simulations

The evaluations here proposed relate on the impact of advance reservation strategy adopted on the performances of the scheduling algorithms adopted

in a CE. For the simulation an *ad hoc* software has bee used: it was a bind chosen because several aspects needed for our proposal are not foreseen in the common used grid/cluster simulators.

They will be measured both the resources utilization and the jobs queue waiting time.

The strategy that have been compared to the Best Effort (from here called ̈BE ̈) are: the typical standing alone FIFO scheduling; it's arrangement with the resources reservation (from here called ARFifo) and the effect of three different versions of Backfill techniques to order and execute the jobs out of order: ̈first fit ̈policy(from here called ARBFff), the "shortest job first" policy (from here called ARBFsf) and the ̈biggest job first ̈policy (from here called ARBFbf). To refer to a concrete GRID scenario, I take into account the real workload observed in the gLite Grid handled by the consortium COMETA Grid infrastructure [3] within the project PI2S2.

The results of led simulations are summarized in the figures 2.8 and 2.9. The figures show the differences among the compared scheduling policies, respectively, on resources exploitation capability (fig. 2.8) and on average time spend by jobs in waiting for execution (fig. 2.9). Obviously BE has got ever the best performance, as expected, while the simple FIFO algorithm (without reservation support) has the worst result in both cases. Among the algorithms that use AR, as can be observed, the ARBFbf has got the best performance with regard to the resources utilization, while the ARBFff has got the best performance with regard to the queue waiting time. These last results coincide with the expectations: the choice of the biggest job to fill the *hole*, i.e the time range not covered by the reservation requests, increases the average utilization of the entire CE, but contemporaneously increases time spends in queue

---

[3]www.consorzio-cometa.it
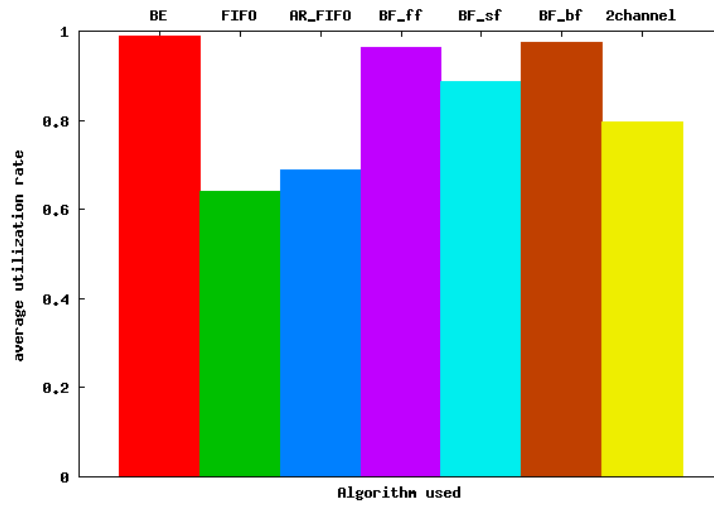
Figure 2.8: Comparison of all algorithms, parameter: average utlization rata
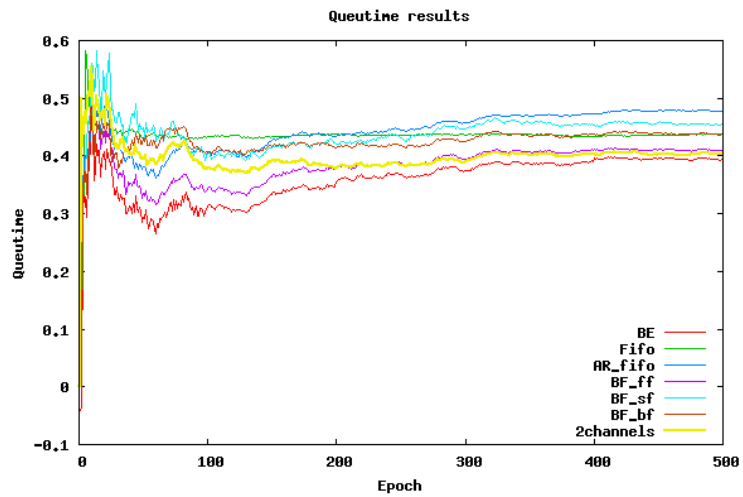
Figure 2.9: Comparison of all algorithms, parameter: queuetime
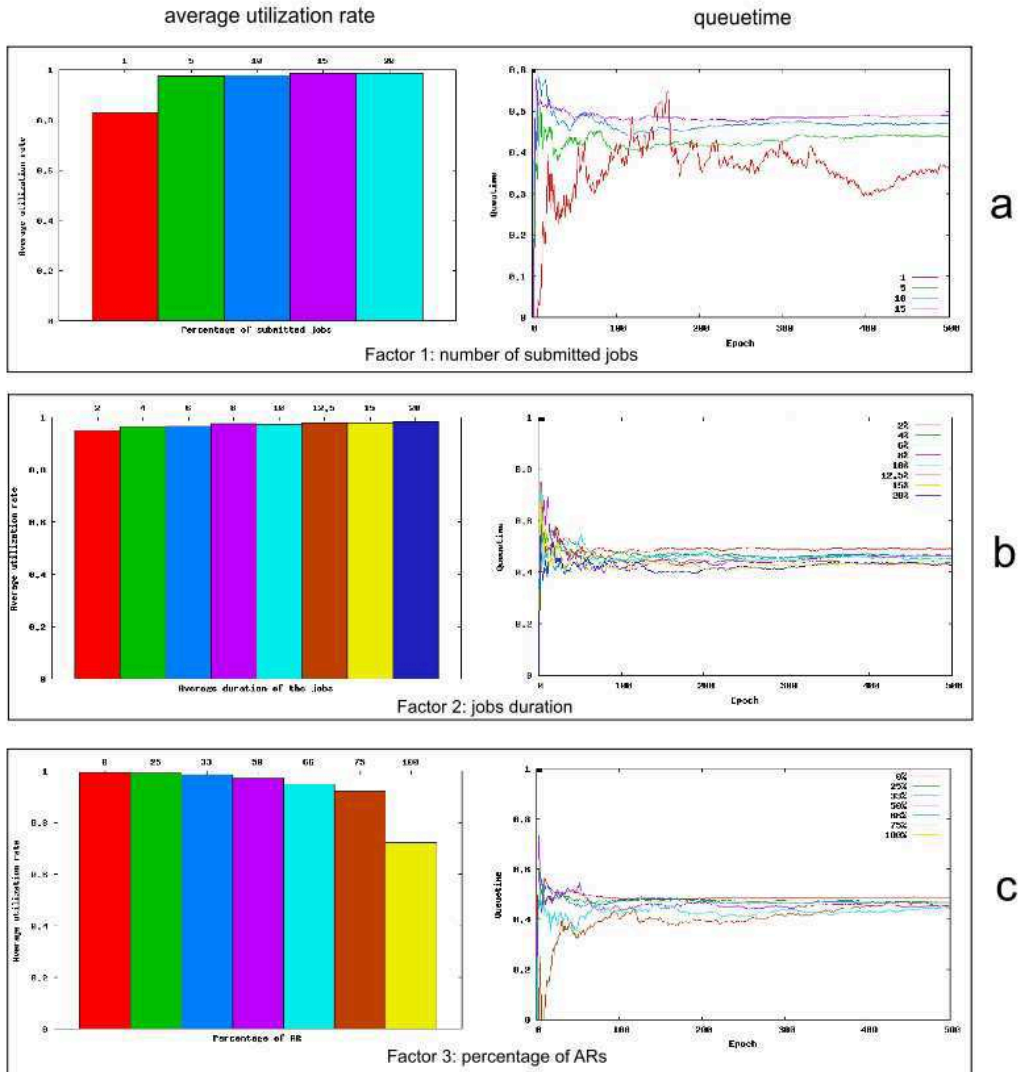
Figure 2.10: Second step of simulations

by the remaining jobs. On the other hand, the choice of the first fit policy results less advantageous with regard to the average utilization rate, but has good performance in term of queuetime, because a greater number of job is dispatched in the same time range.

Focusing on ARBFff (AR with backfill using the biggest job first for the allocations) that shows best performances in terms of average resources utilization and jobs queue waiting time, as previously discussed, it is interesting to analyze its behaviour depending on three factors:

1. **Number of submitted jobs**. Fig. 2.10.a shows the behaviour of the ARBFff strategy when the number of jobs, submitted in an epoch, changes. **In particular, the simulation have been done submitting, for each epoch, a maximum number of jobs equals respectively to 2(1% of total available resources), to 10(5%), to 20(10%), to 30(15%) and 40(20%), respectively**.

   The fig. 2.10.a shows that the increment of the job rate implies the increment of the average resource utilization (till to saturate), while the queuetime worsens. This behavior was foreseeable in that a greater number of jobs permits a better exploitation of the resource in the CE; on the otjer hand, this means also that the resources are busy for a wide range of time, increasing of the queuetime.

2. **Jobs duration**. Fig. 2.10.b shows the performance of the ARBFff strategy on changing the jobs duration (in epoch). **The simulation is led submitting, for each epoch, jobs having, respectively, a maximum duration equals to 5(1% of total available resources), to 25(5%), to 50(10%), to 75(15%) and 100 (20%) epochs**(on capisco).

   The results show that the performance of the ARBFff, both in terms

of resources exploitation and in terms of reduction of average queue waiting time, are better when the duration of the jobs grows up. In particular, the fig.2.10.b shows that the average resources utilization ranges widely while the queue waiting time remains, substantially, the same.

3. **Percentage of Advance Reservation**. Fig. 2.10.c shows the variations of the parameters taken into account when the ratio between the number of BF and of AR requests changes. A percentage of 33% of AR means that, for each burst of input, the 33% of total requests are advance reservation requests. The fig. 2.10.c shows that as the frequency of AR increases, the utilization of the CE decreases and the queuetime increases. This trend can be easily explained: an high percentage of advance reservation requests creates a great number of "holes", increasing the probability of wasting resources. A similar discussion can be done for queuetime: the greater the number of reserved resources, the lesser the probability of dispatching BF jobs.

The results of led simulations have demonstrated that, although the introduction of advance reservation reduces the utilization of the resources belonging to the CE, the adoption of an effective backfill strategy is useful to mitigate this performance reduction. In particular, the use of the "biggest jobs" strategy gives results very close to the best effort case.

2.6 Related works

Since the first implementation of Grid architecture, the advance reservation was an open question and many studies have investigate it.

The Grid Resource Agreement and Allocation Protocol (GRAAP) Working Group (WG) has produced a *state of art* document which explores properties, attributes, roles, mechanisms about resource reservation in grid environments [24]. We envision that our reservation, employed in the proposed system, can be used to support the reservation properties outlined by the GRAAP-WG.

The General-purpose Architecture for Reservation and Allocation (GARA), storically, was the most commonly known framework proposed for supporting Advance Reservation in the context of computational grids. It takes care the issue to provide advance reservation with uniform treatment of various types of resources, such as network, computation, and storage. Although GARA has gained great popularity in the Grid community, it has strong limitations in copying with current application requirements and technologies: i) GARA is not OGSA-compliant, and therefore all the current grid application, that often are OGSA-enabled applications, cannot directly make use of GARA; ii) GARA does not support the concept of agreement, therefore it does not support agreement protocol and SLA; iii) GARA, finally, does not support QoS monitoring and QoS run-time adaptation, key mechanisms to provide QoS-guaranteed services [25]. Contra, ARM is OGSA-compliant, is explicitly thought to support SLA negotiation and can be easily extended whith adaptive functions.

Many studies and tests have been conducted on advance reservation-based Grid scheduling technologies. Most of them focused on the benefits for Grid jobs profiting from advance reservations. Some fo them have been devoted to evaluate the impact from advance reservations for Grid jobs on the performances of CE [26–29], taking care of the problem of fragmentation generated in the computing environments due to these reservations. [30] investigates the influence of AR from the Grid on local parallel scedulers, using discrete

event simulations with real trace from the parallel workload archive.The performance are increased by means of two backfill algorithms: conservative and aggressive(also called: EASY). In the simulation section of this chapter I give some considerations about the choice of job that must be backfilled (the biggest in the queue or the shortest or the first).

[31] discusses the design and the prototype implementation of a QoS system (QoS Grid Services - QGS), that presents some analogies with the one proposed in this chapter: the provision of advance reservation to enable Grid applications to become QoS compliant, the care on the computational parameters of QoS, and the management of the allocation at CE tier. Differently from our implementation, QGS is mainly oriented toward the issues related with the management of QoS and related SLA, instead of those related with AR. Our work results more complete, especially regards on the management policies of the relationships between AR and allocation. Moreover, ARM, thanks its design, could easily be integrated in the QGS, with the role of CE handler.

[32] examines the requirements on a grid's infrastructure to support SLAs and focuses on a dynamic offload infrastructure to match a set of SLA requirements under variation of workload conditions. It assumes that a pool of servers and bandwidth exists from which the commercial grid can draw resources under high-load conditions and to which it returns resources when the load decreases. This approach results too binded to both the dimension of this pool and its acessibility. There are not real guarantees on the availability of the resource necessary to a job. In ARM, on the contrary, through AR, this guarantee exists and, thank to it, it's possible to draw SLA with specific QoS requirement.

More researchers have become interested on how to improve system utilisation by including flexibility factors in advance reservation. Chen and Lee [33]

propose a flexible reservation model based on flexible intervals for the parameter: start time of the advance reservations. Kaushik et al. [34] use the term "flexible time intervals" to define the flexible intervals for advance reservations. They investigate the relation between the interval size and the request waiting time, assuming that the request inter-arrival time follows the Pareto distribution. Both [33] and [34] does not consider requests for multiple resources.

[35] proposes a flexible advance reservation model where start and end time, duration and numbero of requested CPUs are flexible. All these last approaches can be compared with our work. The flexible advance reservations conceptually are equals to our allocation, since they are advance reservations without no hypotesis on the start time and end time, but with a duration specificated in the request stage. The approaches differs for: the target of the optimization critera (cost, average load, completion time) and for the algorithms used for this optimizations. Our work considers both the typologies of AR: strict reservations (with defined start time/end time) and flexible (only duration defined).

In [36] the authors propose an algorithm for a negotiation based scheduler (called NARPS) that dinamically analyses and assesses the incoming jobs in terms of priorities and requirements, reserving them to resources. NARPS reacts to the dynamic behaviour of a large number of users with different requirements seeking computational services, like the work presented in this chapter. The reservation of NARPS, however, are priorized jobs. It does not provide any guarantee on their start time/end time and this strongly reduces the attractives of this solution in a Market-based scenario.

Castillo et al. [37] use concept of computational geometry to handle resoruce fragmentation caused by the introduction of advance reservations. they also develop a set of implementations for scheduling algorithms that have a good

scalability degree. In their studey they consider only jobs with strict time intervals and only jobs requiring a single resource.

2.7 Conclusions

In this chapter two crucial aspects for the present development of Grid environments are considered: the SLAs and the resources reservations. The resource reservation has been considered as a strategy to guarantee that jobs have what they need at the proper time, while the SLA has been studied as a list of all the parameters characterizing a service, both functional and QoS related, and the rules and conditions for the proper use of the service.

This chapter focused on the SLA management in gLite middleware, analyzing a first set of features that are related with the issues of (i) QoS provision and monitoring and (ii) creation, monitoring, negotiation and run time adaption of the SLA.

It also discuses the issues related to job allocation and scheduling and advance reservation of the resources.

The results of this investigation have brought about the implementation of a prototype of an approach created in the context of the gLite middleware, called SLAM (described in the next chapter). Every component of SLAM has been described, also facing the issues of interacting with the gLite middleware and the JMS LSF Platform EGO.

The findings of this study have a number of important implications for future practices.

As described above, the execution of each service is monitored continuously to detect whether some QoS parameter achieve a value that violates the agreements and, if so, to attempt when possible, strategies for run time adaption.

Chapter 3

Service Level Agreement Management according to specific QoS requestes

Providing and guaranteeing quality of service (QoS) of shared resources in a Grid environment is a crucial challenges to obtain a flexible and dynamic services management. This represents a basic feature to guide Grids towards the concept of trading services, wherein providers and consumers of services and their relationship are clearly defined.

This chapter focuses on modelling QoS in a gLite platform [38] and designing suitable protocols to manage different kinds of constraints for supplying QoS guarantees.

It will be taken into account the issues related with the creation, monitoring, modification, termination of SLAs in a grid environment. Since the definition of SLAs strongly affects the profile of reservation that each grid has to provide, the chapter also deals with the issues related to advance reservation
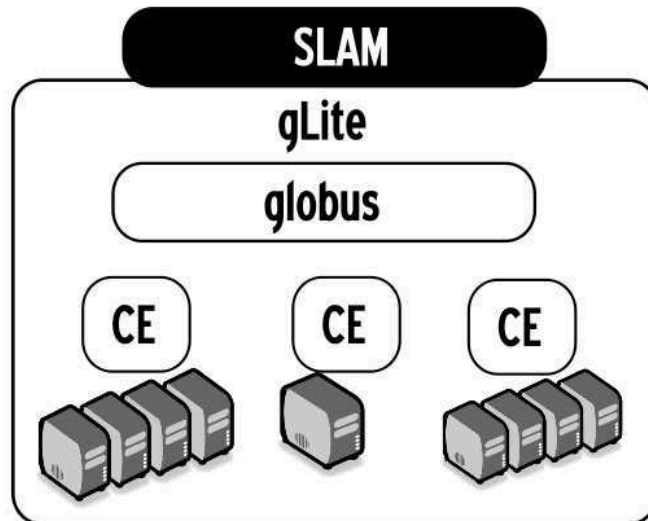
Figure 3.1: SLAM

and allocation, facing the issues related to the advance reservation of a resource, defining the role of the Job Management System, describing the implementation pattern of advance reservation that can follow the choices made in development are taken into account of the planned SLA provision.

3.1   SLA Management

The introduction of the QoS levels [11] in the requests for generic resources allows users to better characterize their needs.
Using the non-functional parameters, in fact, the users can formulate fine-

grained requests, requiring specific function or imposing some executing constraints. On the other hand, the provisioning of services considering the QoS parameters forces the providers to adopt more accurate and complex management policies, to have a continuous and complete control of their resources (i.e. resource monitoring) and to improve system reliability and fault tolerance. For a service provider, the QoS management means:

1. collecting and parsing the user requests in order to translate their QoS requirements in term of specific resources (i.e. quantity and type)

2. discovering, among the available ones, the resources that are able to satisfy the user requests.

3. choosing, among the suitable resources, the best ones according to the scheduling policy adopted.

4. reserving the specific resources (i.e. to guarantee that one or more resources belong only to a user for a specific range of time)

5. monitoring the service execution to check the QoS requirements fulfilment.

Although some solution have been investigated [39], the present implementations of Grid middlewares do not allow the users to specify QoS parameters in their requests; the permissions and the limitations of a generic Virtual Organization user are established thought an "*a priori*" agreement between the grid and the VO managers. In order to overcome these limits, this thesis proposes a web services based framework, SLAM (Services Level Agreement Manager), able to create an advanced support for QoS provisioning in the grid. SLAM has been developed and tested on gLite middleware but it can be used on top

of different grid middlewares. In the following, an overview of gLite will be given and then the framework infrastructure will be shown.

gLite

The middlewares are the software infrastructures used by grids in order to integrate services and resources provided by different vendors and/or belonging to different organisations. The middleware used in the EGEE project [40], the one considered in this thesis, is gLite [38], a middleware stack that combines and refines some components developed in previous related projects (Condor [41], Globus [42], LCG [43], and VDT [44]). This middleware provides the user with a set of high level services (i) for scheduling and running computational jobs, (ii) for accessing and moving data, and (iii) for gathering information about the Grid infrastructure and the Grid applications. According to architecture standardization given by OGSA [45], gLite adopts the Services technologies and the Service Oriented Architecture, in order to facilitate interoperability among the different components and to allow easier compliance with upcoming standards in this field.

For the proposal of this thesis, two components of gLite will be taken into account: the Relational Grid Monitoring Architecture (R-GMA [46]) and the Workolad Manager Service (WMS [40]).

R-GMA is an implementation of the Grid Monitoring Architecture (GMA) proposed by the Global Grid Forum (GGF), that provides a service for information, monitoring and logging in a grid environment. This component abstracts the complexity related to the resources information management, virtualizing and collecting them in one large Relational Database that may be queried directly by the users via web services. The main feature of this architecture is the ability to provide a useful and predictable information system

built on the data provided by losser-coupled providers across the grid environment. The R-GMA architecture consists of three components: i) *Producers* which publish information into R-GMA, *Consumers* which obtain the information by subscribing to the interested services and *Registry* which mediates the communication between the Producers and the Consumers offering them a common interface.

The Workload Manager Service (WMS) is the component that handles the jobs allocation issues. It works on i) accepting requests for job submission and management coming from its clients and ii) taking the appropriate actions to satisfy them. The specific kind of tasks that request computation are usually referred to as *jobs* and are described using the Job Description Language (JDL). The WMS provides a set of client tools (that will be referred to as WMS-UI from now on) allowing the user to access the main services (job management services) made available by the WMS itself. The main funcionalities of the WMS include:

1. job submission for execution on a remote CE

2. automatic resource discovery and selection of the CE

3. listing of suitable resources to run a specific job according to the user requirements

4. monitoring of the job state for all its lifecycle

5. cancellation of one or more submitted jobs

6. handling i) the stage-in and stage-out files and ii) bookeeping and logging information

Interacting with the R-GMA and the WMS services, the gLite user can fully manage its jobs submitted to the grid.

3.1.1   The gLite framework for the SLA Management

SLAM provides to the user an unique interface (i) to request one or more resources specifying the needed QoS parameters, (ii) to negotiate the value of each QoS parameter, (iii) to establish an SLA that collects all the information related to the agreement and to (iv) monitor the requests life-cycle. The framework consists of four main components:

1. RequestInterpreter

2. SLAManager

3. ResourceDiscover

4. ResourceBooker

The RequestInterpreter receives from the user all the information related to the jobs submission. These include the input parameters (and related datasets), the libraries required for the execution and, different from the present implementation, the list of QoS requirements. The aim of the RequestInterpreter is to translate these requests into the amount of required resources(both in physical and temporal terms) needed.

These results are then given to the SLAManager that probes, through the ResourceDiscover , the information service (MDS, RGMA) to obtain the list of the resources which are able to execute the job following the user's demands. All the available resources discovered are passed to the ResourceChoser , that selects the resources to be allocated: this decision strongly depends on the load balancing policy adopted. The ResourceChoser provides the selected resources to the SLAManager which communicates them to the JAM . Finally, the resources will be reserved and the SLAManager composes the information in a SLA.
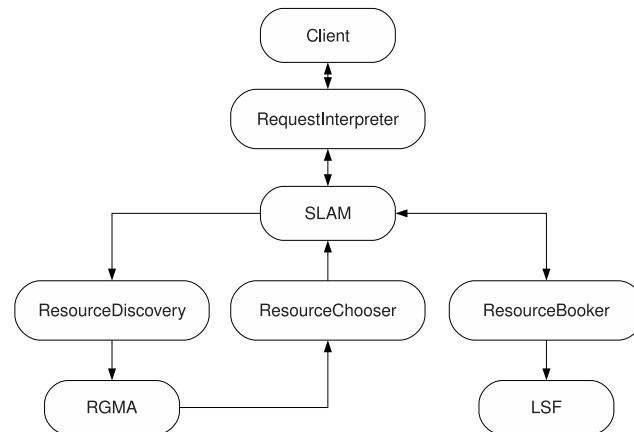
Figure 3.2: framework's component

The SLA is sent to the user so that he can sign it thus reaching to an agree-
ment with the service provider. If some resources are not available with the
desired QoS, the SLAManager starts a negotiation phase to find a compromise
between available resources and user requirements. If the client accepts the
new values for some parameters of QoS requests, he will sign a new SLA,
otherwise it will refuse the agreement (see fig.3.2).

Description of the components

This paragraph reviews each component of the framework:

- **SLAManager** It coordinates the other components to guarantee the
  correct service fruition. The SLAManager attempts to receive the user's
  requests translated as resource requirements, to invoke the ResourceDis-

cover (to discover the resource) and the JAM (to manage the reserva-tions).

- **ResourceDiscover** It attempts to recover the information on the avail-able resources. The ResourceDiscover is able to request the information from information services through a standard query mechanisms, be-cause it is implemented as Web Service.

  It is composed of two sub-components: 1) a standard front-end imple-mented by a WS and 2) a standard back-end, platform dependent, that represents a logic driver that can translate every high level query into lower level commands.

  Actually, in gLite there are two standards: an information service that uses the BDII technology [47] with MDS, GRIS and GIIS [42], while the other uses RGMA [46].

- **ARM** It has a central role for the whole architecture. It reserves the resources that the ResourceChoser has discovered.

  It has a two level pattern: the first level consists of a WS tier that is able to receive the requests originating from the SLAManager while the second, the reservation maker, adapts the high level command into low level directives to drive the underlying JMS, e.g. PBS, LSF or Condor.

- **ResourceChoser** It selects the resources that will host the task among all the available resources. Through the selection of the resources, it implements a policy of task management. It contains several responses of the ResourceDiscover , grouped in entries.

- **RequestInterpreter** It tries to translate the user request in parameters that are understandable by the framework.
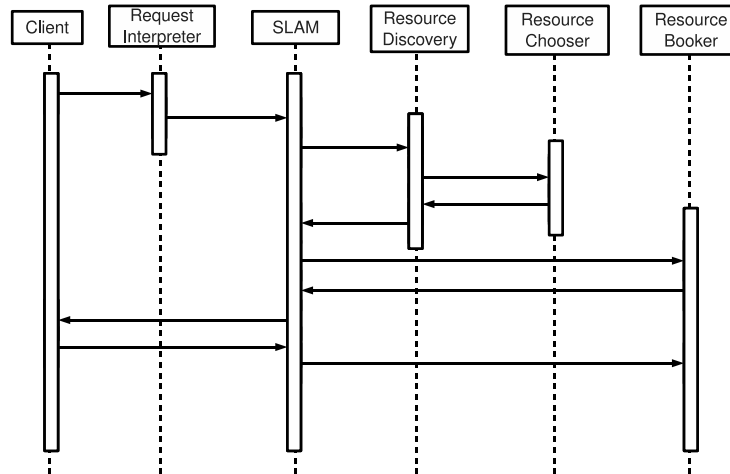
Figure 3.3: Sequence Diagram

3.2 Conclusions

This chapter focused on the SLA management in gLite middleware, analyzing a first set of features that are related with the issues of (i) QoS provision and monitoring and (ii) creation, monitoring, negotiation and run time adaption of the SLA.

It also discuses the issues related to job allocation and scheduling and advance reservation of the resources.

The results of this investigation have brought about the implementation of a prototype of an approach created in the context of the gLite middleware, called SLAM. Every component of SLAM has been described, also facing the issues of interacting with the gLite middleware and the JMS LSF Platform EGO.

The findings of this study have a number of important implications for future practices.

As described above, the execution of each service is monitored continuously to detect whether some QoS parameter achieve a value that violates the agreements and, if so, to attempt when possible, strategies for run time adaption. This last issue will be faced in the chapter 5.

Chapter 4

QoS-aware discovery protocol

A rapid and effective mechanism for service provider localization is a fundamental and necessary tool in a distributed Services MarketPlace.

In this scenario, in fact, the services are distributed among several remote providers: to find the providers of a specific service, the consumer has to refer to a services information index.

The design of these services is an important aspect because it has a great impact on the system as a whole.

If a centralized (or hierarchical) services index is used, as with classical grid solutions (e.g. MDS for Globus Toolkit [48] [49] and BDII for gLite [47]), it is possible to obtain a discovery mechanism simple to manage and characterized by a low response time (i.e. the time needed to obtain the service reference). Unfortunately, this approach compromises both the scalability and the reliability of the services information index, especially in terms of robustness and fault tolerance, in that it represents a *single point* of failure.

For instance, if there are too many clients that query the index service at the same time, the response time could become very high wasting perfor-

mance. Furthermore, if the service crashes, the information stored within it becomes unreachable. If, instead, a structured distributed information mechanism is used, like Distributed Hash Tables (DHT), it is possible, exploiting their well-defined structure and their self-organization capabilities, to obtain a low response time and to solve the problems related to scalability and fault tolerance. Unfortunately, this information organization foresees that the information related to a provider and to its offered services could be maintained by another provider: it could be a problem in a system where each service provider is in competition with the others. Using DHT, in fact, it is possible that the information related to several providers of a specific service are stored in another provider of the same service: thus there are no guarantees that this provider responds to a query giving the users only its own service reference, excluding other providers.

To overcome these drawbacks, this chapter proposes an unstructured distributed services discovery based on a selective flooding algorithm. Considering the P2P-Grid based services Marketplace, presented in the first chapter, the proposed algorithm propagates the searching messages through the "near nodes" of the underlying overlay network. Each peer/node autonomously maintains the information related to the number and type of services provided locally and a *cache of "partial knowledge"* of the services offered by a limited amount of other nodes in the overlay network; the required service will be discovered by a *selective propagation* of the queries through the network which will be sorted through a suitable management of partial information previously collected on each peer.

This distributed searching algorithm is fault resilient: the scope of any crash is bound only to the node involved while the remaining ones will continue to work guaranteeing a high fault tolerance and reliability. Furthermore, since each peer/provider manages locally their own information, the algorithm does

not suffer of problems generated by the malicious behaviour that could be assumed using the DHT. Unfortunately, because the proposed discovery protocol is based on flooding-based algorithm, like the well known Gnutella [50], it can generate a large amount of frames routed through the network: if the number of peers involved becomes very high, the flooding strategy can cause high bandwidth consumption.

The algorithm proposed overcomes this issue through:

1. a flexible and dynamic management of neighbour relationships

2. a control mechanisms for the quantity of frames created by a query

3. a *selective routing* protocol based on node reputation

4. a cache of partial information related to the knowledge of services offered by some other peers

Furthermore, the proposed algorithm is able to search the services under QoS constraints, allowing the users to obtain only the references of those services able to satisfy its requirement. This ability, that is an innovation in the field of discovery algorithm for distributed systems, represents an important value-added in a service market place scenario in that it allows not only to identify different providers for the desired services but also to discern among them the most suitable one to satisfy the user requirement.

4.1   The Discovery Protocol

The proposed discovery algorithm is characterized by five core issues, outlined below.

- **Distributed discovery is based on flooding queries through neighbours.** Each peer is able to directly interact only with the nodes (other peers) it knows and it can address, that are considered its neighbours in the overlay network. Each peer may forward the queries to its neighbours according to a flooding strategy, similar to the one used in the Gnutella v0.4 protocol. The concept of neighbourhood is very important for this scenario because every interaction between a peer and the rest of the network is mediated by its neighbours.

- **The services are searched under QoS constraints.** The queries used to search services contain a list of QoS parameters that allow selection of the service that is most suitable for the user application requirements. This avoids an initial negotiation process with the service providers which, even if offering the needed resources/services, are not able to provide the required QoS level.

- **Services Level Agreement (SLA) based mechanism for service negotiation.** Within every response message, each service provider specifies the values of QoS parameters involved in the negotiation process and an SLA template [51, 52] to manage them. The information in the template will be used to build an SLA that contains (i) all the considered parameters, (ii) their values, (iii) the aim of the agreement and (iv) the possible penalties in case of breach of contract.

- **A cache of known services is used to speed up discovery.** The discovery protocol is supported by a local cache that maintains couples of service-peers that can be used as a shortcut to speed-up the discovery process, allowing a "provider-consumer" direct link.

- **A reputation mechanism to improve neighbours management.** In

order to increase the probability of finding the desired service and to reduce the number of messages spread over the network, the peer sends the queries, through flooding, to a subset of its "better" neighbours. The quality of a neighbour is given by a reputation value, that measures the number of times that a contacted peer answered a service request by offering either a local service or useful information for service discovery.

4.1.1 The phases, the actors and the exchanged messages

In the proposed protocol it is possible to distinguish three different phases, that can be summarized as:

- Overlay network joining (explained in the first chapter).

- Services discovery.

- Service Level Agreement management.

The first phase takes into account the operation that each peer has to accomplish in order to become member of the overlay network and to be able to interact with other peers. The messages involved in these first steps are:

- GetNearNode (GNN): it is used by a peer to obtain a list of available peers from some well-known entry point servers (EPS).

- NearNodeList (NNL): it is used by EPS to return a list of available peers to the peer that had sent the GNN message.

- NearNodeRequest (NNR): it is used by a peer to establish a neighbour relationship with another peer on the overlay network.

- NearNodeAccept (NNA): it is used by a peer to accept a neighbour relationship request.

All the operations related to services lookup are envisaged in the service discovery phase.

Each of these operations involves two different categories of actors. The first one is represented by client/consumer, i.e. the peer that wants to exploit a generic service offered by some remote peers. In the rest of the chapter this kind of peer will be called Requiring Peer (RqP).

The second category consists in the server/provider of service, i.e. the peer that lends its computational and storage resources to satisfy the requests of clients. In the following, the service provider will be called Provider Peer (PrP). However, according to the peer-to-peer philosophy, each peer of the overlay network can assume both consumer and provider role.

The messages exchanged between RqPs and PrPs during the discovery phases are:

- Discovery Query (DQ): it is used by RqP to search for a specific service among the peers belonging to overlay network. It contains the name and level required for the service (deterministic, statistical, best effort), both expressed as globally known and unambiguous code, and a list of parameter-value couples used to characterize the service request. The complete DQ structure is shown in Fig.4.1:

```
1  struct DQ{
2        struct bitmap id_frame;
3        int hops;
4        struct peer RqP;
5        struct id_service serv;
6        struct id_level serv_level;
```

```
7        struct ParamList list;
8  }
```

- Discovery Query Hit (DQH): it is used by the PrP to answer service requests coming from RqPs. The DQH contains the reference of PrP location, the service name, the service level, an SLA template, the values of the required parameters and a local timestamp for temporal reference. The DQHstructure is shown in Fig.4.2(a).

```
1  struct DQH{
2        struct bitmap id_frame;
3        struct id_service serv;
4        struct id_level serv_level;
5        struct peer PrP;
6        struct peer infoPeer;
7        struct timeStamp time;
8        struct Template tmp;
9        struct ParamList list;
10  }
```

- Service Indication (SI): it is used by peers involved in the discovery protocol (i.e. peers which have been reached by a DQ message but not able to provide the needed service) to give the RqP an indication about the position, on the overlay network, of PrPs able to provide the relevant service. It contains both the name and service level requested in the DQ, the address of the SI sender peer and the list of IP addresses of the PrP found with the relevant time reference indicating the known last time it provided the service. The SI structure, shown in Fig.4.2(b)), is:

```
1  struct SI{
2      struct bitmap id_frame;
3      struct id_service serv;
4      struct id_level serv_level;
5      struct peer InfoPr;
6      struct SIinfo[] PrPList;
7  }
8
9  struct SIinfo{
10     struct peer PrP;
11     struct timeStamp time:
12  }
```

The interaction between the same two actors characterized also the last
phase, the one related to the management of agreement. During this phase the
two parties try to find a common agreement that will set the rules for a correct
service fruition. All information related to agreement is collected in an SLA
that all involved entities must accept. The messages exchanged in this phase
are:

- Negotiation Message (NM): it is used by RqP to request to PrP the re-
  negotiation of some values of the QoS parameters. It contains a list of
  QoS parameters with the new needed values. The NM structure, shown
  in fig4.3(a) is:

```
1  struct NM{
2      struct bitmap id_frame;
3      struct ParamList list;
4  }
```
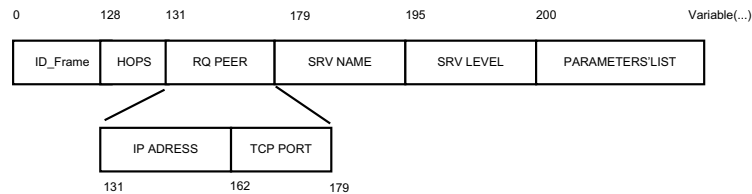
Figure 4.1: DQ frame's structure

- SLA Accept (SA): it is sent to RqP to the chosen PrP to notify its interest in completing the agreement; each SA contains the code of the template and the list of the agreed parameters.

```
1  struct SA{
2       struct bitmap id_frame;
3       int SLAcode;
4       struct id_service serv;
5       struct id_class serv_class;
6       struct Template tmp;
7       struct ParamList list;
8  }
```

- SLA Confirm (SC): it is used by PrP to confirm and to end in a conclusive way the agreement process. SC has the same structure (shown in fig4.3(b)) as SA.

Network joining and neighbors management

A peer can be considered part of our overlay network if it is able to interact with at least another peer of the network. In order to join the network, it sends a GetNearNodes(GNN) message to well-known entry point servers. The aim

0	128	176	192	208	256	288	Variable(...)
ID_Frame	RqP PEER	SRV NAME	SRV LEVEL	Info PEER	TIMESTAMP	PARAMETERS'LIST	

(a)

0	128	144	160	208	224	Variable(...)
ID_Frame	SRV NAME	SRV LEVEL	PrP PEER	TEMPLATE	PARAMETERS'LIST	

(b)
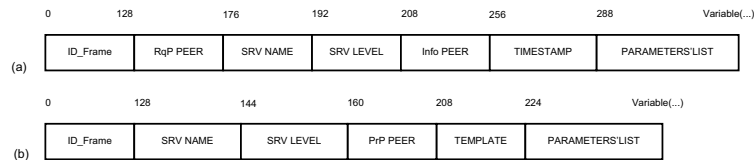
Figure 4.2: (a)DQH and (b)SI frames' structure

0	128	Variable(...)
ID_Frame	PARAMETERS'LIST	

(a)

0	128	144	160	176	Variable(...)
ID_Frame	SRV NAME	SRV LEVEL	TEMPLATE	PARAMETERS'LIST	

(b)
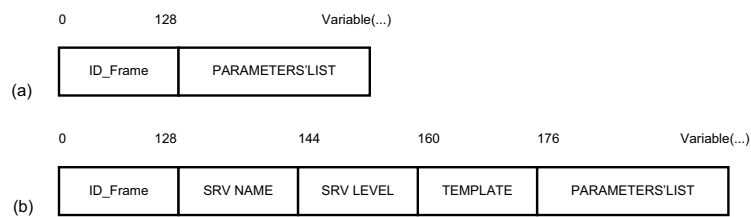
Figure 4.3: (a)NM/NA and (b)SC/SA frames' structure

of this type of server, which handles a repository of peer addresses, is to provide to the applicant, within a NearNodeList(NNL), a list of on-line available peers.

Once the NNL message is received, the peer tries to establish a neighbour relationship, by sending a NearNodeRequest(NNR), to every single node in the NNL. The NNR message stores the information related to (i) the location of the neighbor (IP address, service port) and the (ii) provided services, e.g. their number or the number of available queues. A peer that receives an NNR can decide to agree or to refuse the neighbor relationship: the choice depends on either the number of neighbors it is able to manage or on the value of service-related parameters stored in the NNR message (e.g. type and quality of services locally offered). A neighbor relationship is accepted by means of the NearNodeAccept (NNA) message. The two peers involved ratify the neighbor relationship by adding the remote peer address to their own *neighbors list*.

The neighbors list collects the neighbors with a relevant value of reputation
that measures the "strength" of the neighbor relationship. The reputation will
be used in the discovery process to select from the neighbors list only those
peers that meet certain criteria. Every time a peer receives a response mes-
sage, it uses the information contained to update the value of the sender rep-
utation. By associating the reputation value with the number of fruitful inter-
actions among peers it is possible to find the more collaborative nodes inside
the list. Each peer can take into account the new neighbors for flooding as
their reputation value grows, according to the frequency with which it uses
their services or obtains useful information about discovery. The reputation
value plays an important role in the discovery process since it allows dynamic
change of the neighbor list and, consequently, a change in the routing rules
for the messages . It should be noted that the neighbor list may also increase
when a peer finds a new provider peer by means of the discovery protocol (as
will be described in the following section).

4.1.2   Discovery phase

Each time that a RqP wants to use a remote service, it has to find at least one
of the relevant providers (PrPs) present on the overlay network. If the RqP
requires the remote service with some specific QoS parameters, it has to de-
scribe the values of these parameters in its discovery request together with the
needed QoS level.

The RqP may obtain these information by means of a distributed discovery
protocol, based on selective flooding involving only neighbours with ade-
quate *reputation*, or by searching in a local *services cache*, that holds a list
of service-PrP couples, obtained from discovery activities in the last period.
The behavior of peers involved in the discovery phase is influenced by:

- local services actually offered.

- service QoS level required in the request message.

- local services cache entries

- overlay network routing table, depending on reputation of neighbours list peers.

- the length of the path from RqP (distance in hops)

The discovery protocol is based on the flooding of Discovery Queris (DQ, see section 4.1.1). Generically, in a flooding schema, the number of messages (M_{out}) spread over the network for each peer is given by:

$$M_{out} = N^H \tag{4.1}$$

where N is the number of neighbour for each peer and H is the number of max hops foreseen by the protocol. Considering that in an overlay network the number of neighbour changes for each peer, i.e. each peer is free to establish several neighbour relationship, the formula 1 becomes:

$$M_{out} = \prod_{i=1}^{H} \bar{N_i} \tag{4.2}$$

where $\bar{N_i}$ is the average number of neighbours for each peer having a distance in hops equal to i. Before starting a distributed discovery on the overlay network forwarding a DQ message, each peer searches inside its services cache to look for entries that could match its requirements. If any entries are found, i.e. it knows the information related to the location of one or more possible PrP of requested service, it tries to contact the PrP(s) directly sending it(them)

a DQ (with $hops = 1$).

If no entries for the desired service are available in the local cache, the peer starts a distributed discovery, flooding the DQ to a subset of its neighbour(in a *selective* manner driven by their reputation value) and sets a timeout which measures the time interval within which RqP can wait for a useful response. This strategy determines a reduction in the M_{out}; the formula 2 becomes:

$$M_{out} = \prod_{i=1}^{H} \bar{N_i} * r \qquad (4.3)$$

where r is the reputation factor, having a value belonging to $[0.5, 1]$ represents the concept of selectivity. When a peer receives a DQ, it verifies if it is able to provide the required service and, if it can do it, checks the service level and the values of the parameters required. If it can satisfy all the QoS requirements, i.e. it is a PrP of the specific service, the peer sends the response frame named DiscoveryQueryHit (DQH, see section 4.1.1).

The DQH contains the reference to PrP location and all the useful information related to service fruition. In particular, the DQH contains a reference to an *SLA template* which is used, together with the values of the QoS parameters, to create the SLA that RqP will use to evaluate the convenience of the contract offered. Moreover, the DQH has a field, called infoPeer , used to support the reputation mechanism by allowing it to update the neighbour list with the identity of the last node that sent DQ to PrP. The RqP uses these indications, together with PrP address, to increment the value of the reputation for these nodes (see 4.1.1); if the latter is not present in the neighbour list, it will be added.

Unlike Gnutella, sending the DQH stops the propagation of DQs originating

from the PrP. This decreases the number of DQs transmitted on the network:

$$M_{out} = \prod_{i=1}^{H} [(\bar{N_i} * r) - P_i] \qquad (4.4)$$

where P_i is the number of peers able to provide the desired service. The reduction of M_{out} is greater the closer the PrP is to the RqP.

If the specified service is not present among the ones locally provided or if the peer is not able to match the QoS requirements of RqP, the peer that has received the DQ searches inside its services cache. If the peer finds one or more other peers providing the specified services, it stops the DQ propagation and forwards the request directly towards the found peers (with $hops = 1$). This operation both (1) speeds up the discovery algorithm (trying to establish a direct contact between PrP and RqP) and (2) reduces the number of DQs flooded through the network. In fact, as PrP, the peer that has found an entry of desired service in its cache stops the DQ propagation.

In addition, the above mentioned peer sends a Service Indication (SI), in backward routing, to all the peers that belong to the same network branch that links it with the RqP, thus allowing them to update their services cache. In fact, if the DQ has reached this peer it means that no previous peer has a reference to the required service in its cache.

This causes a new modification of the M_{out} formula, adding a new detracting term, P_{SI_i} that represent the number of peers sending an SI at ith hop.

$$M_{out} = \prod_{i=1}^{H} [(\bar{N_i} * r) - P_i - P_{SI_i}] \qquad (4.5)$$

The SI will also be used by the RqP to update its neighbour list: in particular, the SI sender address will be added to the list and, if already present, its repu-

tation value is increased.

If the peer that received the DQ does not find entries in the cache for that request, it checks the hop value in the DQ. If it is set to a value greater than 1 ($hops >= 1$) it then decreases it and sends the DQ to each peer of a neighbour subset chosen based on a reputation mechanism that allows a peer to select a subset of "better" ones. If $hops = 1$, the discovery protocol stops and the DQ is not forwarded.

The pseudocode below summarizes the actors' behavior:

```
1  RqP :
2
3  void directContact(DQ, ip,TTL){
4   send(DQ, ip);
5   if(DQHmanagement(request))
6       return SUCCESS;
7   else
8   distribuitedDiscovery(DQ,TTL);
9  }
10
11  int DQHmanagement(request){
12    waitFor(DQH);
13    if(receive(DQH, Template)){
14       startNegotiation();
15       cacheUpdate();
16       return(TRUE);
17         }
18    else
19       return(FALSE);
20  }
21
```

```
22  void distribuitedDiscovery(DQ, TTL){
23    neighboursList=choiceNeighour(idPeer);
24    send(DQ, neighboursList);
25  }
26
27  int main(){
28    TTL=N;
29    requestedService request;
30    ip=searchIntoCache(request);
31    if(!ip)
32      distribuitedDiscovery(DQ,TTL);
33    else
34      directContact(DQ,ip, TTL);
35    }
36  }


1
2  Intermediate Peer or PrP:
3
4  typedef struct service{
5     serviceId;
6     serviceClass
7     parametersList;
8  }requestedService;
9
10  int main(){
11    requestedService s;
12    waitFor(DQ);
13    s=extractFrom(DQ);
14    if(isPresent(s)){
```

```
15      send(DQH, Template);
16      startNegotiation();
17      return 0;
18    }
19    ip=searchIntoCache(s);
20    if(!ip){
21      TTL--;
22      if(TTL>=1)
23        distribuitedDiscovery(DQ,TTL);
24      else return 0;
25    }
26    else{
27      TTL--;
28      if(TTL >=1)
29        directContact(DQ,ip,TTL);
30        else send(SI,ip);
31          }
32  }
```

Fig.4.4 shows an example of DQ propagation.

4.1.3   Response's reception and agreement management

After having forwarded a DQ, the peer waits a fixed amount of time for the arrival of the DQHs. After every DQH has arrived, the peer extracts from its SLA field the template that the PrP had provided with their agreement. According to the proposed template and the negotiated QoS value, the RqP derives from the DQH the information needed for a contract and creates an SLA. The peer receives as many possible SLAs as the number of received DQHs: generally this number varies based on the required service and on the
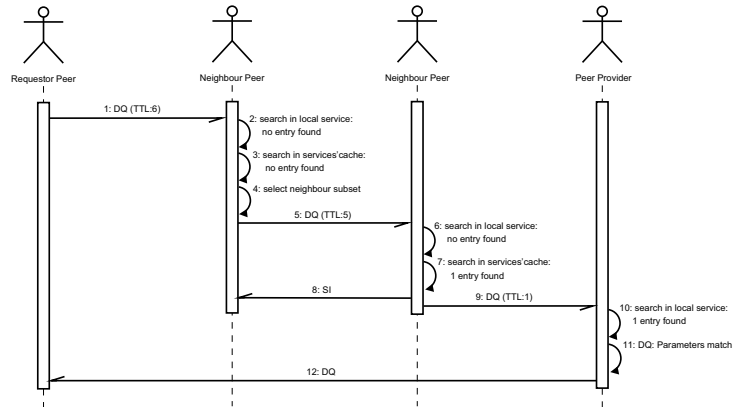
Figure 4.4: An example of DQ's propagation
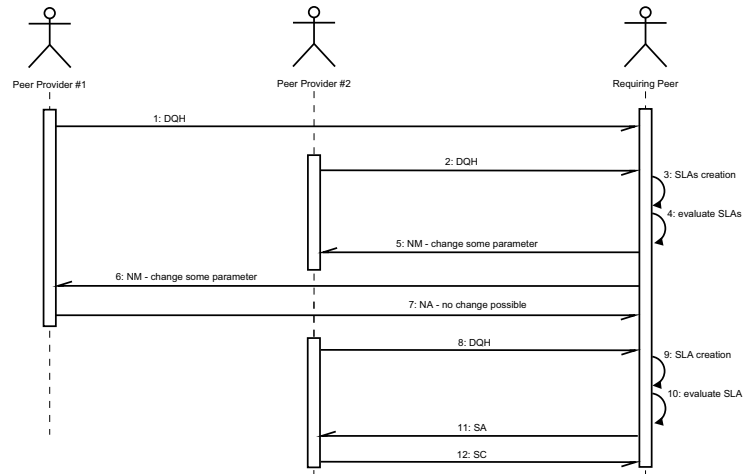
Figure 4.5: An example of negotiation phase

value specified for each parameter. The RqP is also able to re-negotiate the value of the QoS parameters by sending a Negotiation Message (NM), to a PrP, wherein it specifies the new values required. In the renegotiation phase, the PrP can decide whether to accept or refuse the new RqP requests, based on the state of the local resources. If the renegotiation is accepted, the PrP verifies its own availability and sends a new DQH to the RqP.

Finally, once the parameters for negotiation are terminated and the RqP has collected all the SLA proposed by PrPs, the RqP sends an SLA Accept (SA), to the chosen PrP that completes the SLA negotiation process by sending an SLA Confirm (SC). The structure of these message is simpler than the ones related to the service discovery, because they are exchanged between the PrP and RqP only to negotiate an agreement so they do not have any functionality linked to routing.

Fig4.5 illustrates an example of the negotiation phase.

4.1.4   Services cache

In order to speedup the discovery protocol, each peer maintains a local cache named *services cache*. This cache contains a limited amount of direct links to known service providers. In particular, the services cache records information relative to the locations (IP, port) of peers which have recently provided specific services. This information will be used as a shortcut whenever possible, thus reducing the number of messages sent through the network. The services cache is a hash table indexed by two keys: service name and service level. It contains, in addition to information about the location of the PrP and the name and the *level of service*, a *priority field*, which considers the type of frame that has provided the cached information (DQH, SI and SC), and a timestamp.

```
1  struct cacheEntry{
```

```
2          struct id_service serv;
3          struct id_class serv_class;
4          struct peer PrP;
5          struct timeStamp time;
6          struct priority prior;
7   }
```

   The cache (initially empty) is updated during the discovery phase. Each time a response frame is received, the peer extracts the service name, service level and PrP from it and then analyzes the cache content using this information to obtain the index of the above mentioned hash table. If an entry does not exist with this primary key, a new cache entry can be inserted in the hash table. Otherwise the priority field has to be checked: if the priority has the same value, it updates the entry if the new timestamp is more recent. If not, it updates the entry only if the new priority is higher.

To avoid large increases in the cache size, which is easily foreseeable with a distributed search algorithm, the peer uses a *reclaiming policy* when it reaches a predetermined threshold (90% of total size). This policy considers priority, timestamp and the number of entries for a given service. Initially the peer detects the services with the highest number of cache entries; for any specific service, the number of entries removed from the cache is proportional to their total number: the selection of an entry that must be removed is based on priority (SC, DQH and SI in thar order) and on timeStamp (oldest reference first). This system ensures that both: (1) the proportion between the services number and references number will be respected and (2) more reliable and recent information is maintained.

4.2 Results evaluation

The behavior of the proposed QoS-aware discovery protocol has been assessed by means of a network simulator.

The aim of these simulations is to measure the advantage derived by the use of the proposed strategy in terms of reduction of the number of frames involved in a wide area distributed discovery, with respect to the classic flooding schema, as is the one adopted in Gnutella v0.4.

In particular, the increment of scalability has been evaluated in terms of the number of messages spread over the network that represent the major drawback in flooding-based discovery algorithms.

It should be noted that the comparison focuses only on "quantitative" information, i.e. the number of total frames sent through the network, and it does not consider the frame structure or size. Remember that, there is a significant difference between Gnutella and the proposed discovery protocol: while Gnutella is used to share files, the proposed algorithm has been designed for discovery services under QoS constraints.

Each peer of the considered overlay network has a list of local services provided, a services cache of known couples *Service - PrP* and a list with the address of its neighbour and their relevant reputation value. The number and the type of services provided locally, as the initial cache size and its contents have been chosen randomly in the simulation. We evaluated:

- the average number of DQ and DQH frames spread over the network during a distributed discovery phase, used to measure the obtained increment of scalability.

- the ratio of number of DQHs and the number of DQs, that can be considered as an index of search efficiency.
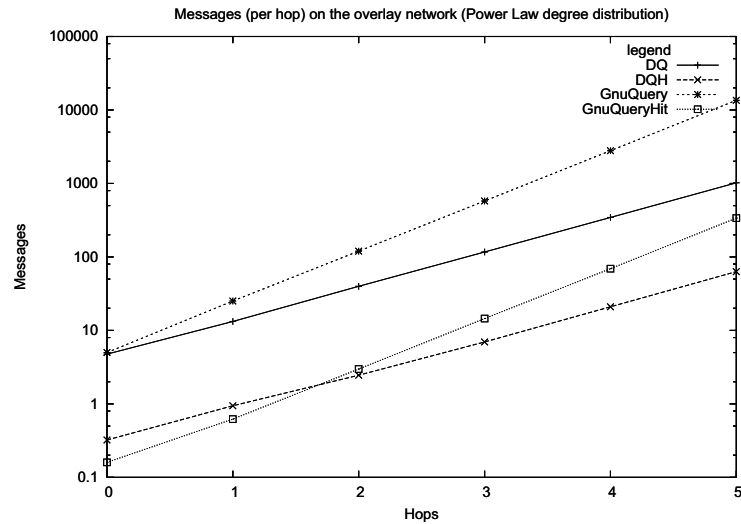
Figure 4.6: Average number of DQs spread in the (power law) network during discovery phase

The measures have focused on comparing the performance of the proposed algorithm and Gnutella v0.4. In addition, some measures aim to show the trend of our algorithm in changing same of internal aspects: in particular it will be shown the influence of reputation mechanism (used to select the best neighbour subset), of the increment of cache hit probability and of clustering coefficient ([53] in follow clustering index) .

Since the network topology influences the performance of flooding based search, all measures have been carried out for two networks topologies featured by a kind of neighbour distribution in power law and multimodal way.

Fig.5.3.2 and fig.4.7 show a comparison of the trend related to the average number of DQs (and DQH) per hops spread over the network (respectively for power law and multimodal ones) using the proposed algorithm and the Gnutella one. These measures have been done considering a reputation value
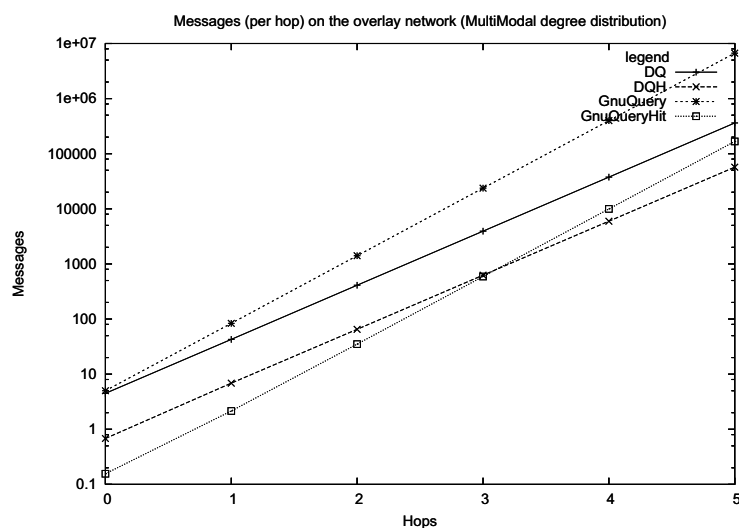
Figure 4.7: Average number of DQs spread in the (multimodal) network during discovery phase

threshold equal to 0.8 (i.e the flooding involves 80% of neighbour) and a clustering index equal to 0.2. The number of neighbour for each peer varies (in relation with selected distribution function) in the range [3, 40] and the hops value is equal to 6 (the Y axis has a log scale). As can be observed, our approach allows a significant reduction in the number of sent messages. In particular, the number of DQs created by our algorithm is less than Gnutella by a multiplication factor equals about 10. This trend can be observed in both overlay network topologies explored. The consistent reduction in DQs could suggest a proportional reduction in DQHs. Instead, figures 5.3.2 and 4.7 show that the number of DQHs is similar for both the algorithms: this is due to the reputation mechanism that allows an increase in the search efficiency of our approach with respect to the Gnutella one. This claim is supported by figure 5.3.2 and fig.4.9 that show the trend of search efficiency ($\frac{nDQH}{nDQ}$) for power
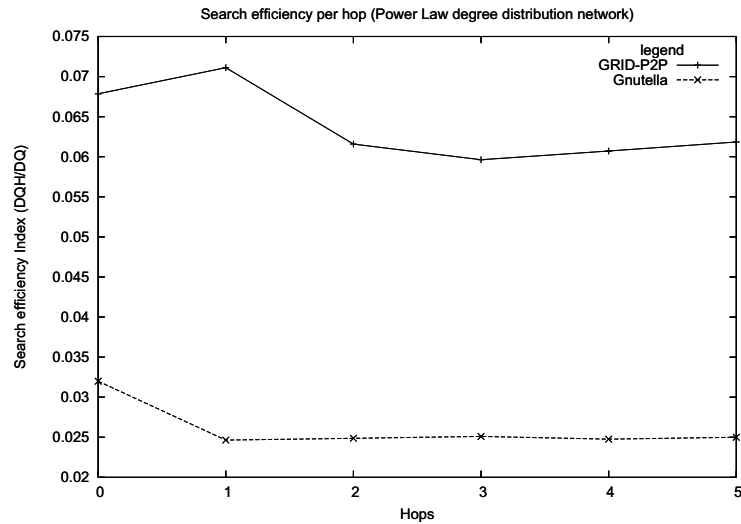
Figure 4.8: Search efficiency in the (power law) network during discovery phase

law and multimodal network respectively.   Figures 4.10 and 4.11 show
the effectiveness of the reputation mechanism on the discovery phase in both
overlay network topologies : using the reputation field, each peer is able to
route the DQ only to the neighbours that have demonstrated an active and effi-
cient role in the previous discovery. This means that the messages are flooded
in the peers having a high informative value.

Figures 4.12 and 4.13 show the effects of reputation mechanism on search
efficiency index. Each figure shows the trend of DQ and DQH when the repu-
tation value decreases, i.e. the number of neighbour involved in the distributed
discovery increases.

It can be noted that the reputation mechanism has a big influence on the dis-
covery algorithm. In fact, selecting only the best 80% of neighbour (i.e. a
reduction of neighbour equal to 20%), the algorithm produces only 25% of
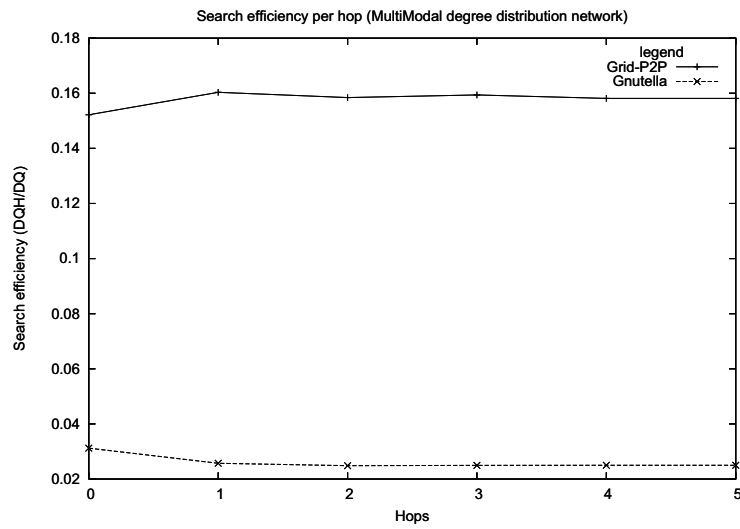
Figure 4.9: Search efficiency in the (multimodal) network during discovery phase
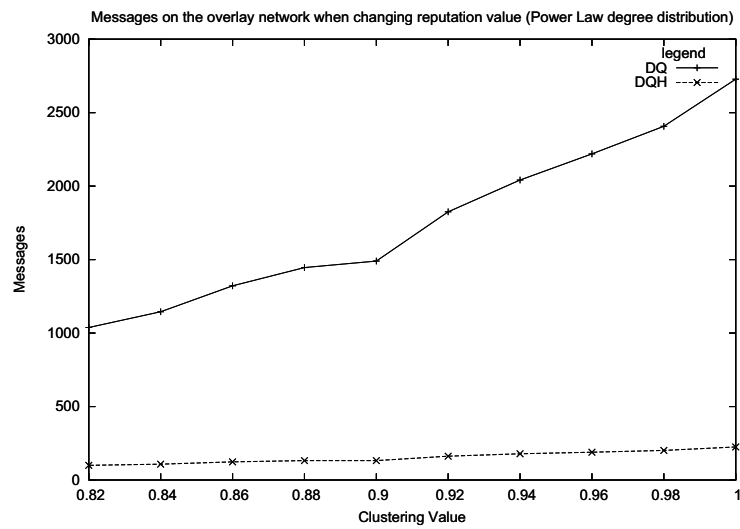
Figure 4.10: Influence of reputation on messages spread in the (power law) network
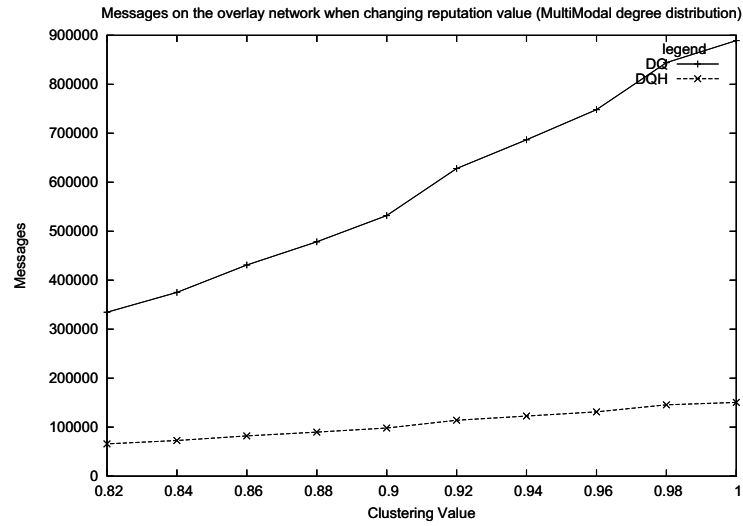
Figure 4.11: Influence of reputation on messages spread in the (Multi Modal) network
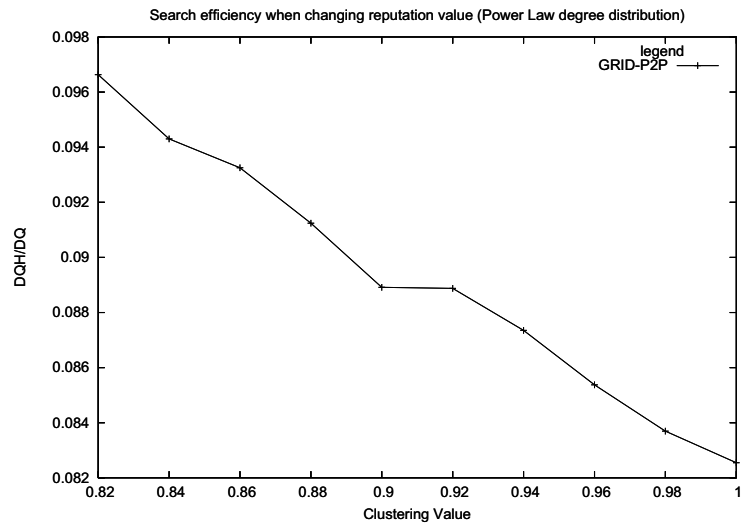
Figure 4.12: Influence of reputation on search efficiency index (power law network)

Search efficiency when changing reputation value (MultiModal degree distribution)
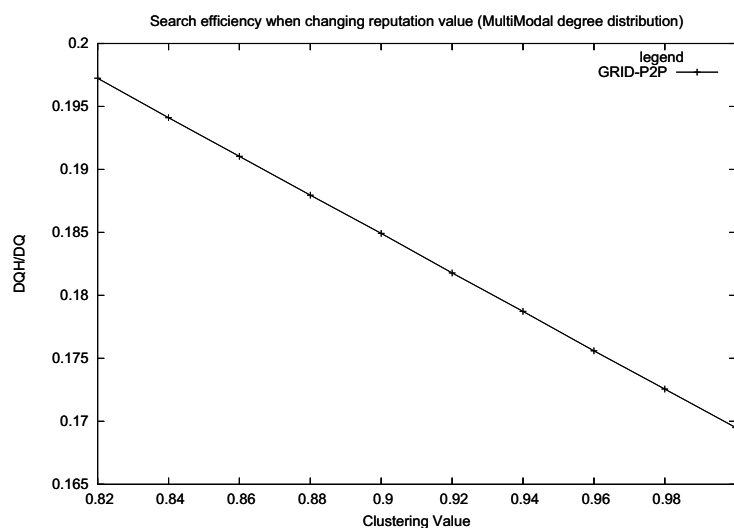
Figure 4.13: Influence of reputation on search efficiency index (Multi Modal network)

the messages it would have produced had all its neighbour been involved.

Figures 4.14 and 4.15 show the influence of cache hit probability on the discovery phase for both overlay network topologies under study. The measures have been done by increasing the cache hit probability from 0% (empty cache) to 5%. The results obtained illustrate the great importance that the services cache holds in our approach.

Another aspect that has been considered in these measures is the influence of the clustering index. It is important to consider this parameter since it is related to neighbour management. In fact, this index gives a measure of the existing relationship between the neighbour of a peer: in particular, this index indicates how the neighbours of a peer have themselves a neighbour relationship. This parameter influences the discovery phase because a peer that receives the same request message from several other peers, sends only
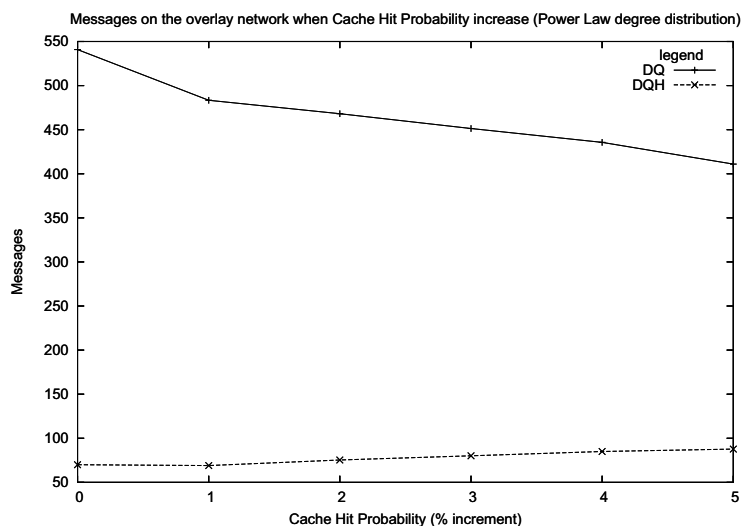
Figure 4.14: The influence of cache hit probability on messages spread in the (power law) network

a single copy of the message. This means that the bigger the clustering index, the lower the number of DQ messages flooded on the network will be. Unfortunately, as shown in figures 4.16 and 4.17,the decrease in DQs, due to the increase in the clustering index, corresponds in proportional decrease of obtained DQH.

 In conclusion, it can be said that this approach allows a reduction in the number of messages flooded on the network and, as a consequence, increases the scalability without compromising, in a strong manner, the number of obtained responses: these capabilities weigh on the search efficiency index that, in fact, is better than the Gnutella one. Moreover, the measures demonstrate that these improvements are independent from the network topology.

Messages on the overlay network when Cache Hit Probability increase (Multi Modal degree distribution)
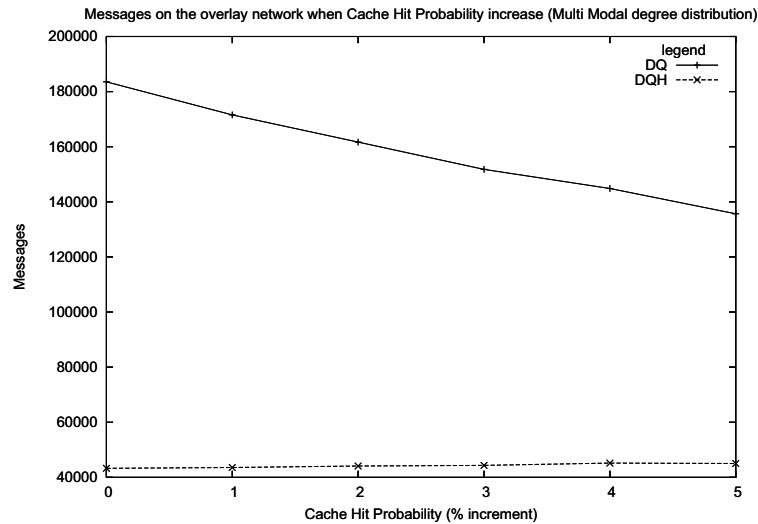
Figure 4.15: The influence of cache hit probability on messages spread in the (power law) network

4.3 Improving protocol performance using mobile agents

Although the led tests have shown the validity of adopted solution, the performance of the proposed protocol are bound by the slow update mechanism of service cache that, as said above, is based on the analysis of the synchronous messages exchanged . To overcome this drawback, it has been studied a different update cache technique, based on mobile agents technology [54–58], no more focused on analysis of frame forwarded through the network (passive update), but on autonomous research of information held in the other peers (active update). The main advantage of the agent-based update mechanism is represented by its independence from flooding-based discovery: each peer decides autonomously when updating its cache, sending a mobile agent
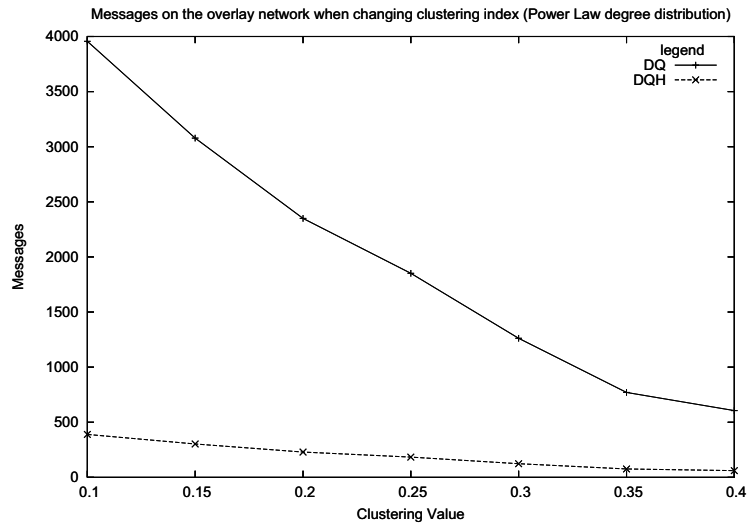
Figure 4.16: The influence of clustering index on messages spread in the (power law) network

to retrieval the information maintained in the various service cache of all the peers. This represents a great benefit because it allows to increase the informative content of the cache without waiting for the DQH or SI frames, reducing the start-up time and reaching quickly a *steady state* conditions.

Furthermore, this technique can be used to obtain some information about neighbour relationship maintained by host peer: this information will be exploited by mobile agent both to update own peer neighbour list and to create a discovery path based on reputation values.

In this context, each agent is defined by a behavior and by a dynamic search path. The behaviour specifies the set of operations that a peer executes when it arrives in the other peers. The search path, instead, specifies the list of peers that the agent will visit.

When a peer joins an overlay network, it creates a software agent, assigns it a

Messages on the overlay network when changing clustering index (MultiModal degree distribution)
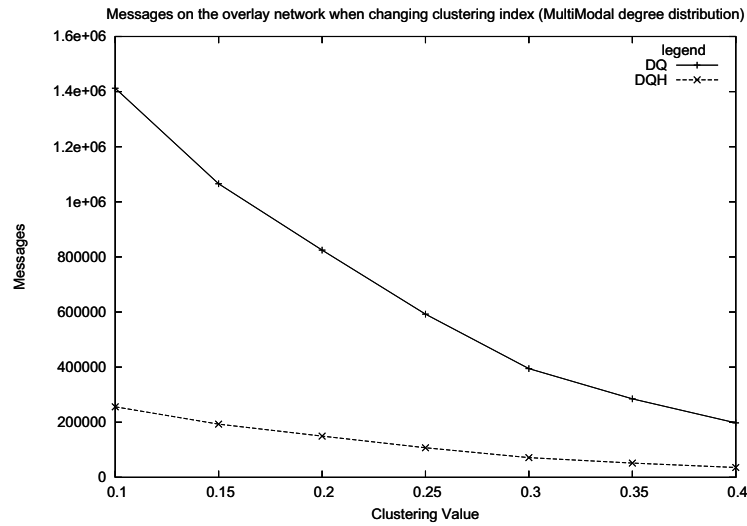
Figure 4.17: The influence of clustering index on messages spread in the (power law) network

particular behaviour and a starting search path, encodes it in a ACL message and, finally, forwards it to remote peers basing on the assigned path.

The mobile agent, visisting each peer belonging to its serch path, executes the operation assigned by its source peer.

In this work are defined:

- three basic behaviours

- two path management policies

The basic behaviour are:

1. cache update

2. services discovery

3. neighbour update

In the *cache update* mode the agent, routing through the peers of assigned search path, makes a copy of information maintained into local service list and on service cache of the currently node and, at the end of each travel, carries back this information to source peer.

The *services discovery* mode represents an alternative way to discovery service: it allows to search one or more services references on peers belonging to search path.

The last basic behaviour, *neighbour update*, is substantially used to enlarge the portion of overlay network seen by peer and to increase its neighbour "quality" level. Both features are obtained through a selection of the best peers (based on reputation values) belonging to neighbour list of visited ones. The path management policies are:

1. source-chosen approach: the path is assigned, each time, by souce peer

2. adaptative approach: the path is defined, at run-time, by agent

In the source-chosen approach, the path is assigned *a priori* by source peer: usually, the peer assigns as path a subset of its known neighbours to deepen the knowledge of its overlay network portion.

In the adaptative approach, instead, the search path is established by the mobile agent. At each migration step, the agent selects a target peer based on some parameters, like peer reputation value in neighbour list or the number of references of a peer into service cache.

The selection criteria and the number of migration step is fixed by souce peer. Each agent, futhermore, has a mechanism to control to avoid an excessive grown of information size collected. Its retrieval algorithm, in fact, is able to recognize and delete the redundant information, with a policy similar to the

one used by source peer to manages its services cache.

The policy used by agent takes into account (1) the number of entries for a given service and (2) the value of priority[1] field.

4.3.1 Results

The software infrastructure has been developed through Java Agent DEvelopment Framework (JADE) v3.4.0 [59, 60], an open source framework that allows implementation and deployment of multi-agents system (complies with FIPA [61] specification), and through an specific add-on, called *migration*, that enables the *interplatform* agents migration, a foundamental requirement in a distributed environment like the considered one. The use of this add-on is necessary because the basic JADE version allows only the *intraplatform* agents migration mode that enables agents transfer only if the target destination belongs at the same platform.

Figure 4.18 shown the two different ways to exchange agents.

Although this platform can be distributed on various machines, it is characterized by a single AMS agent[2] and by a single DF agent[3] hosted in the server machine containing the JADE main container, creating a single fault point[4]. The *migration* add-on permits that each peer can create the own platform with specific AMS agent (indipendent from other ones): in this way the mobile

---

[1]Each entry has a priority value assigned by peer basing on the type of frame providing the information (DQH or SI)

[2]Agent Management System, identifies and register the agent present on managed platform

[3]Directory Facilitator, performs the Yellow pages service

[4]If server containing AMS crushs, all agents belongin to platform became unmanageable
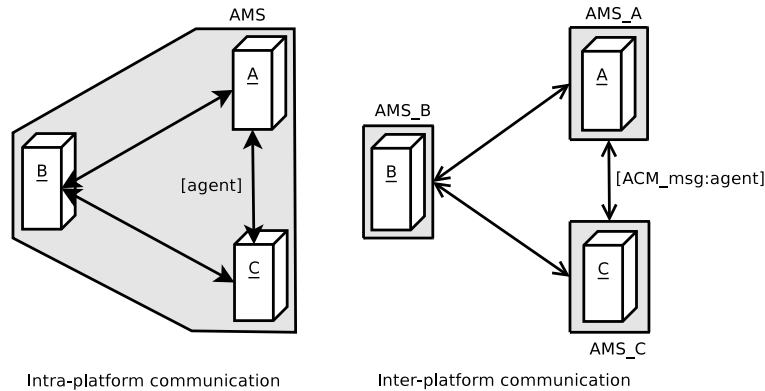
Figure 4.18: Intraplarform vs interplatform

agents is encoded and exchanged among peers[5] through ACL[6] messages.

All developed software has been implemented and tested on Globus Toolkit v4.0.1. A simulation campaign has been led to show the growth model of information maintained into the service cache related to both "search path" policy used and visited node number.

The simulation has been done considering an overlay network composed by 5000 peers: each peer has a service cache and a neigbours list (30 neighbours for each peer). The initial cache size and its content are choosen in a statistical way.

Figures 4.19 shows the cache hit probability trend(CHP) using "cache update" behaviour and both source -chosen and adaptative approach.

In this example the source chosen "search path" contains all peer's neighbours, while, in the adaptative approach, the "search path" has been made selecting the best neighbour on each visited peer (30 step). The differences

---

[5]Local AMS sends agent to remote AMS without estabish an additional communication channel.

[6]ACL messages complies with FIPA specification.
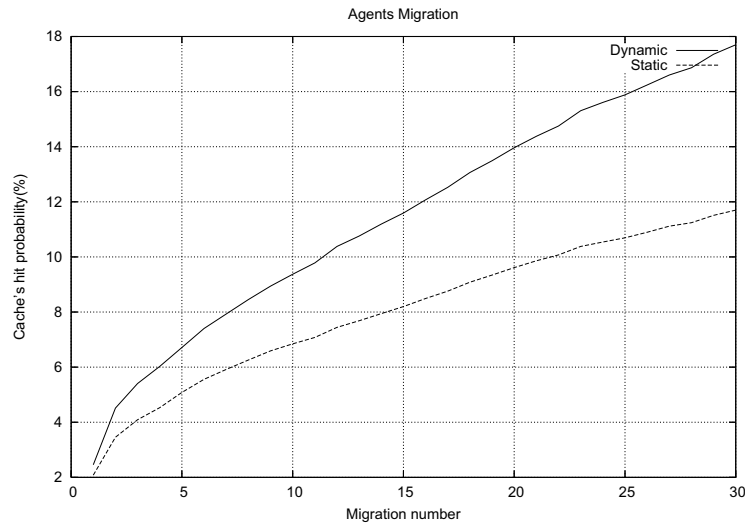
Figure 4.19: Adaptative Vs Source chosen policy

between the two trend is mainly due to "local nature" of source chosen approach, i.e. caches of near peers tend to contain the same information.

When the agent accesses to cache of a neighbour peer, it has a low probability to find a new service and, as a consequence, a low probability to increase the source peer's CHP. Otherwise, it is more probable that the agent find a new reference of a known service: this explain the trend shown in figure 4.20, that represent the relationship beetwen cache size[7] growth and cache hit probability for both approaches.

---

[7]The size depends by (1) service number and by (2) total entries number (each services can have one or more references).
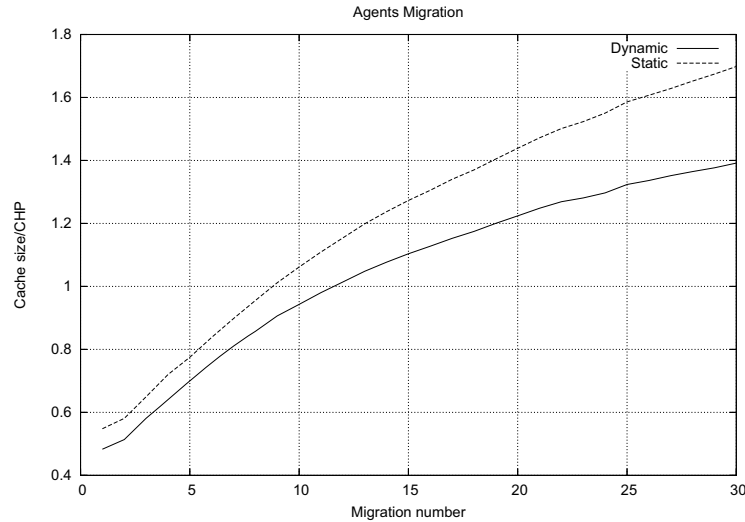
Figure 4.20: Size cache / Cache hit probability

4.3.2   Security policies

Security is an issue particularly critical when using mobile agent [62–65] so-lutions.

For the scope of this work, it will be taken into account only the agent-to-platform threat[8] and, in particular, the problems related to unauthorized ac-cess to local information repository.

When an agent arrives on another peer, it is able to accesses to the services cache and the neighbour list to achieve its task: if it is a malicious agent, it could replace correct information with mistaken ones. This misconduct can carry a discovery protocol performances worsening because both cache and neighbour list can address wrong service provider peer, putting in the overlay network lazy traffic.

---

[8]according to threat definition proposed in [65]

In order to avoid these problems, all resources which could be accessed by external agent have been hooked by an access control list. In this way, only the administrator can modify their content: the others can access the shared resources in a *read-only* mode. No one policy, instead, has been adopted to solve reverse problem, i.e malicious platform that provides misleading information to mobile agents. This choose has been taken because the information provided by untrusted peer have only a light effect on source peer service. In fact:

1. the mobile agents takes information by many peer: if search path include an enough high number of peer, the malicious information will represent only a minimum subset.

2. the information provided by mobile agent are used by source peer mainly to update cache, not to replace its entry.

Furthermore, the reputation mechanism used to manage neighbour list tends to isolate the peers that provide incorrect information compared to peer providing trusted ones.

4.4 Conclusions

In this chapter an algorithm for services discovery in a P2P-Grid-based service marketPlace has been depicted. The algorithm is based on *selective flooding*, where any service is searched for specific QoS constraints forwarding queries to the overlay network of the CE through sets of neighbours. The queries propagate according to a reputation strategy that allows a peer to select a subset of "better" ones and exploiting a cache of frequently used

services in order to speed-up discovery mechanisms. The performance evaluation results allowed to highlight the low average number of frames spread in the network due to the single discovery request. This demonstrated that the technique adopted reduces the bandwidth consumption related to the flooding and, at same time, is still able to discover a high number of peers that provide the required services.

Chapter 5

QoS-aware Service Composition

The cooperation between resources managers belonging to different administrative domains has been one of the open issues of Grid Computing [**?**]. The different solutions used for the description of resources or jobs, the different policies for the management of security, of authorizations and of accounting policies, in fact, represented a strongly limit the possibility of integration among different middlewares. The adoption of interfaces based on Web Services technologies (above all WSDL [66] and SOAP [67]) has considerably simplified these problems making easier the interoperability between different Grids.

This has permitted the creation of new applicative scenarios [6–8, 68] where grids become peers of a strongly "service oriented" overlay network, and where these peers interacting each other to provide and consume services. This scenario is intrinsically competitive: different peers can provide many versions of the same services, different from the others for the cost, the availability, the execution time and for other parameters related to the Quality of Service (QoS)([11]).

In a the last chapter, I have introduced a possible model of P2PGrid scenario and have proposed a solution for the discovery of services under QoS constraints.

That work, that considers the same P2PGrid scenario, proposes a technique for composing services belonging to different providers, guaranteeing the maintenance of a specific QoS level and high resilience.

The proposed solution, in particular, is characterized by a fast recovery strategy, based on the concept of *QoS compensation*, and takes advantage by the peculiarities of the graph representing the execution path for composed services.

This process of services composition is realizes in three different phases: discovery of services, creation of execution path and, in case of services failures, management of exception.

5.1 The management of QoS in composed services

The management of Quality of Service (QoS) represents an important aspect in *service oriented computing*. In a services marketplace as the one considered in this work, in fact, the QoS gives users the basic support for identifying and choosing, basing on users expectations, the best one among different implementations of a given service.

The concept of QoS is based on the expectations of users about services execution and takes into account the aspects related with the question "How is the service executed ?" rather than "What does the service do?".

Two services, in fact, can have the same functionalities but different performance. A service, for example, can run over a better workstation (having

better hardware devices), connected to a faster network, or can be distributed over multiple workstations or rely on optimized computing libraries and ad hoc software. All these aspects do not change the output (in terms of given results) of a service but they influence several "*non functional* parameters", such as the "response" time, the "queue waiting" time, the availability, the reliability and the cost, that are related with the "quality" of execution.

For a composed service S_{cmp}, the QoS is a function of the QoS of the n basic services S_i, $i = 1..n$:

$$S_{cmp}|_{QoS} = f(S_1|_{QoS}, S_2|_{QoS}, .., S_n|_{QoS}) \tag{5.1}$$

where $S_i|_{QoS}, i \in [1, n]$ represents the set of values of *non functional* parameters related with the ith service.

In order to give a measure of composed QoS it is necessary to specify the function $f(*)$.

If it is possible to identify one or more common parameters among the basic services, e.g. response time, availability or cost, the function $f(*)$ can be simply expressed as the "sum" or the "average" of each single value. For example, if it wants to execute a service composed by n different basic services, executed sequentially within a fixed time frame, it is necessary to impose that $S_{cmp}|_{maxT} <= MaxTime(maxT)$, where $MaxTime$ is the deadline specified by the user.

In this case $f(*)$ can be represented as:

$$S_{cmp}|_{maxT} = \sum_{i=1}^{n} S_i|_{maxT} \tag{5.2}$$

If, instead, the services can be executed in parallel, the function $f(*)$ becomes:

$$S_{cmp}|_{maxT} = \max_{i=1..n} \ S_i|_{maxT} \tag{5.3}$$

If it wants to have a composed service with an average availability (*av*), at least, equals to 80%, the function $f(*)$ can be expressed as:

$$S_{cmp}|_{av} = \frac{\sum_{i=1}^{n} S_i|_{av}}{n} \leq 80\% \tag{5.4}$$

Many times, however, especially when the composed service takes into account many different parameters, it is very difficult to express the function $f(*)$ using a simple formula. For example, when the required QoS takes into account different features of each single basic service (e.g. the number of CPUs in a render farm or the replicas number in a storage system), it is difficult to provide an function $f(*)$ expressed using all the involved QoS parameters.

In order to simplify the management of heterogeneous parameters, I introduced the concept of *user satisfaction*. The *user satisfaction* value, obtained applying the normalization process to each considered function, represents a set of "non uniform" QoS requirements through a single numeric value. This value, ranging between a min($= 0$) and a MAX($= 1$), is calculated for each basic service: depending on the parameters involved, the "sum" or the "average" of these numbers can be considered the index of QoS for the composed service.

An example of normalized function is the *normalized differences average*. This function is not related to a single parameter but to the ratio between the difference of the value of parameters required by the user ($U|_{QoS}$) and the one

provided by the service ($P_i|_{QoS}$), as numerator, and the value of parameters required by the user ($U|_{QoS}$), as denominator.

This function is:

$$0 < \frac{(U|_{QoS} - P_i|_{QoS})}{U|_{QoS}} < 1 \tag{5.5}$$

In order to better explain the advantages obtained using the *normalized differences average* function, a simple example is given below. A service composed by two simple services is taken into account: the first requires a fixed amount of CPU (n_{CPU}) and bandwidth (bdw), the second a certain number of data replica(n_{rpl}).

The function providing the total QoS is:

$$S_{cmp}|_{QoS} = f(S_1|_{QoS}, S_2|_{QoS}) \tag{5.6}$$

$$= \frac{\left(\frac{n_{CPU}|_u - n_{CPU}|_p}{n_{CPU}|_u} + \frac{bdw|_u - bdw|_p}{bdw|_u}\right) + \frac{n_{rpl}|_u - n_{rpl}|_p}{n_{rpl}|_u}}{N} \tag{5.7}$$

where N is the number of involved parameters.

If the user require $n_{CPU} = 8$, $bdw = 4GBit$ and $n_{rpl} = 4$ and the services are execute with $n_{CPU} = 6$, $bdw = 3.5GBit$ and $n_{rpl} = 3$, the *user satisfaction*, calculated using the above mentioned formula, is:

$$= \frac{\left(\frac{8-6}{8} + \frac{4-3.5}{3}\right) + \frac{4-3}{4}}{3} = 0.78 = 78\% \tag{5.8}$$

Instead, if the services are execute with $n_{CPU} = 4$, $bdw = 3GBit$ and $n_{rpl} = 2$, the *user satisfaction* is:

$$= \frac{\left(\frac{8-4}{8} + \frac{4-3}{4}\right) + \frac{4-2}{4}}{3} = 0.58 = 58\% \tag{5.9}$$

It is possible to refine the *normalized differences average* function assigning
a weight to each involved parameter: in this way, each parameter can have an
own specific importance.

$$S_{cmp}|_{QoS} = \frac{f(w_i * S_1|_{QoS}, .., w_n * S_n|_{QoS})}{\sum_{i=1}^{n} w_i} \qquad (5.10)$$

5.2   Service Level Agreements

The Service Level Agreements (SLA) represent important instruments for the
QoS-aware management of services. An SLA is a formal deal, established
between the user and the provider, about all the parameters characterizing a
specific service, both functional and QoS, and the rules and the conditions for
the service fruition.

There are many implementations of SLA (WSLA [69], WS-agreement,
SLAng), each one taking care of the specific aspects of the applicative context
where it is used. In general, each SLA implementation foresees two different
parts.

The first one contains some technical specifications:

1. service name

2. service description, in terms of required input and provided output pa-
   rameters.

3. involved participants: provider, user and any *third party entities* able to
   ensure a "trusted" execution of the service.

4. service access mode, i.e. protocols or exchanged messages.

The second part, instead, is used to specify the *non functional* parameters related to the service:

1. service cost

2. QoS section, consisting of the list of all the QoS parameters, negotiated between user and provider, with their respective values(or ranges).

3. exceptions section (optionally), i.e. the list of couple ("condition over parameters", "cost"), negotiated between user and provider, used as penalties (in terms of reduction of the service cost) if faults happen or as benefits in case of better (in terms of values of QoS parameters) service execution.

In this chapter, I take into account two important topics related to SLA management.

The first one regards the use of *SLA templates*, i.e. standard pre-defined structures that define the behavior that provider and user have to adopt to reach a common target. These structures are very useful in that they simplify considerably the negotiation phase because they put the focus directly on "what is provided".

The second aspect, fundamental in the proposed SLA management system, regards the capability of establishing specific boundaries when, after an agreement violation, recovery activities have to be started. An SLA is a binding agreement: a provider, that is selected based on its promised performance, is bound to guarantee the proper service fruition. If this does not happen, the user must be refunded, in a partial or total way, based both on the service level and on the parameters involved in the failure.

An example of "exceptions" section in an SLA is:

```
QoS section:
```

```
responseTime = 375 sec
service cost = 2$
4
Exception section:
if responseTime <  350 sec
7       then cost = 2.5$
if 400 < responseTime < 500 sec
9       then cost = 1.8$
11if 500 < responseTime < 600 sec
11      then cost = 1$
11if responseTime > 600 sec
13      then cost=0$
```

The exceptions are very important in services composition because they allow to build alternative solution for recovering the possible faults. In fact, using ad hoc techniques for the management of exceptions, it is possible to change, at run time, the set of services involved in the composition in order to recover delays and agreement violations.

5.3   A fast technique for the composition of services

The composition of services is an important issue in Service Oriented Architectures because it allows the creation of new complex services by the use of autonomous and independent simple services. The proof of this is given by the great quantity of scientific works (CANS, BPEL4WS, SWORD, QUEST [70]) that take into account this issue (see Section 5.4) and propose different strategies for having robust, adaptable, reliable and fault tolerant

composed services. Each composition strategy, in general, foresees three main phases:

1. the discovery phase, i.e. the phase where the possible providers of each basic service is identified;

2. the execution phase, i.e. the phase where the execution path is set up and the composed service is run;

3. the exception management phase, i.e. the phase where it tries to restore the correct execution of a composed service after that a fault in one or more involved services has happened.

After a short description of the service discovery protocol adopted in this work, the attention will be focus on the strategies used both in execution and exception phases.

5.3.1 The services discovery phase

The first step in a process of services composition consists in the search of providers for every basic service. It should be note that it is important to have a certain number of providers for each service in order to increase the probability to find providers able to fulfill the user requirements. There are many strategies for services discovery, different both for the technology used and for applicative scenario: e.g. web services use UDDI, grid services use MDS or RGMA. The protocol proposed in the last chapter, uses an optimized version of "flooding algorithm" to distribute the discovery queries over the entire network. The use of QoS constraints inside the queries allows the identification of only those providers able to supply services fulfilling the user requirements. Here, a brief description of that protocol is given: for details

see chapter 4.

Using the nomenclature introduced in that work, the service user and the service provider will be indicated as RqP (Request Peer) and PrP (Provider Peer) respectively. The RqP sends, in flooding to its neighbor, a DiscoveryQuery(DQ), a message containing the service name and the QoS requirements; the RqP sends a DQ for each basic service and waits for the arrival of a DiscoveryQueryHit (DQH). The DQH is a response message, sent by the PrPs that are able to satisfy the RqP request: this message contains information about the service location, the values of QoS parameters that the PrP can supply and an SLA template (5.2) used to create an agreement between parties.

The RqP can receive a huge amount of DQHs: for each received DQH, the RqP extracts information about the provider location and QoS values proposed and, according to the template, creates an SLA.

The RqP can accept the PrP SLA terms confirming it through an SLA Accept (SA) message; on the contrary, the RqP can renegotiate some QoS values exchanging a given number of Negotiation Messages (NMs). This ability is an important feature because, as will be explained in 5.3.3, it allows to create a robust "execution path", increasing the fault tolerance of the composed service. Once the negotiation is finished, the RqPhas an SLA for each DQH and, as a consequence, more providers for each service: the services discovery phase is completed and it is possible to start the next phase: the execution one.

5.3.2   Setting the path

A composed service S_{cmp} can be considered as a combination of basic services $S_1, S_2, .., S_n$, arranged in serial or parallel manner or, more generically,
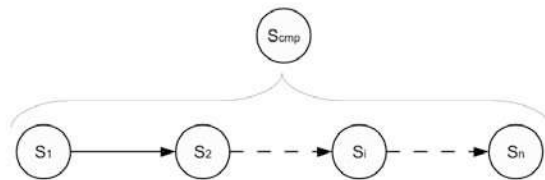
Figure 5.1: Example of serially composed service

in a more complex DAG. In order to give a clearer explanation, I take into account only serially composed services(see fig. 5.1): the proposed solution, however, can be easily extended to other cases.

The creation of a execution path for the S_{cmp} consists in the selection of a provider for each service $S_i|_{i \in [1,n]}$, and their sequentially invocation.

Considering the providers as vertexes and the links[1] between them as edges, the "optimal" execution path can be obtained referring to the graphs theory.

All the PrPs involved in the composition process are clustered basing on type of supplied service (see fig. 5.2): e.g. the providers PrP_{11} and PrP_{12}, both suppling the service S_1, are collected together and are divided from PrP_{21}, PrP_{22} and PrP_{23} (suppling the service S_2) and from PrP_{31} and PrP_{32}

---

[1]Two providers are linked if they are invoked sequentially
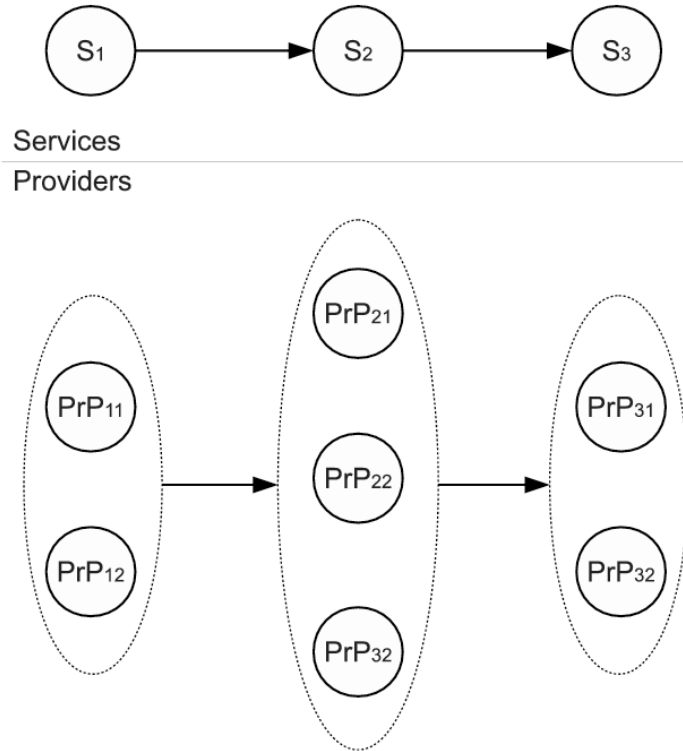
Figure 5.2: Clusterization of SLAs

(suppling the service S_3).

It should be noted that, since the execution path must have only a single PrP for each basic service, there cannot be links between two PrPs belonging to the same service.   Under the above mentioned considerations, each composed service can be considered as a directed graph: $G(V,E,C,Q)$ where

1. $V = \{PrP_{ij} \mid i \in [1,n], j \in [1,n_{S_i}]\}$ is the set of providers;
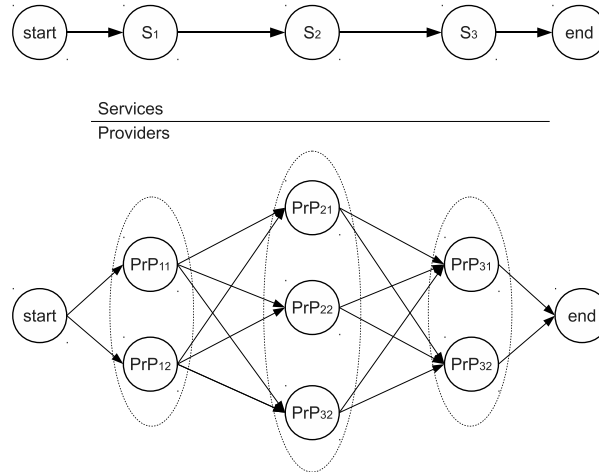
Figure 5.3: The graph representing S_{cmp}

2. $E = \{e_{ij} \mid i, j \in V\}$ is the set of edges. An edge between two providers exists if and only if the providers are invoked in sequence;

3. $C = \{c_{e_{ij}} \mid i, j \in V\}$ is the set of costs ($c_{e_{ij}}$ is the cost of the edge e_{ij});

4. $Q = \{q_{S_i} \mid i \in [1, n]\}$ is the set of values of the *user satisfaction* required for the execution of services.

The figure 5.3.2 shows the complete graph with two additional sham vertexes that represent, respectively, the points of departure and of arrival.

The fundamental characteristics of this graph are the following:

- the PrPs belonging to the same service have the same number of element in the InDegree and in the OutDegree set.

- the weights (i.e. the elements belonging to set Q) related to the edges belonging to the InDegree set of each PrP of the same service S_i have the same value q_{S_i}.

- the weights related to the edges belonging to the OutDegree set of each PrPof service S_i, have the same value q_{S_i+1} .

Considering these conditions, the problem of service composition can be expressed in terms of searching of the "less expensive" path (abstraction of the minimum path) that guarantees a global QoS level, given by the *user satisfaction* calculated with the function "normalized differences average", higher than an established minimum threshold (in reference to the search of a constrained minimum).

This problem, that can be expressed like a multi-constrained path selection problem, can be resolved using the well-known Dijkstra algorithm. This is possible since the cited(5.3.1) algorithm for services discovery, guarantees that every PrP in the graph is able to provide, at least, the desired QoS level and, as a consequence, the selection of providers can be done considering only the costs associated to each edge. Under these conditions, applying the Dijkstra algorithm, it's guaranteed that not only the total path has the minimum cost but also that whichever path, between any two nodes PrP_{i*} and PrP_{j*}, with $j > i$, is the minimum. This particular property gives more robustness to the execution path because it guarantees that, if the PrP related to S_i fails, it is not necessary to renegotiate the agreements with the whole set of services between S_i and S_n (the last one) for obtaining again the minimum path[2], but only for the failed service(S_i).

In fact, as said above, the path included between S_{i+1} and S_n remains the minimum path: choosing the PrP with the minimum cost among the known ones

---

[2]As happens in the classical Dijkstra approach.
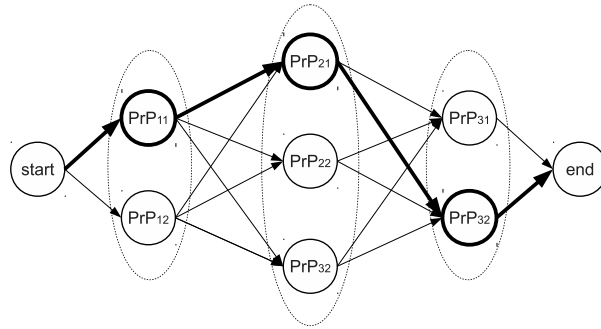
Figure 5.4: An execution path for S_{cmp}

that offering the service S_i, it is possible to rebuild again the whole path with minimum cost.

The figure 5.3.2 shows a possible "execution path" for the composed service S_cmp.

5.3.3 The management of exceptions

As described in "ten pillars world class business process management", nearly 80% of the time spent in building composed business processes is spent in exceptions management. This means that each technique of services composition has to focus particular attentions on this critical step. The correct execution of a composed service depends on the correct execution of all basic services that compose it: if not adequately managed, a failure in one of basic

services can degrade the performance the whole process. Since the considered
scenario, due to its dynamism, its heterogeneity and above all, its distributed
nature, is subject to several kinds of failures, it is essential to adopt several
procedures and/or alternative behaviors in order to restore, in total or - at least
- partially, the correct functioning of composed service in case of failures.

Usually, when an failure in a basic service S_i occurs, the value of one or more
QoS parameters can be corrupted, degrading the QoS of the entire process.
In order to avoid this degradation, it is necessary to recompute a new path,
from the service S_{i+1} to the service S_n, in that the ones previously chosen,
having pre-negotiated agreements, are not be able to restore the QoS level.
The re-computation of a path requires:

- a new discovery phase in order to identify the new providers.

- a new negotiation phase in order to establish the new values of param-
  eter in order to complete the process with the desired level of QoS.

- the break of the agreements (SLAs) with those providers that belonged
  to first path and do not belong to the new one.

The solutions proposed in literature manage all these issues trying to avoid, or
at least minimize, the degradation of QoS level. Here, I proposed a different
approach for a fast path re-computation: this techniques, based on the concept
of "compensation", aims to overcome the mentioned issues exploiting the
"exceptions section" of the SLAs established with the providers of the basic
services involved in the composition.

The idea behind this approach is very simple: if the service S_i does not respect
the agreements stated in the related SLA, degrading the overall QoS level, the
user (or an proxy acting on behalf of it) uses the penalties that the provider of
the *ith* service has to pay to improve the performance of one or more services

among the ones remaining. Considering, for example, the SLA stated with the service S_i:

```
QoS section:
responseTime = 150 sec
service cost = 5$
4
Exception section:
if 160 < responseTime < 200 sec
7       then cost = 4$
if 200 < responseTime < 300 sec
9       then cost = 2$
11 if responseTime > 300 sec then cost=0$
```

If the service has a responseTime greater than 150 sec, the SLA is considered broken.

The user, knowing that an SLA violation is occurred, it can invoke one of the next services warning them, through an apposite signal (e.g. the boolean variable FAULT), that there was a fault. Invoking a service using FAULT=true, the user forces the provider to try fault recovery.

E.g, the service $S_j|_{j>i}$, with an SLA like:

```
QoS section:
responseTime = 100 sec
service cost = 3.5$
4
Exception section:
if FAULT=true
7       then if 90 < responseTime < 60 sec
8            then cost = 4$
```

```
9                    else if responseTime < 60 sec
10                       then cost = 5.5$
11   if 110 < responseTime < 150 sec
12          then cost = 3$
13   if 150 < responseTime < 200 sec
14          then cost = 2$
15   if responseTime > 200 sec
16          then cost=0$
```

it is able to recover the given QoS maintaining the total cost under an established threshold.

Using this technique, under given conditions, it is possible avoiding the QoS degradation (or reducing it to an acceptable value) maintaining the overall cost under an established threshold, without starting a new path computation, thus to reduce the number of new involved providers and, as a consequence, both the negotiation phases and the number of broken SLAs.

Let S_{cmp} the composed services and $S_i, i \varepsilon [1, n]$ the services that compose it.

Let QoS_{cmp} the value of expected QoS and $QoS_i, i \varepsilon [1, n]$ the value of QoS related to the correct execution of S_i.

Let D_{QoS_i} the degradation of QoS due to the fault occurred in S_i (if S_i goes down $\triangle QoS_i$ will be equals to QoS_i) and $\triangle QoS_i$ the added value of QoS offered by S_i if the user requires a compensation approach.

The "compensation" of the D_{QoS_i} due to a fault in S_i consists in the identification of the provider of k*th* service, $S_k, k > i$, able to satisfy the follow conditions:

$$\begin{cases} D_{QoS_i} - \triangle QoS_k < th_1 \\ \\ Cost_{\triangle QoS_k} - Cost_{D_{QoS_i}} < th_2 \end{cases} . \qquad (5.11)$$

where th_1 and th_2 are two thresholds related with the QoS degradation and with the maximum cost acceptable, respectively.

In order to automatize the selection process and to make these operation faster, the user (or an proxy acting on behalf of it) is supported by two *ad hoc* hash tables that suggests the best solution for both th_1 and th_2 parameters. The first hash table is indexed by the value of $\triangle QoS$ and, each entry, contains a list, ordered by the cost, of all the services that are able to compensate that QoS value. The services are inserted in the hash table at the end of negotiation phase, after that all the SLAs are stated. For each SLAs, the exceptions section is analyzed: for each additional opportunity given by the provider, the value of $\triangle QoS_i$ is calculated and the service S_i is inserted in the related entry. It should be note that each service can appear several times, as many as the opportunities given in the SLA. When different services provide the same value of $\triangle QoS$, they are inserted in the hash table under the same index (collision) and sorted using their cost. The second hash table, instead, is indexed by the cost and the services, in each entry, are sorted by the $\triangle QoS$. The process for the creation of this hash table follows the same rules (inverting the cost and the $\triangle QoS$) used to built the first one. If, during the execution phase, an error occurs (e.g. in the service S_i), the user (or an proxy acting on behalf of it) calculates the value of D_{QoS_i} and the penalties that the provider of service S_i has to pay (in term of refunding cost). Basing on these values and on th_1 and th_2, the user can obtain the reference to one (or more) provider that:

- at the same cost (considering the th_2 value), allows to minimize the QoS degradation,

- maintaining the required QoS level (considering the th_1 value), allows to minimize the increment of cost,

- is able to maintain the required QoS level at the same cost (considering the fluctuation due to both $th1$ and $th2$).

In case that the provider of S_1 fails, going down, the user has to find a new provider and complete the negotiation phase. In this case, as for services discovery phase, the negotiation phase can obtain a significant advantage using the searching technique discussed above: this protocol, in fact, foresees the use of Negotiation Message (NM), a special message that allows SLA renegotiation, adding new clauses at runtime. This technique is very useful when:

- there are a composed services with "deadline", in that it avoids searching of new providers for establishing new path and negotiation QoS values, saving time;

- the failure involves the first services, because there are many other providers that can be used to compensate the QoS decrement, and, as a consequence, there is a greater probability to minimize, or eliminate at all, both additional costs and/or the reduction of QoS level;

- there are many failures: in this case, in fact, the proposed solution could be able to restore the QoS level on a single provider whereas the other solutions must re-compute the path each time an error occurs, wasting time.

The proposed technique, instead, could be ineffective when the fault happens at the end (i.e. on one of the last services available) of given work-flow. In fact, the smaller the number of services available for the "compensation" process, the smaller the probability to satisfy the above mentioned recovery conditions (see eq. 11).

5.4 Related work

The issues related to the composition of services under QoS constraints are
gaining attention and have been addressed in a number of recent works.

[71] presents a framework for autonomous and coordinated SLA negotiation
and re-negotiation through an agent-oriented solution, in order to face the
adaptive service composition provision. In this framework autonomous ne-
gotiation can be carried out in a coordinated fashion to determine QoS con-
straints for individual services that collectively fulfill end-to-end QoS using
the Stochastic Workflow Reduction (SWR) algorithm for the decomposition
of total QoS in a collection of atomic QoS. Another important aspect of this
framework is the possibility to re-negotiate the SLAs in case of breaks through
the Iterated Contract Net Protocol (ICNP). However this approach lacks sup-
port for the management of penalties. [72] presents a framework for the re-
source provisioning problem in service composition subject to multiple QoS
constraints and an SLA violation penalty for differentiated customer service,
defining, therefore, a QoS index for service selection and quantifying these
QoS metrics and their SLA violation penalties. The central question of this
paper is similar to those described in my work: to solve a QoS-constrained
resource provisioning problem, i.e. minimize the overall cost of the selected
service resources required while satisfying SLA requirements. The solutions
adopted, instead, are different: while we use a decentralized approach to com-
pose the service, [72] refers to a central QoS manager that handles each aspect
of the composition. Differently from our approach, the failure of this com-
ponent and/or of the associated service broker could be catastrophic for the
whole system. Moreover the management of the penalties simply focused on
the selection of new service sites for all the services that are broken, without
any consideration about the possibility to handle the QoS parameters to try to

contain the cost of the SLA violation (that represents the core of the solution here presented).

[73] presents Agflow, a middleware platform able to address the issue of selecting Web Services for the purpose of their composition in a way that maximizes user satisfaction expressed as utility functions over QoS attributes. The paper discusses two alternative QoS driven service selection approaches for composite service selection: one based on a local optimization and the other on global planning. Only the second alternative can be compared with our work. Our approach is based on the optimization of a parameter, the user satisfaction, by means of a particular version of Dijkstra's algorithm, while Zeng et al. uses an extension of Multiple Criteria Decision Making (MCDM) technique [74]. The differences between the two approaches depends on many factor, the main is the reference architecture: we refers to a fully decentralized Grid-P2P context with could be extended to a generic SOA, while Zeng et al. refers explicitly to a Web Services context, where the MCDM technique is a good solution. [75] focuses on the composite service re-planning during execution. This paper presents a trigger that replannes the composite service when intercepts a QoS deviation from the initial estimation within a services that is executing.To realize this issue, Canfora et al propose an approach that relies on a proxy-based architecture to permit the binding between abstract and concrete services.

Determining the best discretization of a composite service is an optimization problem, aiming to: i) maximizes a fitness function based on a set of QoS attributes, and ii) meet the constraints specified for some of those attributes. The authors analyze the above problem like a NP-hard, investigating the effect of different strategies: Integer Programming [73], Genetic Algorithms [76], the Constraint Programming [77]. These three approaches provide good results if there are not specific time constraints. For services requiring a quick

response (as those considered in our work) it is necessary to pursue a trade off between a possible improvement gained with re-planning and the replanning overhead on response-time. For these reasons our approach uses a replanning strategies that doesn't assure the optimization of the whole composite service, but is very fast and present a crucial features (in according to our opinion): it chooses the replacement that best contains the costs of the composition. [78] is focused on the guaranteed path selection inside a Service Overlay Network (SON) wherein there are a set of component services that act as building blocks for more advanced services, which can be created by combining these component services in series or parallel configurations. The composition of the service is performed through the selection of a path between a source and a destination that passes through a specific set of overlay nodes that satisfy one or more QoS requirements specified by the user. This approach uses the K-Closest Pruning (KCP) algorithm to solve the problem of multi-constraint service path selection for SON in polynomial time.

[79] presents a QoS-aware service composition heuristic algorithm able to select a set of interconnected domains with specific service classes. Moreover when one or more domains fail to deliver their promised QoS performance during the service session, it performs a self-adaption by seeking an alternative communication path that satisfies the original QoS requirements. The target of the adaptation is to cause as little service disturbance as possible, through a simple network adaption algorithm that tries to find a minimal cost alternative path that utilizes as much of the old path as possible.

[79] uses a selection approach based on global allocation of tasks of services using integer programming. It presents AgFlow a middleware platform that enables the quality driven composition of web services. In this approach the adaption of the system related to a service breach is performed through an adaptive execution engine which reacts to changes occurring during the exe-

cution of a composite service by re-planning the execution in order to ensure that the total QoS is respected.

QUEST, [70], a framework that provides both initial service composition, which can compose a qualified service path under multiple QoS constraints, and a dynamic service composition, which can dynamically recompose the service path to quickly recover from service outages and QoS violations. For the initial composition it uses a modified Dijkstra algorithm by comprehensively considering multiple constraints. For the dynamic service composition it uses two algorithms: DQSC (dynamic Qos-assured Service Composition)-complete and DQSC-partial. In the first case, it tries to rebuild the whole path, in the second it tries to reuse as much as possible the old path. However, the search heuristic only attempts to find a feasible path, rather than an optimal one.

In eFlow, [80] the definition of a service node contains a search recipes represented in a query language. When a service node is invoked, a search recipe is executed in order to select a specific service. No QoS model is explicitly supported.

Finally, an approach for monitoring service compositions is presented in [32]. Here, monitors are defined as additional services of a process and used to validate contracts of the individual services, expressed through assertions in the process specification. [81] proposes a interesting framework for checking requirement compliance during the execution of a service. In this approach the expected behavior and assumptions are expressed in event calculus.

5.5   Conclusion

In this chapter I described an innovative technique for guaranteeing the maintenance of QoS level in composed services belonging to different providers within a P2PGrid environment.

The proposed solution is innovative because it introduces the concept of "QoS compensation". If the ith service fails, violating the agreements and degrading the overall QoS, the provider has to refund the user: the user can use this refund as additional resources for asking better performance to one or more of the following services in order to compensate the QoS violation.

Using the concept of "compensation" and negotiating effectively the values of refunds stored in the "exception" section of SLAs, it is possible to implement a fast recovery strategy for avoiding QoS level degradation due to services failures. This technique works well when there are errors in "long-terms" composed services, in presence of many failures or when those failures happen in one of the first basic services. This happens because this technique avoids the re-computation of "execution path", ever necessary in the other strategies proposed in literature, reducing the references to new services and, as a consequence, to new processes for services discovery and negotiation.

Chapter 6

Applicating ARM among a multicore Cloud environment

Cloud computing is, today, the most significant and diffused example of Services Oriented Architecture.

This is mainly due to the widespread adoption of virtualization technologies that, ensuring the isolation among the VMs, allow Cloud to manage a huge amount of software and hardware resources in a simple and extremely flexible way.

However, if not properly managed, the virtualization can adversely affect the underlying hardware performance: for instance, the current solution are not able to exploit, by default, all the functionalities offered by the multicore systems.

Moreover, these solutions do not guarantee the performance isolation among virtual machines and this makes impossible the creation of any form of QoS-guaranteed services management.

This requires the full control over the hardware resources: every QoS-aware

Figure 6.1: Reference Scenario

request done by an user, containing parameters as execution time, waiting time, security and robustness, is translated into specific physical and logical constraints about the number and the types of resources managed at low layer. In this chapter the author proposes a Resources Management System (RMS) based on ARM, a behavioral pattern designed to provide, through VM allocation and advanced resource reservation, QoS-guaranteed services in a multicore-based cloud system.

6.1   Reference Scenario

The reference Cloud environments consists of a set of multicore machines, connected via a shared communication channel (shared memory, dedicated bus as PCI-Express or socket over Giga-Ethernet or Infiniband). In my work I refer to this system in terms of Cloud Elements (CEs), logical entities involving both the underlying hardware resources and the software components

for managing and exploiting these resources. ARM is the component of each CE representing the "access point" for the considered cloud: it handles the scheduling issues and manages the *allocation* and the *reservation* of the resources for the execution of the Virtual Machines (VMs). In this chapter, as commonly happens in cloud, services requests of users are mapped on VMs, logical containers providing "well defined" environments for their execution. ARM does not act directly on the resources but it exploits the management functionalities offered by a Virtual Machine Monitor that, in my work, is the Hypervisor of Xen [82] (see the subsection 6.1.3). For running, each VM needs of a specific set of resources: one or more cores, a fraction of the system memory and a set of I/O devices. There are many ways for mapping VM on the resources (e.g. one VM - one core, one VM - multicore, many VMs - one core): ARM makes this choice basing on the performance and the QoS requirements required by the users. As described in fig. 6.1, the *end user* forwards to CE a request for a service execution with a given *QoS profile*(see subsection 6.1.1). The CE processes the request in order to understand if the specific service can be provided; in positive case, the VM containing the "execution environment", created by the user itself or chosen among the ones provided by the CE, is allocated, by the ARM and Hypervisor, on the most suitable resources ready for its execution.

6.1.1   QoS profile and SLA

Each time it makes a request for the execution of a service, the user can specify a list of attributes on the software and hardware facilities needed for the service execution. In the following, *QoS parameters* defines the attributes/items of the list and *QoS profile* indicates the requirements and constraints on the service execution expressed by that list. In general, it is possible

to distinguish two different types of QoS parameters:

1. functional, those based on the *quantitative* characteristics of the hardware infrastructure, i.e. network latency, CPU performance, storage capacity;

2. non functional, those based on the *qualitative* characteristics of the service required: i.e. reliability, availability, robustness, responsiveness, costs.

If the *end user* defines in his requests a QoS profile, the related service will be referred as *QoS-guaranteed service*; conversely, the service will be referred as *best effort service*. For each *QoS-guaranteed service*, there is a negotiation phase between the *end user* and the service provider: if it concludes with success, the two parties stipulate a Software Level Agreement (SLA) [51, 52, 83], a formal document that contains all the information about services exploitation, as *protocol adopted*, *messages exchanged*, *third-party entities involved* and *penalties in case of agreement breach*, including, overall, the list of the values of both functional and non functional parameters negotiated between the two parties.

6.1.2   The influence of resources virtualization on QoS management

The provisioning of QoS-aware services requires in general a management system divided in two different layer.

The higher one, the applicative layer, is the coordinator and provides the user interface: it accepts the requests coming from the users, translates the
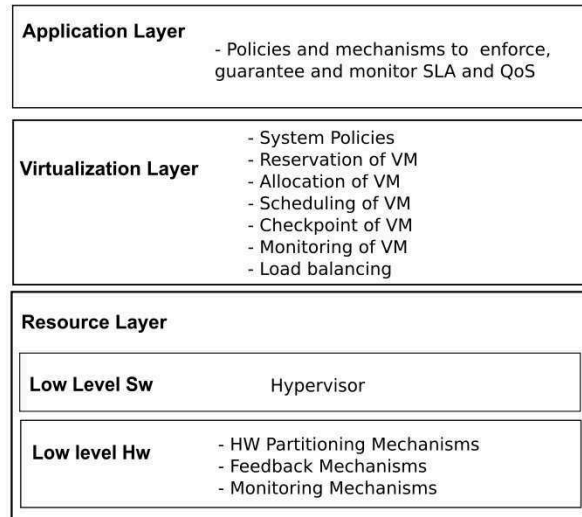
Figure 6.2: QoS management layers

required QoS parameters in constraints over the underlying resources, creates the SLA if the negotiation process is competed successfully and, finally, supervises the execution of service in order to avoid breaches of SLA.

The lower layer, the resources one, plays a fundamental but passive role. It consists of a software component, acting as an "access point" for resources functionalities, that enforces the commands coming from the applicative layer.

However, in the considered scenario,the adoption of virtualization technologies modifies the mapping QoS requests-resources in QoS requests-virtual machine-resources, introducing another layer, the "virtualization" one (see fig. 6.2). In this scenario, the user requests are firstly translated, in the application layer, in parameters used for the configuration of one or more VMs able to execute the required service and, subsequently, in the virtualization

layer, these VMs are mapped on the underlying resources according to the allocation and reservation policies adopted in the Cloud system. The mapping VMs-resources is done using the functionalities offered by the resource layer that, in this case, is divided in turn in two sub-layer: the SW-resource layer, constituted by the Hypervisor, and the HW-resource layer, that takes into account all the hardware resources managed by the Hypervisor.

The interactions between *virtualization* and *resources* layers, and in particular the mapping VMs-resources, have a crucial importance in the management of *QoS-guaranteed services*. In fact, the number and the type of resources dedicated to a VM influences the performance of the services running inside the execution context represented by the VM itself. This mapping, usually, is done automatically basing on the *QoS profile* required by the user according to predefined allocation policies. However, especially for expert users, the system foresees the possibility to mapping manually the VM over a given set of resources.

As shown in fig. 6.1, the user requests contains, in addition to the requirements and constraints expressed in the *QoS profile*, a section for the configuration of the VMs over the physical hardware resources. This section, in particular, contains indication about OS, number of core involved, size and type of memory required, number of I/O channels, etc, useful to understand which resources have to be reserved for obtaining guarantee about VM performance. Moreover, the proposed system gives the ability to associate a start time and duration to a reservation, making possible the both just-in-time allocation and advanced resources reservation.

The use of virtualization, finally, allows the user to estimate the behavior of its service with an high degree of accuracy.

The service, in fact, runs within a specific VM on a well-known hardware architecture. This means that it is possible executing specific profiling of that

service by means of appropriate benchmarks.

The results of these benchmarks, done over different hardware configuration, constitutes a *VMs performance catalog* that the providers offers to their users for foreseeing the performance of their own services.

runs within a specific VM on a well-known hardware architecture. This means that it is possible executing specific profiling of the user applications, by means of appropriate benchmarks. Since the applications will run on the same VM and on the same hardware configuration used in the tests, the user has got the guarantee that the execution of the own application will follow the behavior profiled.

6.1.3   The resources reservation and the Xen Hypervisor

The concept of resources reservation is strictly related to the concept of resource usage: a resource is reserved by one or more tasks if and only if they are the only entities allowed to run on the resource. This means that a resource can be reserved if and only if it is possible to have a complete control on the accesses on the resource. This ability, in general, is a prerogative of the resources manager. In the reference scenario here considered, the computational resources belonging to the cloud are managed by Xen, an "open source" virtual-machine monitor able to give guarantees on the precise management of the resources. Xen offers a set of virtualization technologies for several hardware architectures (IA-32, x86-64, Itanium, ARM). Born in the Computer Laboratory of the University of Cambridge Computer, Xen is a free software, licensed under the GNU General Public License (GPLv2), developed and maintained by the Xen Community. The core component of Xen is the Hypervisor (i.e. the Virtual Machine Monitor -VMM-): it provides

both the abstraction layer for the hardware resources of the host machine and some fundamental mechanisms, like CPU scheduling and main memory partitioning, that the above level, the *Domain 0*, can exploit for overseeing and managing the VMs that will run over that machine. The *Domain 0* (*dom0*) is a special VM having a modified version of the Linux kernel: it starts automatically when the Hypervisor boots and manages and monitors the behavior of all the other VM, called *Domain U*(*domU*) running on the Hypervisor. This is possible since it is the unique VM having special management privileges and direct access to the physical hardware. *dom0* of Xen represents, indeed, the key component for controlling the exploitation of the hardware resources and, as a consequence, the key component on which building the reservation support. Configuring the *dom0*, it is possible to state precisely and in a deterministic way the quantity of hardware resources to be assigned to each one of the running VMs and, thus, to guarantee the required performance. This configuration phase is mandatory: in fact, while Xen provides faults and security isolation for VMs runnings on the same resource with respect to faults and security, it does not provide performance isolation. As shown in QClouds [84], for example, the performance of a set of VMs consolidated on a multicore hardware with a shared Last Level Cache (LLC) can not be foreseen "a priori" because of the VMs interfere with each other. To overcome these issues, Xen provides a set of commands that allow to specify, for each VM, the quantity of hardware that can be accesses. The reservation manager here proposed uses the commands to drive *dom0* to enforce the required hardware exploitation assuming the complete control of the underlying system. It is possible, for example, to assign a given amount of main memory to a specific VM (using mem-set of mem-max commands), to dedicate one of more core for the execution of a single VM (using vcpu-pin command) or to regulate the scheduling of several VMs on the same CPU (with sched-credit command). The control

is not only for computational resources: using the pci command, it is possible to dedicate a given pci device, for example the network controller, to a single VM.

6.2   Advance Reservation

6.2.1   Resources Management Systems

A computing system can offer an environment for secure, reliable, robust and QoS-aware services execution only if it can rely on an effective Resources Management System (RMS).

The RMS represents the *access point* for resources exploitation: its main task concerns the managing the life-cycle of services execution requests. In general, the functionalities provided by the RMS can be summarize in three fundamental steps. The first step concerns the collection of the requests coming from the users. It foresees that the RMS gets the requirements of the user on the service execution and *translates* them into constraints for the underlying hardware resources. For example, if an user asks for executing a services without any *waiting time*, the RMS have to find a free resource. Or, if an user asks for executing a services with an high level of robustness, the RMS has to execute the service, in parallel, on two or three different resources.

In the end of this initial *interpretation* and *translation* phase, the RMS has a certain number of physical and logical constraints about the type and the number of resources needed for executing the service.

These constraints represent the input for the second step. It consists in checking if, among the ones available, there are enough resources able to execute the services according to both temporal and physical constraints.

If the RMS does not find an adequate amount of resources, the request is rejected. When, instead, the RMS identifies these resources, reserves them for given time intervals. Some RMSs (the more sophisticated one) could provide a *negotiation* step for offering to users the ability to execute their services with a lower QoS with, for example, a lower cost. The reservation of the resources and the subsequent execution of the services over them represents the third and last step.

Each RMS characterizes each of these steps with own specific functionalities, basing on the applicative scenario. In the next section, I will introduce a behavioral design pattern for a RMS supplying QoS-guaranteed services in cloud environments called ARM.

6.2.2   Model of Advance Reservation implemented with ARM

ARM provides the software tools for supporting both just-in-time VMs allocation and advanced resources reservation in a Cloud environment.

Fig. 6.3 shows the Finite State Machine for the management of an advance reservation (AR) performed using ARM.

The starting state is represented by the *Waiting for request* state: the ARM is idle. When a request arrives, the ARM goes in the *Evaluating request* state. Here, the ARM extracts the user requirements, translates them in physical (hardware requirements) and logical (software requirements) constraints for the underlying resources and checks if these constraints can be satisfied. If ARM find the needed resources, i.e. the request can be satisfied, it goes in the *Reservation established* state. Here, ARM sets up the Hypervisor for configuring the hardware resources.
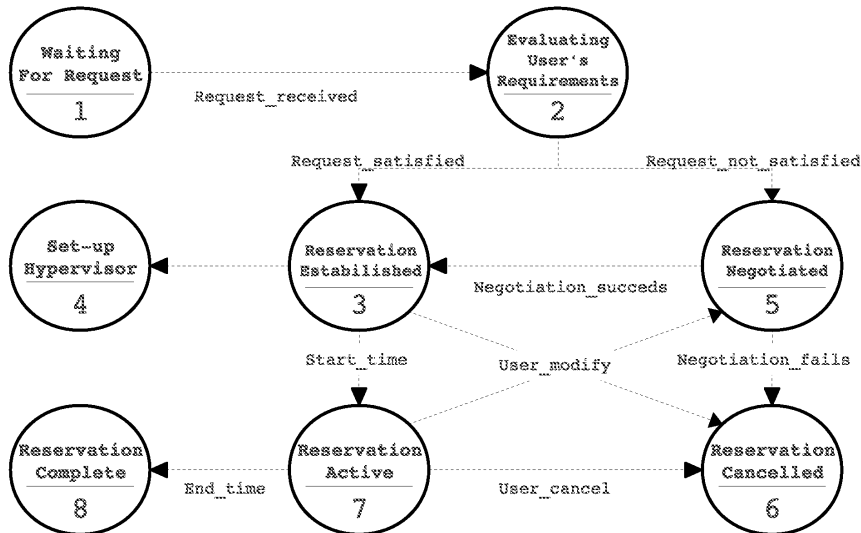
Figure 6.3: Extended state diagram for advance reservation in ARM

Instead, if there are not enough resources to guarantee the required QoS level, ARM goes in the *Negotiation* state. In this state the ARM using a given negotiation protocol (e.g the one used for stating the SLAs in chapter 4), try to reach an agreement with user for the provisioning of a service with a lower level of QoS (or an higher one, if it needs to user). If the user accepts this QoS degradation (e.g. obtaining the service at lower cost), the resources involved are reserved and ARM goes in the *Reservation established* state. If, instead, the user does not accept the QoS degradation, he withdraws his request: ARM goes in the *Reservation canceled* state. In the *Reservation established* state, the reservation has been accepted but *start time* is not yet reached. From here, the reservation can become:

- active, if the start time is reached: the ARM goes in the *Reservation*

active state and the VM is allocated on the resource;

- canceled, if the user deletes its reservation request: the ARM goes in the *Reservation canceled* state;

- altered, if the user user asks for modifying some parameters: the ARM goes in the *Negotiation* state;

From the *Reservation active* state, the ARM can go in three different states. If the user wants to delete or modify his request, ARM goes, respectively, in *Reservation canceled* and *Negotiation* states. If, instead, the reservation ends normally, the ARM goes in the *Reservation completes* state: its work ends and the resource returns to be free. It is important to note that ARM allows users to modify the parameters of reservation both when the reservation is established and also when he has some service requests already in progress. In this way, the user that has overestimated the reservation time can release the resources once its services is terminated and, on the contrary, the user that has underestimated the duration of reservation can extend it to complete its service.

ARM is able to provide all this functionalities through the coordination of the set of software components that I have organized in a pattern. The next section illustrates the objective, the structure and the functioning of this pattern.

1: RSVrequest()
2: askforRSV()
3: responseRSV()
4: RSVreply()
5: checkAvail() :: insert/delete/
modify.Table()
6: SlotAvailability()
7: Allocation/SchedulationVMs()
8: NotifyVMStatus()
9: DrivesHypervisor()
10: MonitorVMStatus()
11: UpdateInformation()
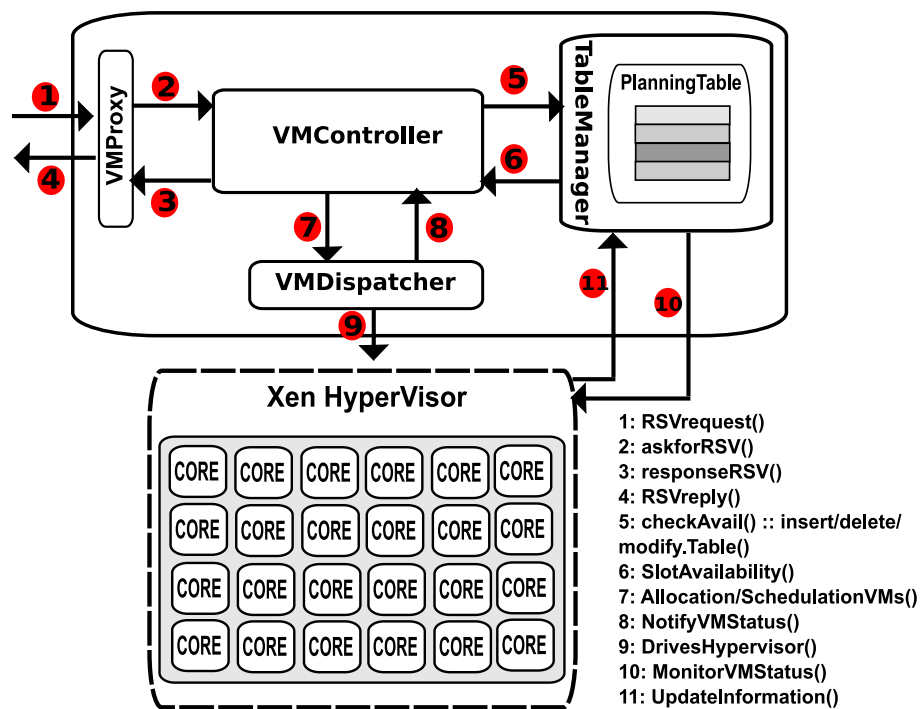
Figure 6.4: Interaction among the components constituting the pattern

6.3 The Pattern: objectives, components and functioning.

6.3.1   Objective

The pattern here proposed aims to provide a reliable, flexible and easy-to-adapt mechanism to manage both *just-in-time allocation* and *advanced reservation* of resources in a Cloud environment.

Although the pattern can be used for managing several kind of resources, for the aims of this chapter, I will focus on the reservation of computational resources. In particular, since the considered scenario is based on multicores machines, the reservation will be referred to each single core of the available hardware. In the following, resource and core will be used as synonymous.

The proposed pattern:

- provides a general solution adaptable to different Cloud middlewares;

- enforces QoS resources management allowing to meet requirements, constraints and, in general, all the parameters characterizing the reservation requests;

- exploits the functionalities of underlying *resources* layer (in this case, XEN) to perform *VM submission*, *VM execution*, *resources state updating*, and *VM checkpoint and migration* on behalf of the reservation manager;

6.3.2   Structure of pattern

The architectural structure of the ARM pattern is shown in the fig. 6.4 and consists of five main elements: ARMServer, Table, TableManager, ARMCon-

troller, and (JAM).

- ARMServer is a *event-driven proxy* responsible for accepting the requests coming from the users (or software agents working on behalf of them), parsing these requests and, finally, forwarding them to the ARMController.

- ARMController is the *reservations coordinator*. It performs all the activities related to the allocation and the reservation: it looks for the availability of the cores, maps the VMs on the cores, manages the reservation life-cycle, monitors the status of active reservations and finally coordinates the execution of best effort requests allocating the VM on the resources left free by the manager (i.e not reserved). ARMController also includes a local fault recovery engine that, in case of failures, tries to restore the correct execution of the VM using the checkpointing and migration abilities provided by the JAM .

- JAM is the pattern component that manages the allocation of VMs on the cores. It also takes care of the issues related with checkpointing and migration of the VMs among the cores. It represents the lower level of ARM and the interface with the Xen Hypervisor, by means of which performs its tasks.

- The Table represents the information repository of the reservation system that ARM can consult for obtaining information on the actual and future state of the cores.

   In Table, the information are organized in a doubly hashed table, as shown in fig. 6.5. The first hashtable, on the left in the fig. 6.5, indexes all the cores by means of their identification (ID, primary key) and of
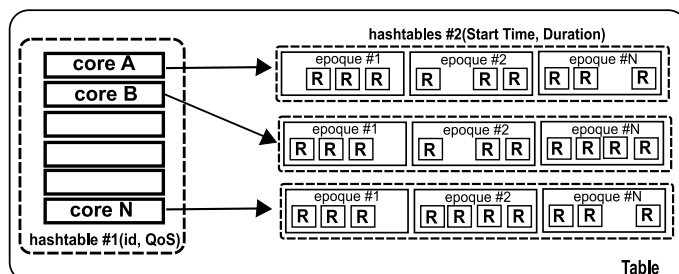
Figure 6.5: Hash tables details

the value of QoS level they can provide (QoS, secondary key). Each
one of the elements of the first table contains a second hashtable, on
the right in the fig. 6.5, that holds the information about the state of
the specific core. In particular, this second hash table contains, for
each core, the indication about the time intervals in which the core is
reserved ([R]) with its start time (Start time, primary key) and duration
(Duration, secondary key). This information, organized in epochs
having fixed duration, covers the whole *time horizon* taken into account
for the reservation. Maintaining references only about the reserved
slots time instead of all the slots that could be reserved, it is possible
to have an infinite *time horizon*: this means that the pattern can accept
reservation requests for any future slot time.

- The TableManagerrepresents the unique access point to the information
  maintained by the Table. In order to add, delete or modify a parameter
  related to the reservations, the ARM has to use the functionalities of-
  fered by this component. The use of a centralized manager guarantees
  the correctness and the consistency of the information in Table.
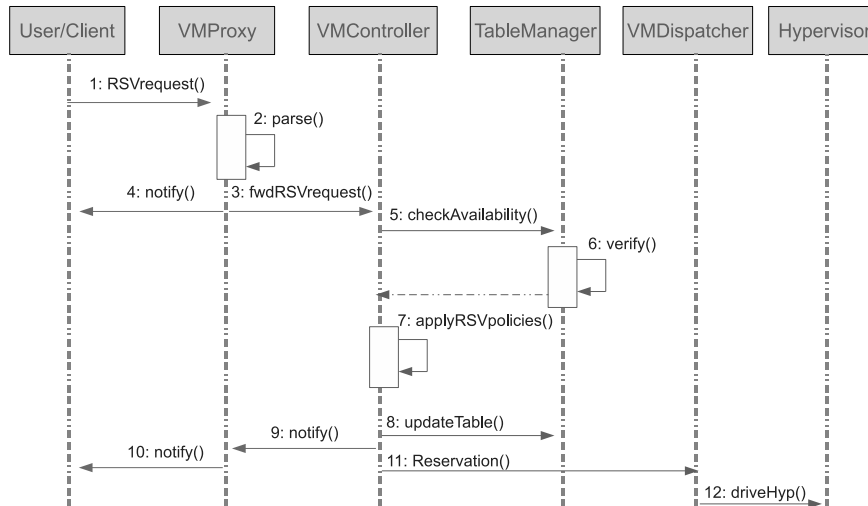
Figure 6.6: Resource reservation: the sequence diagram

6.3.3 Functioning

The functioning of the Advanced Reservation Management Pattern is summarized in sequence diagram shown in Fig. 6.6.

ARMServer waits for reservation requests coming from user. When a request is received (act. 1), ARMServer parses it (act. 2) to verify whether reservation parameters are correctly defined (e.g. if they belong to a valid range of values) and if the user can obtain the specified type and the given amount of needed resources. If the checks succeed, ARMServer forwards the request to ARM-Controller (act. 3) and sends a reply back to the client (act. 4) thus notifying the beginning of reservation process.

ARMController checks (act. 5), through the TableManager whether the cores

pool under its control can satisfy the user requirements. The TableManager (act.6), through the Table identifies the cores able to provide the required QoS level and verifies whether there is a sufficient quantity of free slots for completing the user service. Then, TableManager returns this information to the ARMController which take a decision, basing on the reservation policies (act. 7). If the reservation can be performed, ARMController asks (act. 8) TableManager to reserve those slots time for executing the user service. Instead, if it is not possible, it notify to ARMServer that the reservation can not be done. At the same time, ARMController forwards (act. 9), through ARMServer (act. 10) the process result to the user that has submitted the request.

When the start time of an advance reservation is reached, ARMController through JAM (act 11) and Xen Hypervisor, instantiates the VM on the core and start the service execution.

When the reservation time expires, JAM alerts ARMController.

If the duration of the reservation has been correctly estimated, i.e if the all the services related with the reservation have been completed before the expiration of reservation time, the ARMController frees the cores, releasing the VMs associated with it, and updates the associated entries in Table, through TableManagerİf, instead, the time for reservation expires while the VM is even running on the reserved resources, the ARMController has to decide whether:

- to extend reservation time (if it is possible, i.e. if the availability of the specific resource exceeds the request);

- to checkpoint the VM and, eventually, migrate it on another free core;

- to kill it, stopping the execution of the VM on the reserved resources.

ARMController takes this decision basing on the agreement stated with the user, on the resources management policies and on the state of available re-

sources.

6.3.4   Some notes about JAM and VMM

The JAM uses Xen as Virtual Machine Monitor (VMM) for managing the
resources of the Cloud.

In particular, JAM exploits the Xen's Hypervisor i) for having a uniform
vision of the resources available in the Cloud and ii) for executing a VM
maintaining the information about its state.

As said before, JAM is driven by the ARMController the core component of
ARM pattern.

JAM is not able to distinguish between an allocation request and an advance
reservation request: it considers the allocation a *border line reservation* with
start time set up on the current time. The different management of both ty-
pologies of requests is demanded to the ARMController.

When the ARMController forwards the allocation requests to the JAM , the
requests are enqueued on the specific resource.

The JAM is responsible for getting the request from the resource queue and
forwarding it to Xen Hypervisor, using the commands introduced in section
6.1.3.

JAM uses Xen also for gathering information about resources state. Finally,
as explained in the next section, JAM and Xen cooperate for making possible
checkpointing and migration of VMs.

6.3.5   Checkpointing and migration

Checkpointing and migration of VMs are two fundamental functionalities
in Cloud environments: they improve the resources exploitation allowing to

modify the scheduling and allocation policies at runtime, adapting the system to accepts new unforeseen requests or optimizing its performance via load balancing. These abilities, that are available by default in all the VMMs (Xen Hypervisor included), are fundamental to avoid the "reservation holes" effect that wastes the performance of those system adopting advanced reservation of resources.

Checkpointing is the operation that allows ARMController to freeze a snapshot of the current state of a VM so as it can be restarted from that state at a later time in another core. This enables the ARMController to modify scheduling decisions basing on its needs. If the ARMController decides to no longer allocate a VM on a core, it can checkpoint the VM and preempt it without loosing the work already made by the service that runs on it. The VM can resume its execution starting from a checkpoint when the ARMController allocates it on a new core. Among the several benefits introduced by checkpointing mechanisms, one of the most important concerns the robustness of the system: using periodic checkpoints, it is possible restarting a failed VM from the most recent checkpoint instead of restarting it from the beginning. The ARMController implements checkpoint and migration mechanism exploiting the functionalities offered by the Xen Hypervisor.

The implementation of checkpoint mechanism uses the following algorithm:

1. save and store the VM identifier

2. stop the VM

3. dump the VM state into a file representing its disk image;

4. flush all the I/O flows

The implementation of VM migration uses the following algorithm:

1. check if the disk image of the VM is accessible on the remote node,

2. allocate resources on the remote node,

3. start a lazy copy of the VM memory to the remote node (the VM is still running),

4. when the memory is copied, stop the VM and complete the transfer of the VM to the remote node

The components involved in these issues, as mentioned in section 6.3.3, are the JAM and the Hypervisor.

6.4   Related Work

There are many different solutions for the management of resources in Cloud Systems, coming both from business companies and scientific community. ARM differs from these solutions because the allocation and resources reservation policies it provides are strictly oriented to the QoS provision.
Infrastructure as a Service (IaaS) systems, such as Elastic Compute Cloud (EC2) [85] of Amazon, do not provide such guarantees on the QoS level of the provided services: they rely on the availability of a huge amount of computing resources and often delegate, to the users, the issues related with the QoS management. For instance, the SLAs supported by Amazon EC2 consider only the *annual uptime percentage*, an metric related to the resources availability, without providing any guarantees that this computing capacity will be supplied.
Platform as a service (PaaS) systems (such as Google AppEngine [86] and MS Azure [87]) offer a set of APIs for allowing users to develop services

able to exploit the features offered by the these kind of systems. Although they supply many functionalities for automatic scaling up/down and fault-tolerance management, these systems do not give the ability to define, at the lowest layer, the mapping between the VM and the hardware resources as, instead, the proposed solution does.

The problem of mapping VM(or services)-resources is taken into account both in [88] and in [89].

In [88], the authors describe an utility-based approach for adaptive workload execution in a cloud environment. In particular, authors design a dynamic model that, basing on a predefined utility function and on a costs model, is able to evaluate the performance of any workload on a given execution environment and to identify the mapping workload/resources that maximize the utility function. Similarly Grounds et Al. in introduce a cost-minimizing scheduling algorithm (CMSA) for scheduling requests of multi-level workflows of various types and degrees of complexity [89]. Even this algorithm assumes that a cost function is provided, and operates by making scheduling decisions in order to minimize the estimated value of cumulative cost. Differently from the approach proposed in this chapter, [88] and [89] focus only on the problem of workload/resource mapping, without considering the issues related with advance reservation of the resources.

In [90], instead, the authors tackle the issue related with the resources management under an economical point of view. They analyze a scenario on which different providers (Amazon, Microsoft, etc) co-existing in a cloud-based services marketplace and propose a sensitivity analysis of Nash Equilibrium of the variation of the prices and QoS level on each provider in respect to the variation of prices and QoS level in the other providers. In that competitive scenario, where all providers offer to users the same services with different prices and QoS guarantees, the high manageability provided by the

ability to have the complete control over the underlying resources, obtained using ARM, could represent the winning strategy to adapt, quickly and easily, the management policies to the market trends.

6.5 Conclusion

In this chapter I proposed a solution for designing a Resources Management System (RMS) for cloud-oriented multicore environments.

In particular, I proposed ARM, a behavioral design pattern that defines the architecture and the interactions among a set of specific software components for providing, through VM allocation and advanced resource reservation, QoS-guaranteed services in a multicore-based cloud system. This chapter introduced the issues related with the QoS management in Clouds, explainned how the use of virtualization technologies influences the management policy and given a detailed description of the proposed pattern in term of objectives, architecture and functioning. Particular attention, is given to the interaction between ARM and the Xen Hypervisor, that acts as unique access point to the underlying hardware resources. The complete control of the resources obtained interacting with the Hypervisor, together with the ability to planning resources reservations in the future, represents the key aspects of the ARMfunctioning.

Chapter 7

Dime Technology

Cloud computing is essentially the ability to acquire or deliver a resource on demand, configured however the users chooses, and paid for according to consumption. From a supplier's perspective, including both internal IT groups and service providers, it means being able to deliver and manage resource pools and applications in a multi-tenancy environment, to deliver the user an on-demand, pay-per-use service. A cloud service can be infrastructure for hosting applications or data storage, a development platform, or even an application that you can get on-demand, either off-site at a provider, such as SunGard or Salesforce, or built onsite within IT.

The aim of these technology consists in the decoupling of the hardware infrastructure and the virtual computing infrastructure. By this way they implement a virtual grid services network, where they offer a grid of virtual computing, network and storage resources, overcoming the limitations of present Wide area distributed system (as Grid) due to their rigid schema.

Today, the virtualization represents the key techonology of this trend, [1] at least, for two reasons: i) it guarantees the isolation of tenants (to guarantee fault, configuration, accounting, performance and security (FCAPS) management) and ii) improves scalability. Ont the other hand, the ongoing development of virtualization poses some at least three questions that must be addressed.

The first concerns the real scalability of the actual solutions. This is strictly related to the scalability of the software used for assuring the virtualization, i.e. the Virtual Machine (VM). The number of guest VMs that can be effectively consolidated on a hypervisor without disruptive performance impact within a physical server with the actual technology at best is 10. This means that a ratio of 1 to 10 is possible to accommodate multiple tenants. Any further scaling can only be obtained with an architectural transformation to provide isolation and FCAPS management at much finer granularity at an object level providing a service in an individual (physical or virtual) server or at a transaction (end-to-end customer interaction including execution and persistence) spanning across multiple virtual servers.

The same scalability problem exists in many-core servers because the current generation operating systems, such as Linux and Windows, can support only few tens of CPUs in a single instance and are inadequate to manages servers that contain hundreds of CPUs, each with multiple cores. It is possible to definte this gap as *operating system gap* (the difference between the number of cores available in an enclosure and the number of cores visible to a single image instance of an OS). The solutions currently proposed for solving the scalability issue in these systems, i.e. the use of SSI [91] or the introduction of multiple instances of the OS in a single enclosure with high-speed con-

---

[1]as said form Charles King, Principal Analyst at Pund-IT: "Without virtualization there is no cloud that's what enabled the emergence of this new, sustainable industry."
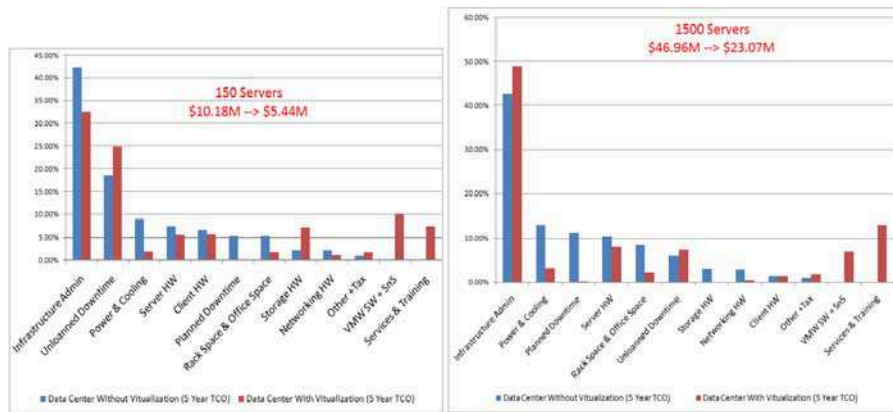
Figure 7.1: Current data center estimates of Total Cost of Ownership

nectivity (e.g. [92]), are inefficient because they increase the management complexity.

The second concerns the communication among the VMs. The current virtualization technologies provide only TCP/IP coomunication patter even in presence of other possibilities (PCI Express, shared memory, etc.), reducing the performances of the clouds. A last question concerns the future trend of cloud marketplace. Fig. 7.1 shows current data center estimates of Total Cost of Ownership over a 5 year period with infrastructure that is virtualized compared to the infrastructure that is not virtualized.

Few key points to note are:

1. The TCO over a five year period is 1.87 better with virtualization with 150 servers and is 2.04 times better with 1500 servers.

2. First year investments are $1.24M and $5.36M with 150 and 1500 server data centers respectively while the first year savings are $0.57M and $2.59M respectively.

3. The infrastructure management costs over 5 years are about 40% to 45% of the total cost with or without virtualization.

4. The software and services costs are approximately 20% of the total cost. In a data center with 1500 servers, the total cost of infrastructure management and virtualization software and services costs add up to 70% of the TCO.

The main conclusion one can draw is that there is still room for improvement to reduce the infrastructure management and virtualization software and services costs as a percentage of TCO. Even though the TCO reduces substantially by about a factor of about 2, it is interesting to note that Infrastructure management cost remains around 40% of the total cost. Virtualization software and services add about 20% of the total cost. Real progress cannot be made till the management cost is substantially reduced with real-time dynamic and automated computational workflow management.

Facing these three questions probably means rethinking (or at least reexamine) the current server centric computing model that has evolved over decades when bandwidth abundance and multi-core architectures were not the norm, because the scenarios are evolved.

In this regard it is possibile to identify two main directions on which to coordinate efforts:

1. Develop a simplification of the architecture to facilitate the decoupling of hardware infrastructure management from virtual services management so that the service workflow implementations are truly distributed and managed based solely on their latency tolerance and not on geographical or physical infrastructure boundaries [93–95]. This will allow eliminating layers of management infrastructure which burdens the

data centers today that has evolved from a server-centric architecture with limited bandwidth networks.

2. While both Grids and Clouds offer pooling and sharing of distributed resources, management of resources becomes a critical factor to resolve contention based on business priorities of the consumers of the shared resources. In addition, in order to meet the wildly fluctuating demand for resources in a mass market, dynamic configuration, Fault, Accounting, Performance and security management must accompany the resource sharing infrastructure. Therefore, Investigate ways to implement telecom grade textittrust in the cloud so that services can share resources distributed over different clouds based on their latency tolerance and business priorities.

In a recent work, [96], authors proposes to fill these issues using a new computing model called the Distributed Intelligent Managed Element (DIME) network computing model.

The model incorporates FCAPS management using a signaling network overlay and allows the dynamic control of the computing element with respect to its configuration, Fault management, Performance management, security management and accounting management. Such parallel implementation of signaling and computing networks is feasible today with multi-core, multi-CPU hardware assisted virtualization.

In my PhD work, my contributions to this techonology mainly focused on: i) the definition of the basic idea of the DIME computing model, ii) the realization of the first demo of DIME Network Architecture (DNA), iii) the implementation of a Linux, Apache, MySQL, and PHP (LAMP) based services architecture that exploit the functionalities of DNA to demonstrate end-to-end transaction management with auto-scaling, self-repair, dynamic

performance management and distributed transaction security assurance.

7.1   Dime Computing Model

7.1.1   FCAPS, signaling channel, execution channel

A DIME is an autonomous computing entity endowed with self FCAPS management capabilities along with computing capabilities. The concept of FCAPS management is derived from the Telecommunications Management Network (TMN). It defines the FCAPS framework:

1. Fault management, by detecting and correlating faults in network devices, isolating faults and initiating recovery actions

2. Configuration management, by providing change tracking, configuration, installation and distribution of software to all network devices

3. Accounting management capability through comprehensive network usage reports generated by collecting and parsing accounting data

4. Performance management by providing real-time access for the monitoring of network performance (QoS) and resource allocation data

5. Security management by providing granular access control for network resources

Project management is a specific example where Fault, configuration, accounting, performance and security are individually managed to provide an

optimal network configuration with a coordinated workflow. Functional organizations, and hierarchical and matrix organizational structures are all designed to improve the efficiency and agility of an organization to accomplish the goals using both FCAPS management and signaling. Connection management is achieved through effective communications framework. Over time, human networks have evolved various communications schemes and signaling forms the fundamental framework to configure and reconfigure networks to provide the agility. There are four basic abstractions that comprise signaling: Alerting, Addressing, Supervision and Mediation. Thus organizational hierarchies, project management, process implementation through workflows are all accomplished through the network object model with FCAPS abstractions and signaling.

In the same way, each DIME encapsulates the management of the FCAPS issues at its internal, and uses two channels for coordinating the execution of workflow wwith the other DIMEs:

- *Signaling channel* Through it, one DIME signals each other to implement DIME network management which configures, secures, monitors, repairs and optimizes the workflow based on workload characteristics, latency constraints and business priorities specified as service management profiles [2] (we call this profile the Service Regulator SR)

- *Execution channel* Through it, one DIME communicates with each other to implement a distributed business workflow as a DAG (in the

---

[2]Profile based resource management is inspired by Plain Old Telephone Service (POTS). Everytime, a user want to call anotherone, the network knows both the the service characteristics and requirements of the initiating party and the service characteristics and requirements of the dialed party, so as to allocate the appropriate resources and initiate the connection. Moreover, the network monitors the connections, bills for the duration and assures the connection reliability, availability, performance and security. The DIME computing model performs the same function for the services that utilize computing, network and storage resources.

von Neumann Stored Program Control Computing model). The tasks implemented by the DIME node is specified as a DAG (called the Service Package SP).

DIME integrates the computation and its management at the computing element level. By this way, it exploits distributed resources, parallelism and networking to provide real-time telecom grade availability, reliability, performance, and security management of distributed workflow services execution. In addition, it uses signalling to monitor and control the business workflow implementation using the parallelism of multicore systems. Signaling allows prioritization of the network objectives and allocates resources in the form of distributed DIME to accomplish the objectives and provides management control to mitigate risk. Elaborate workflows are implemented using the signaling mechanism to specialize and distribute tasks to various DIMEs. The DIMEs are used to collect information, analyze it and control themselves as a group to accomplish the required goals. A key factor of the success of this model is the parallelism of service delivery and the service management networks using a signaling OS to manage the FCAPS element of the individuals and the group connection by leveraging the individual FCAPS management capabilities. According to this vision, the computation spans over a network of DIME that adapts its composition and its features so as to optimize specific performance parameters. This is in total contrast to current approaches where the Operating Systems and management systems, which were designed in the days of CPU and memory resource constraints, and labor dependent administration paradigms are used to implement both computation and management workflows. Figure Referencespicture2 shows a comparison between the von Neumann SPC computing model and the DIME computing

model. There are three key features in this model that differentiate it from all other models:

1. The self-management features of each SPC node with FCAPS management using parallel threads allows autonomy in controlling local resources and provide services based on local policies. Each node keeps its state information and history1 of its transactions. The DIME node provides managed computing services, using the MICE to other DIMEs based on local and global policies.

2. The network aware signaling abstractions allow a group of DIMEs to be programmed to manage themselves with sub-network/network level FCAPS management based on group policies and execute a service workflow as a managed directed acyclic graph (DAG).

3. Run-time profile based FCAPS management (at the group level and at the node level) allows a composition scheme by redirecting the MICE I/O to provide recombination and reconfiguration of service workflows dynamically at run-time.

7.1.2   DIME Components

Each DIME is an autonomous computing entity endowed with self FCAPS management capabilities along with computing capabilities. It is implemented as group of multi-process, multi-thread components, as shown in Figure 7.3. Each one of the components constituting the DIME performs a
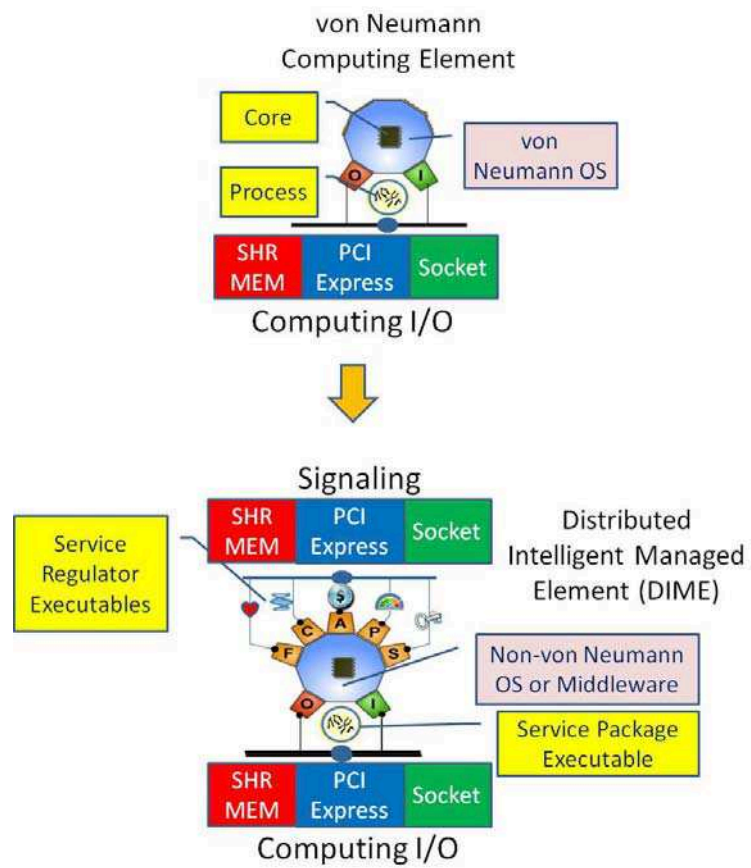
Figure 7.2: The Resiliency, Efficiency and Scaling of Information Technology Infrastructure
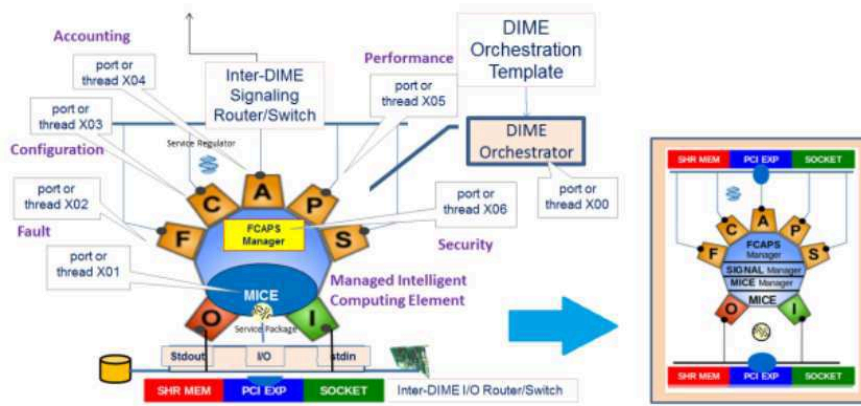
Figure 7.3: The Anatomy of a DIME with Service Regulator and Service Package Executables

specific function and, based on its configuration given to it by an external configuration file or by commands; each DIME can assume several roles (see next section) in the management of workflows.

The main components are the following:

- The DIME local Manager (DLM) is the core of a DIME. It sets up the other DIME components; it monitors their status and manages their execution based on local policies. Upon a request to instantiate a DIME, the DLM, based on the role assumed by the DIME, sets up and starts three independent threads to provide the Signaling Manager (SM), the Managed Intelligent Computing Element (MICE) Manager (MM) and the FCAPS Manager (FM) functions.

- The SM is in charge of the *signaling channel*. It sends or receives commands related to the management and setting up of DIME to guarantee a scalable, secure, robust and reliable workflow execution. It also pro-

vides inter-DIME switching and routing functions

- The MICE is in charge of the execution of a task on the assigned resources. It is istantiated from the MM. All the actions related to the task execution, which are performed by the MICE including memory, network, and storage I/O, are parameterized and can be configured and managed dynamically by the SM through the FM via the MM. This enables both the ability to set up the execution environment on the basis of the user requirements and, overall, the ability to reconfigure this environment, at run- time, in order to adapt it to new, foreseen or unforeseen, conditions (e.g. faults, performance and security conditions).

- The MM is a passive component which starts the MICE, monitors its execution and, on completion, notifies the event to the SM.

- The FM is the key component of the architecture. It processes the events received from the SM or from the MM and configures the MICE appropriately to load and execute specific tasks (by loading program and data modules available from specified locations). The main task of FM is the provisioning of FCAPS management for each task loaded and executed in the local DIME. This means that it handles autonomously all the issues regarding the management of faults, resources utilization, performance monitoring and security. For this reason it provides a *separation of concerns* which decouples the management layer from the execution layer

The DIME network computing model thus allows the description and management of the service to be separated from the execution of the service (leaving it in MICE). The signaling control network allows parallel management of the service workflow. In step 1, the service regulator instantiates the

DIME and provisions the MICE based on service specification. In step 2, The MICE is loaded, executed, and managed by the service regulation policies. At any time, the MICE can be controlled through its FCAPS management mechanism by the service regulator. The MICE provides the logical type that performs everything that is feasible within that logical type (a Turing machine) and the DIME FCAPS management provides a higher logical type (management of the Turing machine) which describes and controls what is feasible in the MICE. These features provide the powerful genetic transactions namely, replication, repair, recombination and reconfiguration that have proven to be essential for the resiliency of cellular organisms [97].

Figure 7.4 describes the separation of service regulation and service execution workflows.

7.2 Dime Network

7.2.1 Architecture

The execution of a workflow (formalized by a DAG) needs the creation of an ad hoc network of DIMEs. In order to have two separated and parallel layers for managing and executing the workflow,each DIME network is implemented using two classes of DIMEs: Signaling DIMEs and Worker DIMEs.

The first class (Signaling DIME) foresees two type, called respectively Supervisor and Mediator, for the management layer at the network level.

The Supervisor sets up and controls the functioning of the sub network of DIMEs where the workflow is executed. It coordinates and orchestrates the DIMEs through the use of the Mediators. Mediator is a specialized DIME for providing predefined roles such as fault or configuration or accounting or performance or security management.Each one of this roles is performed by
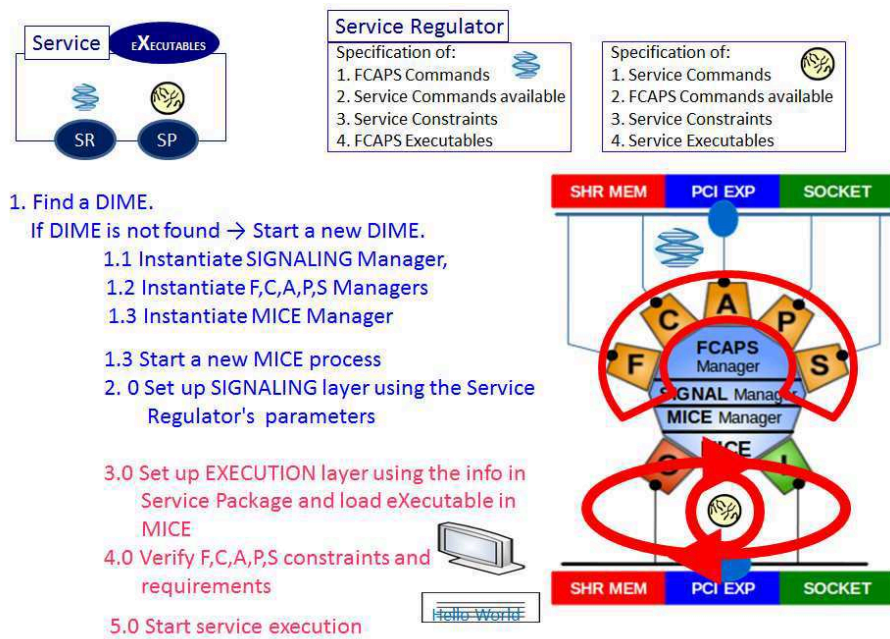
Figure 7.4: The Anatomy of a DIME The Anatomy of a DIME and the separation of service regulation and service execution workflows

a specific manager:

1. Fault Manager guarantees the availability and reliability in the sub network by coordinating the *Fault* components of the FM of all the DIMEs involved in the workflow provisioning. The Fault Manager DIME detects and manages the faults in order to assure the correct completion of the workflow.

2. Configuration Manager performs network-level configuration management and provides directory services. These services include registration, indexing, discovery, address management and communication management with other DIME networks.

3. Account Manager tracks the utilization of the network wide resources by communicating with the individual DIMEs.

4. Performance Manager coordinates performance management at the network level and coordinates the performance using the information received through the signaling channel from each node.

5. Security Manager assures network level security by coordinating with the individual DIME component security.

The second class, the Worker DIMEs, constitutes the execution layer of the network. They perform domain specific tasks described in the DAG. A worker DIME, in practice, provides a highly configurable execution environment built on the basis of the requirements/constraints expressed by the developers and conveyed by the Service Regulator. The deployment of DIMEs in the network, the number of signaling DIMEs involved in the management level, the number of available worker DIMEs and the division of the roles are not prefixed, but they are established on the basis of the number and the type of

tasks constituting the workflow and, overall, on the basis of the management profiles related to each task.

Note that the profiles play a key role in the DIME model; each profile, in fact, contains the indication about the control and the configuration of both the signaling layer and execution environment for setting up the DIME that will handle the related task. Each node of the DAG related to a workflow contains both the task executables [3] and the profile DAG as a tuple ¡ task (SP), profile (SR)¿: in this way, it is possible not only to specify *what* a DIME has to do or execute, but also *how* DIME has to do everything and under what constraints (i.e. its management). These constraints allow the control of FCAPS management both at the node level and the sub-network level.

7.2.2 Functioning

Eeach workflow assigned to the Supervisor DIME consists of a set of tasks arranged in a DAG.

The supervisor DIME, upon receiving the workflow, identifies the number of tasks and their associated profiles. It instantiates other DIMEs based on the information provided, by selecting the resources among the ones available, both the management and the execution layers. This selection is carried out using two simple criteria: the number of DIMEs constituting the set of workers is equals to the number of tasks in the workflow, while the DIMEs composing the signaling layer are known a priori but their number can vary based on the requirements and on the constraints of the workflow. The Supervisor, in fact, can decide to use six different DIMEs (one acting as local Supervisor and five as the different managers - Fault, Configuration, Account, Performance and Security managers -) or a single DIME playing all the foreseen roles or, in

---

[3]which itself could be another DAG

addition, to use some dedicated DIMEs only for specific roles. The information about the profiles becomes instrumental to define 1) the signaling sub-network, 2) the type of relationship between the mediator DIMEs composing the signaling sub-network and the FM of each worker DIME and, finally, 3) the configuration of all the MICEs of each worker DIME to build the most suitable environment for the execution of the workflow. Following these criteria, the Supervisor creates a sub-network able to implements specific workflows that are in turn FCAPS managed both at management layer (through the mediators) and at execution layer (through the FM of each worker DIME).

So, the Supervisor can decide either to create and use new DIMEs for the signaling layer or to share the managers with other sub-networks. This choice depends on the requirements of the workflow and on the current workload managed by Superivsor. For example, if the workflow management requires a great number of messages for its coordination, the Supervisor can use a DIME for each manager. Instead, if the coordination of workflow requires only few signaling messages, the Global Supervisor can decide to use only a DIME for all the managers.

The decisions taken by Supervisor, however, can be modified at run-time, simply activating or halting some components in DIMEs and/or modifying the used addressing schema. This ability, which represents one of the major advantages provided by the signaling based network-aware DIME organization, is fundamental for tackling the workload fluctuation typical of systems providing services on-demand. Using it, the Supervisor can tune optimally the behavior and, as a consequence, the performance of the workflow in execution.

The features become very important especially, in those system where there can be several workflows working in parallel; based on workflow profile, on QoS parameters, on resources required and on costs paid by the users, the

Supervisor can be used to organize the DIME network (and sub-networks) for optimizing given parameters (for example, for maximizing the resources utilization, for maximizing the system gain or for improve robustness of entire system).

Similar considerations can be used for choosing the position/deployment of worker DIMEs. The physical location of the DIMEs in the sub-network, in fact, can influence strongly the performance of the workflow.

If, for example, the workflow needs low-latency communications, the Supervisor is used to deploy the worker DIMEs as close as possible in term of communications delay. The options among that Supervisor can choice are three: deploying the DIMEs on the same PC (using shared memory-based or PIPE-based communication), deploying DIMEs on the same rack/server/blade (using dedicated-bus communication, gigabit-Ethernet or Infiniband) or to distribute DIMEs over the internet (using socket-based communications). These three options present obviously different performances. The initial choice of the Supervisor will depends on the available resources and on the performance requirements. This flexibility about the choice of deployment of Worker DIMEs is an improvement over existing cloud technologies anchored to the socket communication among the VMs. Moreover,the DIMEs belonging to the computing workflow layer are not also configured in a static way but can be re-arranged at run-time to adapt the workflow performance to the workload fluctuation or to react to unforeseen events. The MICE of worker DIMEs, in fact, can be dynamically re-configured for redirecting I/O at run-time. This ability, obtained using a proxy-like communication management system in both layers (signaling and execution), allows the Supervisor to modify the sub-network based on running workflow and on its workload needs.

Once the best solution for organizing the sub-network of DIMEs is defined statically to execute the workflow, the Supervisor uses the Mediator to starts

a Local Supervisor and all the managers involved in the FCAPS management of the workflow execution. Each manager, based on its offered functionalities and on workflow requirements, executes a list of controls: for example, they check the "heartbeat" of the system for fault prevention [4], authorization and authentication for security, or the timers for accounting and performance evaluation. If all the checks are successful, the Local Supervisor sets up the MICEs (through the Cmanager) of the entire worker DIME network for starting the execution of the workflow. If no problems occur, the workflow completes its execution correctly and the output is sent to the user; the sub-network is de-allocated and the DIMEs are made free.

7.2.3   Fault Management

There are several type of fault: some depend on the technological support used to build a DIME network (network delays, resources leaks, security vulnerabilities, etc), while others depend on the functioning of the DIME network (one or more DIME crash during the workflow execution). In this paragraph I focus only on the analysis about the last type of fault.
Consider the case of the occurence of a DIME fails during the execution of workflow (as shown in 7.5) .

Once that the fault is detected, the system starts the recovery of the errors and restores the correct functioning of the entire sub-network, but the behaviour of the network change on the basis of the role that the DIME played within the network. We can have four cases.

First case: a worker DIME faults while is executing a task. In this case, the fault manager, which monitors the "heartbeat" signal of each worker DIME,

---

[4]the absence of one heartbeat indicates a fault of the DIME related to that heartbeat
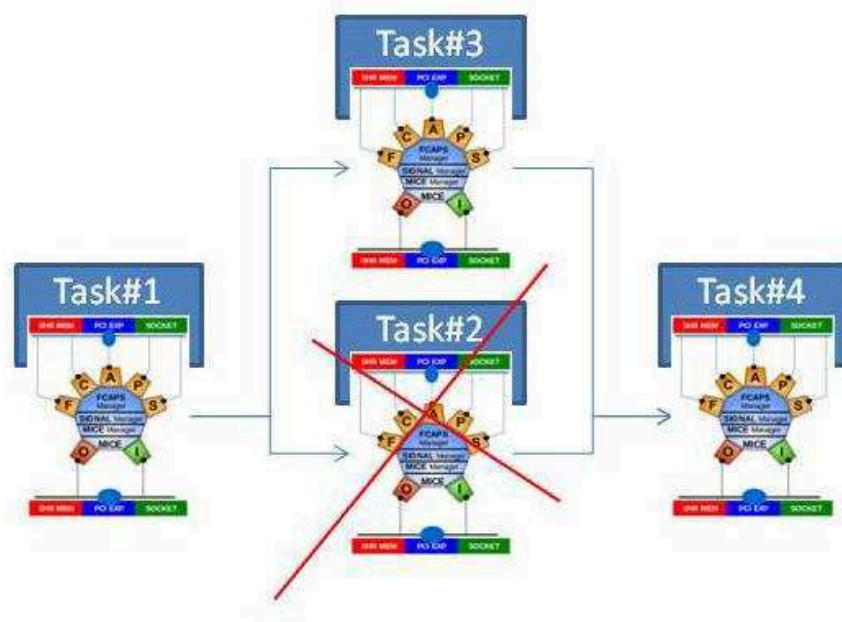
Figure 7.5: Fault during execution of a workflow

identifies immediately that a DIME has become unreachable, deduces that an error has occurred and asks the Local Supervisor for a new worker DIME. The Local Supervisor, then, obtains the reference of a new DIME from the Configuration Manager; the Local Supervisor then configures the new DIME with the same SR and SC of the task not completed and sends the reference of this new DIME to the worker DIME executing the task directly related to the task not completed. The CManager of this worker DIME, then, modifies at run-time the I/O ports of the MICE in order to redirect the output of its task toward the new worker DIME. All these actions, done autonomously and in a transparent way in relation with the other DIMEs of the sub-network, can be collected in both standard and ad hoc procedures (as described in figure 7.6).

In the latter case, the user can specify to system a given behavior for reacting to specific faults.

Second case: one signaling DIME faults during the execution of a workflow. A mechanism similar to the "heartbeat" is implemented between the Supervisor and the signaling DIMEs of each sub-network. If one of the managers becomes unavailable, the Global Supervisor loads in a new DIME (or in a DIME shared with another sub-network) the "templateâ of the old manager. Modifying the references on the Configuration DIME, the Supervisor recovers the correct functioning of the signaling network without alerting any of the other components in the signaling layer or the other worker DIMEs.

Third case: a Local Supervisor DIME's failure. For each subnetwork, a mechanism similar to the "heartbeat" is implemented between the Supervisor of the main network (called Global Supervisor) and the Supervisor DIMEs of each sub-network (called Local Supervisor). If one of the Supervisors becomes unavailable, the Global Supervisor loads in a new DIME the *template* of the old Supervisor. Modifying the references on the Configuration DIME,
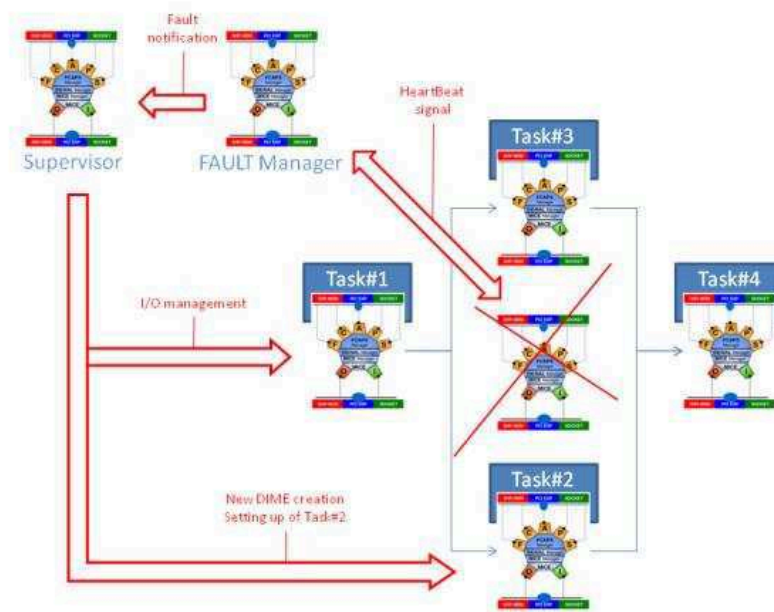
Figure 7.6: Fault management during execution of a workflow

the Supervisor recovers the correct functioning of the signaling network without alerting any of the other components in the signaling layer or the other worker DIMEs.

Fourth case: Fault of the Global Supervisor. Each local Supervisor is linked with the Global Supervisor through the heartbeat. These heartbeat are bidirectional: the Global Supervisor responds to one heartbeat of a Local Supervisor, sending in turn one heartbeat to the Local Supervisor. By this way, the absence of a heartbat can be intercepted from the Local Supervisors' network. If there are sufficient resources available, the recovery procedure is similar to that I described for the Local Supervisor (case 3). Otherwise, local Supervisors starts a election phase to estabilish a new Global Supervisor. The new global Supervisor continue to be also the local Supervisor for its subnetwork till new resources are available to create a new local Supervisor.

7.3 Case study: LAMP,an application of the Dime computin model

In this section, I describe the use of a DIME network (each DIME encapsulating a Linux process with FCAPS management) to implement LAMP based web services architecture to demonstrate end-to-end transaction management with auto-scaling, self-repair, dynamic performance management and distributed transaction security assurance.

7.3.1   LAMP Web Services Using DNA

The DIME network architecture takes its cues from parallels in cellular biology where regulatory genes control the actions of other genes which allow them the ability to turn on or turn of specific behaviors. As affirmed by Philip Stanier and Gudrun Moore, [98] *In essence, genes work in hierarchies, with regulatory genes controlling the expression of âdownstream genesâ and with the elements of âcross-talkâ between the regulatory genes themselves.* The same parallel, furthermore, exists between the task profile and the concept of gene expression. Gene expression is the process by which information from a gene is used in the synthesis of a functional gene product. Fig. 7.7 shows a DIME network of Linux processes implementing a web services workflow using a MySQL database and Apache and PHP services. Key innovation in DNA enables the application or service running in the MICE under the control of the FCAPS manager to provide Fault and performance information through the intra-DIME signaling which is utilized by the end-to-end DIME network management infrastructure using the Inter-DIME signaling. The policies are implemented by the DIME service network managers (the Supervisor, Fault Manager, Performance Manager, Accounting Manager and the Security Manager designed to execute policy implementation workflows.) A simple workflow is as follows:

1. Local performance manager in DIME 2 monitors Apache using an ad hoc library that notifies the response time of the web sites deployed in Apache.

2. When the performance exceeds a threshold, the signaling channel is used to notify the service network manager which in turn requests the Supervisor to instantiate an additional Apache in a new DIME.
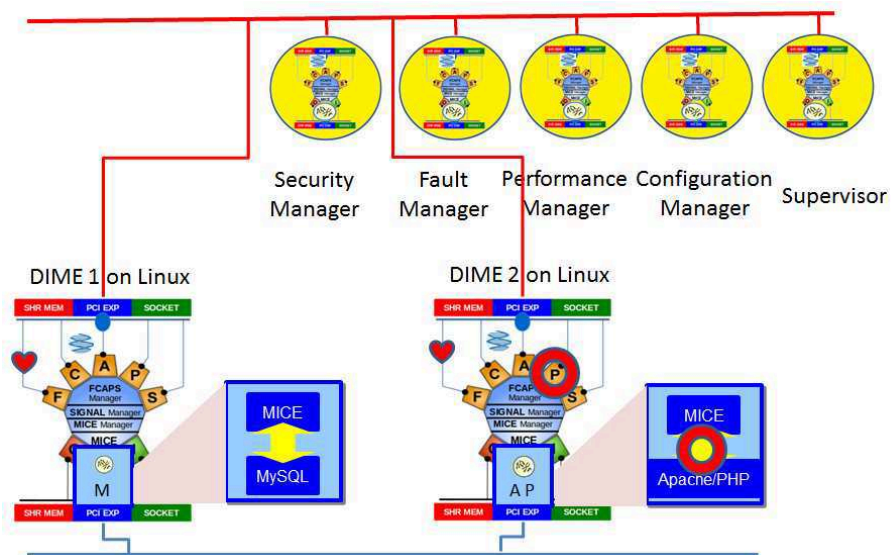
Figure 7.7: DIME network implementing web services using LAMP services with FCAPS management at both the node level and at the network level

3. The supervisor based on business priorities, workload management policies and latency constraints coordinates with configuration and security managers to instantiate a new instance of Apache with appropriate configuration.

4. The configuration manager instantiates the new service and adjusts the work-loads based on policies including DNS splitting.

A similar management workflow regulates the fault management using the heartbeats provided by each DIME at a regular interval to the network fault manager. The database response time similarly is also monitored by periodically querying the MySQL database and appropriate policies are enforced to meet business priorities. The scaling up or down by the configuration manager is implemented based on the workloads, latency constraints and overall business priorities. The policies are implemented at various levels; at the node level and at the sub-network or network level. In addition to domain specific service management workflow, each DIME implements its own local FCAPS management independent of what MICE processes are doing. This allows programming DNA level DIME instantiation, and its life-cycle management to assure 100% infrastructure availability, performance and security service levels. While a simple end-to-end transaction security check is performed with a login and password scheme that allows service network management, a more elaborate authentication, authorization and accounting scheme is discussed in another paper by Tusa et al [99]. We believe that the DIME network architecture represents a major departure from the current cloud approaches [100, 101]. Using the non-von Neumann approach, it radically improves the resource exploitation using parallelism even within the cloud environments, hiding the complexity of the management of FCAPS issues both from the developers and users of cloud services.

7.4 Conclusions and future direction

While the last section discusses the use of DNA injection into Linux OS, we see no technical obstacle to do the same in other operating systems. The key requirement is the multithreading capability to implement parallel management workflow to control the Turing machine implemented as a process in the conventional OS. It is also proven that the DNA can be injected at the core with a native OS written from scratch that scales and provides the resiliency. It is also proven that we can leverage current service oriented architecture, development environments and workflow implementations using a network of Turing machines by migrating them to a managed network of Turing machines using DNA.

It is interesting to note that the services management decoupling from the infrastructure hardware management using DNA does not require any new standards or approaches except exploiting parallelism to separate the management workflow and computing workflows at the core or at the process level using self-management, signaling and network management abstractions.

In designing this new class of distributed systems, it behooves us to go back and seriously study von Neumanns' views on the subject [97]. Talking of cellular organisms and how they operate across errors, he points out that textitthe system is sufficiently flexible and well organized that as soon as an error shows up in any part of it, the system automatically senses whether this error matters or not. If it does not matter, the system continues to operate without paying any attention to it. If the error seems to the system to be important, the system blocks that region out, by-passes it, and proceeds along other channels. The system then analyzes the region separately at leisure and corrects what goes on there, and if correction is impossible the system blocks the region off and by-passes it forever. The duration of operability of the

automaton is determined by the time it takes until so many incurable errors
have occurred, so many alterations and by-passes have been made, that finally
the operability is really impaired. This is a completely different philosophy
which proclaims that the end of the world is at hand as soon as the first error
occurred. In order to benefit from the approach adopted by the cellular organ-
isms, current services management approaches must implement two features
at the core computing element (a von Neumann computing node). First, they
must implement self-management based on local history and local policy re-
quirements. Second it must provide a parallel signaling channel for a network
of self-managed computing elements to communicate and collaborate to im-
plement global policies.

While current cloud and grid management systems implement services man-
agement by monitoring various application or service characteristics with the
use of various management systems, the applications or services that use local
operating systems in each node still have their resource and service manage-
ment serialized using the von-Neumann SPC computing model. The DNA
addresses this by implementing the separation at the computing node by ex-
ploiting parallelism.

Discussing the work of Francois Jacob and Jacques Monod on genetic
switches and gene signaling, Mitchell Waldrop [102] points out that tex-
titDNA residing in a cell's nucleus was not just a blueprint for the cell - a
catalog of how to make this protein or that protein. DNA was actually the
foreman in charge of construction. In effect, DNA was a kind of molecular-
scale computer that directed how the cell was to build itself and repair itself
and interact with the outside world. We believe that the DIME network archi-
tecture enabling the execution of a workflow as a managed directed acyclic
graph provides at least a mechanism for a blueprint for enterprise business
process description, replication, execution, and control using a lengthy recur-

sive sequence of nested programs which unfold in the von Neumann computing world using a non-von Neumann computing model.

Future directions of this research are self-evident.

First, the non-von Neumann middleware can be exploited to improve resiliency, efficiency and scaling of current grid and cloud services by decoupling services management from the infrastructure hardware management. This approach allows implementing reliable and resilient services using unreliable hardware just as the cellular organisms do. Few immediate applications present themselves:

1. Dynamic many-core cluster communication management across multiple Linux images to choose the type of communication based on available resources and service requirements.

2. Implement WAMP services architecture using DIME network architecture

3. Application aware resource allocation (dial-up and dial-down) at run time.

4. Resilient services oriented architecture (RSOA) implementation through the migration of the services microcontainer [103] into a DIME.

5. High performance computing (HPC) resource scheduling and management.

Secondly, the hardware infrastructure itself can be redesigned (exploiting the many-core architecture) to become signaling aware and respond to application requests at run time. Future storage and networking hardware thus can be simplified with hardware assisted DIME architecture to eliminate current

layers of management software and special purpose ASIC implementations. They can be designed to dial-up and dial-down raw resources (number of cores, memory, bandwidth, throughput, storage capacity etc.) based on application requests at run time.

Finally, DNA can be implemented by chip vendors in hardware to provide self-management and signaling awareness exploiting parallelism at the core. This allows uniformity in hardware device drivers with self-management and signaling awareness. On the theoretical side, it is worth examining the intriguing remarks of von Neumann about Godel's theorem and its implications on the descriptions of complexity [104]. In his Hixon Symposium talk, von Neumann remarks textitIt is a theorem of Godel that the next logical step, the description of an object, is one class type higher than the object and is therefore asymptotically infinitely longer to describe. He goes on to say textitIt is one order of magnitude harder to tell what an object can do than to produce the object. The DNA attempts to describe and assure what an object does; in this case the object happens to be a von Neumann computing node. In the light of the new resiliency of DNA (e.g. the DIME can be instantiated and managed to provide 100% availability and recoverability), it is worthwhile to revisit the classic distributed computing issues such as the dining and drinking philosopher problems, the CAP theorem etc. In conclusion, we observe that the evolution of living organisms has taught us that the difference between survival and extinction is the information processing ability of the organism to:

1. Discover and encapsulate the sequences of stable patterns that have lower entropy, which allow harmony with the environment providing the necessary resources for its survival,

2. Replicate the sequences so that the information (in the form of best

practices) can propagate from the survived to the successor,

3. Execute with precision the sequences to reproduce itself,

4. Monitor itself and its surroundings in real-time, and

5. Utilize the genetic transactions of repair, recombination and rearrange-ment to sustain existing patterns that are useful.

That the computing models of living organisms utilize sophisticated methods of information processing, was recognized by von Neumann who proposed both the SPC computing model and the self-replicating cellular automata. Later Chris Langton created computer programs that demonstrated self-organization and discovery of patterns using evolutionary rules which led to the field of artificial life and theories of complexity.

Chapter 8

Conclusions

Cloud computing and grid computing are, today, the most significant and diffused example of SOA for wide-area distributed systems. This thesis studied these two techonologies, focusing on the aspects of the resources' management and services' provision.

In keeping with this premise, the results of my investigations can be grouped in two macroareas: results about the SLA and Advance Reservation management in the Grid macroarea and results about the SLA and Advance Reservation management in the Cloud multicore macroarea.

8.1 Macroarea: Grid

The introduction of SOA features has deeply modified the way to exploit the functionalities of the computational grids.

These features have allowed to open grid systems toward new applications, embracing not only the field of scientific research, where the grid computing paradigm was born and developed, but also the business, financial and

educational ones. Offering access to their services through the web, grids have contributed to the creation of a competitive marketplace where the user can choose and use those *grid services* that better satisfy its requirements. In this scenario, the satisfaction of the user's QoS requirements becomes a fundamental issue because each user will select the provider based on the performance and cost of the offered service.

This thesis has accurately investigated this typology of wide area distributed systems, facing some important challenges and proposing, for each one, an effective solution.

The main issue examined in this dissertation regards the QoS management in grids. The present implementations of grid middlewares manage their underlying resources respecting the static policies imposed by the Virtual Organizations. These policies, defined by an *a priori* agreement between Virtual Organizations managers and grid resources owners, are fixed and can not be modified textitat run time. This type of resources management policies are not adequate in the considered scenario, where there are different typologies of users that need to set up and to tune up the service execution based on their heterogeneous and dynamic requirements.

In order to make grids able to satisfy these requirements, this thesis has proposed a strategy able to overcome the limitations due to Virtual Organization management policies. This strategy foresees that when a user, belonging to a Virtual Organization, requires a generic service, it is able to specify a set of QoS parameter in order to customize the service execution based on its expectations.

To support this feature, it has been necessary to make the grid manager able (i) to establish an agreement, i.e. the SLA, with the service user and (ii) to reserve the resources needed for a correct fruition of the considered service. The resource reservation is a fundamental support to guarantee that the ser-

vice has what it needs, in terms of hardware and software requirements, at the proper time; the ability to manage the SLA, instead, is fundamental to establish univocally all the parameters involved in the agreement, the rules and conditions for the proper use of the service.

All these features allow the service provider to adapt the service execution on the user requests (i.e the user can indirectly influence the resource management policy of the provider). This mean that a provider can execute the same service in different ways (e.g. using or not reserved resources, modifying the waiting time or guaranteeing the complete execution before a deadline) for different users.

On the other hand, a user can require, to different providers, the same service with different guarantees of execution: the user has now the ability to request not only a service but a customized version of it. This means that when a user is in looking for a service provider, it has to have the ability to search for a provider able not only to execute the service but also able to satisfy its QoS requirements.

In this thesis I describe ARM, a pattern that faces the issues related with the grid resources management in terms of allocation and advance reservation. This pattern represents the base on which it's possible to build a Service architecture able to guarantee services provision under SLA constraints. This new architecture uses a set of conmponents: SLAM (described in chapter 3), to manage the SLA in grid environments.

The design of this QoS-aware services discovery protocol has been another important aspect treated in this thesis. Herein, the adoption of an unstructured distributed services discovery protocol based on a selective flooding algorithm has been proposed. This strategy has been chosen in order to obtain a reliable, scalable and fault tolerant solution that would be impossible to have using centralised o DHT based approaches. The proposed algorithm has

relied on (i) a flexible and dynamic management of neighbour relationships, (ii) a control mechanism for the quantity of frames created by a query, (iii), a *selective routing* protocol based on node reputation and on (iv) a cache of partial information related to the knowledge of services offered by some other providers. The experimental results have demonstrated the validity of the proposed algorithm. Furthermore, the proposed algorithm is able to manage the SLA in order to simplify the agreement phases between client and provider of the service. The ability to search services under QoS constraints, that is an important innovation in the field of discovery algorithm for distributed systems, represents a fundamental value-added in a service marketplace scenario in that it allows not only to identify different providers for the desired service but also to discern among them the most suitable one to satisfy the user requirements.

This discovery protocol, furthermore, has been helpful in the management of QoS for composed services, the last issue taken into account in this thesis. The users, in fact, are usually interested in executing complex services, consisting of a composition of (simpler) services, rather than a single one. The management of QoS for composed services, however, is made difficult by the need of coordinating services offered by different providers and working independently by the others. Whenever one or more service providers do not supply the foreseen agreement with the client (because of a fault or an unpredicted overloading), the QoS for the whole composed service could be compromised. This condition forces the use of a QoS management strategy that not only has to continuously monitor the state of every single agreement (i.e of each provider, as it happens with the management of the QoS for a single service), but also it has to involve a compensation strategy that, trough agreement renegotiations, makes it possible for a run time adaptation in order to maintain the desired QoS level for the whole composed service.

The solutions proposed in this thesis have permitted the building of a reliable, scalable and flexible environment where each user can find, use and compose services based on their QoS requirements. The guarantee that these QoS requests can be successfully supported by the services providers is assured by the adoption of an effective resources management strategy that allows each user to accurately tune up the desired service. The scenario here depicted gives a great importance to the user.

The user, in fact, has not only the ability to compose easily services offered by other organizations as it happens in the *web 2.0* scenario: it is has the ability to customize them, influencing their execution.

8.2   Macro-area: Cloud

As I said in the introductino, Cloud computing is, today, the most significant and diffused example of Services Oriented Architecture. This is mainly due to the widespread adoption of virtualization technologies that, ensuring the isolation among the VMs, allow Cloud to manage a huge amount of software and hardware resources in a simple and extremely flexible way.

However, if not properly managed, the virtualization can adversely affect the underlying hardware performance: for instance, the current solution are not able to exploit, by default, all the functionalities offered by the multicore systems. Moreover, these solutions do not guarantee the performance isolation among virtual machines and this makes impossible the creation of any form of QoS-guaranteed services management. This requires the full control over the hardware resources: every QoS-aware request done by an user, containing parameters as execution time, waiting time, security and robustness, is translated into specific physical and logical constraints about the number and the

types of resources managed at low layer. In this thesis, I proposed two solutions to overcore this issues: RMS and DIME.

The Resources Management System (RMS) is an extension of ARM for cloud-oriented multicore environments. As described above ARM is a behavioral design pattern that defines the architecture and the interactions among a set of specific software components for providing, through VM allocation and advanced resource reservation, QoS-guaranteed services in a multicore-based cloud system. In the chapter 6, the thesis introduces the issues related with the QoS management in Clouds, explains how the use of virtualization technologies influences the management policy and gives a detailed description of the proposed pattern in term of objectives, architecture and functioning. Particular attention, is given to the interaction between ARM and the Xen Hypervisor, that acts as unique access point to the underlying hardware resources. The complete control of the resources obtained interacting with the Hypervisor, together with the ability to planning resources reservations in the future, represents the key aspects of the ARM functioning.

DIME is a new approach for the Cloud field. It introduces self-management of computing nodes and their management as a group using a parallel signaling network to execute managed computational workflows. The DIME computing model does not replace any of the computational models in use today. Instead, it complements these approaches by decoupling the management workflow from computing workflow and providing dynamic reconfiguration, Fault Management, Utilization Management, Performance Management and Security Management. I worked with my colleagues Morana and Mikkilineni to implement the DIME network computing model using Linux operating system and exploited the parallelism to demonstrate the dynamic Fault, configuration, performance and security management.

By this way, It's possible to demonstrare two significative enhancements from

current computing models:

1. The parallel implementation of FCAPS management and computing integrates workflow execution and workflow management provides a new degree of reliability, availability, performance and security management in allocating resources at the node level

2. The signaling scheme allows a new degree of agility though configuration management of each node at run time.

There are two ways in which the workflow can be dynamically changed. The first one is through the control of the MICE component to load and execute a specific program at run time. The second, instead, is through the reconfiguration of the I/O channels of the DIME to compose and decompose new services.

We believe that the DIME network computing model represents a generalization of other distributed computing models. The capability to set up the management layer, in fact, allows replicating the functioning of the other models such as, Clusters, P2P or Grid computing; one of the next steps of our research will be focused on the definition of well-defined profiles, used as templates for Signaling DIMEs, for setting up of DIME computing networks that are able to offer the same functionalities of those computing models with an added benefit of dynamic reconfiguration and real-time end-to-end integrated FCAPS management. The separation of service regulator executables from the service executables and their implementation in parallel with dynamic FCAPS management facilitated by signaling perhaps provides an object one class type higher, which von Neumann was talking about in his Hixon Symposium lecture [105]: textitThere is a good deal in formal logic which indicates that when an automaton is not very complicated the description of the

function of the automaton is simpler than the description of the automaton itself but that situation is reversed with respect to complicated automata. It is a theorem of Godel that the description of an object is one class type higher than the object and is therefore asymptotically infinitely longer to describe. It is for themathematicians to decide.

Supporting dynamically reconfigurable communication mechanism (shared memory, or PCIexpress or Socket communication) under Linux operating system, the Dime computing model can improve the communication efficiency between multiple Linux images based on context that will make manycore servers more useful with less complexity of cluster management. By encapsulating Linux based processes with FCAPS management and signaling, DIME provides live migration, fault management and performance management of workflows without the need for a Hypervisor-based server virtualization. Incorporating signaling in hardware provides a new level of agility to distributed services creation, delivery and assurance.

Bibliography

[1] C. Morris, "Sony playstation facing yet another security breach." `http://www.cnbc.com/id/43079509`.

[2] Thibodeau and Vijayan, "Amazon ec2 service outage reinforces cloud doubts." `http://www.computerworld.com/s/article/356212/`.

[3] A. Moscaritolo, "Rsa confirms lockheed hack linked to securid breach," `http://www.scmagazineus.com/article/204744/`.

[4] D. Patterson, "The trouble with multi-core." IEEE Spectrum, July 2010, p28.

[5] C. Kesselman, I. Foster, and S. Tuecke, "The Anatomy of the Grid - Enabling Scalable Virtual Organizations," *The International Journal of Supercomputer Applications and High Performance Computing*, May 2001.

[6] Foster and Iamnitchi, "On death, taxes, and the convergence of peer-to-peer and grid computing," 2003. In 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03), LNCS.

[7] D. Talia and P. Trunfio, "Adapting a pure decentralized peer-to-peer protocol for grid services invocation," *arallel Processing Letters*, vol. 15, no. 1-2, pp. 67–84, 2005.

[8] J. Tang and M. Zhang, *An Agent-based Peer-to-Peer Grid Computing Architecture - Convergence of Grid and Peer-to-Peer Computing*, vol. 54 of *Conferences in Research and Practice in Information Technology*. Hobart, Australia: ACS, 2006. Fourth Australasian Symposium on Grid Computing and e-Research (AusGrid 2006).

[9] F. Gagliardi, B. Jones, M. Reale, and S. Burke, "European DataGrid Project: Experiences of Deploying a Large Scale Testbed for E-science Applications," 2002.

[10] A. D. Stefano, G. Morana, and D. Zito, "An agent based component for qos management in a grid-p2p architecture," 2007. Conference on Self-Organization and Autonomous Systems in Computing and Communications (SOAS'2007), Leipzig, Germany.

[11] G. von Bochmann and A. Hafid, "Some principles for quality of service management," *Distributed Systems Engineering*, vol. 4, no. 1, pp. 16–27, 1997.

[12] M. Page and P. Blanche, "TERMS AND DEFINITIONS RELATED TO THE QUALITY OF TELECOMMUNICATION SERVICES recommendation E.800 QUALITY OF SERVICE AND DEPENDABILITY VOCABULARY," Nov. 17 1988.

[13] Enabling Grids for E-sciencE (EGEE) , "glite home page." http://glite.web.cern.ch/glite/.

[14] www.pi2s2.it.

[15] W. A. W. Jr., C. L. Mahood, and J. E. West, "Scheduling jobs on parallel systems using a relaxed backfill strategy," in *JSSPP* (D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, eds.), vol. 2537 of *Lecture Notes in Computer Science*, pp. 88–102, Springer, 2002.

[16] Altair corporate. , "Openpbs home page." `http://www.pbsgridworks.com/`.

[17] Platform Computing , "Platform lsf homepage." `http://www.platform.com/Products/platform-lsf`.

[18] Gamma, Helm, Johnson, and Vlissides, *Design Patterns Elements of Reusable Object-Oriented Software*. Massachusetts: Addison-Wesley, 2000.

[19] J. Frey, T. Tannenbaum, M. Livny, I. T. Foster, and S. Tuecke, "Condor-G: A computation management agent for multi-institutional grids," *Cluster Computing*, vol. 5, no. 3, pp. 237–246, 2002.

[20] Globus toolkit version 4: Software for service-oriented systems , "globus home page." `http://www.globus.org/toolkit/`.

[21] LCG project , "official website." `http://lcg.web.cern.ch/LCG/`.

[22] VDT project, "official website." `http://vdt.cs.wisc.edu/components/vdt.htm`.

[23] D. De Roure, M. A. Baker, N. R. Jennings, and N. R. Shadbolt, "The Evolution of the Grid," in *Grid Computing - Making the Global Infrastructure a Reality* (F. Berman, G. Fox, and A. J. G. Hey, eds.), John Wiley and Sons Ltd, August 2002.

[24] GRAAP-WS, "Grid resource allocation agreement protocol wg."
`https://forge.gridforum.org/sf/projects/graap-wg`.

[25] R. J. Al-Ali, A. Hafid, O. F. Rana, and D. W. Walker, "An approach for
quality of service adaptation in service-oriented grids," *Concurrency -
Practice and Experience*, vol. 16, no. 5, pp. 401–412, 2004.

[26] W. Smith, I. Foster, and V. Taylor, "Scheduling with Advanced Reser-
vations," in *Proceedings of the 14th International Conference on Par-
allel and Distributed Processing Symposium (IPDPS'00)*, (Los Alami-
tos), pp. 127–132, IEEE, May 1–5 2000.

[27] Q. Snell, M. J. Clement, D. B. Jackson, and C. Gregory, "The per-
formance impact of advance reservation meta-scheduling," in *JSSPP*
(D. G. Feitelson and L. Rudolph, eds.), vol. 1911 of *Lecture Notes in
Computer Science*, pp. 137–153, Springer, 2000.

[28] F. Heine, M. Hovestadt, O. Kao, and A. Streit, "On the impact of
reservations from the grid on planning-based resource management,"
in *Computational Science – (5th ICCS'05, Part III)* (V. S. Sunderam,
G. D. van Albada, P. M. A. Sloot, and J. Dongarra, eds.), vol. 3516
of *Lecture Notes in Computer Science (LNCS)*, pp. 155–162, Reading,
UK: Springer-Verlag (New York), May 2006.

[29] K. Krauter, R. Buyya, and M. Maheswaran, "A taxonomy and survey of
grid resource management systems for distributed computing," *Softw,
Pract. Exper*, vol. 32, no. 2, pp. 135–164, 2002.

[30] B. Li and D. Zhao, "Performance impact of advance reservations from
the grid on backfill algorithms," in *GCC*, pp. 456–461, IEEE Computer
Society, 2007.

[31] R. J. Al-Ali, K. Amin, G. von Laszewski, O. F. Rana, D. W. Walker, M. Hategan, and N. J. Zaluzec, "Analysis and provision of qoS for distributed grid applications," *J. Grid Comput*, vol. 2, no. 2, pp. 163–182, 2004.

[32] A. Leff, J. T. Rayfield, and D. M. Dias, "Service-level agreements and commercial grids," *IEEE Internet Computing*, vol. 7, no. 4, pp. 44–50, 2003.

[33] Y. T. Chen and K. H. Lee, "A flexible service model for advance reservation," *Computer Networks (Amsterdam, Netherlands: 1999)*, vol. 37, pp. 251–262, Nov. 2001.

[34] N. R. Kaushik, S. M. Figueira, and S. A. Chiappari, "Resource co-allocation using advance reservations with flexible time-windows," *SIGMETRICS Performance Evaluation Review*, vol. 35, no. 3, pp. 46–48, 2007.

[35] T. Roblitz, F. Schintke, and A. Reinefeld, "Resource reservations with fuzzy requests," *Concurrency and Computation: Practice and Experience*, vol. 18, pp. 1681–1703, Nov. 2006.

[36] M. Siddiqui, A. Villazón, and T. Fahringer, "Grid allocation and reservation - grid capacity planning with negotiation-based advance reservation for optimized qoS," in *SC*, p. 103, ACM Press, 2006.

[37] C. Castillo, G. N. Rouskas, S. Member, and K. Harfoush, "PARALLEL AND DISTRIBUTED SYSTEMS 1 on the design of online scheduling algorithms for advance reservations and qoS in grids," June 16 2009.

[38] EGEE gLite , "Home page." `http://glite.web.cern.ch/glite/`.

[39] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke, "SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems," in *Proceedings of 8th International Workshop on Job Scheduling Strategies for Parallel Processing*, vol. 2537 of *LNCS*, (Edinburgh, UK), pp. 153–183, Springer-Verlag, July 2002.

[40] EGEE Project, *Enabling Grids for E-sciencE*, 2007. `http://public.eu-egee.org`.

[41] J. Frey, T. Tannenbaum, M. Livny, I. T. Foster, and S. Tuecke, "Condor-G: A computation management agent for multi-institutional grids," *Cluster Computing*, vol. 5, no. 3, pp. 237–246, 2002.

[42] I. Foster, "Globus toolkit version 4: Software for service-oriented systems," 2006.

[43] "Lcg project website." `http://lcg.web.cern.ch/LCG/`.

[44] VDT, "project homepage." `http://vdt.cs.wisc.edu`.

[45] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," in *Open Grid Service Infrastructure WG, Global Grid Forum*, June, 22 2002.

[46] E. project, "Rgma homepage." `http://hepunx.rl.ac.uk/egee/jra1-uk/r-gma/`.

[47] . Berkley Database Information Index. Twiki site, "twiki.cern.ch/bdii."

[48] I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems," *Journal of Computer Science and Technology*, vol. 21, no. 4, pp. 513–520, 2006.

[49] X. Zhang and J. M. Schopf, "Performance analysis of the globus toolkit monitoring and discovery service, MDS2," *CoRR*, vol. cs.DC/0407062, 2004.

[50] Gnutella Protocol Development, "Home page," 2006. `http://www.the-gdf.org/`.

[51] J. Hathaway, "Service level agreements: keeping a rein on expectations," in *SIGUCCS*, pp. 131–133, ACM, 1995.

[52] L. jie Jin, V. Machiraju, and A. Sahai, "Analysis on service level agreement of web services," Tech. Rep. HPL-2002-180, Hewlett Packard Laboratories, July 10 2002.

[53] M. E. J. Newman, "The structure and function of complex networks," 2003. SIAM Review,Society for Industrial and Applied Mathematics,Philadelphia,Vol. 45 num. 167, pp. 167-256.

[54] N. Jennings and M. Wooldridge, eds., *Agent Technology : Foundations, Applications, and Markets*. Springer-Verlag, Berlin, 1998.

[55] D. Chess, B. Grosof, C. Harrison, D. Levine, C. Parris, and G. Tsudik, "Itinerant agents for mobile computing," *IEEE Personal Communications*, vol. 2, pp. 34–49, Oct. 1995.

[56] M. Condict, D. Milojicic, R. Franklin, and D. Bolinger, "Towards a world-wide civilization of objects," in *Proceedings of the 7th SIGOPS European Workshop*, (Connemara, Ireland), Sept. 1996.

[57] D. Lange and M. Oshima, *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley, 1998.

[58] A. Lingnau, O. Drobnik, and P. Dömel, "An HTTP-based infrastructure for mobile agents," in *Proceedings of the 4th International WWW Conference*, (Boston (MA), USA), Dec. 1995.

[59] J. Agent, "Development framework." `http://jade.tilab.com/`.

[60] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE — A FIPA-compliant agent framework," in *Proceedings of the 4th International Conference on the Practical Applications of Agents and Multi-Agent Systems (PAAM-99)*, (London, UK), pp. 97–108, The Practical Application Company Ltd., 1999.

[61] Foundation for Intelligent Physical Agents, *FIPA Abstract Architecture Specification*, Dec. 2002. http://www.fipa.org/specs/fipa00001/.

[62] A. Poggi, M. Tomaiuolo, and P. Turci, "Extending JADE for agent grid applications," in *WETICE*, pp. 352–357, IEEE Computer Society, 2004.

[63] I. Chao, R. Sangesa, and O. Oardaiz, "Grid resource management using software agents," *ERCIM News*, vol. 1, no. 59, 2004.

[64] "Agent-based resource management for grid computing," 2002.

[65] W. Jansen and T. Karygiannis, "Mobile agent security," tech. rep., National Institute of Standards and Technology, 1999.

[66] E. Christensen, F. Curbera, G. Meredith, and S. Weer-
awarana, "Web services description language (WSDL) 1.1."
http://www.w3.org/TR/wsdl, 2001.

[67] S. 1.1, "Simple object access protocol," *http://www.w3.org/TR/SOAP*.

[68] D. Puppin, S. Moncelli, R. Baraglia, N. Tonellotto, and F. Silvestri,
"A grid information service based on peer-to-peer," in *Euro-Par 2005,
Parallel Processing, 11th International Euro-Par Conference, Lisbon,
Portugal, August 30 - September 2, 2005, Proceedings* (J. C. Cunha and
P. D. Medeiros, eds.), vol. 3648 of *Lecture Notes in Computer Science*,
pp. 454–464, Springer, 2005.

[69] A. Keller and H. Ludwig, "The WSLA framework: Specifying and
monitoring service level agreements for web services," *J. Network Syst.
Manage*, vol. 11, no. 1, pp. 57–81, 2003.

[70] X. Gu, K. Nahrstedt, R. N. Chang, and C. Ward, "QoS-assured
service composition in managed service overlay networks," in *23th
International Conference on Distributed Computing Systems (23th
ICDCS'03)*, (Providence, RI), pp. 194–, IEEE Computer Society, May
2003.

[71] J. Yan, J. Zhang, J. Lin, M. B. Chhetri, S. K. Goh, and R. Kowalczyk,
"Towards autonomous service level agreement negotiation for adaptive
service composition," in *Proceedings of International Conference on
Computer Supported Cooperative Work in Design (CSCWD)*, IEEE,
May 3-5 2006.

[72] K. Xiong and H. G. Perros, "SLA-based resource allocation in cluster
computing systems," in *IPDPS*, pp. 1–12, IEEE, 2008.

[73] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware middleware for web services composition," May 01 2004.

[74] M. Zeleny, *Multiple Criteria Decision Making*. McGraw-Hill, 1982.

[75] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani, "QoS-aware replanning of composite web services," in *ICWS*, pp. 121–129, IEEE Computer Society, 2005.

[76] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, Mass.: Addison-Wesley, 1989.

[77] P. Van Hentenryck, H. Simonis, and M. Dincbas, "Constraint satisfaction using constraint logic programming," *Artificial Intelligence*, vol. 58, no. 1–3, pp. 113–159, 1992.

[78] M. Jain, P. Sharma, and S. Banerjee, "Qos-guaranteed path selection algorithm for service composition," in *Proceedings of the International Workshop on Quality of Service (IWQoS)*, pp. 288–289, IEEE, June 2006.

[79] J. Xiao, *Service-Driven Networking*. 2010.

[80] F. Casati and M.-C. Shan, "Dynamic and adaptive composition of e-services," *Inf. Syst*, vol. 26, no. 3, pp. 143–163, 2001.

[81] G. Spanoudakis and K. Mahbub, "Requirements monitoring for service-based systems: Towards a framework based on event calculus," in *ASE*, pp. 379–384, IEEE Computer Society, 2004.

[82] Xen Hypervisor, "homepage." `http://www.xen.org/`.

[83] A. Keller and H. Ludwig, "The WSLA framework: Specifying and monitoring service level agreements for web services," *Journal of Network and Systems Management*, vol. 11, no. 1, 2003.

[84] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: managing performance interference effects for qoS-aware clouds," in *EuroSys* (C. Morin and G. Muller, eds.), pp. 237–250, ACM, 2010.

[85] Amazon.com. (EC2) , "homepage." `http://aws.amazon.com/ec2/`.

[86] Google.com,   "Google   application   engine   homepage." `http://code.google.com/appengine/`.

[87] Microsoft.com,   "Microsoft   azure   homepage." `http://www.microsoft.com/windowsazure/`.

[88] N. W. Paton, M. A. T. Aragão, K. Lee, A. A. A. Fernandes, and R. Sakellariou, "Optimizing utility in cloud computing through autonomic workload execution," *IEEE Data Eng. Bull*, vol. 32, no. 1, pp. 51–58, 2009.

[89] N. G. Grounds, J. K. Antonio, and J. T. Muehring, "Cost-minimizing scheduling of workflows on a cloud of memory managed multicore machines," in *CloudCom* (M. G. Jaatun, G. Zhao, and C. Rong, eds.), vol. 5931 of *Lecture Notes in Computer Science*, pp. 435–450, Springer, 2009.

[90] B.-Q. Cao, B. Li, and Q.-M. Xia, "A service-oriented qos-assured and multi-agent cloud computing architecture," in *CloudCom* (M. G. Jaatun, G. Zhao, and C. Rong, eds.), vol. 5931 of *Lecture Notes in Computer Science*, pp. 644–649, Springer, 2009.

[91] R. Buyya, T. Cortes, and H. Jin, "Single system image," *The International Journal of High Performance Computing Applications*, vol. 15, pp. 124–135, Summer 2001.

[92] seamicro, "homepage." `http://www.seamicro.com/`.

[93] V. Sarathy, P. Narayan, and R. Mikkilineni, "Next generation cloud computing architecture: Enabling real-time dynamism for shared distributed physical infrastructure," in *WETICE* (S. Reddy, ed.), pp. 48–53, IEEE Computer Society, 2010.

[94] R. Mikkilineni and V. Sarathy, "Cloud computing and the lessons from the past," in *WETICE* (S. Reddy, ed.), pp. 57–62, IEEE Computer Society, 2009.

[95] I. Sriram and A. Khajeh-Hosseini, "Research agenda in cloud technologies," Jan. 19 2010. Comment: Submitted to the 1st ACM Symposium on Cloud Computing, SOCC 2010.

[96] G. Morana and R. Mikkilineni, "Scaling and self-repair of linux based services using a novel distributed computing model exploiting parallelism," in *WETICE* (S. Reddy and S. Tata, eds.), pp. 98–103, IEEE Computer Society, 2011.

[97] J. Neumann, "Theory of Natural and Artificial Automata," in *the History of Computing vol 12.*, pp. 408 – 474, Mit Press, 1987.

[98] P. Ferretti, *Embryos, genes and birth defects*. Wiley, 2006.

[99] F. Tusa, A. Celesti, and R. Mikkilineni, "AAA in a cloud-based virtual DIME network architecture (DNA)," in *WETICE* (S. Reddy and S. Tata, eds.), pp. 110–115, IEEE Computer Society, 2011.

[100] R. Buyya and R. Ranjan, "Special section: Federated resource management in Grid and cloud computing systems," *Future Generation Computer Systems*, vol. 26, pp. 1189–1191, Oct. 2010.

[101] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Comput. Syst.*, vol. 25, pp. 599–616, June 2009.

[102] M. M. Waldrop, *Complexity: The emerging science at the edge of order and chaos*. New York: Simon & Schuster, 1992.

[103] M. Mohamed, S. Yangui, S. Moalla, and S. Tata, "Web service microcontainer for service-based applications in cloud environments," in *WETICE* (S. Reddy and S. Tata, eds.), pp. 61–66, IEEE Computer Society, 2011.

[104] J. von Neumann, "Theory and organization of complex automata," in *Theory of Self-Reproducing Automata* (A. Burks, ed.), Urbana and Chicago: University of Illinois Press, 1966.

[105] J. V. Neumann, "The general logical theory of automata," *Cerebral Mechanisms in Behavior–The Hixon Symposium*, 1951.