### Table of Contents

Chapter 1: Introduction .........................................................................................................................19
1.1 Outline ........................................................................................................................................19
1.2 Cloud Computing and Inter-Cloud ............................................................................................19
1.3 Brief Review of Research Specifics ...........................................................................................21
1.4 Motivation ....................................................................................................................................22
1.5 Problem Statement ......................................................................................................................23
1.6 Research Aim and Objectives ......................................................................................................23
1.7 Potential Major Contribution ........................................................................................................24
1.8 Thesis Organization ......................................................................................................................24
Chapter 2: Literature Review ..............................................................................................................27
2.1 Outline .........................................................................................................................................27
2.2 Large Scale Distributed Systems ...............................................................................................27
2.3 Taxonomies of Scheduling and Meta-Scheduling .....................................................................28
2.4 Topologies of Meta-Scheduling ..................................................................................................32
2.4.1 Centralized Scheduling .........................................................................................................33
2.4.2 Hierarchical Meta-Scheduling ............................................................................................37
2.4.3 Decentralized Meta-Scheduling ..........................................................................................38
2.5 Job Distribution in Large-Scale Systems ....................................................................................46
2.6 Discussion of meta-scheduling approaches for federated clouds and inter-cloud .................49
2.7 Discussion of Meta-Scheduling Approaches ............................................................................51
2.8 Summary .....................................................................................................................................57
Chapter 3: Modeling the Inter-Cloud Meta-Scheduling (ICMS) Framework .....................................59
3.1 Outline .........................................................................................................................................59
3.2 The ICMS Topology .......................................................................................................................59
3.3 Architecture of ICMS ...................................................................................................................62
3.3.1 The Intra-level Job Distribution in ICMS .............................................................................69
3.3.2 The Inter-level Job Distribution in ICMS .............................................................................70
3.4 The ICMS Algorithmic Structure ..............................................................................................71
3.4.1 The Service-Request Algorithm ..........................................................................................71
3.4.2 The Service-Distribution Algorithm ...................................................................................73
3.4.3 The Service-Availability Algorithm ......................................................................................75
3.4.4 The Service-Allocation Algorithm .......................................................................................76
3.5 Metrics and Performance Analysis ..............................................................................................78
3.6 Summary .....................................................................................................................................83
Chapter 4: The ICMS Optimal Schemes ..............................................................................................85
4.1 Outline .........................................................................................................................................85
4.2 The Message Exchanging Optimization (MEO) .........................................................................85
4.2.1 The Mathematical Underpinning ..........................................................................................85
4.2.2 The Algorithmic Structure ...................................................................................................93
4.2.2.1 The Message-Formation Algorithm .............................................................................94
4.2.2.2 The Message-Collection Algorithm .............................................................................96
4.2.2.3 The Message-Ranking Algorithm ..................................................................................97
4.2.2.4 The Message-Redistribution Algorithm .....................................................................99
4.2.2.5 The Message-Assignment Algorithm ..........................................................................100
4.3 The VM instantiation models .....................................................................................................101
4.3.1 The static VM instantiation model .......................................................................................102
4.3.2 The dynamic VM instantiation model ..................................................................................103
4.4 The Local Resource Management System (LRMS) Optimal Scheme .....................................106
List of Figures

Figure 1: Large-scale and centralized schedulers ................................................. 32
Figure 2: The meta-scheduling topology formation ............................................ 33
Figure 3: The relationships of message passing and network bottleneck algorithms 49
Figure 4: Mapping of reviewed approaches to their extracted characteristics used for performance evaluation (key: numbers denote summarised reference tables 1, 2 and 3 in Appendix A) ............................................................................................................ 55
Figure 48: ICMS execution time, energy consumption, total costs and utilization levels
Figure 46: Turnaround times for 1 job
Figure 45: Makespan times for 1 job per user (IC vs. ICMS)
Figure 44: The ICMS topology of the decentralized meta
Figure 43: The IC topology of 5 clouds and 1 job per user (5 users)
Figure 41: Throughput percentages for 10 jobs (non-IC vs. IC)
Figure 40: Response ratio times for 10-50 jobs (non-IC vs. IC)
Figure 39: Turnaround times for 10-50 jobs (non-IC vs. IC)
Figure 38: Execution times for 10-50 jobs (non-IC vs. IC)
Figure 37: Makespan for 10-50 jobs (non-IC vs. IC)
Figure 36: The inter-cloud topology (datacentre to datacentre communication)
Figure 35: The map experimental map and the simulation deliverables
Figure 34: The topology of the static VM instantiation
Figure 33: The makespan values of SimIC when 160 identical jobs submitted by 16 users (10 jobs per user) - all clouds offer job execution.
Figure 32: The turnaround and the trend line of the turnaround of SimIC for 160 identical jobs submitted by 16 users (10 jobs per user) - all clouds can offer job execution (SLA matchmaking).
Figure 84: Total simulation time of 1000 jobs with the dynamic migration of reactive VM instantiation in comparison with the static and dynamic case ........................................173
Figure 85: Number of migrations, SLA violation and energy consumption for reactive VM instantiation ........................................................................................................174
Figure 86: Comparison of makespan times for ICMS cases (1, 4 ms delay, 40% availability for static and dynamic VM instantiation) ........................................................................................................175
Figure 87: Comparison of makespan times for ICMS cases (1, 4 ms delay, 40% availability for static and dynamic VM instantiation) ........................................................................................................175

List of tables
Table 1: List of message-exchanging notations ....................................................................................86
Table 2: Simulation frameworks and their characteristics .................................................................115
Table 3: Host specification files .......................................................................................................130
Table 4: Cloud configuration parameters for input in CloudSim and SimIC .....................................132
Table 5: User configuration parameters for input in CloudSim and SimIC .....................................132
Table 6: The centralized use case specification ...............................................................................161
Table 7: The decentralized use case specification ...........................................................................161
Table 8: The cloud experimental configuration of hosts and VMs ..................................................168
Table 9: The experimental comparison ............................................................................................176

List of algorithms
Algorithm 1: Service-Request ........................................................................................................72
Algorithm 2: Service-Distribution ..................................................................................................73
Algorithm 3: Service-Availability ....................................................................................................75
Algorithm 4: Service-Allocation .....................................................................................................77
Algorithm 5: Message-Formation ...................................................................................................94
Algorithm 6: Message-Collection ..................................................................................................96
Algorithm 7: Message-Ranking .......................................................................................................98
Algorithm 8: Message-Redistribution ............................................................................................99
Algorithm 9: Message-Assignment ...............................................................................................100
Algorithm 10: Static VM instantiation job distribution algorithm ...............................................102
Algorithm 11: Dynamic VM instantiation job distribution algorithm ........................................104

Abbreviations
LRMS: Local Resource Management System
ICMS: Inter-Cloud Meta-Scheduling framework
MEO: Message Exchanging Optimization model
SimIC: Simulating the Inter-Cloud toolkit

List of formulas
Formula (1): VM execution time ......................................................................................................78
Formulas (2, 3): Cycles per instruction and execution time calculation as presented into Brady (2012) ........................................................................................................79
Formula (4): VM total execution time ..............................................................................................79
Preface
The study aim is on modeling the Inter-Cloud Meta-Scheduling (ICMS) framework. The first contribution of this thesis is the ICMS model, a meta-brokering solution for efficient job distribution in inter-clouds. The architecture of the model encompasses four algorithms, namely Service-Request, Service-Distribution, Service Availability and Service-Allocation in order to demonstrate the job distribution stages. The second contribution is the optimal schemes that extend the model to effectively control message exchanging, virtualization and local resource management. The experimental analysis is extended, and based on the CloudSim and the SimIC toolkit. The SimIC constitutes the third contribution of this thesis as it implements an inter-cloud enabled simulation toolkit to explore various cloud configurations, user submissions and integration of modular optimal schemes. The extracted results are productive, as it is observed that the model outperforms the standard inter-cloud configuration. Although the model is aimed at a decentralized and large scale inter-cloud the utilization of novel optimal schemes further optimizes results; for instance the evaluation shows that the Message Exchanging Optimization policy further improves performance.
Abstract
Inter-cloud is a recently emerging approach that expands cloud elasticity. By facilitating an adaptable setting, it purposes at the realization of a scalable resource provisioning that enables a diversity of cloud user requirements to be handled efficiently. This study’s contribution is in the inter-cloud performance optimization of job executions using meta-scheduling concepts. This includes the development of the inter-cloud meta-scheduling (ICMS) framework, the ICMS optimal schemes and the SimIC toolkit. The ICMS model is an architectural strategy for managing and scheduling user services in virtualized dynamically inter-linked clouds. This is achieved by the development of a model that includes a set of algorithms, namely the Service-Request, Service-Distribution, Service-Availability and Service-Allocation algorithms. These along with resource management optimal schemes offer the novel functionalities of the ICMS where the message exchanging implements the job distributions method, the VM deployment offers the VM management features and the local resource management system details the management of the local cloud schedulers. The generated system offers great flexibility by facilitating a lightweight resource management methodology while at the same time handling the heterogeneity of different clouds through advanced service level agreement coordination. Experimental results are productive as the proposed ICMS model achieves enhancement of the performance of service distribution for a variety of criteria such as service execution times, makespan, turnaround times, utilization levels and energy consumption rates for various inter-cloud entities, e.g. users, hosts and VMs. For example, ICMS optimizes the performance of a non-meta-brokering inter-cloud by 3%, while ICMS with full optimal schemes achieves 9% optimization for the same configurations. The whole experimental platform is implemented into the inter-cloud Simulation toolkit (SimIC) developed by the author, which is a discrete event simulation framework.
Dedications

I lovingly dedicate this thesis to my fiancé Maria, to my parents Nikolaos and Olga and to my good friend Nikolaos who supported me each step of the way.
Acknowledgements

The author would like to express his appreciation to all those people who made this thesis possible and overwhelming experience.

First of all, the author is grateful to his PhD advisors Professor Nik Bessis and Professor Nick Antonopoulos for giving him the opportunity (including the financial support) to study a PhD at the University of Derby, UK. Both have been steady influence and became supportive in many instances. Special thanks go to research colleagues for their valuable suggestions and help. The author expresses his gratitude to Professor Pierre Kuonen, Dr. Eleana Asimakopoulou, Dr. Ye Huang, Professor Valentin Cristea, Dr. Florin Pop, Dr. Ciprian Dobre, Dr. Ashiq Anjum, Professor Richard Hill and Professor Fatos Xhafa.

The author is indebted to his fiancé Maria, to his parents Nikolaos and Olga, to his brother and his wife Kostas and Eleni for their support during this study and to his family members Nikolaos and Chrisanthi. Their encouragement in many instances has been indispensable.
List of main publications


Chapter 1: Introduction

1.1 Outline
This chapter introduces the concepts and key terms that are discussed in the thesis. These include clouds, federated clouds and inter-clouds. It further presents an overview of the aim, motivation and the methodological approach taken to produce this research work. Finally, it presents the contribution of this research, namely the development of an inter-cloud metascheduling model. Finally, the thesis organization is presented.

1.2 Cloud Computing and Inter-Cloud
The concept behind cloud computing is to provide a computer-based environment where a range of computational services are available to be consumed by users. Various cloud providers such as Amazon (Amazon Elastic Compute Cloud – Amazon EC2) and Salesforce (Salesforce.com) have made such computational services available to clients so they can access them remotely on a pay-on-demand model. Cloud computing is also known for its virtualization where on-demand services are available through virtual machines (VMs) in order to support greater levels of elasticity (Zhao and Huang 2009). Specifically a VM is a representation of a physical machine and it is used for server consolidation to manage scalability and availability.

This is the important cloud computing advantage, to scale user-leased resources on demand (a process called elasticity). There are three main roles in a cloud, namely, service consumer, service provider and service creator (Lens et al. 2009). Traditionally, the service provider generates a service request that is utilized by the consumer to represent the user hardware and software requirements for leasing cloud capacity. This request is hosted in the premises of a service provider.

A cloud service life cycle contains various user requests for services submitted to a cloud service provider. There are various types of clouds depending on the service availability and accessibility level. The most common are public, private, virtual public, and hybrid clouds. All these types are directly related to the way that services are provisioned to the end-users (Cisco Cloud Computing n.d.):

a) The public cloud provides services to clients over the public Internet using a pay-per-usage model.

b) The private cloud provides services to be consumed by a single organization and could be hosted internally or externally to the organization.
c) The virtual private cloud is managed by a single organization and allows specialized services to be offered by the provider to the clients. This includes migration of workloads from private to virtual private clouds.

d) The hybrid cloud is a combination of different types of clouds that are bounded together in order to be developed as a multiple deployment model. For example, a cloud that shares characteristics found in a private and virtual private cloud is known as a hybrid cloud.

Clouds could also offer a variety of services with regards to the content that is to be delivered to the users (Mateescu et al. 2011). These are as follows:

a) Infrastructure as a Service (IaaS) includes traditional computing resources. These are cluster servers, storage, and other forms of low-level network and hardware resources.

b) Platform as a Service (PaaS) includes an execution environment as a runtime-system for serving applications.

c) Software as a Service (SaaS) includes specialized software that is delivered to users who intend to develop real world processes.

Lately, inter-clouds as a new cloud approach has been emerged. Inter-clouds involve public clouds forming a collaborative partnership that allows distribution and common management of services. An inter-cloud represents the communication glue between the different services available from the different cloud infrastructure layers. The vision of this research is to focus on inter-clouds by decoupling resource consumers from providers by allowing providers to offer resources on demand and consumers to access the inter-cloud by having an infinite view of elastic services.

Cloud elasticity and scalability allows users to manage their leased resources easily e.g. to increase the computational capacity of their services without shutting down the service. It is perceived that an inter-cloud as an extended interoperable environment of clouds will offer additional advances along with elasticity and scalability. This includes new services to users in a coordinated workload management setting. To achieve this, a new model is required to inherit the requirements and characteristics of typical large-scale distributed systems (e.g. high performance computing systems and grids).

A new approach suggests that clouds need to come together and agree on common coordination and collaboration for improving the whole quality of service (QoS). Several research bodies including Global Inter-Cloud Technology Forum (2010) have highlighted this need. For that reason, Buyya et al. (2010) introduced a cloud federation to develop new cloud capabilities by resource sharing. Specifically, that study suggests that the federated
Cloud will require organizing a set of co-operated infrastructures identical to a dynamic distributed system. These include sophisticated protocols for controlling trust standards, resource discovery, naming and mining systems, scheduling of jobs, and workload exchange services.

1.3 Brief Review of Research Specifics
This section presents a brief discussion of the problem area. Cloud computing is one of the most important technologies for delivering on-demand services including computing infrastructure and software via the Internet (Foster et al. 2008). Leading cloud vendors develop datacenters that may be situated at dispersed geographical locations. However, as the number of resource consumers increases the elastic clouds may start suffering in terms of resource provisioning capabilities. A common solution is to aggregate various clouds that agree on mutual protocols for improving the overall QoS. The Global Inter-Cloud Technology Forum (2010) highlights the need for this direction by suggesting that current efforts do not support a coordinated distribution of different clouds workloads. To address this limitation, this research study proposes an inter-cloud that expands cloud capabilities by providing a flexible initiative for sharing resources. Specifically, inter-cloud forms a pool of collaborated and federated sub-clouds that target to the service provisioned capacity.

Currently, most resource providers develop clouds that are usually limited to the size of their physical capacity (cluster size). This is because clouds are usually datacenter facility oriented (Jin et al. 2011), thus a lack of providing a high level of data redundancy could be observed. As a result, while the number of resource consumers increases, the mapping of resource provision to consumption is becoming unbalanced, and the overall potential for improved QoS is limited to the cloud capacity. So, clouds could come together, leading to a form of inter-cloud infrastructure in which sharing is motivated by an overall scope of performance improvement. HP, Intel, Yahoo, etc. (Buyya et al. 2010), who are the big players in this area, highlight such a growing interest. Their innovative efforts have led to the establishment of a federation of collaborated clouds with joint initiatives.

However, the vendors’ effort has a specific control plane rather than a setting based on future standards and open interfaces. In reality, to support service provider cooperation, the establishment of communication among geographically distributed sites is essential. Cerf (2011) stated that current efforts in this direction show that even the most famous vendors do not support scalable geographical coordination on workload balancing, thus a shredding of services among their datacenters is observed. Further, they suggest that the biggest providers,
such as Amazon, request their clients to select their best location for hosting services. This somehow questions the provider’s QoS level as the decision for cloud location selection is left to the users. In such an environment the overall performance is related to users’ random decisions. Based on that, this study focuses on developing an inter-cloud model to allow service distribution in order to achieve better elasticity and scalability. This could lead to improved QoS.

One of the most important features in job distribution in large-scale settings is the scheduling. A scheduler deals with the selection of resources considering the characteristics of the actual system (centralized or decentralized) as well as the requirements of the desired scenario. Various taxonomies of schedulers have been introduced, including schedulers for operating system (OS), high performance computing, parallel and meta-computing. Among all these, the task of scheduling in meta-computing has proven to be the most complex (Xhafa and Abraham 2010), mostly due to the involvement of a mixture of local resource management systems (LRMS). Since an inter-cloud involves multiple LRMS, meta-computing is chosen as the key research area of this study.

1.4 Motivation
The motivation of this study relies on the statement of Vinton Cerf (The Inter-Cloud and the Future of Computing 2011) on the inter-cloud and the future of computing. Specifically, he notes that today there are different cloud providers that develop different systems; however they share the same characteristics in terms of remote machines that aim to solve customer problems. By discussing the Internet’s evolution he suggests that an inter-cloud shares similar characteristics, however without having inter-cloud standards. In this area meta-scheduling forms a key requirement as one of the most important paradigms for large scale job distributions (Bessis et al. 2012b). Thus, the present research study’s inspiration is based on this limitation of current cloud and federated cloud infrastructures to extend service distribution in a similar way as the Internet setting as recently stated by Cerf (2011), Buyya et al. (2010), the Global Inter-Cloud Technology Forum (2010) and Chapman (2012)

The research study’s motivation is based on the fact that the cloud paradigm shares similarities with other technologies including high performance and grid computing. For example, the grid characteristic of resource geographical distribution highlights new requirements for inter-clouds. In view of these, an inter-cloud could also be seen as an infrastructure allowing access to resources and services available from one or more of another cloud, cluster, HPC or grid infrastructure. So, inter-clouds can be considered as a
wider research effort, like efforts undertaken in HPC-to-grid and/or cloud-to-grid research where issues including resource discovery, allocation and scheduling are the most important (Majumdar 2011). One of the most important reasons for selecting this area is the dynamics of the system and the unpredictability of resource availability. These constitute the challenges that determine the present research study motivation. Dynamics were the focus of attention for many years and these allow decisions during the execution time.

In brief, the focus of this research study is on the service distribution concept in inter-cloud. The aim is to effectively achieve an efficient QoS by systematically assigning services in the form of job-to-cloud low-level resources. This involves incentives for users (e.g. by always executing requests or offering more competitive service execution costs and times) and for clouds (e.g. through lower energy consumption rates). In this setting, the services (jobs) encapsulate the actual capabilities offered by the cloud environment. Thus, the challenge is to identify the rationality behind the decisions of service provisioning by the cloud provider in an inclusive approach (overall inter-cloud) of process management.

1.5 Problem Statement
The research problem encompasses the following three research questions.

a) Is it possible to develop a meta-scheduling architecture that facilitates effectiveness in terms of performance inter-cloud service distribution? The term ‘effectiveness’ is defined as the ability of the inter-cloud to execute the service requests in the required amount of time.

b) Is it possible to identify and define the key optimal schemes that optimize the service distribution in order to improve inter-cloud performance?

c) Is it possible to design a toolkit to simulate the underlying model and optimal schemes?

So the research problem of this study relies on linking clouds in order to achieve wider service distribution and optimize inter-cloud elasticity as a whole.

1.6 Research Aim and Objectives
This section presents a brief discussion and methodology of the research aim and the study objectives.

The aim of the research study is to develop a meta-scheduling model to optimize service elasticity for inter-clouds.

In view of this, the research study’s objectives are ordered as follows.
A. A literature review of local and meta-schedulers for highly distributed and dynamic environments (grids and clouds) with the aim of identifying relevant critical issues. The requirements analysis process addresses the most important features towards the inter-cloud meta-scheduling approach as drawn from the literature review.

B. The development of a novel meta-scheduling approach that assimilates various heuristic scheduling objectives for achieving an efficient resource management of the different inter-cloud entities. This includes the identification of inter-cloud meta-scheduling performance metrics with the aim of identifying benchmarks that lead to the evaluation of the model.

C. The identification of optimal schemes that affect the performance of the model. This includes the presentation of algorithms and their mathematical underpinning of the model’s optimal scheme.

D. The development of a toolkit to simulate the inter-cloud meta-scheduling model and the optimal schemes. The evaluation of this will support the novelty of the study.

1.7 Potential Major Contribution
This research study presents the inter-cloud meta-scheduling framework for achieving wider service dissemination. This study encompasses potential contributions in the area of meta-scheduling in inter-clouds, optimal schemes for such models and simulation settings for experimental purposes. In detail these are as follows:

a) To propose the inter-cloud meta-broker (ICMS) model inspired by the meta-computing paradigm for realizing communication among low-level entities of the underlying infrastructure.

b) To propose a set of optimal schemes that encompasses the enhancement of the architectural design features of ICMS including the message exchanging optimization, the virtual machine deployment and the local resource management system.

c) To propose an experimental setting (SimIC) wherein inter-cloud is developed with respect to internal cloud optimal schemes for resource request, distribution, availability, allocation, execution and monitoring. These include allocation and deployment optimal schemes for both user services and cloud VMs.

1.8 Thesis Organization
The thesis is structured as follows.

Chapter 2 presents a survey of computational models for scheduling decisions in dynamic and flexible distributed environments. Specifically, a state-of-the-art literature review of
existing works is discussed as part of the study’s Objective A by focusing on delimiting the problem area of meta-scheduling in inter-clouds. The approaches presented herein recognize the needs that lead to the development of the proposed model.

Chapter 3 presents the model of the Inter-Cloud Meta-Scheduling (ICMS) framework that covers all the important components of the core functionality for wide service dissemination as part of Objective B. In addition, the research study describes the conceptual model of the development process as well as the justification of performance metrics and measures. In advance, it illustrates that the ICMS is characterized by a distributed and decentralized meta-computing operation known as the meta-broker. The chapter also illustrates the algorithm pseudo-codes along with the mathematical model of the ICMS.

Chapter 4 presents the ICMS optimal schemes that include the message exchanging, the VM deployment and the LRMS optimal scheme management. These implement the novel functionalities of the ICMS where the message exchanging represents the job distributions method, the VM deployment offers the VM management features and the LRMS details the management of the local cloud schedulers. The next section details the operations of each optimal scheme. These are related to Objective C.

Chapter 5 presents the experimental platform of the ICMS wherein all the crucial components are simulated in the SimIC infrastructure. The discussion encompasses the technical characteristics as well as the realization of the algorithmic models and benchmark metrics as developed in previous chapters. Benchmark results are demonstrated from the CloudSim simulation toolkit. During the simulated service life cycle various operations are taking place including resource request, distribution, availability, allocation, execution and monitoring. The study evaluates results and compares different experimental configurations as part of Objective D.

Chapter 6 presents the experimentation and optimization that illustrates the meta-broker service distribution, the dynamic scheduling, the message exchanging model, the virtual machine deployment model and the local resource management system evaluation. Additionally, related experimentation is based on the ICMS message exchanging, VM deployment and local resource management optimal schemes.

Chapter 7 summarizes the contributions and outlines future research steps. The remarks and conclusions present the overall study accomplishment by evaluating the entire research study. Also, the applicability of the solution includes the granularity of the proposed model when applied to various systems and scenarios. Finally, the study’s limitations and future prospects conclude with a discussion of future ICMS development phases.
Chapter 2: Literature Review

2.1 Outline
This chapter presents a survey of computational models for resource management with a particular emphasis on scheduling and meta-scheduling approaches. This includes decision-making processes as executed in clusters, high performance computing (HPC), grids and clouds. A state-of-the-art literature review of related works is discussed as part of the study Objective A that concludes with critical evaluation of the study’s requirements.

2.2 Large Scale Distributed Systems
An inter-cloud is a resource management setting where different clouds communicate with each other in order to exchange services (also known as jobs). So the topology of interconnected sub-clouds could be considered similar to large-scale settings (e.g. grids and clouds). This forms the key research topic for this study. In this context, scheduling and meta-scheduling refer to the way in which resources are allocated, and both have a major role in the management of jobs in an inter-cloud. This includes that a scheduler receives user jobs, selects available resources according to different conditions (e.g. availability) and evaluates performance criteria to plan jobs to resources (Xhafa and Abraham 2010). The study is focused on identifying scheduling and meta-scheduling architectures of large-scale distributed systems.

The notion of resource management involves the orchestration of resources in such way that tasks are efficiently executed within a resource. The term efficiency refers to various criteria posed by the system including performance metrics as task execution times, latencies etc. The local and meta-scheduling concepts aim to resource allocation and management, however each one from a different perspective. The local schedulers (also known as local resource management systems – LRMS) are used at the cluster level, to achieve workload balancing. Meta-schedulers are used to assign user jobs to resources based upon user-defined requirements (Leal et al. 2009). In large-scale settings meta-schedulers constitute the communication link among different distributed systems.

Fundamentally, the concept of distributed systems involves multiple autonomous machines (also known as hosts or nodes) that communicate through a network or Internet to achieve a common target (e.g. to deliver computational capacity for executing jobs). During the last decades several computing architectures have been appeared with regards to different operational scenarios such as small enterprise networks of hosts or large-scale interconnected machines. The best known computing paradigms are high performance computing
(HPC) or high throughput computing (HTC), grids and more recently clouds (Mateescu et al. 2011). Originally HPC represents an owner-centric resource provisioning architecture (cluster-based) where resources are locally owned, and clients have private access to the owner organization. Here, the allocation happens in relation to the user submissions (tasks) and jobs are shared among resources with respect to the workflow defined by the user. Huang et al. (2011) suggests that the aim of HPC-HTC is to gain great computational power for solving complex problems normally in a small-scale administrated environment.

Grid implies that resources are locally and/or externally owned, and thus includes a wider administrated resource field. Members of the grid constitute virtual organizations (VOs) (common interest groups) and have access to resources in a public manner (Forster and Kesselman 1999). In grids, heterogeneous resources could enter and leave the grid dynamically. This makes the administration and scheduling in grids a challenging issue (Rodero et al. 2009). A way to achieve a feasible allocation (all jobs will be executed on time) is by using the meta-scheduling concept. Each grid has a resource management component named as meta-scheduler that is placed on the top of the resources to allow communication with remote sites. So, different meta-schedulers are linked to allow job distribution and allocation and execution.

Finally, clouds aim for wide resource management where resources can be externally or internally owned forming a public, private or hybrid setting. The pay-on-demand cloud model allows a user to access resources by using virtualization technology (Buyya et al. 2009). This provides the cloud’s elasticity and allows cloud providers to dynamically create, migrate and destruct virtual resources according to user demands (Mateescu et al. 2010). So, the cloud is a large-scale cluster where resources are locally owned and there is no support for wider coordination and service distribution. In this setting, users can access cloud resources remotely, so it is considered as a large-scale distributed system. Next, the study focuses on the taxonomies of scheduling and meta-scheduling.

2.3 Taxonomies of Scheduling and Meta-Scheduling
This section presents the static and dynamic taxonomies of scheduling and meta-scheduling. Static is defined as the scheduling case where decisions are done before execution of scheduling, in contrast to the dynamic case in which all decisions are made during the execution of the scheduling. Static or dynamic schedulers could be adapted in various hybrid schemes (local or meta-scheduling) depending on the requirements of the system selected. For example, in HPCs most schedulers are static and centralized as in Guim and Corbalan
(2007) and Brawn et al. (2001), however in grid most schedulers are dynamic and either local or meta-schedulers as in Huang et al. (2011), Iosup et al. (2008a) and Leal et al. (2009). This is because the large-scale task distribution involves dynamic meta-schedulers while static schedulers are mostly used in local batch systems (Christodouloupolous et al. 2008).

At first, the static schedulers assign tasks (processes) to CPU cycles through a ready queue. This is a fundamental characteristic for job execution at the lowest CPU level. Several local schedulers have been developed to achieve scheduling in small-scale systems by taking simple queuing decisions. In brief, the best known are the first-come-first-served, round robin, shortest job first, highest priority first, multi-level feedback and backfilling. Stallings (2004) suggests that these are divided into two types, preemptive and non-preemptive. Preemptive processes suspend execution after a specific time, and non-preemptive one run until they finish their execution. A discussion of well-known static schedulers is presented below.

- **First Come First Served (FCFS)** is a non-preemptive scheduling method where the jobs are placed at the end of the queue and wait for the previous jobs to be executed (Schwiegelshohn and Yahyapour 1998). This algorithm is widely adopted and can be found in real systems with low overhead features. It is a simple algorithm as it is fair in the sense of service arrival (Rodero et al. 2009). However the algorithm does not consider short or important jobs that could be delayed for long periods or while less important jobs are completed. Nevertheless, real-world local grid schedulers such as Maui (Bode et al. 2000) and Sun Grid Engine (Gentzsch 2001) use this classic queue local scheduler as part of their overall scheduling implementation.

- **Smallest Job First (SJF)** (Yoo and Das 2001) and **Largest Job First (LJF)** (Li 2005) schedulers are associated with the size of the job in a non-preemptive fashion. Initially, SJF schedules the queue by the least estimated time process to be first, in contrast to the LJF that gives priority to the longest jobs. The advantages of such a solution is that the scheduler maximizes throughput times (for SJF) as well optimizes the average turnaround time (for LJF). Nevertheless, jobs waiting in the queue might never be executed (a case called process starvation) because they never become the smallest or largest by size respectively.

- **Fair Share** (Alam et al. 2009) scheduling is a simple strategy where resource usage is equally distributed to the jobs. The algorithm offers a fair environment in terms of job serving. However, important processes with high computational demands share the
same processing power with the least important ones. Fair share is found in cluster management systems as part of their scheduling algorithm, such as the Sun Grid Engine (Gentzsch 2001).

- **Round Robin (RR)** (Ramabhadr and Pasquale 2006) is a hybrid preemptive version of the FCFS that treats jobs in the ready queue in a circular manner. When a job arrives at the queue it is placed at the end. The scheduler executes jobs until completion of their length, or until a specific quantum expires. In the second case the jobs are again placed at the end of the queue. The algorithm offers simplicity and low overhead depending on the quantum value. However, if the quantum is too high the algorithm turns into FCFS, and if the quantum is too small then the algorithm waists time for switching from one process to another.

- **Shortest Remaining Time First (SRTF)** (Harchol-Balter et al. 2003) is a preemptive version of SJF. In this algorithm the scheduler selects the shortest job to run or the job with the shorter remaining time until completion. This solution handles short processes very quickly; however, it is possible for long processes to wait in the queue while short processes are continuously added. Similar to SJF, Huang (2012) states that this algorithm is rarely used in real time environments due to the possible starvation of long jobs.

- **Highest Priority First (HPF)** (Kamoun 2008) is a general class of algorithms (preemptive or non-preemptive) where each process is assigned a priority. In this solution, the ready queue is ordered with jobs based on their priority with the highest one first. Several variations of this solution have been developed based on aging factors of priorities in order to prevent starvation of jobs with low priorities. HPF algorithm includes complexity that involves the effective computation of priorities; however it cannot handle efficiently a variety of jobs such as interactive batch jobs. Real world schedulers such as the Portable Batch System (Bode et al. 2000) and the Sun Grid Engine (Gentzsch 2001) use advanced priorities as part of their scheduling functionality.

- **Earliest Deadline First (EDF)** (Srivastava 2006) is a dynamic priority algorithm in which the priority of tasks can change over time. This algorithm uses priority of job as its deadline, so the job with the highest priority has the smallest deadline. The scheduler prioritizes the jobs by ensuring that no deadlines are missed during the period.
• **Multi-level queue (MLQ)** and multi-level feedback queue (MLFQ) (Hoganson 2009) are both more sophisticated schemes for addressing the management of a selection of jobs. The MLQ scheduler offers a multi-queue solution for each type of job. Using this method processes can be categorized, e.g. system, interactive and background, and different scheduling algorithms applied for each type of job. The MLFQ uses the time-slice method of the RR. Each time a process exceeds the quantum, it is moved to a different queue with a longer quantum. Other criteria, e.g. priorities, may be responsible for non-permanent queue assignment. The last level of the algorithm uses the FCFS scheme.

• **Backfilling** (Zotkin and Keleher 1999) is a class of algorithms in which jobs are moved to the front of a queue in order to fill any spaces created by a different algorithm, usually the FCFS. The algorithm requires that each job specifies a maximum execution time, thus jobs that are moving forward in the queue would not delay existing jobs. Specifically two common backfilling solutions are the aggressive and conservative backfilling solutions discussed below.
  o In **aggressive backfilling** (Leinberger 1999) small jobs in the queue will be moved forward as long as they do not delay the first job in the queue. This type of algorithm is usually called Extensible Argonne Scheduling sYstem (EASY) and has been adopted by many local scheduling systems such as Maui scheduler (Bode 2000) and Load sharing facility (Zhou et al. 1993) that utilize backfilling techniques as part of their scheduling functionality.
  o **Conservative backfilling** (Franke et al. 1999) algorithm is a similar to the EASY algorithm. Instead of preventing the delay of the first job, this algorithm moves small jobs ahead in the queue only if none of the jobs in the queue are delayed. Based on that functionality this method offers a more deterministic solution compared with EASY.

• **BiggestFree** (Hamscher et al. 2000) algorithm selects a resource from a list of currently free resources that has the largest amount of free computational power. Since this algorithm does not support priorities, the biggest drawback is that small or non-critical may be executed before biggest or highly critical jobs.

• **BestFit** (Hamscher et al. 2000) algorithm selects a resource that is able to execute a job and has enough free resources to start the job (least free resources if the job started). A
study by Hamecher et al. (2000 p.6) suggests that “in comparison with the BiggestFree this strategy does not unnecessarily fill up larger machines with smallest jobs”.

To conclude, the list of schedulers forms the basic strategies for static and centralized distributed systems. Figure 1 shows a map of large-scale schedulers for static and centralized systems.

![Diagram of schedulers](image)

Figure 1: Large-scale and centralized schedulers

The schedulers for dynamic settings are more complex as they organize a group of machines, each of which has a local scheduler (Christodoulopoulos et al. 2009). This includes a setting that is dynamically changed as resources could join or leave at any instant. In addition, the environment could contain heterogeneous machines with different configurations. For this reason the meta-computing paradigm has been presented where scheduling is considered as a set of policies (Xhafa and Abraham 2004) due to the different needs of users and resources. This includes that job scheduling is heterogeneous as jobs have diverse requirements for execution. The large scale of the environment represents a large amount of local schedulers that require interacting. Finally, meta-scheduling distributed systems are geographically dispersed and belong to several owners. So resource sharing based on Service Level Agreements (SLAs) signed between vendors as well users and vendors. These agreements can change dynamically and could affect the availability of resources. The next section presents the topologies of the meta-scheduling approaches.

2.4 Topologies of Meta-Scheduling

A scheduling topology defines that resources are linked with a specific structure with respect to their resource accessibility level. In meta-scheduling a resource manager is placed on the top of the local queue (that represents the static schedulers) in order to methodically distribute requests. Meta-schedulers are organized in three topologies: centralized, hierarchical and
decentralized. Figure 2 demonstrates the actual formation of the meta-scheduling topologies. In case (a), the meta-scheduler can access all resources directly, in case (b) the meta-scheduler can access resources in a hierarchy of levels, and in (c) the decentralized meta-schedulers can access some of the resources. In the latter case, the decentralized meta-schedulers could communicate in order to exchange resource job requests. Next, the study focuses on each topology and presents a discussion of the meta-scheduling approaches.

Figure 2: The meta-scheduling topology formation

2.4.1 Centralized Scheduling

The centralized topology includes management of resources from a central component that maintains information for all of the resources that constitute the distributed system (Hamscher et al. 2000). Each time new jobs are submitted; the centralized meta-scheduler assigns jobs in LRMS queues. So it does not perform scheduling (Subramani et al. 2002) but only acts as dispatcher for a wide and sophisticated distribution.

In Carrol and Grosu (2008), the authors discuss the scheduling problem with regards to multiple processors. By scheduling tasks on a subset of processors in a centralized topology the authors prevent random scheduling. Also, the authors have developed a variety of theoretical algorithms and formulas for independent task scheduling. This is a simple solution that does not consider heterogeneity of resources. In addition, the majority of these algorithms are difficult to be implemented in real cases (Gehring and Preiss 1999). Freund et al. (1998, p. 8 for direct quote) discuss a scheduling system that gains benefits when “the scheduler considers both the computer availability and the performance of each task on each computer”. The results from the evaluation were encouraging for localized systems. However, the authors conclude that as the system grows larger, the performance decreases dramatically.

In the same direction, Gehring and Preiss (1999) study meta-scheduling issues by considering real workloads that demonstrate decent performance. The aim of this work is to
develop a centralized approach for providing solutions for the NRW meta-computer (Bitten et al. 2000) that is a country-wide meta-computer (based in Germany). The authors suggest that a major feature for meta-computing scheduling is the existence of local queues to which the “jobs submitted without being passed through the meta-computer system” (Gehring and Preiss 1999, p. 20). Therefore, they have developed Interleaved Hybrid Scheduling (HIS) as an approach to use well-known scheduling algorithms for meta-computing prototypes. The scheduling case is to deny the direct submission of jobs to the local dispatcher, thus the meta-scheduler works similarly to a single parallel machine scheduler.

The same authors suggest that the HIS method has been evaluated mostly by using FCFS, First Fit Increasing Height (FFIH) and Adaptive Static Partitioning (ASP). FFIH is derived from FCFS and sorts all requests for increasing resource demands. ASP, on the other hand, is more sophisticated, as it stores incoming requests in a waiting room until there are free processors. They conclude that a major advantage is that the HIS method with the FFIH algorithm decreases the average response times of moderated workloads significantly. The results were encouraging but limited. This is because powerful algorithms have not been evaluated yet. Also, another drawback is the fact that the authors do not consider the reliability of jobs as well as the bandwidth factor of the geographically distributed resources. So such a method is applicable to consistent and non-dynamic environments. Finally, the ASP shows good performance on monolithic machines.

In a different view from the above discussion, Snell et al. (2000) present an advanced reservation mechanism based on meta-scheduling. The work aims to build a method in which jobs are exclusively used by a resource (Teo et al. 2004). Specifically, the approach allows the local scheduler to select resources for job execution by assigning resources to job usage. The scheduler called Ursala, utilizes the Maui scheduler that supports various scheduling policies, e.g. FCFS and SJF (Jackson et al. 2001). The Maui scheduler reports its findings to the Ursala. The latter receives the list, computes start times and attempts to create reservations in the Maui schedulers. This centralized solution allows full optimization of scheduling, by minimizing the negative impact of meta-scheduled workloads (Snell et al. 2000). Also experimental results show an improvement of response times of supercomputer centers. The major drawback of this centralized approach is that for a realistic case it does not provide advanced reservation support by all schedulers (Teo et al. 2004).

Scheduling for centralized supercomputers has been discussed also in Brawn et al. (2001). The study presents a static mapping of jobs to resources in a predictive manner for minimizing the total execution time of the jobs. The authors define the mapping as the
matching and scheduling procedure. They further suggest that the problem of optimally mapping is a NP-hard problem due to the large resource number. By comparing eleven different heuristic algorithms they provide a discussion that reveals the appropriate heuristic to be used for each case scenario. Finally by evaluating the performance of the eleven algorithms, the authors conclude that the GA consistently gives the best results.

An advanced scheduling strategy for multi-site scheduling in grid environments has been discussed in Ernemann et al. (2002a). The algorithm contains three improvements; firstly the jobs are split into fragments in the case where a job cannot be allocated to a specific machine. This happens by using backfilling for communication and migration for fragments. Secondly when a resource can be placed directly then a BestFit (Huang 2012) policy occurs. Thirdly, the algorithm utilizes a global dispatcher for achieving geographical distribution of resources. The evaluated results in Ernemann et al. (2002a, p. 11) show that “the usage of small systems in combination with multi-site scheduling cannot perform as well as single machines with the same amount of resources”.

The backfilling algorithm is also presented as a solution for centralized meta-scheduling in (Yue 2004). The authors design a global backfilling scheduler and the results show that the policy outperforms the independent site execution, however again in a centralized manner. Dissimilar to the above strategy, a placeholder monitoring and throttling algorithm has been introduced in Pinchak et al. (2002) that works across distributed and local schedulers. The basic idea is to centralize the jobs of the workload into a meta-queue, then use the placeholder to move the job to the next accessible queue and use late binding to offer flexibility. The results show that the scheduler can load balance workloads between heterogeneous and centralized administrative domains.

Guim and Corbalan (2007) discuss a self-scheduling policy for high performance computing. The algorithm works on a two level architecture. At the top the dispatcher schedules the job, and then the resource local scheduler executes the job. The scheduling policy uses a set of tasks assignment policies to decide where jobs will be finally submitted. They suggest that their algorithm is non-centralized, however it behaves better in a centralized solution by achieving efficient global performance. The evaluation shows that waiting times as well as resource usage has been improved.

The meta-brokering concept was introduced by Kertesz et al. (2009) and aims to ease the addition and usage of different resource managers (brokers). To achieve that, a brokering portal has been designed to reach resources of different grids in an automated way. “This way a workflow can be brokered over several grids based on different underlying technologies but
still providing optimal utilization of resources” (Kertesz et al. 2009, p. 1). The portal or meta-broker is a scheduler standing on top of the local meta-scheduler. The evaluation results show significant improvements when compared to the conventional meta-scheduling systems.

In Kertesz et al. (2008) a meta-brokering approach is discussed to support grid interoperability. Specifically, the meta-broker sits on the top of a resource broker and uses meta-data to decide where to send the job. Such scheduling methods are called meta-brokering solutions (Leal et al. 2009). Specifically, it “creates a meta-level above current resource management solutions by using technologies from the area of the semantic web”, as Leal et al. (2009, p. 4) state. However, this is a centralized solution unadoptable in complex and dynamic systems such as grids and inter-clouds.

The work of In et al. (2004) presents an infrastructure for policy-based scheduling in resource allocation in grid computing. The scheduling model is centralized and uses a hierarchical representation of the most important features of scheduling. Specifically, by assigning resource usage accounts and priorities the method controls the request assignment. This approach provides varying levels of load balancing and resource utilization based on the resource consumers’ requests.

The work of Grosu and Chronopoulos (2004) presents selfish agents in order to achieve a total load-balancing infrastructure. Specifically, the Algorithmic Mechanism Design (AMD) theory is utilized as the specification of payments to agents in a way that results in an environmental equilibrium (Feigenbaum and Shenker 2002). The work is based on a centralized model in which the local dispatcher decides the allocation and payments. Finally, the work concludes with a protocol that implements the mechanism.

Finally, the Bellagio (Auyoung et al. 2004) contains a market base resource allocation system for federated distributed infrastructures. Auyoung et al. (2004, p. 1) suggest that in this system “users identify resources of interest using a resource discovery mechanism, and express preferences in the form of combinatorial auction bids”. A centralized auctioneer who controls the bids for resources coordinates the whole procedure. Typically a bid includes the required resources, the computational processing duration, and an amount of virtual currency.

To conclude, all the aforementioned works aim to a centralized meta-scheduling environment in which a central component is responsible for the management of various local schedulers. The key advantage of this method is that central administration complications like starvation could be easily predicted. In addition, the meta-scheduler assigns jobs constantly to the best possible resource for execution by selecting jobs from the centralized pool list. This is the main reason that the above works claim to have effective performance results.
However, for each centralized meta-scheduler a local system administrator maintains complete control, thus making systems’ dynamic changes unpredictable, e.g. bottleneck of requests and single point failures.

2.4.2 Hierarchical Meta-Scheduling

The hierarchical meta-scheduling scheme shares similarities with the aforementioned centralized scheduling. In this setting jobs are submitted to a central instance of the scheduler as discussed in Subramani et al. (2002) but additionally each host has its own local scheduler with different policies. However, the hierarchy allows meta-brokers to be linked either vertically or horizontally in levels (layers) of accessibility.

An advanced solution of hierarchical scheduling has been presented in Brune et al. (1999) where authors present a geographically distributed high performance computing setting. Its architecture is based on a three layer structure, namely the Computer Center Software (CCS) for scheduling system components; the Resource and Service Description (RSD) for specifying hardware and software components; and the Service Coordination Layer (SCL) for brokering and registering applications. This solution is reliable and scalable as it is hierarchically organized in autonomous “CCS islands” and performs scheduling in space-sharing using deadlines. The authors compare CCS with the Globus meta-computing directory service (MDS) (De Assuncao et al. 2008), which provides a flexible data model, in a theoretical way without presenting any relevant benchmark results. A noteworthy difference is that Globus realizes the environment as a huge virtualized meta-computer in contrast with CCS where resources may be distributed but must be accessible in one domain (Huang 2012).

Another solution is a framework called Sharp for secure distributed resource management, discussed in Fu et al. (2003). The system is based on the barter economy in which exchange must be made using a cryptographically signed object called Resource Ticket (RT). In Sharp, each site runs local schedulers for physical resources. Sharp is a comprehensive architecture, which combines “cryptographically protected authorization, leases, hierarchical resource management, probabilistic resource tickets, and self-certifying decentralized trust management.” (Fu et al. 2003, p. 13)

To conclude, the hierarchical scheduling scheme has not been fully utilized by developers, mainly because this is very similar to the centralized topology. Fundamentally, this has been underlined by Feitelson and Rudolph (1995) who suggest that this solution inclines to a centralized rather than decentralized scheme, as there is one central scheduling instance in which jobs are submitted. In general, both approaches, centralized and decentralized, always
offer remarkable results, and it could be a good practice to be the basis of comparison when developing highly dynamic distributed meta-schedulers for large-scale environments. Besides, various scheduling approaches, Leal et al. (2009) compare their results with a centralized and/or hierarchical solution to present their performance analysis. The next section presents the distributed meta-scheduling topology and a variety of related scheduling approaches.

2.4.3 Decentralized Meta-Scheduling

The distributed meta-scheduling theme originally defines that each resource has a local and a meta-scheduler. Thus jobs are directly submitted to a meta-scheduler that decides which local scheduler to relocate it to. In the simplest of the cases, meta-schedulers query each other at regular intervals so as to collect current load data (Christodoulopooulos et al. 2009), and to find the site with the lowest load for transferring the job. This solution is the more advanced and complex, compared with centralized and hierarchical themes as it is more scalable and flexible. Specifically, the meta-scheduler has an incomplete and instantaneous knowledge of the environment. This partial knowledge based solution is usually related to granularity of the system that defines the measure in which a system is broken down.

In centralized and hierarchical schemes the schedulers have a complete knowledge of the actual resource infrastructure. This includes the number of hosts, number of jobs submitted, the workload of each host, and the topology of the system. Dissimilar, in the distributed theme, this information is partial and the jobs received from the meta-schedulers are assigned to local schedulers of the same or a different host. As all the tasks are submitted locally the distributed scheme allows jobs to be transferred to remote hosts for achieving performance criteria, e.g. improved local resource utilization levels to lead into global load equilibrium.

Distributed meta-scheduling algorithms have been studied for many years. In 1996 the work of Weissman and Grimshaw (1996) proposed a wide-area scheduling system based on a Local Resource Management System (LRMS) and wide-area scheduler (the meta-scheduler). Each member of the site has to instantiate a) a LRMS that manages the local resources and b) the wide-area scheduler (WA) for achieving a global scheduling. Specifically, the WA scheduler contains two interfaces; firstly the scheduling manager to local schedulers and secondly a grid scheduler to remote scheduling managers. The job distribution is based on message passing where nodes send requests for job allocations to all addresses of the static file. The model has been evaluated using the Legion system (Grimshaw et al. 1995) in order to demonstrate the effectiveness of the approach.
Schwiegelshohn and Yahyapour (1999) present a meta-scheduling mechanism called NWIRE (Net-Wide-Resources). The scheduler consists of a MetaManager that is responsible for controlling a set of domains (MetaDomains) and has access to the ResourceManager that represent the LRMS. NWIRE has been evolved over the years to consider several scheduling characteristics including existence of conventional schedulers and resource reservations and trade resources based on the economic mechanism of Ernemann et al. (2002b). This includes remote domains that create offers and send their best combinations back (as messages) to the initial source without any particular order. In this solution, the job allocation is decided by the description of the requests. In general NWIRE offers a high fault tolerance mechanism as the failure of a single trader will not affect the whole procedure.

Anand et al. (1999) present a decentralized dynamic algorithm named Estimated Load Information Scheduling Algorithm (ELISA). The method first estimates the load awaiting service (queue length) at the neighborhood processors and secondly reschedules the loads at the current resource based on these estimations. The aim is to increase the possibilities to gain load balancing by estimation based on updated information after large time intervals. The ELISA basic concept is that at periodic intervals called status exchange interval the processors exchange their queue length and an estimate job arrival rate. The instants in which information exchange takes place, called the exchange epoch that is further subdivided into intervals called estimation intervals. The points of such divisions are called estimation epochs. The results of Anand et al. (1999) have shown that ELISA is an efficient solution for achieving load balancing in large distributed systems. In terms of message distribution, ELISA processes exchange status information (at a periodic interval) that contains the queue length and an estimate of the arrival rate. However, there is no specific method for messaging.

The necessity for coordinated resource management in distributed systems is presented by Daval-Frerot et al. (2000). The work presents a model, namely the federation of distributed resource traders and parallelizing jobs submissions to user-defined services. By coupling several resource providers, the resource trader acts similarly to a meta-scheduler as the intermediate among consumers and providers. Several traders cooperate with each other in order to develop a federation of traders in which local users, clients and resources managed trade for resources. The collaboration of traders happens with the aim of maximizing a trading function. Specifically, the trader contains two interfaces; the first one is the local scheduler and the second is a remote interface to other traders. Within this cooperative setting the traders can negotiate for various parameters, e.g. response time. When a request is
submitted to a trader, then the last one tries to execute it locally. In the case in which the current trader cannot satisfy, the request is transferred to a different trader. The results presented in Daval-Frerot et al. (2000) shows that by using trader federation an improvement in the resolution times could be achieved. However, this method does not present how data consistency is managed, as well as there is no discussion about the actual simulation environment. The message exchanging is based on distribution of offers that are sending from requesters to responders. The requesters collect all the information coming from traders; however the authors do not specify what happened if there are no offers from traders.

The work of Subramani et al. (2002) demonstrates a distributed computing scheduling model. The key idea of the proposed meta-scheduler is to redundantly distribute each job to multiple sites, instead of sending the job to the most lightly loaded. Specifically, when a job is placed in multiple sites the possibility of effective backfilling is higher. The technique measures the average job turnaround time and average job slowdown. The actual algorithm requires minimal data and decides scheduling on current global picture of the system. Briefly, the K-Distributed Model and the K-Dual Queue Model compose the meta-scheduler.

These authors continue that firstly the K-Distributed Model includes jobs that are scheduled to the least loaded sites. Then the job is placed in multiple queues for better backfilling facilitation. Secondly the K-Dual Queue Model incorporates priorities to the local jobs instead of the remote ones. When a job is placed in a remote queue, at the same time it is being placed at the local queue with higher priority for backfilling (easy backfilling), so higher odds to execute in the local place. The results show that the algorithm might be very efficient as the average slowdown is improved in the K-Distributed Model. Similarly the K-Dual Queue scheme performs best for the lightly loaded sites thus, bringing better fairness to those jobs. However, this slightly decreases the overall performance of the system. The authors present a messaging approach that cancels messages in K-1 clusters when one of K multiple requests starts execution. However, authors focus on message requests and they do not present the message reply operation.

Butt et al. (2003) present a model for connecting various Condor work pools which yields to a self-organizing flock of Condors. This work is more focused in the area of resource discovery by using a P2P routing pastry model as suggested by Hentschel et al. (2009). However, the model uses the Condor resource manager to schedule jobs to various idle resources, and invokes the flocking mechanism only in the case in which the machines are busy. Specifically, the scheme compares queue lengths, average pool utilization and resource availability and creates a list of pools. Those pools are characterized as most suitable to least
suitable and stored in a list. The results show that the flocks can reduce the maximum job waiting time in the queue. Butt et al. suggest (p. 3) that “each pool that has resources available sends a message announcing the available resources to all the pools specified in its routing table”. By this way a node that receives a message becomes aware of available resources in the pool. This includes messages sent from all resources to the rest resources of the pool.

Andrade et al. (2003) present a scheduling infrastructure based on the Bag-Of-Tasks (BOT) applications and called OurGrid. OurGrid is a collection of peers constituting a community. Specifically, the system contains the following components; the Swan which is the software system for making possible access to resources from community members, the OGBroker which is the resource consumer brokering system and the OGPeer which is the means to connect OGBroker to OurGrid. When a user submits a request (normally an application) to his OGBroker, the last one sends a request for peers to OGPeers by using a JXTA overlay. Then scheduling happens by the site’s reputation and resource availability. The authors consider a broadcasting solution of messages between resource providers; however providers that could execute a job reply back only in case of availability.

In Lai et al. (2004) authors discuss also a market-based resource allocation system. The scheduling mechanism in this system is based on auctions. Specifically, each resource provider or owner runs an auction for his resources. The meta-schedulers communicate with a Service Location Service (SLS) which contains an index of resource auctioneers. In SLS auctioneers record their status every thirty seconds. The bid for resources is created by the meta-schedulers, which act on behalf of their resources. However, with this solution resources can be under-utilized as meta-schedulers may bid always for a specific set of resources. This concludes to a coordination lacking of the meta-scheduling method. The authors suggest that messages are exchanged among the SLS auctioneers based on an interval and without any particular method.

Shah et al. (2007) suggest two scheduling algorithms, namely Modified ELISA (MELISA) based on Anand et al. (1999) and load balancing on arrival. Both algorithms are based on the distributed scheme of sender-initiated load balancing. Their difference is in the grid scaling as MELISA works better in large-scale systems, and load balancing on arrival works well with small-scale environments. Specifically, MELISA calculates the neighboring nodes load by considering jobs arrival rates, service rate and node loads. However, in contrast with ELISA (Anand et al. 1999) the job transferring decision is based on the comparison of node load and not queue length. To improve MELISA performance, the authors conclude that the load
balancing on arrival method will balance the high job arrival rates. The messaging approach is related to the interval that defines the time in which messages are transferred. In addition, each process exchanges status information with its “buddy” set without any specific method.

The delegated matchmaking (DMM) approach presented by Iosup et al. (2008a) is a novel delegated technique which allows the interconnection of several grids without requiring the operation of a central control point. This occurs by temporarily binding local resources to remote resources. Specifically, in this decentralized approach when a user cannot be satisfied at the local level, then through a delegated matchmaking procedure remote resources are added to the user transparently. The DMM utilizes a hierarchical architecture in which resources in the same level may cooperate with each other. So, by delegating resources and not jobs the DMM aims to minimize the overhead caused by the management of jobs. The results of the simulation show that DMM can have significant performance and administrative advantages. However, this work raises heterogeneity issue questions in large-scale distributed settings. Also, job failures and unmovable loads at the cluster level are not considered. The authors adopt this solution in the Koala centralized meta-scheduler (Mohamed and Epema 2005). The authors suggest that messages are exchanged among components for any case of delegation (e.g. DelegationReject message). Since the negative responses are returned, an increased message overhead could be observed.

Ernemann et al. (2002b) present a model for the InterGrid as a sustainable system. The authors first discuss existing research studies with the aim of creating national and continental Grids. So they suggest that there is a need for new settings that will allow grid to evolve from local to global scale. Specifically, InterGrid suggests interlinking grid islands using peering arrangements. Thus, by providing a flexible and scalable construction a sustainable connection can happen among grids. This happens by the use of InterGrid Gateways (IGGs) which allows a cross collaboration among grids. In detail, IGGs have agreed arrangement among themselves and can perform resource allocation to different grids. In the same line, De Assunsao and Buyya (2009) evaluate the performance analysis of the InterGrid architecture by using conservative backfilling, multiple resource partition, least loaded resource policy and earliest start time policy. Finally, the results show that the average response time has been improved in the aforementioned evaluated scheduling algorithms. However, Leal et al. (2009) suggest that this approach reflects a more economical view when business application support is the primary goal. This algorithm allows reject messages to return in the case that a grid does not have the required slots for allocation.
Leal et al. (2009) present a decentralized model for addressing scheduling issues in federated grids. This solution proposes the utilization of the GridWay, a meta-scheduler to each grid infrastructure of the federated grid. The method is an alternative to the centralized setting. The authors suggest four algorithms that could be executed in the GridWay meta-scheduler, namely; the static objective (SO), the dynamic objective (DO), the static objective and advance scheduling (SO-AS) and the dynamic objective and advance scheduling (DO-AS). Starting with the SO algorithm which aims for a higher throughput, an objective decides the number of jobs to be submitted to a host. The DO is a more complicated approach that determines objectives which are actually processed during the execution time. Finally, both SO-AS and DO-AS share similar functionalities to SO and DO, however with one major difference. Specifically, jobs are advanced scheduled to desired resources without waiting for free nodes. Experimental results presented by the same authors reveal that DO-AS is the best strategy as it outperforms other solutions SO-AS in minimizing makespan times. The last one is completely transparent to the users by not requesting information. Its great advantage is that the method considers past performance requirements when it forecasts new objectives. Thus, authors suggest that this flexible method of DO-AS is fast enough to be used in realistic scheduling. In addition, they assume a complete knowledge setting in grids so they do not present any messaging discussion.

The work of Folling et al. (2009) presents an Evolutionary Fuzzy System approach for identifying situation adaptive and robust algorithms for workload distribution in decentralized grids. The authors suggest a decoupled grid resource management system (GRMS) that decides the delegation of jobs from site to site. Jobs are submitted to the local resource management system (LRMS) as usual; however a submission component intercepts these and forwards them to a local GRMS for further investigation. Folling et al. (2009, p. 35) states that “the decision mechanism is established by using a fuzzy controller system with flexible rule sets that are optimized using evolutionary computation, using a pair-wise training approach and performance metric-based rule base selection”. This happens because in some cases resource utilization, throughput and average response times remain confidential, for example due to resource competitions or security reasons. Therefore, such information is not sharable during the scheduling process. The evaluated results are based on real world data and show that it is possible to exchange policies, which lead to response time and utilization improvements. The authors suggest that enhancement of performance can come from a stable basis for workload distribution.
Rodero et al. (2010) address the problem of broker selection in multiple grid scenarios by describing and evaluated several scheduling techniques. In particular, a system entity, e.g. hosts and virtual organizations, are represented as meta-brokers which might behave as gateways. Every scheduling method discussed in this work consists of the “bestBrokenRank” broker selection policy along with two different variants namely bestBrokerRank_AGGR (AGGR_SIMP and AGGR_CAT) policy and bestBrokerRank_SLOW policy. The first one utilizes the resource information in aggregated forms as input, and the second one utilizes the dynamic performance metric “broker average bounded slowdown”. Authors claim that performance is not penalized significantly as resource information accuracy may be lost as well; the better results come by using the dynamic performance information. However, although the interoperable grid scenarios can improve workload executions and resource utilization, this work did not address issues in matching time with aggregated resource information. The authors do not present the messaging approach.

The work of Wang et al. (2010) discusses the problem of overloading by suggesting an alternative approach of resource selection called bidding. Since there is no global information available in a dynamic environment, e.g. grid and cloud, bidding cannot facilitate the optimum decision. For this reason, a resource selection heuristic method has been proposed in order to minimize the turnaround time in a non-reserved bidding based grid environment. The first heuristic is called random selection and the probability of selection is given by a mathematical formula. The second is the minimum execution time-deterministic and selects resource providers with the minimum execution time. The third is called minimum execution time-probabilistic and the selection of a provider is proportional to the CPU capability. The fourth is the minimum completion time-deterministic, which is similar to the second heuristic with an added characteristic: selection happens according to the waiting time plus the execution time. Finally the fifth is the dissolve-probabilistic and selection of providers is inspired by the way ice cubes dissolve by calculating the proportion of the served workload to the whole workload. By conducting a series of experiments the authors claim that dissolve-probabilistic performs better than the other heuristics. However, this work did not consider important scheduling issues which might affect performance, such as job workload, CPU capability, job execution deadlines, bandwidth and network features and dynamic availability of resources. The authors present that messages are sent to providers based on intervals and received back based on a bidding model. This includes that messages are sent from all to all resources.
Huang et al. (2011) introduce a decentralized dynamic scheduling approach called community aware scheduling algorithm (CASA). The CASA operates as a two-phase scheduling decision and contains a collection of sub-algorithms to facilitate job scheduling across decentralized distributed nodes. In particular, the first called job submission phase finds the proper node from the scope of the overall grid (job distribution) and the dynamic scheduling phase, which aims to iteratively improve scheduling decisions. Its great difference when compared with aforementioned approaches is that it aims for an overall performance improvement, rather than individual host performance boosting. The authors by conducting a series of experiments have shown significant results. First of all, applying CASA in a decentralized scheduling theme could lead to the same amount of executed jobs comparing with the centralized solution. Also, job slowdown and waiting times have been dramatically improved. This happens mostly because the model does not request detailed resource information from the resources. In addition, the authors claim that improvements were also noticed on the scheduling performance including response and waiting time and the messaging overhead. CASA, in contrast with aforementioned algorithms, is based on contacted nodes’ real time responses. However, the authors suggest that further enhancements should be considered to include backfilling methods and shortest job first. Also further experiments can be considered by using different grid workload traces in order to get a better understanding of the improved performance. Finally, the message approach includes request, accept, assign, receipt and inform messages and it focuses on a probability to find a resource. However, this includes a large amount of messages which increases the messaging overhead.

Rodero et al. (2009) present a scheduling strategy based on backfilling called JR-backfilling and a resource selection policy called the SLOW-coordinated policy. The method uses dynamic performance information instead of job requirements and resource characteristics. The overall algorithm aims for the minimization of workload execution time, job waiting time, job response time, average bounded slowdown and to maximize resource utilization. Obtained results show that the JR-backfilling outperforms FCFS and, in addition, SLOW-coordination is better than the traditional matchmaking approaches in terms of workload execution time, etc. However, the FCFS approach is simple compared with dynamic solutions and more results are expected in order to compare the feasibility of the work. Also, the authors suggest that the method performs better in a homogeneous environment rather than a heterogeneous setting.
Xu et al. (2011) presents a job-scheduling algorithm that considers the commercialization and virtualization characteristics of cloud computing based on the Berger Model (Berger et al. 1966). The model suggests distribution justice based on expectation states that study actors and evaluate their behavior. The authors suggest that “the basic idea of distributive justice is that individual in social system can judge its own gained resources to be fair or not through distribution relations comparison between itself and other ordinary person in referential structure” (Xu et al. 2011, p. 420). Due to job scheduling in clouds, two constraints are established aiming for fairness. The first constraint classifies the user jobs (tasks) by quality of service constraints. Then by creating a general expectation function, they control the fairness of the resource during resource selection. The second constraint affects the resource allocation process by defining the resource fairness justice function. The authors have validated their method in a simulation toolkit and results show that the method brings better fairness. However, it is a more economical approach within a single cloud rather than aimed at an inter-cloud setting. In addition, system dynamics have not been considered, as it is parallelized for a large cluster base setting. Also, the heterogeneity of cloud datacenters is assumed, which is hidden from the virtual machines. Finally, the algorithm may be considered as a single cloud scheduler.

To conclude, this section presents various scheduling categories for a wide range of systems. Next the study focuses on the realization of job distributions in large-scale systems by focusing on messaging approaches.

2.5 Job Distribution in Large-Scale Systems
In meta-scheduling, the message solution is a crucial issue as it defines the communication means. Currently, most of the message solutions are focused on the number of packets that are moved among processors in clusters and grid settings. In such cases, the Message Passing Interface (MPI) (ANL 2012) has been introduced as a portable message passing standard. The MPI is related to point-to-point communication and the collective communication approach. The first case (point-to-point) includes that the processor of a node sends a message to another along with some data. MPI processes are independent and they communicate to coordinate a job submission, so messages are sent between two processes. The actual operation includes that one process sends a message to another one that receives it, and then it replies to the sender. A typical rule is that for a point-to-point MPI case the number of messages that are sent and received should match (one receive per send).
Each message contains a number of properties that include the actual data, the data type of each element, the number of elements, a message tag, and the ranks of the source and destination process (Gupta and Vadhiyar 2007). This kind of exchange could occur in synchronous or asynchronous mode. For instance, the synchronous mode includes the sending of complete information while the asynchronous contains data regarding the time that the message left from the first process. In addition, asynchronous allows high dynamic-ness of the system, so it could be applied successfully for cases of variable workloads as detailed by Karatza (2004). Point-to-point has been considered as a flexible method for messaging, however for a large number of processes the sum of messages will be high; a fact that affects the overall system performance. This shortcoming is based on the request-to-response model and includes that for each request a process should always reply.

A different approach is the collective communication that involves the transfer of many processes at a time in order to have coordinated communication of a group of processes. The solution increases the design complexity as it encompasses the synchronization of processes. Nevertheless, it is a more advanced approach whose operations could be applicable for larger scale distributed infrastructures. The collective communication is always synchronous in the sense that collection will not be completed until all MPI nodes reach the same point (Adhianto and Chapman 2007). In general, by using broadcasting called “broadcast call” one node sends a message to all nodes of the group. Steffenel and Jeannot (2007) suggest that the “reduce call” procedure is called by the MPI at the end of the process for collecting information from all nodes’ processors and stores the result on one node. The collective functions follow the basic MPI requirement that indicates that the amount of data sent should match the amount of data received, thus this makes it an inflexible solution.

For the collective communication procedures, a variety of different routines have been introduced to implement different communication patterns (ANL 2012). This includes AllToAll, AllGather, BCast, Scatter and Gather, AllReduce and other functions. Adhianto and Chapman (2007) detail that the AllToAll model allows complete information distribution among all the node processes of a group. Allgather, on the other hand, collects processes and then broadcasts (BCast) to each contacted node. However, as the number of nodes increases, performance is compromised due to congestion of network resources. Finally, the Scatter and Gather method allows collective processes to be distributed in a different process and gathered again back to the originating processor. A different view of messaging could include the network algorithms for minimizing the network bottleneck.
A number of theoretical models have been developed in order to avoid network congestion, as in Gupta and Vadhiyar (2007) however from the perspective of packets that are exchanged among network entities. For example, these authors present the spreading simple algorithm that allows a node to send data to node \((p+i)\) where \(p\) is the process and \(i\) the iteration and, receive data from node \((p-i+N)\) mod \(N\) (where mod \(N\) is the division modulo).

A different approach called the ring/bucket/circular algorithm is presented in Chan et al. (2006). Specifically, at each iteration \(i\) a process \(p\) sends data to a node with an index \((p-i+1+N)\) mod \(N\) to the right neighbour in the list. The recursive doubling algorithm as discussed by Adhianto and Chapman (2007), requires less time as the number of total transfers is reduced. MPI makes use of the MPICH (ANL 2012) (MPI Chameleon) to recursively reduce the number of messages by utilizing a criterion; when the number of processes is a power of 2 it uses recursive doubling for small message sizes. Next, for the rest of the messages (large size) it uses the ring algorithm to achieve message dissemination. However, this solution is aimed at small-scale parallel computing systems. Steffenel and Jeannot (2007) propose that most of these algorithms have been designed for homogeneous and tightly coupled systems.

In the case of highly heterogeneous and decoupled settings (e.g. grids and inter-clouds) solutions divide the network into hierarchies. Cotronis (2004) demonstrates the MPICH-G2 algorithms to gather data up the hierarchy using recursive doubling and then broadcasts these data by binomial broadcast (according to a probability factor). Similarly, MagPie presented by Kielmann et al. (2001) includes a three-stage algorithm first to AllGather data at coordinators; second to gather data among coordinators, and third to broadcast data by coordinators again using a binomial broadcast. The major drawback is that it follows static network hierarchical schemes in modeling decisions. In addition, data transmission is repeated at coordinators thus keeping bandwidth values in high layers of hierarchy at low levels (Gupta and Vadhiyar 2007).

Gupta and Vadhiyar (2007) illustrate an algorithm that is network topology aware and adaptive to various network loads. This solution follows the transient clustering of nodes based on network characteristics. In contrast, Steffenel and Jeannot (2007) focus on an alternative algorithm for minimizing the number of steps through a wide-area network. They also claim that the reduction has a direct impact on the performance modeling by minimizing the factors that directly interfere with the wide-area communication. Although efficient algorithms have been developed for specific networks, a generic model for heterogeneous and decoupled nodes has been proven to be complex to design. This is because of the
dynamic nature of the resources. An important requirement to be considered is the time-centric information processing and the levels of elasticity and scalability that are required for the jobs. Figure 3 demonstrates the relationships between the various algorithms and approaches discussed.

Today, large-scale systems, e.g. inter-clouds, utilize the notion of the resource manager in order to achieve dynamic capabilities. Specifically Auyoung et al. (2004) present meta-scheduling solutions by allowing communication to be implemented in terms of message exchanges. However, most of these approaches are aimed at decentralized, heterogeneous, decoupled and transparent systems, where entities exchange messages in an AllToAll fashion.

To conclude, in this section the study has highlighted the messaging solutions of the meta-scheduling approaches.

2.6 Discussion of meta-scheduling approaches for federated clouds and inter-cloud

The area of meta-scheduling in inter-cloud and cloud federations has been discussed by literature mainly from a conceptual point. The work of Bernstien et al. 2009 presented scenarios for multi-cloud systems to allow interoperation. They focus on an approach that introduces in interoperability solutions for low-level functionalities of clouds from a sub-contracting approach. In addition, they raise questions on messaging and synchronization by
suggesting that broker mechanisms could act as meta-schedulers for effective resource management. However, they do not extend their study in terms of experimentation. In Buyya et al. (2010), authors present a market-oriented approach that offers inter-cloud by suggesting the need for a mediator that controls communication and interactions. They define their model in a theoretical manner, and underline the need for cloud brokers. They present a basic experimental analysis using the CloudSim simulation toolkit. This study considers this as a fundamental work in the area and uses the toolkit in order to detail the benchmark analysis.

Other works, e.g. Rochwerger et al. (2009) discussed the federated infrastructure as a service cloud management model. The perspective of the work is from the Grid resource management layer. This highlights that grid resources could be shared in order so users to lease available services. Their work does not clarify the usage of meta-scheduling characteristics for job distributions, as they assume that Grids already offer the required operations.

Finally Simarro et al. (2011) demonstrate the optimization of VM deployments in multi-cloud settings. Their approach is focused on an economical model for VM deployment rather than on the part of scheduling, however they assume that there is a mechanism for allowing distribution of VMs among clouds. At last, the work of Marshal et al. (2010) proposes a model for meta-scheduling between utilized resources in an inter-cloud that is comprised from grids and clouds. However they focus on the perspective of the grid resources that could be seen as IaaS rather than the cloud setting. Similar to Rochwerger et al. (2009) authors assume that there the Grid Resource Management System (GRMS) offers the interoperation among various nodes.

Thus, the study concludes that the works that are related with inter-clouds either indicate the utilization of meta-scheduling as a way for inter-clouds communication or focus on other areas such as economical brokers etc. Works that are related with grids utilize the already defined meta-scheduling from the GRMS. Also, other works that utilize meta-scheduling are assume their functionality (e.g. the grid paradigm offers the meta-schedulers in Rochwerger et al. 2009) and they do not discuss a particular architecture. Thus, to the best of this study knowledge, there are no other works related directly with performance and how it is affected by the adoption of meta-scheduling paradigm in inter-clouds. Next the focus is on the analysis of meta-scheduling approaches and their key characteristics based on the literature review approaches.
2.7 Discussion of Meta-Scheduling Approaches

To identify the key concepts of each approach, the study sums up the analysis of the approaches identified in the literature review by focusing on centralized and decentralized solutions. These are summarized in tables 1, 2 and 3 presented in Appendix A. The decentralized or distributed scheme aims at addressing most of the requirements that have been neglected by centralized solutions (Bessis et al. 2012b). These approaches include crucial characteristics towards wider-scheduling decisions in inter-collaborated environments (Sotiriadis et al. 2011), as presented below.

- **Heterogeneous pool of resources** is recognized as one of the crucial subjects in various cases, e.g. Rodero et al. (2009), Rodero et al. (2010), Andrade et al. (2003), Frerot et al. (2000) and Leal et al. (2009). However, the literature study shows that tentative results from these works confirm a low appreciation of the heterogeneity issue during experimentation. However, older works either do not include heterogeneity or assume homogeneity in their requirements scenarios, e.g. Weissman and Grishaw (1996), Schwiegelsohn and Yahyapour (1999).

- **Interoperability** between local and meta-schedulers is subject to the requirements posed by the desired scenario. In any case it is considered in various works by either supporting scheduling autonomy, as in Weissman and Grishaw (1996), temporary binding amongst resources and jobs Iosup et al. (2008a), or by supporting fault tolerance mechanisms as in Pinchak et al. (2002).

- **Dynamic-ness** of the environment is a critical property when developing an interoperable meta-scheduler. Various works try to solve meta-scheduling issues derived from the unpredictability of a dynamic changing environment as in Leal et al. (2009) that considers past performance requirements for forecasting new objectives. Similarly, Huang et al. (2011) present a meta-scheduling tactic that does not expose internal node information and is based on nodes’ real time responses. Equally, work in Rodero et al. (2009) uses dynamic performance information instead of job requirements and resource characteristics. In contrast with those solutions, non-dynamic approaches such as Grosu and Chronopoulos (2004) and Rodero et al. (2010) assume a steady-state setting during simulation. In the last one, the authors suggest a delegated matchmaking procedure in which resources are matched temporarily to remote resources.

- **Geographical distribution** among different pools of resources is considered in most of the works, as they all include meta-scheduling for grid environments. Specifically,
Folling et al. (2009), Rodero et al. (2009), Schwiegelsohn and Yahyapour (1999) and
Huang (2012) present scheduling strategies for geographically distributed resource
pools, e.g. grid virtual organizations. Normally, this issue is part of the overall objective
of wide distribution of jobs.

• *Inter-collaboration for resource sharing and/or jobs* amongst same and/or different
infrastructures, e.g. grid virtual organizations and HPC, grids and clouds is usually
neglected as the complexity in such settings is rises exponentially mainly because of the
additional requirements. Specifically, work in De Assuncao (2008) and De Assuncao
(2009) aim at an inter-grid of interlinking grid collaborated islands using peering
arrangements. Work in Huang et al. (2011) presents a more advanced meta-scheduling
algorithm for job scheduling among distributed grid nodes. Similarly, the works of
Wang et al. (2010) and Rodero et al. (2010) aim at an inter-collaborated theme.

• Load-balancing in different settings has been identified in various works, such as
Christodouloupolos et al. (2009), Rodero et al. (2010) and Huang (2012). Specifically,
the increasing load balancing probability improves the performance of the overall
environment. For example, in Anand et al. (1999) the algorithm estimates the queue
length of neighboring nodes and then performs a rescheduling process. Likewise, in
Shah et al. (2007) the approach calculates the neighboring nodes’ load by considering
job arrival rate, service rates and node loads. In this case, jobs are transferred based on
the comparison of node load and not queue length.

• *Resource allocation mechanisms* in decentralized solutions have been driven by
different scenarios. In Butt et al. (2003) the method connects various Condor pools
happens by site reputation and resource availability. A market-based resource allocation
model is discussed in Lai et al. (2004) in which an auction list of resources is
maintained by meta-schedulers who act on behalf of their resources. Past performance
information in the form of historical data is utilized by Leal et al. (2009) to achieve a
resource allocation mechanism. To conclude, various mechanisms exist in the literature
always based on the requirements of the specific scenario. For example the work of Xu
et al. (2011) presents a scheduling algorithm which considers the commercialization
and virtualization characteristics of cloud computing based on the Berger Model, thus it
is more an economically driven setting within a single cloud rather an inter-cooperative
intensive mechanism. Due to job scheduling in clouds two constraints are established aiming at fairness.

- **Rescheduling** concept and *advance reservation* mechanism are commonly used in various cases for iteratively improving the performance of the scheduling process. Specifically, Huang (2012) claims that during a rescheduling phase a notable improvement has been observed in scheduling performance. Equally, Leal et al. (2009) suggests that by utilizing an advance reservation mechanism based on previous work performance measures, a significant enhancement in performance has also been observed. However, the authors suggest that the overhead during training may be increased significantly, especially in the case in which a large-scale job input arrives in the scheduler.

- Utilization of *historical data* is not considered by the majority of the works, although it could contain a future value for enhancing the rescheduling and advance reservation process. However, the work of Leal et al. (2009) tries to achieve a similar strategy in which the method considers past performance requirements and might forecast new objectives. However, this is only adoptable for specific information systems as it requires a training mechanism for forecasting performance. Similarly, in Huang et al. (2011) the method considers past job delegation records during the rescheduling process.

- **Security** issues, similarly to the decentralized and hierarchical meta-scheduling topology, are usually ignored and resource managers are assumed to do the specific work. Typically, this issue is out-of-scope of the meta-scheduling theme. In the decentralized meta-scheduling the security problem includes issues like information exposure during meta-scheduler collaboration.

- **Neighboring collaboration** is mainly the development of various size cliques that share commonalities in job requirements, while at the same time they could belong or not to the same administration domain. Examples are the Condor pools in Butt et al. (2003), and the grid islands in De Assuncao et al. (2008) and De Assuncao et al. (2009). Both solutions could offer sustainable connections among different communities, however, unfairness among resources could lead to starvation and the dynamics could affect certain connections.

- **Coupling** of specific jobs to resources could lead to a temporarily improved performance setting as presented in Iosup et al. (2008a), however as mentioned
previously, dynamics could affect the coupling relationships. A solution to this problem could be advance reservation mechanisms for coupling jobs to resources as in Leal et al. (2009), or local-to-remote resources as in Iosup et al. (2008a) on a temporary basis to offer a momentary boost of performance. Decoupling, on the other hand, decides the delegation of jobs from site to site without connecting resources. Examples are the work of Folling et al. (2009) in which jobs are submitted as normal from meta- to local scheduler, however a submission component redirects to a global resource manager for further inspection, using an evolutionary computation method to optimize workload exchanging. Similarly, Rodero et al. (2010) presents a policy that considers dynamic performance metrics as detailed by Grosy and Chronopoulos (2004) based on backfilling that uses dynamic performance information processing. Finally, Huang et al. (2011) performs scheduling of jobs based on dynamic real time node responses. Those characteristics of decentralized meta-scheduling approaches are summarized in the list below and could lead to the identification of relevant concerns to each study’s specific scenario, e.g. inter-cloud. It should be noted that the specific characteristics are derived from the cross-evaluation of various literature works.

- **Message exchanging** is considered as a key requirement by most of the decentralized approaches, as in Subramani et al. (2002), Butt et al. (2003), Andrade et al. (2003), Lai et al. (2004), Mohamed and Epema (2005) and Leal et al. (2009). However, most of the works do not detail the whole request-response procedure. For example, Mohamed and Epema (2005) suggest that messages are exchanged among components for any case of delegation (e.g. DelegationReject messages). Since the negative responses are returned an increased message overhead could be observed. Similarly, Leal et al. (2009) suggests an algorithm that allows reject messages to return in the case that a grid does not have the required slots for allocation. Butt et al. (2003) suggest that a node that receives a message becomes aware of available resources in the pool. This includes messages that are sent from all resources to the rest of the resources of the pool. In contrast, Andrade et al. (2003) considers a broadcasting solution of messages where a resource that could execute a job does not reply back.

Figure 4 demonstrates a mapping of various approaches, including centralized (and hierarchical) and decentralized, based on the discussion of extracted characteristics. It shows topologies that are plotted to various characteristics, which in turn are mapped to scheduling approaches. This includes two layers, the centralized and decentralized. Each centralized and
A decentralized approach is mapped to a performance requirement as derived from the discussion in sections 2.7.1 and 2.7.2.

An inter-cloud has been defined as a large-scale and decentralized resource management setting, so the focus is on the decentralized layer. Figure 5 shows that layer and includes the approaches and the performance evaluation characteristics. So, through the literature presentation, analysis and transformation process, the study concludes that the important characteristics are heterogeneity, load balancing, message exchanging, dynamic-ness, interoperability, geographical distribution, neighboring collaboration, inter-collaboration, decoupling, re-scheduling and use of historical data.

Figure 4: Mapping of reviewed approaches to their extracted characteristics used for performance evaluation (key: numbers denote summarised reference tables 1, 2 and 3 in Appendix A)
These characteristics are summarized as follows.

- **Heterogeneity** implies that different clouds could have diverse architectural design. This includes different types of computational units and submission of heterogeneous requests for services. Heterogeneity is considered as a key requirement for the proposed model.

- **Interoperability** includes inter-collaboration for resource sharing and/or jobs in order to present an augmented setting. By default, the inter-cloud approach requires definition of an interoperable and inter-cooperative setting. Similarly **geographical distribution** is considered as part of the interoperability to exchange services among remote clouds. In addition interoperability increases **service elasticity**. This is related to the capacity of
the cloud and it is considered as a key requirement for such settings (Amazon Elastic Compute Cloud – Amazon EC2). The new model expects to expand these capabilities by allowing a wide service dissemination environment in a multi-cloud cooperative infrastructure.

- **Dynamic-ness** means a changing of the system status over time. This is a key requirement as the system will be designed to adapt to unforeseen situations. **Load-balancing** refers to the distribution of workloads by assigning jobs to dynamically changing capable clouds for execution. The proposed model develops a dynamic workload management policy in order to control such behavior.

- **Resource allocation management** includes assignment of tasks to jobs by methodically allocating resources. The study develops policies to regulate the low-level resource sharing. This includes the **rescheduling** concept and **advance reservation** policies for providing a realistic case scenario.

- **Utilization of past service experiences (historical data)** refers to the utilization of past service experiences for adding future value. The proposed model considers historical data in the form of job execution to assist the overall meta-scheduling process. In addition, **neighboring collaboration** is formed according to such data.

- **Decoupling** means the detachment of users from resources or resources from interconnected ones. The proposed model defines a decentralized and decoupled solution wherein users are able to enter and select resources according to different contexts dynamically.

- **Message exchanging** refers to an adaptive way of information exchanging. The proposed model defines a new algorithmic model to allow sophisticated message exchanging among resources.

### 2.8 Summary

To conclude, this chapter presented a detailed analysis of meta-scheduling algorithm taxonomies and topologies according to the current literature approaches. The characteristics are evaluated in order to identify the key requirements. The next section introduces the inter-cloud meta-scheduling framework that includes a set of algorithms to control the job distribution in inter-clouds.
Chapter 3: Modeling the Inter-Cloud Meta-Scheduling (ICMS) Framework

3.1 Outline

This chapter presents the Inter-Cloud Meta-Scheduling (ICMS) framework. This study names the ICMS meta-scheduler as a meta-broker. The model allows distribution of user requests (jobs) between different clouds in order to a) effectively distribute the service by ensuring that all jobs are executed, b) respect the service level agreements (SLAs) of users and clouds and c) optimize the performance of specified cloud metrics.

3.2 The ICMS Topology

The ICMS model is based on a meta-brokering solution for job distribution (Sotiriadis et al. 2013b). A user interacts with the broker to request service execution (one or many jobs). The broker acts on behalf of the user and requests specific resources from the cloud system (datacenter). The architecture of the ICMS is based on the cloud service exchanging. Each time a user requests from a cloud meta-broker the request is forwarded to a local-broker that checks for required resources (based on user SLAs). Figure 6 illustrates the ICMS service exchanging between two clouds. The key entities of the ICMS are the users (represented by jobs), the low-level infrastructure (datacenter and hosts) and the resource management components (local and meta-brokers).

Figure 6: The decentralized inter-cloud service request model

The **datacenter** is the core of the cloud system that includes physical resources. In addition, it handles host, VM allocation and LRMS. In advance, it offers functionalities to monitor the performance of executed jobs and generates the debts of users. A **host** represents the physical machine of the datacenter. Each host represents the computational units (e.g. CPU, number of cores, RAM, storage and bandwidth) and is available for virtualization. The
*local-broker* interacts with the datacenter for acquiring resources. The main functionality of this component is to hide the low level operations of the cloud from the actual user (that is represented by the *meta-broker*). The latter accepts multiple user submissions that are forwarded into the local-broker for resource availability (Sotiriadis et al. 2012b).

The *hypervisor* is also a major component that controls service execution and resource allocation. These include VM deployment and local resource management system (LRMS). The ICMS includes a set of algorithms for managing optimal schemes and job distribution in inter-cloud systems. The meta-broker facilitates a job distribution approach by decoupling users and cloud providers. The vision is based on the fundamental concept of the Internet network that allows various intranets to interconnect with each other. This permits clients of various providers to access a wide range of resources from different providers.

ICMS is based on a decentralized solution so a basic assumption is that various meta-brokers are distributed in different geographical locations and act similar to meta-scheduling distributed systems (Sotiriadis et al. 2012c). Let us assume that there is one inter-cloud composed of a number of interoperable sub-clouds. Individually, clouds include a number of datacenters. Each datacenter contains a number of physical machines called hosts. In addition, each datacenter generates a number of local-brokers. Finally, each host generates a number of VMs based on a VM deployment strategy.

Each local-broker generates a meta-broker that represents the cloud interface. The ICMS default configuration is that one cloud includes one local- and on meta-broker; however this could be altered with regards to the specific experiment. A user requests for a job(s) that contains a requirements specification. Then, it passes the request to the meta-broker that searches for local-availability. If this fails, it distributes the request to interconnected meta-brokers from a personalized catalogue of addresses called the meta-registry. The distribution needs to meet the SLA specification that is generated from the user. The way in which requests are distributed among meta-brokers is defined in a message exchanging optimal scheme. Jobs are sent from meta-brokers to local-brokers and then to the cloud datacenter for requesting resource availability. So each job is assigned to a virtual machine that has been generated in a remote host. To demonstrate this case, figure 7 shows an example case of three clouds (clouda, cloudb, cloudc) that are sub-clouds of the ICMS.
Figure 7: The cloud service request distribution

User user$_1$ requests for resources (job$_a$) by establishing connection with cloud$_a$. Cloud cloud$_a$ assigns to the user local-broker$_a$ and meta-broker$_a$. Local resource management is executed by local-broker$_a$ and the external procedures (e.g. job distribution to remote sites) are handled by meta-broker$_a$. Each meta-broker has a meta-registry that is an index of public meta-brokers to receive requests and send responses for resource availability (in this case these are meta-broker$_a$ and meta-broker$_b$). The study assumes that the meta-registry is formed among common agreed clouds in a previous stage. Every time a new cloud enters the inter-cloud partnership it is recorded in the meta-registry of one or more clouds.

Since ICMS is decentralized, the assumption is that a cloud meta-broker has incomplete access to other meta-brokers. During the service life-cycle, a request is sent to local resources from meta-broker$_a$ to local-broker$_a$. If the SLA is matched and resources are available, the local cloud executes the request. In a different case, the request is send to meta-broker$_b$ and meta-broker$_c$ respectively. Each sends the request to their local-brokers (local-broker$_b$ and local-broker$_c$). The decision-making process happened by the cloud local-broker and hypervisor. The latter manages the following three key issues:
a) The management of the decentralised job distribution based on a novel message exchanging model.
b) The management of VM deployment that defines static and dynamic VM generation features.
c) The LRMS of local clouds that include the local-schedulers of the different sub-clouds.

3.3 Architecture of ICMS

The ICMS includes a layered structure as presented in figure 8. These are the service submission, the distributed resource and the local resource management. In layer 1, the standard topology includes users (represented as nodes) that forward requests to layer 2 where a user could have one to many submissions). Layer 2 generates a random topology based on profiles of other distributed meta-brokers (represented as nodes) that communicate in order to exchange jobs. Layer 3 contains a standard topology that shows the formation of low-level infrastructure that includes local-brokers, datacenters, hypervisors, hosts and VMs. This layer also includes the LRMS schedulers that implement the scheduling of the clouds.

Figure 8: The ICMS layered structure

The layers include the key elements of the service life-cycle, namely plan, deliver and operation. The service submission management layer is responsible to create the job configuration by translating user requirements to system specification. The output is in a form that is recognized by the inter-cloud components. The distributed resource management layer collects job submissions and descriptions, extracts information regarding performance criteria (e.g. job size) and forwards it to the appropriate entity. This could be either a local resource queue or a remote meta-broker to further distribute the job. The local resource management layer offers the job performance and execution environment. Here, jobs are forwarded to the lowest level of the infrastructure into VMs. The whole ICMS is based on the modular optimal schemes that realize the layered structure and consider the dynamic requirements of the previous chapter. Figure 9 illustrates the structure that includes the four modules of the ICMS conceptual architecture, namely service request, distribution, availability and allocation.
The service request module includes the user specification and job formation as in layer 1. The service distribution module contains the message distribution, the meta-brokering and the SLA job description as in layer 2. The service availability module contains the SLA matchmaking, dynamic workload and LRMS as in layer 3. Finally, the service allocation includes the hypervisor host allocation, VM allocation and dynamic queuing as shown in layer 4.

To demonstrate the interactions among ICMS entities, figure 10 shows the user submission and the local cloud job execution. Figure 11 shows the meta-registry and the distribution request. Specifically, in figure 10 a user submits a request to a responder meta-broker that forwards it to its local-broker. The last one executes the dynamic workload optimal scheme to decide the capacity of current available resources. In case of unavailability the request is sent back to the meta-broker which executes a further distribution (figure 11). Otherwise, the job is forwarded into the datacenter that utilizes the hypervisor in order to generate VMs.

Figure 11 illustrates the distribution of jobs in an inter-cloud. In case of non-availability, the requester meta-broker collects addresses from a meta-registry and forwards the request to other meta-brokers. Each one follows the same procedure as in figure 10 to allocate jobs in low level-infrastructure. Finally, figure 12 demonstrates the messaging and job distribution between requester and responder meta-brokers. Initially, a request is sent to the meta-broker (in this case three meta-brokers named meta-broker_k, meta-broker_l and meta-broker_m) to request execution capability. In case of availability, responders send a message back. The next section presents the algorithms of the ICMS.
Figure 10: ICMS: Local submission and execution
Figure 11: ICMS: Meta-registry and distribution request
Figure 12: Inter-cloud distribution request and response model
In ICMS each resource has a component named as a distributed resource manager that assigns jobs to local resources for allocating computational units. The jobs are exchanged in terms of messages that are sent and received among inter-connected nodes in order to find capable resources for execution. The job distribution is based upon a time-centric solution; where a decentralized topology of distributed resource management components allows extensive message requests (messages sent) however during regular time intervals. As shown in figures 10, 11, 12 the following steps demonstrate these interactions.

**Step 1:** The job distribution starts when a number of jobs are submitted to an entity that becomes the requester. Each job contains a set of requirements that are organized as properties regarding time intervals (e.g. waiting time, interval, etc.) and computational units (CPU, memory, bandwidth, etc.).

**Step 2:** For each job received the requester stores it in a list. Each list row has a message with key characteristics including the deadline and the job length as a measurement for calculating resource availability on remote resources.

**Step 3:** The requester defines the deadline, which is related to an interval limit and the size of the list (e.g. for large lists the deadline could be higher as the time needed to transfer is higher). This also defines the cost of communication among entities. So a small deadline could result in a small number of job submissions, while a large one could lead to heavy submission.

**Step 4:** The requester collects addresses of inter-connected nodes from a catalogue. These nodes will become the responders in communication. The study assumes that these are stored in a local profile.

**Step 5:** The requester sends the list with jobs as a message along with data requirements (e.g. deadline, job length). In addition, the message includes the ranking criteria (e.g. turnaround, energy consumption level); so all the responders will use the same classification levels for fair selection. It should be noted that identifications and tags define each message. During communication, the tags are set to unique values in order to characterize the group of messages.

**Step 6:** The responder collects a single request (i.e. the list along with data) and performs an internal resource availability operation according to the ranking criteria. Then it classifies jobs and generates a new temporary list.

**Step 7:** Each job is ranked according to a function and a decision is taken with regards to job availability in requester resources.
Step 7a: In case of full job availability (each job of the list can be executed locally) the responder generates a list with jobs.

Step 7b: In case of non-availability (e.g. responder cannot execute all or few of the jobs contained in the list) a further job distribution occurs, however in an updated time frame. This will loop steps 1 to 6.

Step 7c: In case of non-availability because of an empty responder profile (e.g. resource is the actual terminal) the message is finished instantly. Therefore, responses are not sent back.

Step 8: The new list is created with jobs and rankings in a descending performance order of rankings. This forms the criteria for selecting jobs at the next resource management level. In the case that the responder acknowledges that the job(s) will be executed on a remote machine, the responder re-directs messages to interconnected nodes. All messages are assigned with updated time deadlines.

Step 9: The lists are collected from the requester that compares and classifies jobs by using the desired ranking criteria. This includes unique jobs that are now assigned with an identification of the selected resource for allocation. The decision defines whether a remote resource will be accepted as the host for job execution or not.

Step 10: The procedure ends and each job is sent to the local or remote resource.

Figure 13: The interactions of meta-brokers on job formation and distribution
Figure 13 demonstrates these interactions. Next, the study presents the ICMS levels of job distribution. These are a) the intra-level that defines the accessible meta-brokers, and b) the inter-level that defines the remote meta-brokers.

3.3.1 The Intra-level Job Distribution in ICMS

This is the first level of job distribution. The study assumes that a central entity collects job requests in the form of messages from all other sources (e.g. a user). Each job request contains a list of requirements. To identify resource availability for job allocation and execution, the central entity must send job messages to any inter-connected entity. The latter entities collect the job messages, check for availability and in the case where more than one job requests are contained in the same message, they rank (in terms of preference) each job contained in the message list. The ranking list is sent back to the central entity if there is resource availability. In the case where no availability exists in a remote entity the process is terminated instantly and responses are not sent back. In this way the study reduces the number of messages sent back with the central entity receiving ranked positive responses only.

The central entity sends jobs by a) waiting for an interval call (a time frame to elapse) or b) reaching the number of jobs in the list. Collected jobs are ranked according to a performance criterion and jobs are ready for allocation to different resources. Figure 13 shows the Centralized Message Phase (CMP) and the interactions of a central entity with three responders. The study characterizes intra-level job distribution as acting similarly to a centralized computing system where the nodes have complete knowledge of all resources. This includes that jobs are about to be executed only from the inter-connected resources. In other words, decision-making becomes the centralized component for identifying resources and planning resource management.

![Diagram](Figure 14: The Centralized Message Phase)
Entity\(_{\text{Req}}\) (the central entity) forms a list that is forwarded to entities Entity\(_{\text{Res}}\) (remote inter-connected entities). Then it assigns a tag and a value (e.g. \(\text{tag}=\text{u}\)) that is the same for each submission. Inter-connected entities (e.g. Entity\(_{\text{Res}}\)) receive a message with the specific tag and perform an availability check for each of the jobs in the list. This includes the ranking of jobs so a new list is formed containing the jobs in descending of performance measures (e.g. highest performance rate is placed first). In the case of non-availability (e.g. the list is empty) the request is terminated, thus a response is not sent back. In any other case Entity\(_{\text{Res}}\) reforms the list and sends back its response. Entity\(_{\text{Req}}\) collects jobs by identifying the tag value (\(\text{tag}=\text{u}\)) and performs the global ranking (based on different performance functions, e.g. energy cost). Finally, jobs are assigned to inter-connected Entity\(_{\text{Res}}\) using a different tag (tag=\(\text{p}\)) denoting the next resource management step (job assignment and allocation).

### 3.3.2 The Inter-level Job Distribution in ICMS

The second phase is triggered when messages during intra-level job distribution could not lead to the central Entity\(_{\text{Req}}\) assigning all jobs to any inter-connected Entity\(_{\text{Res}}\). In this case, messages are forwarded to a second level of remote resources. These are resources that are inter-connected with the initial requesters. Figure 14 shows the interactions between the two levels of entity. Specifically, the first level is the entities that are directly related to the requester (centralization), and the second level defines the possible remote sites (decentralizations).

![Diagram showing the Inter-level Job Distribution in ICMS](image)

**Figure 15: The Decentralized Message Phase**

The Decentralized Message Phase (DMP) details the distribution of jobs based on contacted entities’ decision-making processes. This implies that in the case of non-availability of first contacted entities, a request could be further distributed to other remote resources. By assuming that there could be different levels of responding nodes, the possibility for further dissemination is increased. This is also based on the fact that entities could be part of different virtual organizations or collaborated groups. In general, the decentralization offers a variety of advantages such as interoperability and heterogeneity.
management along with increased resource availability (elasticity factor). However, this involves complex topologies of entities. Nevertheless, the use of timing and timestamps allows control of the job spreading.

In intra-level job distribution $\text{Entity}_{b,\text{Res}}$ is transformed to an $\text{Entity}_{b,\text{Req}}$ and forwards a request to its inter-connected resources (e.g. $\text{Entity}_{c,\text{Res}}$) that is initially unknown and unreachable by the $\text{Entity}_{a,\text{Req}}$. In this way the study ensures that jobs are forwarded in terms of messages that are exchanged in a total decentralized approach. It should be noted that each message has a set of intervals. When the initially defined time frame has elapsed an instant termination of message is denote. Then a list of jobs is distributed to the first level of entities, as happened with intra-level job distribution. If availability exists, then the updated list is sent back to requester using the defined tag (tag=$u$). In the case of non-availability of job execution, the contacted entity ($\text{Entity}_{b,\text{Res}}$) forwards the request to remote resources (e.g. $\text{Entity}_{b,\text{Res}}$) and sets a new tag (tag=$y$).

At this time instance the procedure is forwarded to the remote entity ($\text{Entity}_{b,\text{Res}}$) that executes initially the intra-level to identify local resources, and the inter-level in the case of non-availability. In a similar vein, the same procedure is followed until the end of the initially defined interval. For each further distribution the study sets a new interval that is lower than that initially requested. This is because it takes into account the communication delays and the time needed for decision making at the first level of entity. During exchanging the rankings, criteria are configured to the same performance measures of the requester in order to have a fair resource selection strategy. Next the study presents the algorithms that implement the job distribution in ICMS.

### 3.4 The ICMS Algorithmic Structure

The ICMS is formed by a total of four algorithms, namely a) service request, b) service distribution, c) service availability, and d) service allocation. In an inter-cloud setting the algorithms interact with each other in order to find a resource to execute jobs. Each algorithm accesses different operational issues in order to control the messages, the VM management and the LRMS. Next the study presents the algorithms, namely Service-Request, Service-Distribution, Service-Availability, and Service-Allocation.

#### 3.4.1 The Service-Request Algorithm

The algorithmic pseudo-code 1 demonstrates the job (service) formation and request according to a user specification job.
**Algorithm 1: Service-Request**

Require:  
- user_name: user identification or name  
- user_OS: user operating systems  
- user_platform: user desired platform (Intel)  
- user_memory: user RAM memory  
- user_cores: user desired cores number  
- user_CPU_speed: user CPU capacity  
- user_H/D_controller: user controller (e.g. CDROM)  
- user_storage: user storage capacity  
- user_BW: user bandwidth  
- user_spec: user specification on software  
- user_instr: user instructions  
- user_CPI: user cycles per instructions  
- user_hours: user usage duration  
- user_deadline: user scheduling deadline  
- user_pri_level: user priority level  
- user_delay: user delay value  
- user_jobs: user total jobs  
- user_cloud: user cloud selections  
- user_profile: user profile specs  
- user_clock: user required power  
- current: user dynamic event data  
- job\(_i\): user job  
- meta-broker\(_n\): user linked meta-broker  
- monitor_trace: monitor log and traces  
- tag: user tag  
- added_delay: increasing delay figure

Methods:  
- update_user_profile (requirement(s)): updates the require parameter  
- send(entity, event data): send request and data  
- acquire(user, meta-broker): assign user a meta-broker  
- monitor(job, delay, user, MIPS): monitor current data  
- SLA(specification, profile): the SLA configuration of user

1: user_profile ← (user_name, user_OS, user_platform, user_memory, user_cores,  
user_CPU_speed, user_H/D_controller, user_storage, user_BW, user_spec, user_instr,  
user_CPI, user_hours, user_deadline, user_pri_level, user_delay, user_jobs, user_cloud,  
user_profile)  
2: user_CPU_speed ← user_CPU_speed * user_cores  
3: update_user_profile (CPU_clock)  
4: MIPS ← user_CPU_speed / user_CPI * 1/10^6  
5: for all user_jobs  
6: current ← job\(_i\), user_delay, user_spec, meta-broker\(_n\), user_name  
7: acquire (user_name, meta-broker)  
8: added_delay = added_delay + user_delay  
9: SLA (user_spec, profile)  
10: send (meta-broker, added_delay, current, SLA, tag)  
11: monitor (job, user_delay, user_name, MIPS)  
12: end for
Specifically, the Service-Request algorithm (2) forms the user profile, creates the SLA according to specific job and assigns a meta-broker to the user (that represents the user interface). Furthermore, the algorithm calculates the required CPU speed in accordance to the CPU cores and the needed MIPS as a preliminary performance measure. Finally, for each of the user jobs the algorithm sends a request to the meta-broker (each one after the other by allowing a delay to pass) while the meta-broker acts reactively by instantiating the Service-Distribution algorithm. During the whole service life-cycle the monitor component keeps a log of job scheduling traces for future usage.

3.4.2 The Service-Distribution Algorithm

The Service-Distribution algorithm represents the job request exchanging among meta-brokers for large scale scheduling. The aim is to request for capacity and competency of inter-connected nodes (other meta-brokers) in executing certain jobs (user profiles and data that are sent by the Service-Request algorithm) that cannot be executed locally. In a different case (local execution) the meta-broker forwards the job to the low-level local cloud infrastructure. In addition, the decentralized and incomplete knowledge of meta-brokers makes the solution more flexible. The algorithmic pseudo-code 2 demonstrates this procedure.

<table>
<thead>
<tr>
<th>Algorithm 2: Service-Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Require:</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Method:</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

1:   get (user_name, added_delay, current, SLA, tag)
2:   process (meta-broker_delay)
3:   added_delay ← get (added_delay) + meta-broker_delay
4:   if get(tag) is user tag then
send (cloud, added_delay, current, SLA, tag)
monitor (job, user_delay, user_name, MIPS)
end if
if get(tag) is cloud tag then
for all meta-broker_inter ∈ meta-registry
send(meta-broker_inter, added_delay, current, SLA, tag)
monitor (job, user_delay, user_name, MIPS)
end for
end if
meta-broker_inter→ res_meta-broker
if get(tag) is res_meta-broker then
send(bucket, added_delay, current, SLA, tag)
monitor (job, user_delay, user_name, MIPS)
end if
if get(tag) is req_meta-broker then
if flag → True then
update(tag ← user tag)
send(bucket, added_delay, current, SLA, tag)
monitor (job, user_delay, user_name, MIPS)
if count = num {default: 3} then
terminate_messages(meta-broker_inter)
else
terminate_messages(meta-broker_inter)
end if
end if
end if
end if

The Service-Distribution algorithm (2) is based on messages that are sent from one entity to the other according to a message exchanging optimal scheme. Initially, this algorithm collects jobs forwarded from user(s) and processes each of them by considering a processing delay time. Then it offers an adaptive solution according to job input as follows.

- In the case that the message (tag) denotes a communication request from the user then it is forwarded to the local cloud for SLA matchmaking.
- In the case that the tag denotes a message that arrived from the local cloud then it is forwarded to the inter-linked meta-brokers (extracted from the meta-registry). Further to this, the first responder that is capable of executing the job gets the user profile for further delegation based on the message-exchanging algorithm.
- In the case that the tag denotes a message that arrived from the same meta-broker (after a circulation) then either it re-distributes the message or it terminates it and suspends the job based on a criterion (e.g. if there is no SLA matchmaking in terms of software competence of clouds it suspends execution, however in the case of limitation on computational power it redistributes the job until execution happens).
This completes the distribution case wherein requests are spread among the meta-brokers relying on the meta-registry list information. In the algorithmic case if the flag is set to true value then each job is iteratively re-distributed until execution occurs. However, in the case of a continuously SLA mismatching the algorithm keeps a counter that terminates the job after a certain value of iterations (e.g. the default value is set to the desired one).

3.4.3 The Service-Availability Algorithm

The Service-Availability algorithm is responsible for firstly ensuring SLA matchmaking and secondly directing a job to an available resource (in a local or remote location) according to a dynamic workload management optimal scheme.

**Algorithm 3: Service-Availability**

<table>
<thead>
<tr>
<th>Require:</th>
<th>meta-broker_name</th>
<th>meta-broker identity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>local_broker_delay</td>
<td>cloud decision latency</td>
</tr>
<tr>
<td></td>
<td>added_delay</td>
<td>increasing delay figure</td>
</tr>
<tr>
<td></td>
<td>local_broker_SLA</td>
<td>SLA defined by cloud</td>
</tr>
<tr>
<td></td>
<td>user_profile</td>
<td>user formed profile</td>
</tr>
<tr>
<td></td>
<td>workload</td>
<td>figure to current workload</td>
</tr>
<tr>
<td></td>
<td>jobi</td>
<td>job submitted by user</td>
</tr>
<tr>
<td></td>
<td>host_capacity</td>
<td>host augmented capacity</td>
</tr>
</tbody>
</table>

| Method: | get(user_data) | user sent data |
|         | send(entity, event data): | send request and data |
|         | monitor(job, delay, user, MIPS) | monitor current data |
|         | process(cloud_delay) | process of cloud decision making time |

1: get (user_name, added_delay, current, SLA, tag)  
2: process (local_broker_delay)  
3: added_delay ← get (added_delay) + local_broker_delay  
4: if get(user SLA) match local_broker_SLA and get(host_capacity) exists  
5: for all jobi  
6: get(user_profile(user_CPU, memory, storage, BW)) < workload [i]  
7: workload [i++] ← jobi  
8: send (datacenter, added_delay, current, SLA, tag)  
9: monitor (jobi, user_delay, user_name, MIPS)  
10: end for  
11: else  
12: send (meta-broker, added_delay, current, SLA, tag)  
13: monitor (jobi, user_delay, user_name, MIPS)  
14: end if

This is particularly useful for large scale systems wherein various requests are submitted at different times, thus the system requires to decide whether there is computational capacity to execute or not. Thus, the workload management optimal scheme either allows a job to be forwarded for execution into the low level infrastructure, or it decides to return the job back
to the meta-broker for further dissemination (as presented in Service-Distribution algorithm 2). Algorithm 3 pseudo-code illustrates the Service-Availability procedure.

The workload calculation is related to the dynamic workload management that is further discussed in the next section. The Service-Availability algorithm (2) demonstrates that jobs arriving into the cloud systems (into the local-broker) are dynamically controlled for a) SLA matchmaking and b) current workload and capacity of hosts. In the case that jobs can be executed the algorithm forwards each one to the datacenter for host and VM allocation. In any other case requests are returned back to the meta-broker for distribution. In any case delays are considered for providing a realistic solution that increases the overall job executing waiting time.

3.4.4 The Service-Allocation Algorithm

The Service-Allocation algorithm demonstrates the VM allocation optimal schemes and jobs execution that are enclosed in VMs. By default, jobs that arrive in the datacenter are selected for execution according to a first come first served queue system. This implies that a VM is instantiated or generated for each request arriving first in the datacenter management component called the hypervisor. However, LRMS (discussed in following sections) allows scheduling in the local level based on shortest job first, priority scheduling and earliest deadline first algorithms. In any of the cases, the hypervisor takes the decision for the following optimal schemes.

- VM generation happens in a static (create from the beginning) or dynamic (instantiation of VMs through migration) manner. This includes that VMs are either installed or are relying on an external storage and transferred (migrated) to a cloud host for executing a request that has been created at a previous instance (in the form of a recorded VM).

- Requests for VMs are organized in a deferred queue that releases jobs after a real-time criterion passes. This encompasses a combination of the number of jobs that are in the deferred queue and the intervening of an interval value.

Finally, the algorithm allocates a host portion and starts the VM execution. The local-broker that accesses the local resource plan queue monitors this procedure. This allows a dynamic (workload management decision is based on current queues) and real-time scheduling (queues are released after interval criteria) of jobs. It should be noted that the default ICMS static algorithms include the first come first serve, shortest job first, earliest deadline first, and priority algorithms. The Service-Allocation algorithm (4) pseudo-code
demonstrates the procedure that allocates host resources and executes a specific job. Finally, the monitor operation keeps a log of traces for each of the job exchanging among entities.

### Algorithm 4: Service-Allocation

<table>
<thead>
<tr>
<th>Require:</th>
<th>selection of sharing optimal scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>select_VM_allocation_os</td>
<td>parameter selection of queue algorithm</td>
</tr>
<tr>
<td>select_queue</td>
<td>static algorithms</td>
</tr>
<tr>
<td>FCFS, SJF, EDF, PS</td>
<td>user job</td>
</tr>
<tr>
<td>job</td>
<td>user profile data as formed by the meta-broker</td>
</tr>
<tr>
<td>user_profile</td>
<td>hypervisor decision latency</td>
</tr>
<tr>
<td>hypervisor_delay</td>
<td>datacenter decision latency</td>
</tr>
<tr>
<td>dc_delay</td>
<td>increasing delay figure</td>
</tr>
<tr>
<td>added_delay</td>
<td>hash key for queuing jobs</td>
</tr>
<tr>
<td>key</td>
<td>desired queue size for releasing the queue</td>
</tr>
<tr>
<td>queue_length</td>
<td>desired time for releasing the queue</td>
</tr>
<tr>
<td>interval</td>
<td>VM allocation optimal scheme</td>
</tr>
<tr>
<td>static, dynamic</td>
<td>parameter for current workload</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method:</th>
<th>user sent data</th>
</tr>
</thead>
<tbody>
<tr>
<td>get(user_data)</td>
<td>send request and data</td>
</tr>
<tr>
<td>send(entity, event data)</td>
<td>monitor current data</td>
</tr>
<tr>
<td>monitor(job, delay, user, MIPS)</td>
<td>process of meta-broker decision making time</td>
</tr>
<tr>
<td>process(hyper_delay)</td>
<td>keep log of executed jobs</td>
</tr>
<tr>
<td>record(VM data)</td>
<td>scheduling algorithm fashion</td>
</tr>
<tr>
<td>sort(algorithm)</td>
<td>allocating host computational resources</td>
</tr>
<tr>
<td>host_allocation(data)</td>
<td>checking whether VM rely in a pool</td>
</tr>
<tr>
<td>exists(VM)</td>
<td>transfer VM from pool to datacenter</td>
</tr>
<tr>
<td>migrate(VM)</td>
<td>process of hypervisor decision making time</td>
</tr>
<tr>
<td>gen(metric)</td>
<td>generate metric results</td>
</tr>
</tbody>
</table>

```python
1: get (user_name, added_delay, current, SLA, tag)
2: select_VM_allocation_os [static, dynamic]
3: select_queue [FCFS, SJF, EDF, PS]
4: for all job, ∈ queue[]
5: get (user_profile)
6: added_delay ← get (added_delay) + hypervisor_delay + dc_delay
7: queue [key, job,]
8: select_queue.sort(FCS)
9: if queue_length>=s or interval=i then
10: if select_VM_allocation_os = static
11: host_allocation(CPU, memory, storage, BW)
12: process(VM)
13: record(VM, job,)
14: else if exists(VM) then
15: migrate(VM)
16: else
17: goto(line10)
18: end if
19: end if
20: workload ← host_allocation(CPU, memory, storage, BW)
21: send (local-broker, workload)
```
Finally, this section presented the ICMS algorithmic structure for defining the most important internal procedures of the inter-cloud.

### 3.5 Metrics and Performance Analysis

The study clarifies the requirements of the service as posed by the user in order to determine the collection of performance measures. Specifically, the assumption is that at a preliminary stage the user of the cloud requests an infrastructure as a service (IaaS) cloud capacity that encompasses software as a service (SaaS) characteristic. This involves a request for cores, CPU power, memory, storage and bandwidth as well as controller (e.g. drives) and platform specification (e.g. operating system). Eventually, this describes the required VM that will be deployed to a remote location for sandboxing the job.

The SaaS is defined by an average number of instructions per program and cycles per instructions for measuring the software requirements of the hardware capacity (required clock rate of host). This eventually allows the determination of the performance criteria of the various ICMS entities. Thus, starting with the service submission, the assumption includes the following scenario. A user requests for a job (to be sandboxed in a VM) and can get \( x \) out of maximum 1 (for defining percentage value \( x/100\% \)) of the total augmented host capacity. It should be mentioned that \( x \) is defined by the cloud administrator (to leave capacity for the hypervisor) of the datacenter and requires executing a set of software (programs) with \( y \) instructions, and cycles per instruction (CPI) = \( z \) (e.g. 300 cycles / 100 instructions = 3) with clock rate \( w \) defined in MHz (e.g. 25% of 4000MHz of Host with single core).

The CPI value considers the cycles per instructions required from specific software. The performance of the services is analogous to the performance of the VM that executes the service and it is affected by the overall latency till the service execution started. It should be noted that the \( x\% \) denotes the percentage of the machine to be dedicated to the VMs. The rest is required by the host in order to operate at high performance rates (as mentioned before to operate the hypervisor). Formulas (1) and (3) present the performance of the VM in terms of execution time with regards to requirements posed by the user for a request of a mono-processor VM.

Formula (1): VM execution time

\[
\text{ExecTime}_{\text{VM}} = \text{Instruction per job} \times \text{cycles per instruction} \times \text{seconds per cycle} \quad (1)
\]
Also the CPI represents the cycles per instruction count for each user submission and is given by formula (2).

Formulas (2, 3): Cycles per instruction and execution time calculation as presented into Brady (2012).

\[
\text{CPI} = \frac{\text{Cycles}_{\text{user}}}{\text{Instructions}_{\text{user}}} \quad (2)
\]

\[
\text{ExecTime}_{\text{VM}} = \frac{\text{Instruction} \times \text{CPI}}{\text{ClockRate}} \quad (3)
\]

Formula (4) presents the performance of the VM for the total execution time measure by considering a multi-processor request. In addition, the \( h \) parameter denotes the time duration of the VM leasing by the cloud user.

Formula (4): VM total execution time

\[
\text{TotalExecTime}_{\text{VM}} = \text{instructions} \times \text{CPI}_{\text{user}} \times \frac{1}{\text{CPU}_{\text{VM}}} \times \frac{1}{\text{CPUCores}_{\text{VM}}} \times h \quad (4)
\]

This includes the CPU and cores’ capacity as required by the user in the user submission. The study multiplies that value with the number of cores for deciding the requested CPU for a multi-processor system as in formula (3). Similarly, the study places a coefficient value for each of the computational characteristics required by the user, e.g. CPU (5), memory-formula (6), storage-formula (7), bandwidth-formula (8). The VMCount represents the VM quantity that shares computational capacity.

Formulas (5, 6, 7, 8): VM CPU, memory, storage and bandwidth (Brady, 2012)

\[
\text{CPU}_{\text{VM}} = \frac{\text{CPU}_{\text{host}} \times \text{Coefficient}}{\text{VMCount}} \quad (5)
\]

\[
\text{Memory}_{\text{VM}} = \frac{\text{Memory}_{\text{host}} \times \text{Coefficient}}{\text{VMCount}} \quad (6)
\]

\[
\text{Storage}_{\text{VM}} = \frac{\text{Storage}_{\text{host}} \times \text{Coefficient}}{\text{VMCount}} \quad (7)
\]

\[
\text{BW}_{\text{VM}} = \frac{\text{BW}_{\text{host}} \times \text{Coefficient}}{\text{VMCount}} \quad (8)
\]

For instances of the dynamic workload the CPU augmented values is given by formula 9. For the rest of the resources the augmented value is represented by the sum of the hosts memory (10), storage (11) and bandwidth (12).

Formulas (9, 10, 11, 12): Augmented host CPU, memory, storage and bandwidth

\[
\text{AugmentedHostCPU} = \sum_{\text{host}=1}^{\text{i}} (\text{CPU}_i \times \text{CPUCores}_i) \quad (9)
\]
AugmentedHostMemory = \sum_{host=1}^{i} (memory_i) \quad (10)

AugmentedHostStorage = \sum_{host=1}^{i} (storage_i) \quad (11)

AugmentedHostBW = \sum_{host=1}^{i} (BW_i) \quad (12)

The performance measure of a service is given by formula (13) as follows.

Formula (13): Service (job) execution time

\text{ExecTime}_{\text{service}} = \text{Latency}_{\text{user-vm}} + \text{PerformanceTime}_{\text{VM}} \quad (13)

The latency denotes the time that passes till the service execution start. The actual latency is the weighted degree of the meta-broker communication in a weighted graph (e.g. the ping value) given by formula (14). Formula (15) computes the latency of the contacted meta-broker as the sum of the responses sent multiplied by \( \frac{1}{2} \). This denotes the half loops as the algorithm sends messages but not all the nodes respond back. The nodes that reply are given by the variable \( a \). Thus the sum of the vertices that respond are summed up and added to the half-walk vertices. This value is again multiplied with a coefficient property \( c \) that represents priority jobs or advance reservation mechanisms etc. The final latency of the user to VM communication is calculated by formula (16) that includes the hypervisor information processing time. This is calculated as the sum of latencies (processing time) for meta-brokers, datacenters and hypervisors.

Formulas (14, 15, 16): Latency of meta-broker, datacentre and user-VM

\text{Latency}_{\text{mbr}} = \sum_{\text{mbr}_i \in \text{mbr}} \text{degree(meta - broker}_i) \quad (14)

\text{Latency}_{\text{dc}} = \left( \frac{1}{2} \right) \sum_{\text{dc}_i \in \text{dc}} \text{deg(d}_c_i) + \sum_{i \in a} \text{deg(d}_c_i) \ast c \quad (15)

\text{Latency}_{\text{user-vm}} = \text{Latency}_{\text{mbr}} + \text{Latency}_{\text{dc}} + \text{Latency}_{\text{hypervisor}} \quad (16)

Relevant delays involve meta-brokers, local-brokers, and hypervisors (host and VM allocation) decision-making processes. Finally, the performance of the service is non-relevant to the execution time of the service as given by formula (17).

Formula (17): Performance of service (job) execution

\text{Performance}_{\text{service}} = \frac{1}{\text{Executio}} \quad (17)

Similarly, the measurement of the performance of the VM given by formula (18)
Formula (18): Performance of VM

\[ \text{Performance}_{VM} = \frac{1}{\text{ExecutionTime}_{VM}} \quad (18) \]

To illustrate the formulas and to measure the performance of a service submission, the study presents a brief scenario case. A user requests for a VM with 0.25 (25%) of host performance and executes a set of programs with \(100 \times 10^6\) instructions, and CPI= 3 (300 cycles / 100 instructions) with clock rate 1000 MHz (0.25 of 4000MHz of the host with single core). The performance of the VM is calculated as follows:

\[ \text{ExecTime}_{VM} = 100 \times 10^6\text{(ns.)} \times 3 \times \frac{1}{1000} \times \frac{1}{1} = 3 \times 10^5\text{ns.} = 0.3 \text{ms.} \]

Thus, the performance factor of the specific VM is calculated by dividing 1 by the execution time, which is an equal to 3.33. If the total delay of the events from the user to the VM is 10 then the execution time and performance of the service is calculated as follows:

\[ \text{ExecutionTime}_{service} = 0.3 + 10 = 10.3 \text{ms. so, Performance}_{service} = \frac{1}{10.3} = 0.097 \]

Finally, a useful computational metric for measuring the performance capacity of a job is the millions of instructions per second (MIPS) that are required by the user. This demonstrates the application requirements and is utilized as an indicator of the required CPU power by a user. In addition, by using this metric a comparison of user specifications can happen if required. Formula (19) shows how MIPS are calculated.

Formula (19): Calculation of MIPS

\[ \text{MIPS} = \frac{\text{clock rate}}{\text{CPI}} \times 10^{-6} \quad (19) \]

By defining the initially requested performance measures, the study presents the actual algorithmic structure of the ICMS model. Finally, the metrics are also presented to show the performance of the algorithms. The core algorithms represent the service (or job) user submission life-cycle to an inter-cloud setting. The monitoring procedure generates a number of metrics by instantiating data generated from the whole ICMS process as follows. Initially, the throughput value, formula (20), of the cloud includes the number of jobs that are matched and executed by the cloud.

Formula (20): Calculation of cloud throughput

\[ \text{Throughput}_{cloud} = \sum_{i=0}^{i} \text{job}_i \quad (20) \]

This affects the cloud utilization parameter, formula (21) that calculates the number of jobs (percentage) executed from the whole user(s) input.

Formula (21): Calculation of cloud utilization level (percentage)
Utilization_{cloud} = \frac{\text{Throughput}}{\sum_{\text{Jobs}}^\text{counter}} \times 100 \ (21)

The turnaround time for a job is calculated by formula (22) and the response ratio is calculated by formula (23) where \( x \) is the highest value of a metric and \( y \) is the lowest value of the same metric.

Formulas (22, 23): Calculation of cloud turnaround time and response ratios

\[
\text{TurnTime}_{cloud} = \left\{ \frac{\text{instructions}_{VM} \times \text{CPI}_{VM}}{\text{CPU}_{VM} \times \text{CPUcores}_{VM} \times 10^5} \right\}_{\text{cloud}} + \text{CurrentTime} \ (22)
\]

\[
\text{ResponseRatio} = \left( \frac{x - y}{x} \right) \% \ (23)
\]

The makespan formula (24), demonstrates the total values of the VM execution time plus the total delay due to service dissemination. This includes the figure of the total service execution time from the initialization to execution.

Formula (24): Calculation of job makespan time

\[
\text{Makespan}_{job} = \text{VMexecutionTime} + \text{totalDelayTime} \ (24)
\]

Formula (25) demonstrates the energy consumption function (in kW) with regards to the datacenter host consumed watts; the time elapsed during the cloudlet life-cycle, the cost per kWh (average) and a coefficient value as an experimental property for adjusting simulation results.

Formula (25): Calculation of consumed KW

\[
\text{ConsKW} = \frac{\text{watts} \times \text{time}}{1000} \times \text{costPerKWH} \times \text{coef} \ (25)
\]

Specifically, the wattage is set to 300 (average value of a high-power workstation), the cost for kW is set to 8 cents per hour (UK rates) and the coefficient value is set to 10 to slightly increase the values due to low workload. Finally, energy efficiency measures are calculated by formulas (26) and (27) with respect to the host configuration on watts, the usage of machine in terms of hours and the cost per kilowatt per hour. The cloud administrator defines these values prior to the initialization of the ICMS.

Formulas (26, 27): Calculation of host energy consumption and total host cost

\[
\text{HostEnergyConsumption} = \frac{\text{WattsHost} \times \text{UserHours}}{1000} \ (26)
\]

\[
\text{TotalHostCost} = \text{ConsKW} \times \text{CostKwph} \ (27)
\]

To conclude, this section presented the ICMS performance formulas (e.g. throughput, makespan, turnaround times, etc.) as an important cornerstone for measuring the performance of the setting against a number of metrics. The next section illustrates the measures of
dispersion in order to define mean values (along with weights) that can be adapted to each of the performance metrics discussed in this section.

3.6 Summary
The study presented the ICMS as a meta-computing inspired solution for inter-clouds. The algorithmic solution includes a set of four sub-algorithms that aim to classify the service life-cycle steps in submission of workloads. The architectural design of the inter-cloud meta-broker is totally decentralized and dynamic by enhancing the decision-making process for service distribution and real-time job scheduling. Next the focus is on the ICMS optimal schemes.
Chapter 4: The ICMS Optimal Schemes

4.1 Outline
This chapter presents the ICMS optimal schemes, namely, Message Exchanging Optimization (MEO), VM deployment and LRMS. These implement the novel functionalities of the ICMS where the message exchanging implements the job distributions method, the VM deployment offers the VM management features and the LRMS details the management of the local cloud schedulers. The next section details the operations of each scheme.

4.2 The Message Exchanging Optimization (MEO)
This section details the mathematical representation using graph theory of the Message Exchanging Optimization (MEO) model (Bessis et al. 2013) that includes both the intra- and inter-operations presented in Chapter 3. It should be mentioned that MEO optimizes further the ICMS. Then, the focus is on the algorithmic structure that implements the mathematical model.

4.2.1 The Mathematical Underpinning
The study assumes that there is \(v_1 \in V\), where \(V = \{v_1, v_2, ..., v_n\}\) are entities of a distributed system that constitute the message-exchanging meta-brokers (nodes). Each node has an uptime value \(u_{v_n}\) and \(\{u_{v_n} \in \mathbb{R} | u_{v_n} > 0\}\) that defines its operational period (in ms). The graph \(G = (V, E)\) is a directed graph with nodes \(v_n \in V\) and \(v_n \neq v_{n+1}\) etc. If \(v_n\) communicates with \(v_{n+1}\) a trail is created between nodes called \(w_1\). In this case the study aims for a directed graph as nodes communicate with different orders, thus \(w_{n,n+1}\) is considered as a walk that connects \(v_n\) and \(v_{n+1}\). In a similar way, \(w_{n+1,n}\) is considered as a walk that connects \(v_{n+1}\) and \(v_n\). Each \(v_n\) contains a profile \(f_n\) where \(v_{n+1} \in f_n\), so \(v_n\) and \(v_{n+1}\) are inter-connected nodes.

- A list \(f_n = \left[\begin{array}{c} v_{n+1} \\ \vdots \\ v_w \end{array}\right]\) is defined to contain the addresses of nodes to which to send messages.

It is supposed that a request \(r_a \in Rq\), and \(Rq = \{r_1, r_2, ..., r_p\}\) is submitted at a time instance \(t_i\) and \(\{t \in \mathbb{R} | t \geq 0\}\) to an entity \(v_a\). The entity defines an interval \(int_a\) that expresses the waiting time (deadline) and \(\{int_a \in \mathbb{R} | int_a > 0\}\). The entity also defines the maximum size of the file as \(s_a\) and \(\{s \in \mathbb{R} | s > 0\}\). Each list that is generated by \(v_a\) contains a number of data \(d_a \in D\) and \(D = \{d_1, d_2, ..., d_q\}\) that are organized in an array with respect to the message identification. The \(d_a\) contains the job specification and the interval by which it could vary for
each job. Thus, each message $m_a \in M$, and $M = \{m_1, m_2, \ldots, m_i\}$ where $m_a = [L_a, tag_a]$ and $L_a$ is the list of the message with a tag $a$ to be the tag value of the message.

- A list $L_a = \left[ \begin{array}{c} d_1 \\ \vdots \\ d_s \end{array} \right]$ is defined to contain the data of a message where $s$ defines the size of the list and $d_i = \left[ \begin{array}{c} j_1 \int_1 \\ \vdots \\ j_s \int_a \end{array} \right]$, where $j_s$ is the last job with interval $\int_a$.

- The $j = \left[ \begin{array}{c} \text{clocks} \\ \text{cpus} \\ \text{cores} \\ \text{bw} \\ \text{hf} \end{array} \right]$ is the basic metric for calculating performance (h represents the requested hours) of MEO.

The study assumes that a message is sent to a recipient $v_{n+1}$ with a delay $dl_a$ and $\{dl \in \mathbb{R} | 0 < dl_a < \int_a\}$. The node $v_{n+1}$ executes a ranking function according to a performance measure. This is related to information extracted from $j_1$ and $dl_a$, and for the case of the total time taken between the submission of a job for execution and the return of the complete output it is defined by the turnaround ranking formula (22).

The response message is $m_b \in M$ where $m_b = [L_b, tag_b]$.

- The study defines a list $L_b = \left[ \begin{array}{c} j_k \\ \vdots \\ j_1 \end{array} \right]$ where $j_k$ is the first ranked job and $j_1$ is the last ranked job.

- The study defines $j_k = \left[ \begin{array}{c} \text{perf} \\ \text{udl} \end{array} \right]$ as the basic metrics for calculating performance where $udl_a$ is the updated delay $\{udl_a \in \mathbb{R} | 0 < udl_a < dl_a\}$ that includes the decision making time frame of the requester.

In this case the trail $w_{n:n+1}$ defines the distance of the requester and responder thus $w_{n:n+1} = dl_a$. The requester collects messages and ranks jobs according to a function called $\text{RankReq}(L_b)$. Then each job $j_k$ is associated with a resource e.g. $v_{n+1}$. Specifically, $\text{RankRes}(L_b)$ defines the minimization of the performance criteria. Table 1 details the list of notations used in this discussion.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>A set of nodes</td>
</tr>
<tr>
<td>$v_n \in V$</td>
<td>A node</td>
</tr>
<tr>
<td>$w_{n:n+1}$</td>
<td>A trail between nodes</td>
</tr>
<tr>
<td>$u_{v_n}$</td>
<td>A node uptime value</td>
</tr>
<tr>
<td>$f_n$</td>
<td>A profile of $v_n$</td>
</tr>
<tr>
<td>Rq</td>
<td>A set of requests</td>
</tr>
<tr>
<td>$r_m \in Rq$</td>
<td>A request</td>
</tr>
<tr>
<td>$t_a$</td>
<td>A time instance</td>
</tr>
<tr>
<td>$\int_a$</td>
<td>An interval of node $a$</td>
</tr>
</tbody>
</table>
Based on this discussion the following sections present lemmas to address:

- The trail calculation in a bidirectional graph
- The message-exchanging optimization
- The message timing justification
- The message distribution
- The performance optimization of message exchanging (message path and energy consumption rates)
- The performance-efficiency costs (message size and delays)

The lemmas ensure that the performance of the MEO model is always equal to or better than the AllToAll approach for the specification discussed in the lemmas. In addition, the study defines the costs of communication in terms of message size and delay.

**Lemma 1: Trail Calculation**

Let $G(V,E)$ be a directed graph with non-negative edge weights, and suppose that $v_n$ is the starting point and $v_{n+1}$ is the ending point with $v_{v_n} \neq 0$ and $v_{v_{n+1}} \neq 0$. The assumption is that there is at least one path from $v_n$ to $v_{n+1}$ if and only if $v_{v_{n+1}} \in f_n$. For each message $m_i \in \{m_1, m_2, \ldots, m_n\}$ that is sent from entity $v_n$ to $v_{n+1}$ a trail is created with weight $w_{v_n,v_{n+1}}$. Then for all $w_{v_n,v_{n+1}}$ in a directed graph formation there is a bi-directional weight that is calculated as follows:
\[
W_{v_{n+1}|v_n} = \begin{cases} 
\delta_{a} + \eta_{a} : \text{if } \text{RankRes}(L_a), \exists L_b = \emptyset \\
(2 \times \delta_{a}) + \eta_{a} \leq \text{int}_{a} : \text{if } \text{RankRes}(L_a) \rightarrow L_b \neq \emptyset \text{ and and } w_{v_{n+1}|v_n} = w_{v_{n+1}|v_n}
\end{cases}
\]

Proof: Let \( \delta_{a} \) be the distance between \( v_n \) and \( v_{n+1} \) and \( \eta_{a} \), \( \text{int}_{a} \); if RankRes(L_a) as defined by \( L_b \) is an empty set (in other words there is no job execution availability). In contrast if RankRes(L_a) returns the list \( L_b \), then the weight equals the \( \delta_{a} \) multiplied by 2 (if \( d_{v_n|v_{n+1}} = d_{v_{n+1}|v_n} \)) as it includes a request and a response message transfer. Finally, the study adds the value of \( \eta_{a} \) the decision making delay of \( v_{n+1} \).

In the case where \( d_{v_n|v_{n+1}} \neq d_{v_{n+1}|v_n} \), the sum of the delays and \( \eta_{a} \) define the weight of the path. In particular \( \text{int}_{a} \) is always greater than or equal to the weight as it represents the deadline for the message distribution where \( v_{v_n} > \text{int}_{a} \) and \( v_{v_{n+1}} > \text{int}_{a} \). In other words, if \( \text{int}_{a} \) becomes greater than the weight of the path the message is terminated. Finally, the addition of the sum of the delays (\( \delta_{a} \)) and internal decision making (\( \eta_{a} \)) for a multi-level distribution case will give the value of the total delay that is always lower than or equal to \( \text{int}_{a} \) for a non-message termination case.

Lemma 2: Message Number Calculation

Let \( G(V,E) \) be a directed graph with non-negative edge weights, and suppose that \( v_n \) is the starting point and \( v_{n+1} \) is the ending point with \( v_{v_n} \neq 0 \) and \( v_{v_{n+1}} \neq 0 \). Suppose that \( w_{v_n|v_{n+1}} \neq 0 \) then the number of sent messages (SM) during exchanging is calculated as

\[
\text{SM}_{\text{SM}} = \eta \times \frac{\theta}{\theta} \text{ and } \theta \neq 0 \text{ where } \eta \text{ is the maximum number of messages forwarded from the requester and } \theta \text{ is the factor of availability defined by } v_{v_n+1} \text{ as derived from RankRes(L_a).}
\]

Specifically the last operator generates \( L_b \) which is the list of job availability in \( v_{v_n+1} \). The \( \theta \) value is calculated as \( \theta = \frac{\eta}{e} \) where \( e \) is the size of the received messages if the list \( L_b \) is not an empty set, thus \( l \neq 0 \).

Proof: Let \( w_{v_n|v_{n+1}} \neq 0, \theta \neq 0, l \neq 0 \) and \( v_{v_n}, v_{v_{n+1}} \neq 0 \) then the size of the list \( L_b \) defines the capacity of \( v_{v_n+1} \) to reply back that is formed according to RankRes(L_a). In this case the maximum number of jobs is set to \( l \). Then the \( \theta \) factor equals the number of messages sent divided by the number of messages received (based on the \( l \) value). Then the overall value of messages is the sum of sent messages from the requester to the division of sent from
responder expressed by the $v_{n+1}$ capability and described by the $\theta$ factor. The study defines $l \neq 0$ as in any other case (e.g. $l = 0$) the message is terminated, as there is no job availability.

**Lemma 3: Performance of MEO Model**

Let $G(V,E)$ be a directed graph with non-negative edge weights, and suppose that $v_n$ is the starting point and $v_{n+1}$ is the ending point and $v_{v_n} \neq 0$ and $v_{v_{n+1}} \neq 0$. Suppose that $w_{v_n:v_{n+1}} \neq 0$ then the number of sent message (SM) by using MEO is always lower than or equal to the AllToAll collective approach. Thus, the performance of the MEO is always equal to or better than the AllToAll approach. In the case of $\theta < \eta$ and $l \neq 0$ and $\theta > 0$, MEO performance is continually improved compared to the AllToAll approach.

**Proof:** As $w_{v_n:v_{n+1}} \neq 0$ a trail among $v_n$ and $v_{n+1}$ ($v_{v_n} \neq 0$ and $v_{v_{n+1}} \neq 0$) thus this designates that at least one message has been sent. The AllToAll SM is calculated $SM_{\text{AllToAll}} = 2 \times \eta$ while the SM of MEO is $SM_{\text{MEO}} = \eta + \frac{7}{\theta}$. As the hypothesis includes that $\theta \neq 0$ then if $l = \eta$, for all contacted $v_{n+1}, v_{n+1}, \ldots, v_{n+w}$ (w is the last node of the $f_n$ list that can offer availability) $SM_{\text{AllToAll}} = SM_{\text{MEO}}$. However, if $l < \eta$ and $l \neq 0$ the value of $\theta > 0$ thus according to lemma 2 the $SM_{\text{AllToAll}} > SM_{\text{MEO}}$. So, $Perf_{MS_{\text{all-to-all}}} = \frac{1}{MS_{\text{all-to-all}}}$ and $Perf_{MS_{\text{MEO}}} = \frac{1}{MS_{\text{MEO}}}$ then the study concludes that $Perf_{MS_{\text{MEO}}} > Perf_{MS_{\text{all-to-all}}}$.

**Lemma 4: Message Timing Justification**

Let $G(V,E)$ be a directed graph with non-negative edge weights, and suppose that $v_n$ is the starting point and $v_{n+1}$ is the ending point and $v_{v_n} \neq 0$ and $v_{v_{n+1}} \neq 0$. For each message $m_i \in \{m_1, m_2, \ldots, m_n\}$ that is sent from entity $v_n$ to $v_{v_n} \neq 0$ and vice versa, after the ranking operation where $l \neq 0$, the total delay time is formulated as $int_{v_n} \geq d_{v_n} + d_{v_{n+1}} + udl_{v_{n+1}}$. The proposition is that in a MEO model the interval time is always greater than or equal to the sum of the delays in order to have message replies.

**Proof:** According to lemma 1 the $w_{v_n:v_{n+1}}$, represents the weight of the path with respect to the delay of the communication link. Thus, in this case $d_{v_n} \leq int_{v_n}$, $d_{v_{n+1}} \leq int_{v_n}$ and $udl_{v_{n+1}} \leq int_{v_n}$. If the sum ($d_{v_n} + d_{v_{n+1}} + udl_{v_{n+1}}$) is greater than $int_{v_n}$ then a message is terminated immediately. So, as $l \neq 0$ the sum of delays is required to have a lower value than the total interval time based on the hypothesis that a response is sent if and only if there is availability and the time interval has not been exceeded.
Lemma 5: Message Distribution in MEO

Let $G(V,E)$ be a directed graph with non-negative edge weights, and suppose that $v_n$ is the starting point and connected with $v_{n+1}$ that is further connected with $v_{n+2}$ as a finishing point and $u_{v_n} \neq 0$, $u_{v_{n+1}} \neq 0$ and $u_{v_{n+2}} \neq 0$. For each message $m_i \in \{m_1, m_2, \ldots m_n\}$ that is sent from entity $v_n$ to $v_{n+1}$ and forwarded to $v_{n+2}$ the probability of succeeding availability decreases with time. That is to say that $P(v_{n+1}) > P(v_{n+2}) > \ldots > P(v_i)$ for a given interval when $uld_{v_n} < uld_{v_{n+1}} < uld_{v_{n+2}}$ etc., so the probability to find a resource on time tends to decrease with respect to the initial chosen interval of the requesting entity.

Proof: Let $w_{v_n:v_{n+1}} \neq 0$ and $w_{v_{n+1}:v_{n+2}} \neq 0$, with $u_{v_n} \neq 0$, $u_{v_{n+1}} \neq 0$ and $u_{v_{n+2}} \neq 0$, so there is a trail with $w_{v_n:v_{n+2}} = w_{v_n:v_{n+1}} + w_{v_{n+1}:v_{n+2}}$ and $w_{v_n:v_{n+2}} \neq 0$. The $d_{v_n}$ defines the delay of the channel to reach $v_{n+1}$ and $d_{v_{n+1}}$ the delay of the channel to reach $v_{n+2}$. For a non-message termination case, and if $d_{v_n:v_{n+1}} = d_{v_{n+1}:v_n}$ then $int_{v_n} \geq (d_{v_n} + d_{v_{n+1}}) \times 2$. The study defines as possibility of an entity $v_{n+1}$ the division of $d_{v_{n+1}} \times coef$ where coef is a coefficient value defined by the entity (e.g. $int_{v_n}$). The coef is set to the same value by the entire pool of entities. However, as time increases, and $d_{v_n:v_{n+1}} \neq 0$, $d_{v_n} > (d_{v_n} + d_{v_{n+1}})$. Thus the conclusion is that $P(v_{n+1}) > P(v_{n+2}) > \ldots > P(v_i)$, so $P(d_{v_n:v_{n+1}}/coef) > P(d_{v_{n+1}:v_{n+2}}/coef) > \ldots > P(d_{v_{n+1}:v_n}/coef)$; this encompasses that as the number of further disseminations increases the possibility to meet the initial deadline is decreased as well.

Lemma 6: Message-Exchanging Costs

Lemma 6: Message Size Cost

Let $G(V,E)$ be a directed graph with non-negative edge weights, and suppose that $v_n$ is the starting point and $v_{n+1}$ is the ending point and $u_{v_n} \neq 0$ and $u_{v_{n+1}} \neq 0$. For each message $m_i \in \{m_1, m_2, \ldots m_n\}$ that is sent from entity $v_n$ to $v_{n+1}$ and vice versa after the ranking operation the size cost of communication is related to the size of each message sent and received divided by the capacity of the communication channel ($c$ - the bandwidth) as given by the formula (28).

Formula (28): Calculation of cost size in MEO

$$\text{CostSize}_{\text{MEO}} = \frac{S_s + S_r}{c_{v_n:v_{n+1}}} \quad (28)$$

The proposition is that cost$_{\text{MEO}}$ of the MEO operation offers always lower or equal cost results when compared with cost$_{\text{AllToAll}}$. 
Comparing that does not reply back thus coefficient value of the requesting entity. Thus if messages $r$, operation defines the sum of the delays to reach recipients in addition to the sum of delays for

Proof:

Let $w_{v_n:v_{n+1}} \neq 0$ ($v_n \neq 0$ and $v_{n+1} \neq 0$) thus there is a trail from $v_n$ to $v_{n+1}$. The cost operation defines the sum of the file sizes $s_s$ and $s_r$ where $s_r \leq s_s$ divided by the channel bandwidth. Thus if $s_r = s_s$ as in the AllToAll case, $\text{cost}_\text{AllToAll} = 2 \times s_s$. In contrast, if $s_r > s_s$ then $2 \times s_s > s_s + s_r$. The conclusion is that $\frac{2 \times s_s}{c_{v_n:v_{n+1}}} > \frac{s_s + s_r}{c_{v_n:v_{n+1}}}$. In this case cost$_\text{AllToAll}$ is higher than cost$_\text{MEO}$ and this validates that if $s_r > s_s$ and according to lemma 4 MEO offers always lower costs results.

**Lemma 7: Message Delay Cost**

Let $G(V,E)$ be a directed graph with non-negative edge weights, and suppose that $v_n$ is the starting point and $v_{n+1}$ is the ending point with $w_{v_n:v_{n+1}} \neq 0$, $\text{int}_{v_n} \neq 0$, $v_n \neq 0$ and $v_{n+1} \neq 0$. For each message $m_i \in \{m_1, m_2, \ldots m_n\}$ that is sent from entity $v_n$ to $v_{n+1}$ and vice versa after the ranking operation the delay cost of communication is related to delay of each message to reach and return from a recipient divided by the interval time $\text{int}_{v_n}$ as defined by the $v_n$ entity where $v_{v_n} > \text{int}_{v_n}$ and $v_{v_{n+1}} > \text{int}_{v_{n+1}}$ and given by formula (29).

Formula (29): Calculation of cost message number in MEO

$$\text{CostMessageNumber}_{\text{MEO}} = \frac{\eta + \eta}{\text{int}_{v_n}} \tag{29}$$

The study proposes that as the interval increases the cost of the delay function decreases and based on lemma 2 it offers lower costs if $\eta \neq \eta + \frac{n}{\theta}$ and $\theta \neq 0$. Especially, if $dl_{v_n:v_{n+1}} = dl_{v_{n+1}:v_n}$, the $\sum_{i=n}^{\eta} dl_{v_i} > \sum_{i=n+1}^{\theta} dl_{v_i}$, thus the delay cost function can be represented by formula 30.

Formula (30): Calculation of cost of message delay in MEO

$$\text{CostMessageDelay}_{\text{MEO}} = \frac{\sum_{i=n}^{\eta} dl_{v_i} + \sum_{i=n+1}^{\theta} dl_{v_i}}{\text{int}_{v_n}} \tag{30}$$

The proposition is that the delay cost function could be represented in terms of the sum of delays where the MEO solution always offers better performance with lower cost values compared with the AllToAll approach if $\sum_{i=n}^{\eta} dl_{v_i} > \sum_{i=n+1}^{\theta} dl_{v_i}$.

Proof:

Let $w_{v_n:v_{n+1}} \neq 0$ ($v_n \neq 0$ and $v_{n+1} \neq 0$) thus there is a trail from $v_n$ to $v_{n+1}$. The cost operation defines the sum of the delays to reach recipients in addition to the sum of delays for messages returned back (in case of availability) divided by the interval that represents the coefficient value of the requesting entity. Thus if $\eta \neq \eta + \frac{n}{\theta}$ there is at least one responder that does not reply back thus $\eta > \theta$ and if $dl_{v_n:v_{n+1}} = dl_{v_{n+1}:v_n}$, the $\sum_{i=n}^{\eta} dl_{v_i} > \sum_{i=n+1}^{\theta} dl_{v_i}$.

Comparing this with AllToAll message-exchanging, the conclusion is that in the former
case $\sum_{i=n}^{\eta} dl_{v_i} = \sum_{i=n+1}^{\eta} dl_{v_i}$, and since $\sum_{i=n}^{\eta} dl_{v_i} > \sum_{i=n+1}^{\eta} dl_{v_i}$ (lemma 2) the cost of the delay using MEO is always lower.

**Lemmas 8, 9: Performance Optimization**

**Lemma 8: Message Path Performance Optimization**

Let $G(V,E)$ be a directed graph with non-negative edge weights, and suppose that $v_n$ is the starting point and connected with $v_{n+1}$ that is connected to $v_{n+2}$ etc. The connection continues to node $v_b$ that forms the ending point and $v_{n+1} \neq 0, v_{n+2} \neq 0, ..., v_b \neq 0$. For each message $m_i \in \{m_1, m_2, ..., m_n\}$ that is sent from entity $v_n$ to $v_b$ and forwarded among intermediate entities and $int_{v_n} > \sum_{i=v_n}^{v_b} d_i$ the best message distribution path is defined by the minimum of the sum of the costs for paths from $v_n$ to $v_{n+1}$ to $v_{n+2}$ etc. as defined by lemma 6. In other words, the minimum function $f(x)$ that calculates the cost $c_{v_n:v_b}$ defines the best path where $c_{v_n:v_b}$ defines the bandwidth of the channel from $v_n$ to $v_b$.

**Proof:** Let $w_{v_n:v_x} \neq 0$ so there is a path among $v_n$ to $v_x$. In particular due to the decentralized nature of the setting this could include multiple paths of the same message to reach the final destination where each message $m_y$ has a total delay less than or equal to $int_y$ according to lemma 1. Thus, different requests from remote resources are ranked (to the host location) according to a best path selection of the cost measures. In the case of $int_{v_n} > \sum_{i=v_n}^{v_x} d_i$ there is a possibility for job allocation to a remote location, thus the lowest cost value defines the best trail based on the selection of the cost function $f(x)$ that it is either related to cost$\text{MEO}$ or the total delay as given by formula 31.

**Formula (31): Calculation of cloud turnaround time and response ratios**

$$\min[f(x)] = \begin{cases} 
\text{cost}_{\text{MEO}} = \frac{s_x + s_y}{c_{v_n:v_{n+1}}}, & \text{if } x = \text{size} \\
\sum_{z=v_n}^{v_{n+1}} dl_z + udl_x, & \text{if } x = \text{delay}
\end{cases} \quad (31)$$

So, the conclusion is that the best path optimization for multi-level messaging is defined by the minimum cost operation.

**Lemma 9: Energy Consumption Optimization**

Let $G(V,E)$ be a directed graph with non-negative edge weights, and suppose that $v_n$ is the starting point and connected with $v_{n+1}$ that is connected to $v_{n+2}$ etc. The connection continues to node $v_b$ that forms the ending point and $v_{n+1} \neq 0, v_{n+2} \neq 0, ..., v_b \neq 0$. For each message $m_i \in \{m_1, m_2, ..., m_n\}$ that is sent from entity $v_n$ to $v_b$ and forwarded among
intermediate entities the energy consumption of the MEO approach is related to the cost message delay operation and given by formula 32.

Formula (32): Calculation of consumed kW in MEO

\[
\text{ConsKW} = \frac{\text{watts} \times \text{CostMessageDelay}}{1000}
\]

The proposition is that the consumption rates in MEO are related to the cost of the delay function, as this is the consumption frequency in terms of timing, so based on lemma 5 MEO, offers optimized consumption rates.

**Proof:** Let \( w_{v_n,v_x} \neq 0 \) so there is a path among \( v_n \) to \( v_x \). If \( \eta \neq \eta + \frac{\eta}{\theta} \) and \( \theta \neq 0 \) thus the assumption is that there is at least one node that does not reply back thus \( \eta > \theta \). Comparing this with AllToAll message-exchanging, then the study concludes that in the former case \( \sum_{i=n}^{\eta} d_{v_1} = \sum_{i=n+1}^{\theta} d_{v_1} \), and since in MEO \( \sum_{i=n}^{\eta} d_{v_1} > \sum_{i=n+1}^{\theta} d_{v_1} \) (lemma 2) the cost function is MEO consumption \( \sum_{i=n}^{\eta} d_{v_1} + \sum_{i=n+1}^{\theta} d_{v_1} \) to \( \text{all-to-all} \). Thus the energy consumption rates will be optimized as well.

It should be noted that the cost of communication could be calculated either by the size of the message or the delay of the communication link according to the configuration of the requesting entity.

**4.2.2 The Algorithmic Structure**

This section demonstrates the algorithmic model of the MEO. Particularly, the approach includes request and response entities that allow the implementation of the whole set of aforementioned mathematical underpinnings. Figure 15 shows the relationships of the algorithmic structure. Figure 16 facilitates the structure and the relationship of the algorithms that illustrate the message exchanging functionality. It is shown that the four algorithms determine a) the job collection, message formation and sending procedure of the request entity, b) the message gathering, identification of specification, ranking and response procedure of the response entity, c) the message redistribution procedure and d) the message collection, ranking and assignment process.

![Figure 16: The sequence diagram of the algorithmic model](image-url)
4.2.2.1 The Message-Formation Algorithm

The assumption is that a number of nodes have the same uptime and are interlinked in a decentralized topology. In the given setting, users can frequently submit requests that describe the job requirements. Thus, algorithm 5 enables the Message-Formation operation executed by a node (the responder of the setting).

Algorithm 5: Message-Formation

<table>
<thead>
<tr>
<th>Require:</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>the requesting resource</td>
</tr>
<tr>
<td>interval\text{\textsubscript{collection}}</td>
<td>the interval time to collect job messages</td>
</tr>
<tr>
<td>time:</td>
<td>the current time instance</td>
</tr>
<tr>
<td>j\textsubscript{i}</td>
<td>the job submitted by a source</td>
</tr>
<tr>
<td>clocks\textsubscript{i}</td>
<td>the job required clocks</td>
</tr>
<tr>
<td>CPI\textsubscript{i}</td>
<td>the job required CPI</td>
</tr>
<tr>
<td>cores\textsubscript{i}</td>
<td>the job required cores</td>
</tr>
<tr>
<td>bw\textsubscript{i}</td>
<td>the job required bandwidth</td>
</tr>
<tr>
<td>h\textsubscript{i}</td>
<td>the job required duration</td>
</tr>
<tr>
<td>L\textsubscript{i}</td>
<td>the list with the job specification data</td>
</tr>
<tr>
<td>tag</td>
<td>the tag value of the message (e.g. q)</td>
</tr>
<tr>
<td>msg</td>
<td>the message contains the L\textsubscript{i} and the tag</td>
</tr>
<tr>
<td>f\textsubscript{i}</td>
<td>the profile of the entity i</td>
</tr>
<tr>
<td>ad</td>
<td>the address of a node included in the f\textsubscript{i}</td>
</tr>
<tr>
<td>e</td>
<td>the tag value for incoming messages</td>
</tr>
<tr>
<td>interval\text{\textsubscript{distribution}}</td>
<td>the interval time to collect distribution messages</td>
</tr>
</tbody>
</table>

Figure 17: The sequence diagram of the algorithmic model
response the notification of the responder
criterion the performance ranking criterion defined by the entity i

Operations:
get the collection procedure of job data
set the operation to set the required job specification
create the operation to create a list
open the operation to open a profile
size the method to return the size value of the profile
send the method to send a message to address ad
run the method to run an algorithm
wait the method to wait for an interval to expire
update the method to update a list

Algorithms:
Ranking the ranking algorithm that accepts the criterion as value
Assignment the assignment algorithm that accepts the \( L_i \) as value

1. set \( \text{interval}_{\text{collection}}\), criterion
2. while \( \text{time} < \text{interval}_{\text{collection}} \) wait
3. for all \( j_i \)
4. get(\( \text{clocks}_{i}, \text{CPI}_{i}, \text{cores}_{i}, \text{bw}_{i}, h_i \))
5. set \( j_i[\text{clocks}_{i}, \text{CPI}_{i}, \text{cores}_{i}, \text{bw}_{i}, h_i] \)
6. create(L_i[j_i])
7. end for
8. set tag ← q
9. create(msg[L_i,tag, criterion])
10. open(f_i)
11. for all \( f_i\).size() 
12. ad ← get(f_i[k])
13. send(msg, ad)
14. end for
15. set \( \text{interval}_{\text{distribution}}\)
16. while \( \text{time} < \text{interval}_{\text{distribution}} \) wait
17. if tag=e then
18. get(response)
19. run(Ranking algorithm(criterion))
20. update(L_i[j_i])
21. end if
22. end while
23. for all \( L_i\).size() 
24. run(Assignment algorithm(L_i))
25. end for

The algorithm details the message formation and sending procedure by highlighting the ranking and assignment algorithms. Specifically, it configures an interval value for collecting jobs from the source (e.g. users) and creates a list using the incoming job specification. At this point the job submission contains a number of requirements such as job clocks, CPI, cores, bandwidth and the job uptime duration. Other key deliverables are that the algorithm sets a tag on the message, creates a list and sends it to the interlinked addresses as extracted
from a personalized profile. Then, the process waits for an interval in order to collect responses. During that period it ranks (according to a user-defined criterion) and updates the list in order to finally assign jobs.

4.2.2.2 The Message-Collection Algorithm

The message collection algorithm facilitates the assembly procedure for incoming messages and the formation of the ranked list. The algorithm identifies messages for job delegation by identifying port tags (tag=q or w). Then it decomposes the message list (L_i) and sorts required performance measures. Again, it creates a new list with the up-to-date ranked jobs and replies back to the requester. If the list is empty (due to resource unavailability) the responder does not reply back. Specifically, it takes a decision based on a flag variable denoted by the entity. This includes either terminating the job as there is no further distribution, or to collect addresses from a personalized profile. Advance submissions to a next level of resources exemplify further distribution. In this case the message redistribution algorithm is initialized.

Algorithm 6: -Collection implements this functionality.

Algorithm 6: Message-Collection

<table>
<thead>
<tr>
<th>Require:</th>
<th>i</th>
<th>the requesting node</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>i’</td>
<td>the responding node</td>
</tr>
<tr>
<td></td>
<td>msg_i</td>
<td>the incoming message from requester i</td>
</tr>
<tr>
<td></td>
<td>L_i</td>
<td>the list with the job specification data</td>
</tr>
<tr>
<td></td>
<td>flag</td>
<td>the flag variable</td>
</tr>
<tr>
<td></td>
<td>trm</td>
<td>the termination flag</td>
</tr>
<tr>
<td></td>
<td>rds</td>
<td>the redistribution flag</td>
</tr>
<tr>
<td></td>
<td>tag</td>
<td>the tag value of the message (e.g. q)</td>
</tr>
<tr>
<td></td>
<td>q</td>
<td>the tag indication for incoming message from requester</td>
</tr>
<tr>
<td></td>
<td>w</td>
<td>the tag indication for incoming message from redistributor</td>
</tr>
<tr>
<td></td>
<td>L_i</td>
<td>the incoming list from i</td>
</tr>
<tr>
<td></td>
<td>int_responder</td>
<td>the interval of the responder</td>
</tr>
<tr>
<td></td>
<td>int_requester</td>
<td>the interval of the requester</td>
</tr>
<tr>
<td></td>
<td>e</td>
<td>the tag indication for returning messages</td>
</tr>
<tr>
<td></td>
<td>ad</td>
<td>the address of a node</td>
</tr>
<tr>
<td></td>
<td>criterion</td>
<td>the message required performance criterion</td>
</tr>
</tbody>
</table>

| Operations:    | decompose | a message decompose operation |
|                | get | the collection procedure of job data |
|                | rank | the ranking procedure |
|                | set | the operation to set the required job specification |
|                | update | the method to update a list |
|                | size | the method to return the size value of the profile |
|                | send | the method to send a message to address ad |
|                | terminate | the method to terminate a message at the responder |
|                | destroy | the method to delete a list L_i at the responder |
|                | open | the method to open a profile |
The method to execute an algorithm or an operation

Algorithms:
- Ranking algorithm
- Redistribution algorithm

1. set flag ← {trm, rds}
2. for all msg, and (tag=q or tag=w)
3. decompose (msg_i)
4. get L_i
5. run(Ranking_algorithm(criterion))
6. update(L_i)
7. if L_i.size>0 then
8. if int_responder < int_requester then
9. set tag ← e
10. ad ← i
11. send(msg, ad)
12. end if
13. else
14. if f_i.size=0 then
15. flag=trm
16. else then
17. flag=rds
18. end if
19. case: flag = trm
20. terminate(msg_i)
21. destroy(L_i)
22. case: flag = rds
23. open(f_i)
24. for all f_i.size()
25. run(Redistribution_algorithm(f_i))
26. end for
27. end case
28. end if
29. end for

To conclude, the algorithm implements the message collection phase. It is shown that the responder assigns a tag value (tag=e), so that is the means to identify incoming messages. Finally, the algorithm configures the termination or distribution case in the initialization phase. However, in a decentralized system such a decision is defined by the profile size of the responding node. This is to say that if the list is empty then the node is a terminal node; alternatively the message will be further redistributed (thus it is an intermediate node).

4.2.2.3 The Message-Ranking Algorithm

The Message-Ranking algorithm defines the criteria for job classification in the request or respond nodes. Specifically, this includes the minimization of a function that calculates a collection of metrics. These are job execution time, total time, latency, graph degree,
turnaround time, makespan, energy consumption (kW per entity), energy consumption (kW per host resource), cost of the message size, cost of the delay, and probability cost. It should be mentioned that the origin of the experimental analysis defines the criteria, e.g. in a path optimization case, the degree of the graph or the latency will be taken as the ranking conditions. Algorithm 7: Message-Ranking describes the measures of the performance.

Algorithm 7: Message-Ranking

| Require: | the requesting or responding node |
| criterion | the requesting ranking criterion |
| Rank | the output of the criterion |
| instr | the number of instructions |
| cycles | the number of job cycles |
| CPI | the number of CPI |
| cores | the number of required cores |
| h | the uptime of the job in host |
| dl | the delay of the entity |
| int | the interval of an entity (e.g. int\textsubscript{i} is the interval of requester) |
| udl | the decision making time (e.g. udl\textsubscript{i}) |
| Watts | the watts of the host entity |
| consPerKW | the consumption per kW rate of the entity |
| coef | the coefficient value of the entity |
| L | the list of the jobs to be ranked |
| bw | the bandwidth speed of the channel |
| Nomsg | the total number of messages (e.g. from entity i to y is Nomsg\textsubscript{i:y}) |
| Operations: | the method to return the size value of the profile |
| size | |

1. if criterion ← ET (Execution Time)
2. Rank = instr\times cycles
3. end if
4. if criterion ← TT (Total Time)
5. Rank = (instr\times CPI\times 1/CPU)*1.cores\times h
6. end if
7. if criterion ← LA (Latency)
8. Rank = dl+dl\textsubscript{i}
9. end if
10. if criterion ← DE (Degree)
11. Rank = \sum dl\textsubscript{i} + \sum dl\textsubscript{i}'
12. end if
13. if criterion ← TuT (Turnaround Time)
14. Rank = ET + LA
15. end if
16. if criterion ← MS (Makespan)
17. Rank = ET + udl\textsubscript{i}
18. end if
19. if criterion ← CPE (Consumption per entity)
20. Rank = (watts\times TT\times 10\textsuperscript{-3})\times consPerKW\times coef
21. end if
22. if criterion ← CPH (Consumption per host)
23. \[ \text{Rank} = \text{watts} \times 10^3 \]
24. end if
25. if criterion ← MeC (Message Cost)
26. \[ \text{Rank} = (\text{size}(L_i) + \text{size}(L_{i'})) \times (1/bw) \]
27. end if
28. if criterion ← DeC (Delay Cost)
29. \[ \text{Rank} = (\text{Nomsg}_{i:i'} + ((\text{Nomsg}_{i':i})/\text{Nomsg}_{i:i'})/\text{int}_i \]
30. end if
31. if criterion ← PR (Probability Cost)
32. \[ \text{Rank} = \text{dl}_{\text{entity}}/\text{int}_{\text{entity}} \]
33. end if

The Message-Ranking algorithm implements the metrics for measuring performance. This describes the ranking procedures in the requester or responder. It should be mentioned that at the initial algorithm (collection) the requester configures the criterion so all responders classify requests according to identical competencies. The optimization of the algorithmic functions defines the best standings within the ranked lists (e.g. the execution time is calculated as in formula 1).

4.2.2.4 The Message-Redistribution Algorithm

The message redistribution algorithm implements the relocation procedure in the case of further job dissemination. The procedure alters the tag values of messages and forwards each one to a node belonging to the personalized profile list. Algorithm 8: Message-Redistribution demonstrates the process of message dispersal.

Algorithm 8: Message-Redistribution

| Require: | msg | the requesting message |
| i | the requesting or responding node |
| msg\_i | the incoming message from requester i |
| L\_i | the list with the job specification data |
| f\_i' | the profile of the entity |
| flag | the flag variable |
| rds | the redistribution flag |
| tag | the tag value of the message (e.g. q) |
| p | the tag indication for outcoming message from redistributor |
| int | the interval of the requester |
| t | the time instance |
| ad | the address of a node |

| Operations: | create | the create message operation |
| get | the collection procedure of job data |
| set | the operation to set the required job specification |
| send | the method to send a message to address ad |
| open | the method to open a profile |
| Rank | the method to rank jobs of the list |
A key aspect is that the algorithm operates under the initial deadline value (of the requesting node) in order to be terminated in cases of interval violations. Also, the algorithm allows messages to be forwarded only if there is no availability in the local resource pool. In this case, messages are reformed and transferred to remote entities for requesting resource availability according to a specific criterion.

### 4.2.2.5 The Message-Assignment Algorithm

The message assignment algorithm determines the next phase of the resource allocation. Here the messages have been concluded. When the message formation algorithm instantiates this procedure each job is assigned to the desired entity that offers the best rankings (according to algorithm 4). Algorithm 9: Message-Assignment implements the allocation of jobs in entities (thus to their local hosts’ scheduler). The algorithm collects the execution results after the completion of the job.

**Algorithm 9: Message-Assignment**

<table>
<thead>
<tr>
<th>Require:</th>
<th>i</th>
<th>the requesting or responding node</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>j</td>
<td>the job</td>
</tr>
<tr>
<td></td>
<td>j_set</td>
<td>the set of jobs in not i</td>
</tr>
<tr>
<td></td>
<td>tag</td>
<td>the job assignment tag</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>the value to define assignment</td>
</tr>
<tr>
<td></td>
<td>ad</td>
<td>the address of the best ranked resource</td>
</tr>
<tr>
<td></td>
<td>res</td>
<td>the performance results of the job assignment</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operations:</th>
<th>set</th>
<th>the operation to set the tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>allocate</td>
<td>the operation to allocate the job to resource</td>
<td></td>
</tr>
<tr>
<td>send(LRMS)</td>
<td>the sending procedure of job data into LRMS</td>
<td></td>
</tr>
</tbody>
</table>

1. **for all** j ∈ j_set
2. set tag ← a
3. allocate (j,ad)
4. send(LRMS)
5. **end for**
This process is related to the local resource management system (LRMS) of the specific resource that takes the scheduling decision. Specifically, the system moves to the next resource management step (e.g. job scheduling orchestration). This concludes the MEO modeling structure that encompasses the implementation of the whole set of operations as described in each algorithmic section sections. The next section illustrates the use case scenarios of inter-clouds by implementing the MEO algorithms to extract functioning outputs.

4.3 The VM instantiation models

The VM deployment policy defines the management of VMs by the cloud provider in order to manage the creation of VMs. In particular, in cloud systems, remote users enclose each job submission to a VM (a procedure that is called sandboxing). The VM deployment could have a vital role in the whole job execution life cycle. Thus, the study defines two VM deployment strategies, called static and dynamic VMs instantiation, to classify the VM generation process. These are described as follows.

- **Static** is defined as the deployment of VMs where there is a fixed number of VMs that are instantiated by the hosts. In static case the cloud administrator creates VMs at the time of the user submission. So there is no usage of records of previous job summations.

- **Dynamic** is defined as the deployment of VMs where instantiations are based on the use of past VMs executions and pre-installed VMs in order to serve user demands. This is to say that VMs are generated based on old user demands for specific requirements (e.g. CPU capacity). The assumption is based on the fact that users could request for identical VMs in the future. This is mainly because most of the cloud providers offer a specific set of VM specifications. Also, the study assumes that each datacenter has a storage place to save VMs, and each one is migrated in the cloud in order to be instantiated by the users.

In order to explore static and dynamic deployment the study utilizes a VM migration approach as presented by Lagar-Cavilla et al. (2009). In particular the approach utilizes a forking processing method (Lagar-Cavilla et al. 2009 and extended in Lagar-Cavilla et al. 2011) to discover the performance of the VMs. It should be mentioned that in the forking case, the generation happens by inheritance. Specifically, the state of the parent thread creates a distinct child thread within a multithreaded environment. In this case the threads represent the VMs that are generated based on past job submissions. After forking, the VMs are
migrated from the remote location and placed in available hosts for accepting job submissions. The study utilizes this approach in order to demonstrate the performance optimal in terms of VM total execution time. (Sotiriadis et al. 2012a, Bessis 2012b). Next, the static VM instantiation model is presented.

4.3.1 The static VM instantiation model

In the static VM instantiation model the decision of the VMs deployment is taken prior to the job submission phase. So, when new jobs arrive in the cloud hypervisor, the last one selects the appropriate resource for scheduling the tasks.

The static VM instantiation algorithm 10 demonstrates the job distribution in a predefined setting. Here a cloud hypervisor instantiates VMs based on the requests for VMs. Specifically, each host of the resource pool generates a number of VMs according to the criterion of the administrator. The newly deployed virtual resources are ready for job execution as they already have a local queue in the space-shared queuing policy as detailed by Calheiros et al. (2011). The last one schedules jobs on multi-core CPUs by allocating fragments from job executions.

Algorithm 10: Static VM instantiation job distribution algorithm

| Require: | Jobs\textsubscript{num}: the initial jobs number of the workload archive |
| Host\textsubscript{i}: | physical host |
| Reqnode: | requested node |
| Poolhost: | physical resource list |
| PoolVMs: | virtual resource list |
| Res\textsubscript{VM}: | responder virtual resource (the guest) |
| Res\textsubscript{LRMS}: | responder LRMS |
| Res\textsubscript{queue}: | responder queue list |
| Job\text{desc}: | description in requested processing elements, estimated execution time |
| Message\textsubscript{jobAllocation}: | job allocation requested message |
| Message\textsubscript{informative}: | information on job delegation message |
| Message\textsubscript{jobDelegation}: | job execution request |
| Message\textsubscript{results}: | job delegated job results come directly from remote centralized scheduler |
| Del\textsubscript{list}: | A vector with a list of accepted delegated resources |
| OpportunisticCriterion\textsubscript{i}: | Opportunistic execution criterion |

| Require: | Hypervisor(), Send message(), Get message(), Set criterion() |

1: \textbf{for} Host\textsubscript{i} = \{i, i++, n\} \in Pool\textsubscript{host} \textbf{do}
2: \hspace{1em} Hypervisor (Pool\textsubscript{host}, Pool\textsubscript{VMs}, OpportunisticCriterion\textsubscript{i}) \textbf{accepts} Req\textsubscript{node}
3: \hspace{1em} \textbf{for} Jobs\textsubscript{num} = \{y, y++, y\} \in Wai \textbf{do}
4: \hspace{2em} \textbf{for} all Res\textsubscript{LRMS} \in Pool\textsubscript{VMs} \textbf{do}
5: \hspace{3em} Send Message\textsubscript{jobAllocation}(Job\text{desc}) \textbf{to} Res\textsubscript{LRMS}, Res\textsubscript{queue}
6: \hspace{3em} Set criterion(Criterion\textsubscript{i})
When a new job arrives in the queue the request is sent directly to the physical resource, in which the hypervisor has already instantiated the VMs. The forwarded message for job execution goes to the guest VM node and places the job(s) in its local queue, known as LRMS. Finally, the executed job is returned back to the requester node through the physical host. Figure 17 demonstrates the sequence model of this procedure for a FCFS queue.

![Sequence Diagram](image)

Figure 18: The Sequence diagram of the static VM instantiation

In particular, the responder host is includes the hypervisor that creates the VM and sends a message for instantiation the local queue. The last one informs the requesting node that the VM is ready for execution.

4.3.2 The dynamic VM instantiation model

The dynamic instantiation allows VMs to be generated on demand based on the current job characteristics from the analysis of previous job executions. This is to say that the previous submitted workloads affect the number of the VMs to be deployed by the hypervisor.
Basically, the requester node asks for job execution directly from the pool list of the physical resources. Yet, the hypervisor of each physical machine has already generated the VMs for the specific job requirements based on a ranking list of past demands. The generation happens by forking (Lagar-Cavilla et al. 2011), which is a way of generating child VMs from parents by only copying the state of the thread within a multithreading environment. After that, the job goes to the instantiated VM(s) and is queued in their LRMS. Then the results are sent back to the requester node through the responder host. In parallel, the responder host gets a notification message of job completion. That is to say that the VMs developed by forking will be terminated. In case that a VM request is not in the ranking list then a new VM is generated based on the static case.

The dynamic VM instantiation algorithm 11 demonstrates the job distribution within a dynamic setting of a cloud hypervisor for instantiating VMs based on a criterion analogous to the number of jobs and the required computational power of the workload archive as detailed in Bessis et al. 2012a. In addition, the pseudo-code includes functionality for meta-scheduling of advanced and more complex scheduling cases. Specifically, each resource serves as a proxy between VM and LRMS. Thus, a new layer has been added to delegate messages from the meta- to the LRMS of the VM responder host.

**Algorithm 11: Dynamic VM instantiation job distribution algorithm**

**Require:**

<table>
<thead>
<tr>
<th>Job$_num$</th>
<th>initial jobs number of the workload archive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job$_counter$</td>
<td>a variable to store the count of the jobs</td>
</tr>
<tr>
<td>Job$_characteristics$</td>
<td>a variable to store the characteristics of the jobs</td>
</tr>
<tr>
<td>Job$_PES$</td>
<td>a variable to store the PEs of the job workload archive</td>
</tr>
<tr>
<td>coefficient$_i$</td>
<td>a coefficient variable with regards to the jobs total number</td>
</tr>
<tr>
<td>VM$_num$</td>
<td>the number of VMs</td>
</tr>
<tr>
<td>VM$_characteristics$</td>
<td>the computational characteristics of VMs</td>
</tr>
<tr>
<td>Host$_i$</td>
<td>the physical host</td>
</tr>
<tr>
<td>Req$_node$</td>
<td>the requested node</td>
</tr>
<tr>
<td>Pool$_host$</td>
<td>the physical resource list</td>
</tr>
<tr>
<td>Pool$_VMs$</td>
<td>the virtual resource list</td>
</tr>
<tr>
<td>Res$_VM$</td>
<td>the responder virtual resource (the guest)</td>
</tr>
<tr>
<td>Res$_LRMS$</td>
<td>the responder LRMS</td>
</tr>
<tr>
<td>Res$_queue$</td>
<td>the responder queue list</td>
</tr>
<tr>
<td>Job$_desc$</td>
<td>job description in requested processing elements, estimated execution time</td>
</tr>
<tr>
<td>Message$_jobAllocation$</td>
<td>the job allocation requested message</td>
</tr>
<tr>
<td>Message$_informative$</td>
<td>the information on job delegation message</td>
</tr>
<tr>
<td>Message$_jobDelegation$</td>
<td>the job execution request</td>
</tr>
<tr>
<td>Message$_results$</td>
<td>the job delegated job results come directly from the remote centralized scheduler</td>
</tr>
</tbody>
</table>
Del\textsubscript{list}: A vector with a list of accepted delegated resources

**Require:** Hypervisor(), Terminate(), Send message(), Get message(), Set criterion()

1: \textbf{for} Jobs\textsubscript{num} = \{y, y++, y\} \subseteq Wai \textbf{do}
2: \hspace{1em} Jobs\textsubscript{counter} \leftarrow Jobs\textsubscript{num}
3: \hspace{1em} VM\textsubscript{num} \leftarrow Jobs\textsubscript{counter} \cdot \text{coefficient}_i
4: \hspace{1em} VM\textsubscript{characteristics} \leftarrow Job\textsubscript{PEs} \cdot \text{coefficient}_i
5: \hspace{1em} \textbf{for} Host\textsubscript{i} = \{i, i++, n\} \subseteq \text{Pool}\textsubscript{host} \textbf{do}
6: \hspace{2em} Hypervisor (Pool\textsubscript{host}, Pool\textsubscript{VMs}, VM\textsubscript{num}, VM\textsubscript{characteristics}) \text{ accepts} Req\textsubscript{node}
7: \hspace{2em} \textbf{for all} Res\textsubscript{LRMS} \subseteq \text{Pool}\textsubscript{VMs} \textbf{do}
8: \hspace{3em} Send Message\text{jobAllocation(Job\textsubscript{desc}) to Res\textsubscript{LRMS}, Res\textsubscript{queue}}
9: \hspace{3em} Get Message\text{informative}
10: \hspace{3em} Del\textsubscript{list} \leftarrow Res\textsubscript{LRMS}
11: \hspace{3em} \text{Del\textsubscript{list}++}
12: \hspace{3em} \text{Terminate(Pool\textsubscript{VMs});}
13: \hspace{2em} \textbf{end for}
14: \hspace{1em} \textbf{end for}
15: \textbf{end for}
16: \textbf{for all Res\textsubscript{meta-scheduler} \subseteq Del\textsubscript{list} do}
17: \hspace{1em} Send Message\text{jobDelegation (Job\textsubscript{desc}) to Res\textsubscript{LRMS, Res\textsubscript{queue}}}
18: \hspace{1em} Get Message\text{results}
19: \hspace{1em} \textbf{end for}
20: \textbf{if} Del\textsubscript{list} = \emptyset \textbf{then goto step 1}
21: \textbf{end if}
22: \textbf{end for}

Figure 18 illustrates the procedure of the dynamic instantiation of VMs by incorporating the VM queue. Here, it should be noted that the VM queue first implements the meta-scheduling behaviour for coordinating LRMS queue submissions.

Figure 19: The Sequence diagram of the dynamic VM instantiation
In particular, the requesting node sends a call for job execution in the pool list. Each responder executes the ranking deployment and instantiate a job by migrating (the hypervisor deployment strategy). Then the instantiated VM is passed to the VM queue that returns the job back to the requesting node.

4.4 The Local Resource Management System (LRMS) Optimal Scheme

The LRMS scheme encompasses the functional and cross-functional processes that implement the efficient and effective deployment and utilization of provisions when they are needed. As this study is concerned with meta-scheduling concepts, thus scheduling of tasks in suitable resources for execution, the focus is on the operational resource formation in terms of overall service orchestration. These are classified to a) discrete-time reflection, b) dynamic workload management and c) LRMS orchestration scheme. The next sections discuss each of these in order to give insights to their operations.

a) The discrete-time orientation presents an event time case of the meta-brokers information exchanging. Specifically, the study assumes that various clouds (e.g. cloud\textsubscript{a}, cloud\textsubscript{b}) establish a partnership connection and are prepared for services exchange according to specific demands. The assumption is that initialization is determined by the cloud administrators. Then, the following steps demonstrate the time instances (e.g T0 represents time instance 0) that occur for service dissemination for available resources.

\textbf{T0.0:} User\textsubscript{a1} requests resources from cloud\textsubscript{a} (submits job\textsubscript{a1}).

\textbf{T0.1:} Cloud\textsubscript{a} assigns meta-broker\textsubscript{a1} and meta-registry\textsubscript{a1} that contains the information of linked meta-brokers.

\textbf{T0.2:} At this time instance (time instance 0), there are no existing submissions, hence cloud\textsubscript{a} allocates resources and executes job\textsubscript{a1} in VM\textsubscript{a1}.

\textbf{T0.3:} Job allocation is determined by the hypervisor that generates a VM for the sandboxing request.

\textbf{T1.0:} Next (time instance 1), user\textsubscript{a2} requests resources from cloud\textsubscript{a} (job\textsubscript{a2})

\textbf{T1.1:} Cloud\textsubscript{a} assigns meta-broker\textsubscript{a2} and meta-registry\textsubscript{a2} that contains the information of linked meta-broker data.

\textbf{T1.2:} At this time instance (time 1), there is one meta-broker (meta-broker\textsubscript{a1}) of the same cloud that sends a request to local-broker\textsubscript{a1}. The assumption is that resources are plentiful thus a further request for matchmaking is not sent.
T1.3: Job allocation is determined by the hypervisor that generates a VM for the sandboxing request.

T2.0: Next (time instance 3), user_{b1} requests more resources from cloud_b (job_{b1})

T2.1: Cloud_b assigns meta-broker_{b1} and meta-registry_{b1} that contains the information of linked meta-broker data.

T2.2: There are no existing submissions, hence cloud_b allocates resources and executes job_{b1} in VM_{b1}.

T3.3: Next (time instance 3), user_{b1} requests more resources from cloud_b (job_{b1})

T3.4: Cloud_b assigns meta-broker_{b1} and meta-registry_{b1} that contains the information of linked meta-broker data.

T3.5: At this time instance (time 3), there is one meta-broker (meta-broker_{b1}) of the same cloud that sends a request to local-broker_{b1}. The assumption is that resources are limited thus a further request for matchmaking is sent to cloud_a and meta-broker_{a1}.

T4.0: At this time (time 4 as the study assumes that time elapsed) meta-broker_{a1} collects the request and forwards it to local-broker_{a1} for SLA matchmaking.

T4.1: At this time local-broker_{a1} evaluates the request and the study assumes that there are plentiful resources for execution.

T5.0: Job allocation is determined by the hypervisor that generates a VM for the sandboxing request.

T6.0: Next (time instance 6), user_{b1} request more resources from cloud_b (job_{b1})

T6.1: Cloud_b assigns meta-broker_{b1} and meta-registry_{b1}.

T6.2: At this time instance (time 6), there is one meta-broker (meta-broker_{b1}) of the same cloud that sends a request to local-broker_{b1}. The assumption is that resources are limited thus a further request for matchmaking is sent to cloud_a and meta-broker_{a1}.

T7.0: At this time (time 4 as the study assumes that time elapsed) meta-broker_{a1} collects the request and forwards it to local-broker_{a1} for SLA matchmaking.

T7.1: At this time local-broker_{a1} evaluates the request and the study assumes that there are limited resources for execution.

T7.2: At this time (time 7 as the study assumes that time elapsed) local-broker_{a1} sends the request back to meta-broker_{a1} that forwards it to meta-broker_{c1} of interconnected cloud_c for SLA matchmaking. The study assumes that cloud_c contains meta-broker_c.
T8.0: At this time (time 4 as the study assumes that time elapsed) meta-broker_{c1} collects the request and forwards it to local-broker_{c1} for SLA matchmaking.
T8.1: At this time local-broker_{c1} evaluates the request and the study assumes that there are plentiful resources for execution.
T8.2: Job allocation is determined by the hypervisor that generates a VM for the sandboxing request.
T9.0: Next (time instance 9), user_{b1} requests more resources from cloud_{b} (job_{b1})
T9.1: Cloud_{b} assigns meta-broker_{b1} and meta-registry_{b1}.
T9.2: At this time instance (time 9), there is one meta-broker (meta-broker_{b1}) of the same cloud that sends a request to local-broker_{b1}. The assumption is that resources are limited thus a further request for matchmaking is sent to cloud_{a} and meta-broker_{a1}.
T10.0: At this time (time 10 as the study assumes that time elapsed) meta-broker_{a1} collects the request and forwards it to local-broker_{a1} for SLA matchmaking.
T10.1: The assumption is that resources are limited thus a further request for matchmaking is sent to cloud_{c} and meta-broker_{c1}.
T11.0: At this time (time 11 as the study assumes that time elapsed) the job enters meta-broker_{c1}. The assumption is that resources are limited thus the request returns back to meta-broker_{b1} which restarts job execution distribution after an interval.

b) The **dynamic workload management policy** involves workload calculation that is related to up-to-date host capacity in terms of CPU, memory, storage and bandwidth. Similar to real-time solution, this involves a re-active workload request for each job submission to the local-broker. Then the following steps demonstrate the time instances that are taking place during the service dissemination for dynamically request resource availability. This includes the time instance 1 of the discrete-time service distribution optimal scheme.

T1.0: At this time instance (time 1), there is one meta-broker (meta-broker_{a1}) of the same cloud that sends a request to local-broker_{a1}. The assumption is that resources are plentiful thus a further request for matchmaking is not sent.
T1.1: Local-broker_{a1} requests resources from datacenter_{a1} for current workload values.
T1.2: Datacenter_{a1} through the hypervisor monitors internal procedures and controls the local resource management system deferred queues with previous jobs.
T1.3: The last one responds with resource availability (the study assumes that resources are plentiful) while at the same time it alters the current resource availability in terms of computational resources (datacenter_{a1} hosts).

T1.4: Job allocation is determined by the hypervisor that generates a VM for the sandboxing request.

T1.5: VM_{a1} is placed in the queue for execution while the monitoring system releases the computational resources after the user leasing time interval has elapsed.

c) The LRMS demonstrates the deferred queues that are created by the hypervisor for effectively executing VMs. Specifically, during a job execution; the subsequent jobs that enter the hypervisor are placed in the queue. The study implements four different optimal schemes for deciding the VM execution case that includes the First Come First Serve (FCFS), the Shortest Job First (SJF), the Priority Scheduling (PS) algorithm and the Earliest Deadline First (EDF). Specifically, the assumption is that the cloud administrator selects the LRMS that suits the experimental case. This works in dynamic and real-time settings as ICMS develops regular triggers to release queues according to system cases.

4.5 Summary

The section presented the ICMS optimal schemes that include the message exchanging, the VM deployment and the LRMS. The architectural design of the inter-cloud meta-broker is totally decentralized and dynamic. Specifically, the inter-cloud facility distributed the request for service and sandboxes operations into VMs that belong to an interoperable sub-cloud. The next chapter employs ICMS over a simulation toolkit to implement the algorithmic functionality along with the performance metrics.
Chapter 5: Simulation Environment and Benchmark Analysis

5.1 Outline
This chapter focuses on the simulation framework that implements ICMS along with the generic cloud functionality in order to simulate an inter-cloud. Further, the ‘Simulating the Inter-Cloud’ (SimIC) toolkit, (Sotiriadis et al. 2013a) is a discrete event simulation framework that replicates an inter-cloud setting with regards to ICMS literature review extracted requirements (e.g. dynamic and elastic services). Thus, the work initially presents a discussion of related large-scale simulation toolkits towards a dynamic simulator for inter-clouds and presents the remarks and core design issues. For identifying the benchmark performance an analysis is performed on the CloudSim simulation toolkit (Calheiros et al. 2011) that implements identical cloud job submissions as the SimIC.

5.2 Large Scale Simulators (HPC, Grids and Clouds)
In recent years, simulators and analysis tools for distributed systems have been the center of development. This is based on the needs of job composition, setting configuration and resource deployment which is unfeasible in real systems. Specifically, real test bed experiments are difficult to perform due to the large number of different requirements and administrative costs. To this extent, a diversity of toolkits allows modelers to simulate their research hypothesis prior to the actual software or hardware development (Calheiros et al. 2011). This includes the need to evaluate various resource management phases (e.g. meta-scheduling optimal schemes) through several scenarios where the actual experiments are limited to the real test bed system’s scale and capabilities.

Thus the alternative solution is to use simulation tools that allow researchers to assess research questions and to implement and test the actual behavior of the system under several scenarios, metrics and criteria. For the case of clouds and inter-clouds none of the real-world systems allows extending testing based on specific resource management objectives (Buyya et al. 2010). Thus the study identifies the simulators that could meet the problem specification posed by ICMS for real-time dynamic information processing and job meta-scheduling in elastic inter-clouds. Specifically, an inter-cloud meta-scheduler essentially requires a highly dynamic, heterogeneous and decoupled simulation environment. In addition, the support of cloud features such as virtualization and heterogeneity are vital concepts to the final design decision. Based on that the following presents a discussion of advantages, drawbacks and applicability of simulators to the ICMS architectural notion.
• **MicroGrid** (Song et al. 2000) is an environment that offers the basic implemented tools for performing simulation experiments in grid computational settings. In addition the software uses Globus 1.1 that runs on top of the MicroGrid layer. Song et al. (2000) claim that the MicroGrid can offer overall feasible experimental results, although still for a non-real-time scheduling system. Based on that this solution is considered inappropriate for implementing ICMS.

• **GridSim** (Buyya and Murshed 2002), is a toolkit for achieving parallel processing modeling of grid simulation, schedulers and brokers. It includes the time- and space-shared scheduling algorithms and also allows advance reservation mechanism. However, the default version does not include real-time scheduling while dynamics consideration is limited. In advance, virtualization is not implemented.

• **GangSim** (Dumitrescu 2004), dissimilar to the previous solutions, includes the notion of virtual organizations and multi-sites. In addition, it enables repeatable and controllable experimentation with dynamic resource management techniques. In general, it is a powerful simulation setting, yet still the method does not support capturing of local scheduler behavior. In addition, Calheiros et al. (2011) suggest that virtualization and heterogeneity (diversity of requirements) is not considered.

• **GSSIM** (Kurowski et al. 2007) is a GridSim (Buyya and Murshed 2002) based simulation environment for performing scheduling in multi-level and heterogeneous grid systems. It is actually a flexible way of improving the traditional GridSim simulator speed, however, it does not allow the representation of certain optimal schemes.

• **Alvio** (Alvio Simulator 2002, Guim and Labarta 2007) is a simulation model based on C++ to evaluate traditional HPC scheduling approaches. It is a great tool for experiments that include small scale and non-dynamic configuration by providing a statistical estimator model. In addition, it offers a prediction module for incorporating past performance requirements and results for assisting future job scheduling decisions. However, when the system extends to a wide scale, e.g. in clouds, the testing results will not be realistic.

• **DGSim** (Iosup et al. 2008b) offers a framework for developing simulation schedulers of various grid resource management architectures. This high-level simulation environment offers a large-scale testing setting of multi-grid environments. In addition, Iosup et al. (2008b) claim that DGSim considers inter-operation of grids and relevant dynamics. Finally, a great advantage is that the input set is actually workload traces from real world
grid environments. However, an implementation of the simulator has not yet been distributed to the public.

- **SimGrid** (Casanova et al. 2008) offers the core functionalities for simulating distributed applications in large scale heterogeneous distributed systems. In general this is a powerful simulation environment that allows developers to perform experiments in grids, P2P and cloud systems. The great advantage of SimGrid is that allows testing on non-centralized and heterogeneous schedulers that aim at dynamic resource availability models. Also, when compared with the previous simulation environments, it offers sufficient documentation of APIs and components.

- **Alea** (Klusacek et al. 2008), and **Alea 2** (Klusacek and Rudova 2010), is a simulation framework to deal with heterogeneous resources and dynamic job flows. It allows the implementation of schedulers to support advanced techniques, e.g. easy backfilling. In addition, it offers a graphical user interface based on GridSim to provide a simulation environment that supports simulation of various Grid scheduling problems. To this extent, Klusacek and Rudova (2010) demonstrate the Alea environment by implementing a centralized grid scheduling algorithm. In addition, the authors claim that the scheduler could handle dynamic situations during the simulation process.

- **MONARC** (Dobre and Stratan, 2004) is a very strong simulator tool for simulating frameworks aimed at optimizing distributed computing. This toolkit allows the simulation of data replication and scheduling with the aim of improving flexibility and performance. However, it does not support virtualization features.

- **SmartGRID** (Huang et al. 2008) offers a decoupled layered structure and interoperable infrastructure for grid resources by utilizing fully decentralized and bio-inspired algorithms. For supporting scheduling decisions, SmartGRID integrates a simulator that provides services as group communication through asynchronous message passing and resource discovery. Specifically, the actual simulator relies on an overlay network. This is that nodes of the underlying network are connected through physical and logical links each of which resembles different paths. In advance, nodes are connected in loosely-coupled groups in a non-centralized peer to peer connection. It has been claimed by Huang et al. (2011) that the scheduling environment of SmartGRID offers a highly heterogeneous and dynamic meta-scheduling setting for evaluating grid and cloud environments. Moreover, the authors suggest that the simulation is designed to offer a
decisive effect in robustness, heterogeneity and reliability with regards to a dynamic infrastructure.

- \textit{CloudSim} (Calheiros et al. 2011) overcomes the absence of virtualization technology in the vast majority of distributed systems simulation efforts by offering a seamless model with VM support. This could allow the testing of more advanced solutions such as process and/or live migration. In addition, a unique feature of the framework is the flexibility to perform testing in either time- or space related algorithms. Furthermore, it offers a self-contained platform for developing cloud data-centers, hosts and service brokers. Although the other toolkits could offer some of those features, CloudSim’s major advantage is that customized clouds could easily be modeled and tested. Specifically, Calheiros et al. (2011) presents the development of a federation of clouds and presents some primary testing results with an improvement of their selected scheduling performance.

All these simulation environments have been developed in order to mimic advanced resource management decisions of different meta-scheduling approaches. However, each one implements different meta-scheduling criteria and optimal metrics for running experiments in various cases. It has been proposed by Buyya et al. (2010) that none of the conventional schemes, e.g. GangSim (Dumitrescu and Foster 2005) or GridSim or their alternatives such as GSSim, could address directly the cloud modeling requirements (e.g. support of virtualization, dynamics, heterogeneity and loosely-coupled-ness of nodes).

In parallel, the work of Calheiros et al. (2011) evaluates various simulation frameworks, e.g. GangSim and SimGrid, and concludes that although these toolkits offer grid simulation capabilities, none of these could support directly the posed requirements (application and infrastructure) arising from cloud computing and especially for inter-clouds. Accordingly, efficient simulation machines that offer dynamic scheduling decisions such as the Alea, SmartGRID, and CloudSim initially incorporate and then extend conventional simulator schemes by integrating dynamics. For instance, Alea based on GridSim extends functionality for handling dynamic situations. In addition, SmartGrid is based on Alea and GridSim which both bring “the modeling of different kinds of essential grid components, such as grid jobs with various parameters, heterogeneous grid resources, and grid users” as mentioned by Calheiros et al. (2011, p. 2).

Finally, CloudSim, originally bases its design on GridSim, however it includes important features for modeling and simulating large scale clouds datacenters, virtualized servers, customized policies, energy-aware computational resources, federated clouds, user-defined
policies and dynamic insertion of simulation elements (Calheiros et al. 2011). A very strong tool for simulating frameworks that aim to optimize distributed computing systems is the MONARC simulator. This toolkit allows the simulation of data replication and scheduling with the aim of improving flexibility and performance. Recently, a new simulation framework called iCanCloud (Nunez et al. 2011) has emerged that offers significant features, e.g. usability, flexibility, performance and scalability. The authors further suggest that iCanCloud simulates instance types provided by Amazon, and this is including in the simulation framework. Specifically, the study suggests that grid toolkits (e.g. Alea, GangSim, etc.) cannot offer the required design requirements and resource management characteristics for clouds and inter-clouds. However, CloudSim offers support for modeling and simulation of large scale cloud computing data centers as well as virtualized server hosts. Table 2 summarizes each simulator against the study’s key inter-cloud requirements, which highlights the need to design to cover the required operational features.

Table 2: Simulation frameworks and their characteristics

<table>
<thead>
<tr>
<th>Simulator/Requirement</th>
<th>Heterogeneity</th>
<th>Inter-operability</th>
<th>VM</th>
<th>Virtualization management</th>
<th>Rescheduling – Past service logs</th>
<th>Messaging</th>
<th>Dynamic meta-scheduling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alea</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GangSim</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SimGrid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GridSim</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GSSim</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SmartGrid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MONARC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CloudSim</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iCanCloud</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SimIC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.3 Core Design of Simulating the Inter-Cloud (SimIC)

For inter-clouds none of these solutions can address the application level requirements arising. By default, none could mimic such functionality without extending the distribution package. This is because the core design elements are meta-computing inspired, e.g. the large scale that the system could expand to, the decentralization of the distributed resource managers, the dynamic adaptability and the real-time service orchestration. To this extent, the modeling decision has been concluded to the development of the “Simulating the Inter-Cloud” SimIC simulation toolkit that is fundamentally inspired by the CloudSim framework.

By using the SimIC a modeler could configure a diversity of inter-clouds in terms of datacenter hosts and software characteristics wherein the desired number of users could send single or multiple requests for computational power (cores, CPU, memory, storage, bandwidth), software resources (measured empirically in cycles per instruction and million of
instructions per second) and duration of VM utilization. It should be mentioned that the toolkit includes a variety of meta-scheduling inspired characteristics for achieving job dissemination, resource discovery services, dynamic workload management, real time scheduling of jobs in VMs, static and dynamic VM deployment optimal schemes and VMs migration situations. The inspiration of the design of the core entities of the SimIC came from the CloudSim framework; however the classes have been re-designed and considerably extended to include additional features. Principally, SimIC includes a variety of entities that have been modeled for achieving a diversity of meta-computing inspired requirements as follows:

- Large-scale distribution of job requests among meta-brokers as happens in grid systems. In SimIC meta-brokers decide the sub-cloud to execute services by using wide service dissemination algorithms.
- Decentralized topology of meta-brokers including peer-to-peer (P2P) inspired resource discovery. SimIC allows meta-brokers to transfer information and address resource discovery implementations by allowing hashing of meta-brokers ids in P2P networks.
- Static and dynamic management of current workload for each job submission. The cloud (local-broker) is dynamically aware of the current computational capacity for deciding whether to execute jobs locally or forward the request to the personalized meta-broker for further distribution.
- Static and dynamic SLA matchmaking schemes among meta-brokers allow an initial criterion of service execution capability of a cloud.
- Static and dynamic instantiation of VMs with regards to history records. A hypervisor is responsible for deciding whether to generate a new VM (static) or migrate one (dynamic) from a SAN storage device. The decision is based in historical delegation records from previous user submissions to the inter-cloud.
- Real-time job scheduling in VMs according to a variety of heuristic scheduling criteria (e.g. preemptive and non-preemptive cases). The default solution is by triggering entities at regular intervals in order to release deferred queues or to check if the queue length has grown to certain sizes (modeler definition).
- Queuing of VMs according to selected static schedulers. Default developments include first come first serve (FCFS), shortest job first (SJF), earliest deadline first (EDF) and priority scheduling (PS).
- VM migration according to cloud provider requirements. This includes backup of VMs to storage devices in case of emergency.
• Re-active management of heterogeneous service submissions in the form of VMs.
• These requirements are the core of the SimIC framework as they include the key aims of ICMS development.

The next section presents the discrete event simulation to demonstrate the SimJava core classes functionality and the event distribution of the SimIC.

### 5.3.1 Discrete Event Simulation of SimIC using SimJava

SimJava is a discrete event simulation package for developing complex simulations that mimic a number of different processes. In addition, it offers classes to control threads and maintain events in queues for designing advanced scheduling decisions. The general idea includes the design of entities that communicate with each other by sending events that represent messages. Each message could carry a number of information items that are to be utilized by the interconnected entity. Figure 19 demonstrates three entities and their communication, which is initiated between ports.

![Figure 19: Three entities and their communication](image)

**Figure 20: The ICMS SimJava framework**

Figure 20 demonstrates the core features of the SimIC from the perspective of SimJava. The activity diagram demonstrates the starting and the ending points for the communication of two entities in terms of messaging and real-time information retrieval (e.g. current latency). The message initialization happens at time instance 1, and then a message is created at state 1. After, state 2 collects the message (get from out port State 1) and sends the message to in port State 3. During the time instance passes from time 2 to 3 and finally to time instance 4. Finally, the message is terminated (or initialized) from another state in order to continue the information exchanging. In figure 20, the three states are refereeing to one SimIC entity (e.g. datacenter) that creates, collects and pushes the message to the next entity.
The entities and their functionality are organized in a three layer structure as follows.

- **Layer 1** includes the entities representing the objects of the system. In a SimJava simulation each one is represented by a Sim_entity class that encapsulates the core functionalities. Each SimIC class incorporates this design in order to define the actual behavior (layer 2) of entities that are the ICMS resources. Specifically, the core classes are User, Meta-broker, Local-broker, Datacenter, Hypervisor, Hosts, VMs and Bucket. In brief, initialization happens by the user that initializes communication with the meta-broker which represents the user interface. The last one acts on behalf of the user (identical to meta-scheduling systems) in order to forward the request to low level resources (local or remote sites). This procedure is executed by the local-broker that monitors service life-cycle for the meta-broker. The datacenter represents the low level infrastructure and it is the place wherein requests are forwarded to hypervisor for host allocation and VM deployment. In particular, the hypervisor class encapsulates the local policies, e.g. LRMS as well as host and VM scheduling. It should be noted that events are exchanged among entities.

- **Layer 2** shows the behavior of the SimIC that represents the actions happening within the simulator. The core features are the utilization of ports, functionalities and constraints that demonstrate the actual behavior of the entity. Each class contains at least one port for input or output messages to other linked entities. In addition, it incorporates mechanisms for collecting messages, taking decisions (based on policies) and forwarding to the entity decided for delegation. The constraints are the standard features that the entity contains in order to define itself. For example a meta-broker constraint is the linkage to a meta-registry for collecting meta-broker profiles. The actual communication is based on tags that are assigned to messages during exchange and are the means of identifying the origin of the message and the required operation of the responder. The user tag is a class of tags to denote the different user submission; the meta-brokering tag is used for defining the inter-cloud as this class of tags is shared.
among various sub-clouds. The event tag and the dynamic workload are a classification of low-level infrastructure tags. Additionally, queuing refers to the orchestration of events (that are job messages) according to different LRMS (FCFS, SJF, PS).

- **Layer 3** includes the performance and tracing operations that are utilized and produced respectively. The performance measures includes execution time of the VM, turnaround time of service, makespan of the service, throughput of services in an entity, host utilization levels, VM utilization levels, service latencies and VM uptime times. Most of these metrics could be utilized by different entities in order to measure the performance of the SimIC at different instances, or example throughput of datacenter, latency at the hypervisor, etc. Finally, the tracing includes the logging of events and their interchanges, performance results, monitoring of the whole service submission and production of charts and graphs with regards to the simulation case.
Figure 22: The SimIC layered structure
5.3.2 Installation Specification of SimIC

The SimIC (version 1) is based on the process event simulation API of the SimJava version 2 distribution (SimJava 1996). The SimIC (version 1.1) has been developed using the Java™ 2 Platform (JDK 1.6). Also, it includes a JFreeChart 1.0.14 library (JFree 2011), for producing charts and diagrams for the selection of performance metrics in order to monitor performance of entities.

5.4 Architecture of SimIC

SimIC involves automation of service distribution that ranges among decentralized meta-brokers. These are placed on the top of each cloud in order to communicate with others as in a distributed and interoperable topology (e.g. grid computing). The crucial factor of dynamics consideration is implemented by allowing the load of various heterogeneous user specifications (in the form of text files) that contain hardware, software and timing requirements to be uploaded within the simulator. The architecture of the simulator involves a variety of intra-cloud (e.g. datacenter) and inter-cloud entities (e.g. meta-brokers or decentralized resource managers) as well as supporting classes for service distribution, importing user specifications, exporting performance results and drawing simulation charts. In addition, the whole framework has been designed in a segmental format wherein modelers can easily adapt entities; edit their number and relationships as well as select or create various allocation policies by extending current schedulers or creating new instead.

By using this design the study ensures an appropriate solution for implementing various simulation cases in order to identify cloud and inter-cloud meta-scheduling benchmarks. This is the level to compare with novel strategies such as hosts’ allocation policies, VMs dynamic deployments and allocation, service request distribution, etc. SimIC incorporates a variety of user requirements that implement different activities of a distributed cloud system. This design decision increases dynamic factors such as user and service diversity, service elasticity, heterogeneity on resources, scalability of VMs, decentralization and interoperability that are crucial to be defined in order to achieve a supportable simulation. It should be mentioned that the current SimIC version allows simulations of IaaS and SaaS VMs that include the required functionality. The focus of this section is on the entities of SimIC that incorporate ICMS functionality. Specifically, for achieving an efficient design SimIC contains the following classes.
a) The **UserCharacteristics** class instantiates the current service information for each user by incorporating hardware and software requirements as defined in two different text files (txt). The first is the UserHDCharacteristicsFile1 that includes the required hardware resources for user 1, and the second is the UserHDCharacteristicsFile1 that demonstrate the required software specification to be executed within the required hardware from user1. Figure 22 demonstrates the specification as loaded into the SimIC wherein the left part is the hardware while the right part is the software requirements.

![Figure 22: The ICMS SimJava framework](image)

The hardware includes the username, the host operating system, the platform, the memory, the CPU cores and CPU speed, the hardware controller, the hard disk storage, the bandwidth. The software includes the software name, the instructions, the cycles per instruction, the hours, the deadline, the priority and two empty slots for future requirements. Finally, for each of the users, the modeler requires to create pilot files to represent the actual performance requirements. This demonstrates the application requirements and it is utilized as an indicator to the prerequisite CPU power by a user.

b) The **ServiceCharacteristics** class calculates an initial performance request by accepting the user specified program instructions and cycles per instructions part of the UserCharacteristics. These indicators consider the initially required performance (the estimated performance measure to serve as the basis for required computational performance). Initially, the estimated performance is calculated by the number of MIPS as given by the formula

\[
\text{MIPS} = \frac{\text{clock rate}}{\text{cpi}} \times 10^{-6}
\]

(formula 19). This includes the clock rates (CPU) and CPI as given by the user.

c) The **OutputUserRequirements** class generates a dynamic user profile that includes a variety of hardware, software (heterogeneous requirements) and initial performance request measurements (e.g. the millions of instruction per second for a given application(s)). The study defines it as dynamic as it is formed at the time of user submissions and it is terminated after the service submission is finished. Parts of this
profile are accessible from each or specific components of the SimIC v.1. The accessibility is subject to internal information exposition desired levels of the simulation case. In addition, user profiles could be stored in order to add future value in advance scheduling decisions (e.g. based on past user experiences). Figure 23 shows the user profile formation.

Figure 24: The User profile formation

This includes the date of submission, the username to classify username assigned by the system (e.g. 1 denotes that this is the first user that submits), and the user name (defined by the user). The rest of the information is formed according to the UserHDCharacteristicsFile1 and the UserSWCharacteristicsFile1.

d) The **OpenProfile** class accepts each specific user requirement (e.g. user desired CPU, memory, etc.) as defined in the user profile for passing information to the various SimIC entities. Again this is related to the information exposition levels. In general an entity utilizes this class to export data from the user profile and for decision making.

e) The **User** class is responsible to forward a number of requests for resources, wherein each request scheduled after a specific processing delay to a dedicated inter-connected cloud interface, the meta-broker. This relationship is a many (users and requests)-to-one (meta-broker). Each request could be heterogeneous with different configuration, and it is sent after a specific delay in order to simulate the latency of the user entity. In addition, each request is a message that includes information about the job identification, specification, an indication to the user profile, and other information relevant to the simulation data. This is passed to the next entity along with a reference to the position of the requirement files.

f) The **PrintText** class creates a result file that prints each entity submission log, that includes the date, time and current submission specification and simulation time (dedicated delay). In addition, this class is instantiated from various entities that use the
log functionality in order to print specific simulation times that are about to be monitored by the modeler.

g) The **Meta-broker** class implements the interoperability functionality of the SimIC. Specifically, each meta-broker is interconnected with one or more meta-brokers depending on the simulation experiment use case. The addresses are acquired from a **meta-registry** profile. The profile contains the names of the meta-brokers that are Sim_java entity classes. Each remote meta-broker is responsible for a) initially local availability request and b) further service dissemination. However, as this could be an iterative procedure, as requests could travel through the system for a large time frame, the study utilizes a delay termination thread in order to terminate the further service request. This value is decided by the modeler during the simulation setup. The modeler could expand this functionality to address resource discovery based on P2P chord solution, however this solution is not yet implemented in this SimIC version.

In SimIC requests for services could be distributed within an inter-cloud from meta-broker to meta-broker in the case that the local contacted resources cannot serve the performing request or the set of requests (jobs) submitted by the user. This might be due to low computation resources or cloud incompetency on executing certain software specification (limitation on licensing). The SimIC default meta-brokering topology includes that each meta-broker is linked to one (next) meta-broker and so on. Finally each meta-broker is linked to a terminal entity (Bucket) that collects requests that have been unable to be executed in order to keep a log of unfinished jobs. Jobs within the bucket are either forwarded for further dissemination or are terminated depending on the initially chosen experiment configuration.

Figure 24 demonstrates the user profile service distribution wherein a requested meta-broker collects the addresses of the meta-registry prepares the messages and forwards each one to the local resource (local-broker). If the job cannot be executed locally it is forwarded to an inter-connected meta-broker. In any other case, the job is forwarded to the low level infrastructure of the cloud.
h) The **Bucket** class represents the terminal entity that collects the unexecuted jobs and logs job profile information. These could be either re-directed to the inter-cloud after a regular interval or terminated if there is a case of SLA mismatching. It should be mentioned that this class could be instantiated as a terminal to other entities as posed by the scenario case. This implies that events that are possible to travel over the inter-cloud are disposed of in a newly instantiated class.

i) The **Cloud (local-broker)** class includes the SLA matchmaking mechanism for deciding whether the specification of user requirements could be executed to the local resources. In addition, datacenter (host) current performance is dynamically calculated for measuring the available computational power. If there is cloud capability the request(s) is (are) forwarded to the internal cloud entities (e.g. the datacenter). If the cloud local resources are unable to execute the request or some of the set of request(s) submitted by the user, it returns the event or events back to the initiated meta-broker that passes it to the next inter-connected meta-broker for SLA matchmaking. Each event that is returned back to the meta-broker contains the additional latency of the cloud decision making time, as well as the interlinking meta-broker processing time (added to the original delay).

j) The **OpenHost** class imports each host characteristic from a text file to the simulator by allowing the SimIC to access host hardware characteristics (e.g. host name, CPU, cores number, memory, storage, bandwidth etc.), while the **OpenHostsList** opens a list from a text file that contains the individual hosts dedicated to the specific cloud by accessing...
their names. Both classes are instantiated by the cloud for measuring current computation competency dynamically.

k) The **Accounting** class generates the energy consumption and total costs metrics based on the user request for computational resources according to desired VM usage hours and proficiency. This is achieved by opening the user profile that is matched with the user id.

l) The **Datacenter** class accepts events for VMs deployment from the cloud that are determined by a hypervisor. Fundamentally, this class implements an accounting functionality for calculating costs and energy consumption performance measures while it passes all events to a local allocation policy utilizer.

m) The **Hyper** class represents the hypervisor and is responsible for collecting requests for VMs from the datacenter class by accessing the host and VM allocation policies. The first is responsible for the way in which hosts are selected for allocation and the second measures the computational power of a specific host in order to share CPU among current or subsequent VM requests. By default both policies are implemented in a FCFS fashion and their modular structure allows straightforwardly the utilization of other scheduling algorithms. The modular structure of this allows selection among algorithms such as the SJF and PS that have been implemented as parts of the hypervisor (Hyper class). It should be mentioned that a request or a set of requests that cannot be executed directly become delayed requests and are placed in the deferred queue that organizes the scheduling according to the desired scheduling algorithm (LRMS). This is particularly important as it forms one of the key inter-cloud advantages; the capability of the system to accommodation different jobs that could be executed in optimum time according to the local LRMS. The Hyper class finally instantiates the OpenHosts, OpenHostsList, and the OpenProfile classes for calculating the current available computational power and the desired computational resources of the specific event. Finally, this component deploys a VM by considering the VM generation delay. This value is added to the total delay of simulation. It should be noted that discussion of the simulation latency is presented in following sections.

n) The **HyperCall** class generates an internal call thread to the Hyper class in order for the former to release the jobs that have been scheduled in the queue according to the LRMS algorithm. The time interval value is defined by the modeler (static value) or considered by a probabilistic-generated number. In addition, the modeler could generate various
instances of this class according to the desired simulation scenario. Insights on this are presented in the simulation analysis use case.

o) The **HostCharacteristics** class imports each specific host computational capacity as defined in a text file. The SimIC allows the parsing of this information in order to allow the Hyper class to allocate computational units. Figure 25 demonstrates the host specification that includes the name of the host, the host operating systems, the platform, the memory, the CPU cores, the CPU speed, the h/d controller, the storage and the available bandwidth.

```
HostName: Cl.St.Hosts.1
HostOS: Linux
Platform: Intel
Memory (GB): 10
CPU-cores: 4
CPU-speed: 10000
B/D-Controller: CD-DVD
Storage-HD: 1000000
BW: 10000
```

Figure 26: The host characteristics file

p) The **Hosts** class represents a static computing machine. The class gets an event from the Hyper for requesting an instance of the host characteristics. Eventually, this adds an addition delay to the hypervisor decision for allocating a VM. This is the latency of the host for starting job execution.

q) The **VM** class develops the service request execution paradigm that sandboxes the user profile. This is the entity that an event (request) ends after a specific VM processing time. In addition, the VM class generates a result file that principally includes VM specification and performance measures such as event id, event delay, VM name, meta-broker name that executes the request, user name that submits the service, VM execution time, makespan of the VM, energy consumption, and total cost. In addition, user turnaround times (in the case of more than one request), response ratio (turnaround time divided by the execution time of a job), throughput, utilization levels along with average values for each of the aforementioned which are calculated herein.

r) The **VMRescheduler** class allows a VM to be re-selected (re-deployed) for service execution after finishing initial request. This allows a user to re-instantiate the same profile faster than re-developing the VM from scratch. In this way an overall optimization of the VM performance could be observed.

s) The **VMMigrationScheduler** is the class for defining the VM migration strategy among various clouds. Its modular structure allows VMs to be transferred to different cloud SAN devices for various cases (e.g. when high workloads occur or when metrics
approach a standard equilibrium, or in the case of a sensor indication as a trigger for back-up reasons or disaster cases).

t) The **SANStorage** class generates extra hardware space for extending internal hosts storage as well as being used as a temporary saving storage when migration happens. A cloud could have various SAN storage spaces for external VM storage according to the definition of the experiment.

u) The **MigrationSensor** class defines a trigger for starting the VM migration into an external SANStorage space. Specifically, this class is set by the modeler to act on specific or random simulations after a time interval. In advance, the modeler has the opportunity to define other sensors (e.g. heat) that could perform reactively for different situations.

v) The **CreateResults** class generates a log file that contains the performance measures of the specific sub-cloud for various criteria: average delay, turnaround time, average execution time, average makespan, average energy, average cost per hour, throughput, utilization percentage, response ratio, and performance measure.

w) The **DrawVMs (Performance_Metric)**, e.g. the **DrawVMsMakespan** class plots a jpg diagram for each of the selected benchmarks and stores it in a default directory. For that reason the JFreeChart package is utilized.

To conclude, these entities demonstrate the core classes of the SimIC framework. The default relationships of the entities are demonstrated in figure 26. By using the default configuration, the modeler could define different experiments that implement a unique simulation case containing a topology of entities for a required scenario. In addition, a variety of values for different experiments e.g. users, number of jobs, delays, etc. could be defined in simulation classes in order to represents the experimental case.
Figure 27: The default message event model for architecting entities of the SimIC framework.
5.5 Building Simulations with SimIC
The basic simulation includes the job distribution of the default SimIC configuration. The assumption is that 4 users request different requirements (VMs), each user submitting two identical events. The submission happens in an inter-cloud of 4 sub-clouds and each one generates a hypervisor for orchestrating VM allocation on hosts. For the sake of the experiment the study sets up the delay of all components to be 10 ms and utilizes a priority algorithm as queuing solution. Finally, the simulation setting includes that each user enters a cloud according to its id (user 1 to cloud 1, user 2 to cloud 2 etc.), however with different SLA specifications. For example, user 1 requires a specification that can be matched with cloud 1 and cloud 4, user 2 can be matched with cloud 2 and cloud 3, and so on. The job distribution is determined by the meta-broker in order to demonstrate the allocation of jobs to the first capable meta-broker for performing execution cloud.

Each request (whether from the same user or not) is treated by the SimIC as unique. For instance, the user requests a VM with, say, 0.25 of 1 host performance and executes a set of programs with $100 \times 10^6$ instructions, and CPI (cycles per instructions) = 3 (300 cycles /100 instructions) in a machine with clock rate 1000 MHz (0.25 of 4000MHz of Host with single core). The execution time of the VM is calculated as follows:

$$\text{ExecutionTime}_{\text{VM}} = 100 \times 10^6 \text{(ns.)} \times 3 \times \frac{1}{1000} \times 1 = 3 \times 10^5 \text{ns.} = 0.3 \text{ ms}.$$ 

The performance of the VM is calculated to be 3.33 as follows:

$$\text{Performance}_{\text{VM}} = \frac{1}{\text{ExecutionTime}_{\text{VM}}} = 3.33$$

The hosts of the experiment are included in a text file that contains the names of each cloud host. Table 3 demonstrates a typical host list along with the first host configuration of the list.

<table>
<thead>
<tr>
<th>Host List 1</th>
<th>Software Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cl.St.Hosta.1</td>
<td>HostName: Cl.St.Hosta.1</td>
</tr>
<tr>
<td>Cl.St.Hosta.2</td>
<td>HostOS: Linux</td>
</tr>
<tr>
<td>Cl.St.Hosta.3</td>
<td>Platform: Intel</td>
</tr>
<tr>
<td>Cl.St.Hosta.4</td>
<td>Memory(GB): 10</td>
</tr>
<tr>
<td></td>
<td>CPU-cores: 1</td>
</tr>
<tr>
<td></td>
<td>CPU-speed: 10000</td>
</tr>
<tr>
<td></td>
<td>H/D-Controller: CD-DVD</td>
</tr>
<tr>
<td></td>
<td>Storage-HD: 10000</td>
</tr>
<tr>
<td></td>
<td>BW: 10000</td>
</tr>
</tbody>
</table>
Figure 27 shows the output of cloud 1 with regards to the collection of metrics as well as each job that has been executed in this cloud. It should be mentioned that user 1 jobs have the same event id due to their identical configuration.

Figure 28: The log of a typical sub-cloud (cloud 1) of the inter-cloud that includes performance metrics values.

Figure 28 shows the output of the inter-cloud that involves the collection of all jobs being executed. It should be noted that the bucket log is 0 (all jobs have been executed). Also, each Hyper shows the number of jobs that executes within its queue.

Figure 29: The output of the simulator with regards to the inter-cloud job execution pool.

To conclude, this section presented the SimIC capabilities in generating results and outputs. The work demonstrates in detail the algorithms of job distribution along with sequence diagrams that demonstrate event-exchanging aspects. The design and implementation of the solution is based on meta-brokers that are responsible for service dissemination by having spontaneous and dynamic information of the environment. This is more realistic and related to the granularity of an inter-cloud system. The meta-broker profiles the identifiers of other meta-brokers as well as communicates with the local resources for information exchanging. In contrast, centralized and hierarchical schedulers require having a complete knowledge of the actual resource meta-actors, thus representing a non-realistic approach for large size settings. This includes number of hosts, number of
services submitted, workload of each hosts, number of virtual machines (VMs), and topology of the system at any given time.

In contrast, the SimIC implements the ICMS algorithmic structure that relies upon the distributed scheme, and assumes that this kind of information is incomplete and the services received from the meta-brokers are transient and assigned to local or remote hosts (resources). This is inspired by the distributed scheme that allows services to be transferred to distant hosts for achieving a performance criterion (e.g. better local resource utilisation, thus leading to global load equilibrium). In view of that, the ICMS utilizes the meta-brokering architecture for illustrating the inter-cloud service submission, distribution, allocation, and execution orientation.

5.6 Fundamental Performance Evaluation

This section presents the experimental analysis of CloudSim and SimIC when identical user submissions take place. The execution encompasses a basic experiment initially in CloudSim and results are produced based on the configurations in tables 4 and 5.

Table 4: Cloud configuration parameters for input in CloudSim and SimIC

<table>
<thead>
<tr>
<th>Host Requirements</th>
<th>Mips</th>
<th>RAM</th>
<th>Storage</th>
<th>Bandwidth</th>
<th>Host number</th>
<th>Host 1</th>
<th>Host 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment Values</td>
<td>1000</td>
<td>2048</td>
<td>1000000</td>
<td>10000</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 5: User configuration parameters for input in CloudSim and SimIC

<table>
<thead>
<tr>
<th>VM Requirements</th>
<th>CPU size</th>
<th>RAM</th>
<th>Mips</th>
<th>Bandwidth</th>
<th>Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment Values</td>
<td>1000</td>
<td>512</td>
<td>1000</td>
<td>1000</td>
<td>1</td>
</tr>
</tbody>
</table>

Further to this, the results produced by the SimIC are compared with the average execution time of the service submission (presented as VMs) in CloudSim. The performance measures are given by the MIPS formula that calculates the millions of instructions per second (MIPS) as a rate for operations per unit used by CloudSim and SimIC. To identify the performance benchmark of cloud submissions the scenario executes a number of user requests for VMs in CloudSim. This extends from one user to one VM request to 100 users to 100 VM requests. It should be noted that CloudSim shares the computational power of CPU cores in the space-sharing policy, thus the VM execution time is increased for high workloads. In parallel, SimIC utilizes an identical feature that dynamically allocates more resources in order to fulfill the requests.
In this case, both systems perform jobs in a parallel execution trend, however by operating under different policies. For example, CloudSim shares computational power of hosts within a datacenter in order to fulfill all the requests, while SimIC considers a policy for dynamic CPU sharing by considering a latency that increases the VM execution time. In any case, the fundamental benchmark analysis shows that for high workloads (greater than 50 users) both simulators offer corresponding performance with parallel increasing trend lines (VM execution times). Accordingly, based on figure 29 the study suggests that SimIC operates consistent with CloudSim, wherein a slightly optimization of execution time could be observed (mainly due to the low latency of dynamic allocations). Nevertheless, the experiment illustrates that benchmark analysis in both toolkits offers identical result output in VM execution times.

Figure 30: The comparison of CloudSim and SimIC for one cloud specification of 1-100 user submissions for 1-100 VMs.

In order to present the novelty of SimIC the study addresses a more complex objective that allows job distribution among different clouds (based on SLAs acceptance posed by users). In that way it is convenient to implement various realistic scenarios wherein collaborated clouds exchange information on behalf of the user request on time by always ensuring that SLA specification is matched. This is with regards to current execution workload and capacity of sub-clouds to execute certain requests. To explore SimIC behavior the following sections present the implementation of an inter-cloud of 8 clouds wherein various sets of users (16 and 32) submit jobs (10 per user).

The distribution algorithm aims to allocate jobs to clouds that fulfill specific requirements that are sandboxed in VMs. For this use case a scenario is generated with regards to a) runtime decision-making and b) dynamic workload management. The following sections present
four experimental cases that are executed in SimIC within an inter-cloud setting. It should be noted that the inter-cloud meta-brokering topology implies that each meta-broker is interconnected with the next one (e.g. meta-broker 1 to meta-broker 2, meta-broker 2 to meta-broker 3 etc.).

5.6.1 Case 1: 160 Jobs Submitted by 16 Users (Partial SLAs)

In the first case each user submits an identical job specification that can be served only from the clouds 1, 4, 5 and 8. This is because of the heterogeneity factor of the service submissions that require matching with the competency of the system to execute the requests (SLA matchmaking). In addition, the dynamic workload management allows services that cannot be executed locally to be forwarded to capable resources that can offer the computational capacity. In this setting the scenario implements the execution time of the VM that reflects the current system delay (measured as turnaround time). It should be mentioned that the time interval of VM usage is not determined to affect performance (h is set to 1). The second metric is the makespan that demonstrates the sum-up of the VM execution time plus the total delay due to service dissemination. By executing this scenario case, the study extracts results that are presented in figures 30, 31.

![Turnaround times](image)

**Figure 31:** The turnaround and the trend line of turnaround times of SimIC for 160 identical jobs submitted by 16 users (10 jobs per user) - Jobs can be executed from clouds 1, 4, 5 and 8 (SLA matchmaking).

Specifically, figure 30 demonstrates the turnaround time and the polynomial trend line of this value of SimIC for 160 identical jobs submitted by 16 users (10 jobs per user). As initially discussed jobs can be executed from clouds 1, 4, 5 and 8 (SLA matchmaking). The
trend line indicates that as the number of user submissions increase the system tends to offer improved turnaround times by achieving job executions for all the set of jobs.

Figure 31 shows the makespan values of SimIC when 160 identical jobs submitted by 16 users (10 jobs per user) enter the simulators. Again the same constraint includes that jobs can be executed from clouds 1, 4, 5 and 8. In particular, the turnaround times tend to increase due to the distribution of jobs among meta-brokers in order to achieve job execution of the whole input set. In general, the makespan times of the last jobs show an increased tendency to a factor of 0.658 (this is calculated as the division of the average value of result set by 1000), which is considered as an acceptable rate mainly because of the large submission number. Specifically, this will serve as a metric for comparison with next scenario cases.

Figure 32: The makespan values of SimIC when 160 identical jobs submitted by 16 users (10 jobs per user) - Jobs can be executed from clouds 1, 4, 5 and 8 (SLA matchmaking).

5.6.2 Case 2: 160 Jobs Submitted by 16 Users (Full SLAs)

This case scenario involves the experimental input of 160 identical jobs submitted by 16 users (10 jobs per user) wherein all clouds can offer job execution. However, in this case the dynamic workload management defines the dissemination functionality. This involves that if a cloud cannot execute the job due to limited resources then it sends the job back to the meta-broker for further dissemination.

Figure 32 shows the turnaround and the polynomial trend line of the turnaround times of SimIC for 160 identical jobs submitted by 16 users (10 jobs per user) where all clouds can offer job execution. It is apparent that the turnaround polynomial trend line shows an increasing trend for 50 to 100 job submissions; however for 100 to 160 the line shows a decreasing rate. That is considered as an improvement because the system tends to offer better performance (decrease turnaround time trends) for peak workloads.
Figure 33: The turnaround and the trend line of the turnaround of SimIC for 160 identical jobs submitted by 16 users (10 jobs per user) - all clouds can offer job execution (SLA matchmaking).

Figure 33 presents the makespan values of SimIC when 160 identical jobs submitted by 16 users in an identical case as previously. The chart shows an increasing trend of the makespan with ratio factor of 0.59 that it is marginally lower compared with case 1.

Figure 34: The makespan values of SimIC when 160 identical jobs submitted by 16 users (10 jobs per user) - all clouds offer job execution.

To conclude, this section compares identical cloud configurations that are implemented in CloudSim and SimIC respectively in order to show the parallel performance tendency of both toolkits. For achieving experimentation on dynamic and real-time multi-user submissions within an inter-cloud the study presents an extended experimental analysis of four cases that offers productive results. In particular, for high workload submissions with a partial
capability of service execution of clouds the system shows decreased values of selected benchmarks.

In general SimIC allows modelers to configure a diversity of inter-clouds in terms of datacenter hosts and software policies wherein desired number of users could send single or multiple requests for computational power (cores, CPU, memory, storage, bandwidth), software resources (measured empirically in cycles per instruction and millions of instructions per second) and duration of VM utilization. The toolkit contains a selection of meta-scheduling inspired characteristics for achieving job dissemination, resource discovery services, dynamic workload management, real time scheduling of jobs in VMs, static and dynamic VM deployment policies and VMs migration cases.

5.7 Summary
This chapter illustrates the simulation toolkit SimIC (version 1.1) for simulating inter-cloud environments. The design and implementation of the solution is based on meta-brokers that are responsible for service dissemination by having spontaneous and dynamic information of the environment. The next chapter focuses on the performance evaluation.
Chapter 6: Performance Evaluation

6.1 Outline
Further to ICMS model and its optimal schemes shown in chapters 5 and 6, this chapter presents the performance evaluation of the models, algorithms and schemes. The experimental design section describes the definition and the organization of the different sub-experiments along with use-cases. Each experiment is detailed along with a discussion of the simulation configuration.

6.2 Experiment Design
The section introduces the definition of the experiment and the analysis of each of the use cases as follows.

a) The ICMS benchmark and performance analysis demonstrates the implementation of the centralized inter-cloud setting of Buyya et al. (2011) and the decentralized ICMS in the same simulation setting (SimIC). By focusing on the job distribution on both cases, the study illustrates the configuration of meta-brokers and clouds, the discussion of metrics and evaluation of both results. In brief the use case encompasses a variation of user job submissions in an inter-cloud of 5 clouds. Each cloud has been configured to different resource specification. In the centralized inter-cloud of Buyya et al. (2011) the assumption is that all clouds have complete access to the inter-cloud facility making this the benchmark for comparison. In contrast, in the decentralized inter-cloud based on the ICMS the assumption is that the cloud topology is random, so clouds have incomplete and partial access to other clouds.

b) The service distribution based on MEO includes the implementation of a centralized and a decentralized inter-cloud in SimIC. The focus is to demonstrate the comparison of AllToAll (used as benchmark) and MEO models. The single-level message distribution represents a centralized system (e.g. a sub-cloud of an inter-cloud or an LRMS), while the multi-level message distribution shows a decentralized system (e.g. an inter-cloud). The first use case includes a centralized inter-cloud of 3 clouds and a comparison of AllToAll, MEO and MEO with compression. The second use case includes an inter-cloud of 7 clouds and a comparison of AllToAll, MEO and MEO with compression. In particular the second use case includes a variation on resource availability in sub-clouds in the level of 33%, 50% and 100%. The experiment compares configurations and evaluates performances.
c) The VM deployment optimal scheme supports the decision process of cloud hosts for the deployment of virtual machines in the inter-cloud environment. The study explores two configurations; the static case in which virtual machines are generated according to the cloud hosts, and the dynamic case in which virtual machines are dynamically created based on past job submissions, using migration. The solutions are implemented in SimIC for measuring the performance of various combinations of virtual machines, jobs and hosts. In brief the use case encompasses a variation of user job submissions in an ICMS based inter-cloud of 5, 10, 15 and 20 clouds. The assumption is that in the static case (used as a benchmark) the VMs are generated on the fly by the hypervisor. The last one stores VM submissions in a local storage device. The dynamic case includes a submission that shares similar requests, so instead of creating new VMs the optimal scheme migrates instances that match with the required ones from the storage device. The experiment compares configurations and evaluates performances.

Next, the study focuses on the ICMS experimental configuration (based on SimIC) and the scenario cases.

6.3 ICMS Experimental Configuration

This section demonstrates the experimental configuration of the ICMS simulation cases. Appendix B table 4 details the users’ submission that include user name, host operating system, platform, memory, CPU cores and CPU speed, installed hardware controllers, storage hard disk drive, available bandwidth, software specification (availability), job instructions, CPI, resource usage duration (in hours), deadline time, and priority. The priority value is considered only in the case of the priority-scheduling algorithm. Appendix B table 5 shows the cloud hosts. For example, cloud 1 has 16 hosts, each of which has a different configuration. Appendix B table 6 shows the cloud host’s specification.

The next section presents the ICMS specification and presentation of simulation results as extracted from the SimIC simulation toolkit. It is divided into four subsections in order to demonstrate that a) inter-cloud (IC) performs better than or equal to a non inter-cloud (non-IC) setting, b) ICMS performs better than or equal to IC (for the case of 1 job per user and non-dynamic workload management optimal scheme), c) ICMS performs better than or equal to IC (for the case of 50 jobs per user for non-dynamic workloads), d) ICMS with dynamic workload management outperforms non-dynamic cases for a mixing of user submissions with 100% availability. It should be noted that the default LRMS for the experimental scenarios of section 6.3 is set to FCFS. Figure 34 shows the map of the experiment.
6.3.1 Cloud vs. Inter-Cloud Settings

The first experiment aims to demonstrate that IC performs better than or equal to non-IC setting for the case that both settings have exactly the same computational configuration. This includes that in non-IC, cloud 1 has precisely the same capacity (CPU, memory, storage and bandwidth) with an IC that is comprised from two clouds, cloud 1 and cloud 2. For that reason the study adapts the cloud 1 host specification of Appendix B table 5. For the IC case, cloud 1 and cloud 2 are set to 8 hosts correspondingly. By this way the comparison is considered analogous.

Figure 35 demonstrates the topology of the cloud (e.g. cloud 1) when 5 users submit 10 to 50 jobs with 2 milliseconds (ms) latency from each other. Specifically, the first user submits at 0 ms, the second at 2 ms, the third at 4 ms, the fourth at 6 ms and the fifth at 8 ms. Each request is redirected into a datacenter from the local-broker in case of SLA matchmaking) and then the hypervisor manages the VM generation process. Each time a request arrives in the hypervisor, a new VM is generated according to the available resources. If resources are incomplete, the request is transferred to a container class that collects unexecuted requests for monitoring reasons.

Figure 36: The topology of the cloud setting
Figure 36 demonstrates the topology of the inter-cloud when 5 users submit 10 to 50 jobs. This setting encompasses the IC collaboration scheme of Buyya et al. (2011) where jobs are moving from datacenter to datacenter (cloud 1 to cloud 2). Each time the SLA matchmaking happens according to a datacenter optimal scheme that controls the available resources in the cloud datacenter. Similarly to the previous case, if a cloud cannot execute a job then it forwards it to a container class. Both cases (cloud and IC) share the same utilization model. This is to say that resources will be allocated if they are available and to the higher utilization level (100%).

For both cases (cloud and IC) all jobs are executed in local clouds, as there is no option for further job distribution, thus making solutions centralized. The simulation completes 10 to 50 jobs per user that are submitted in a sequence defined by the interval of 2 ms.

Figure 37: The inter-cloud topology (datacentre to datacentre communication)

Figure 38: Makespan for 10-50 jobs (non-IC vs. IC)

Figure 37 shows the makespan values and the linear trend lines for 10 to 50 jobs. It is shown that the IC trend line increases at a higher rate when compared to the cloud case, however the values remain lower. It is clear that as the number of job increases, the IC will increase the makespan value due to the latency (set to 2 ms) that is caused by the job
exchanging. In particular, the study sets this value to a low number (2 ms) because of the assumption that cloud 1 has been split over two datacenters.

Figure 38 shows the execution times (10-50 jobs) for both cases. The average value for non-IC is 5.79 ms while for IC is 5.38 ms. This demonstrates that IC decreases the average value due to the better allocation of the IC. The improvement factor for IC is calculated at 7% (percentage of the division of the difference of higher value to lower value to the higher value). This demonstrates that IC optimizes the average execution times due to the better resource allocation managed by the hypervisor.

Figure 39: Execution times for 10-50 jobs (non-IC vs. IC)

Figure 39 shows the turnaround times (10-50 jobs) for non-IC and IC settings. In this case the average value of the turnaround times is 388 ms for the non-IC and 380 ms for the IC case.

Figure 40: Turnaround times for 10-50 jobs (non-IC vs. IC)

Specifically, the IC case marginally outperforms the non-IC case due to the increased latencies in job distributions of the IC case. The improvement factor is calculated at 2%. This shows that even in the worst case of increasing delays (due to the job transferring from cloud 1 to cloud 2), the optimized execution times keep turnaround times in low levels (the IC
turnaround difference is 8 ms). Figure 40 shows the response ratio times (users 10-50) for non-IC and IC cases.

![Response ratio 10-50 jobs](image)

**Figure 41: Response ratio times for 10-50 jobs (non-IC vs. IC)**

In this study, the response ratio defines the performance value of the experiment. Specifically the statement is that the higher the response ratio the better the performance. The average values have been calculated to 1744.2 for the non-IC case and 1847.1 for the IC case. It is obvious that the IC outperforms the non-IC case by achieving an improvement factor of 6%. This demonstrates that IC outperforms non-IC in the case of same configuration. Figure 40 demonstrates that the response ratio for non-IC offers lower performance when compared to IC for most of the user submissions.

It should be mentioned that due to the high user demands some jobs could be dropped due to resource unavailability. Each non-executed job is forwarded to the container to keep monitoring logs. For instance, in the case of 100 jobs, the non-IC case executes 38%, while the IC executes 41%. Since, neither case implements a dynamic workload management, jobs that cannot be executed locally are directed to the container.

![Throughput percentages](image)

**Figure 42: Throughput percentages for 10-50 jobs (non-IC vs. IC)**
Figure 41 shows the throughput percentages for both cases. The values for each submission are close, so the amount of jobs that are executed are different slightly. On average, the IC throughput of the IC gives an improvement factor of 5%. Figure 42 shows the average execution time and average utilization rates for both cases. It indicates that the average utilization of the IC case is 37.2% and the non-IC is 35.4%. The average execution time shows decreasing value for the IC case.

![Execution time & Utilization (non-IC vs IC)](image)

Figure 43: Average execution time and average utilization for 10-50 jobs (nonIC vs. IC)

To conclude, the following considerations have been highlighted during the comparison of non-IC and IC settings.

- The average execution times have been decreased for the case of IC of 10-50 job submissions.
- The makespan times in the IC case are marginally increased (average calculation) and the linear trend lines show a slightly increasing rate. This is because the optimized execution times keep the overall makespan in low levels.
- The turnaround times are slightly increased (average calculation) mainly because of the latency throughout job exchanging.
- IC increases utilization levels because it executes more jobs (37.2% when compared to the 35.4% of non-IC).
- The average execution times are slightly decreased in IC due to enhanced job allocations. This is related to the number of hypervisors (2 for IC) that achieve optimized job distributions.
- The response ratio values are equal or lower (for case of 30 jobs); thus performance is slightly decreased for non-inter-cloud for cases of 10, 40 and 50 jobs. Performance is significantly decreased for 30 jobs. However, the average performance has been optimized for the IC case.
This discussion concludes that the average performance of the simulation experiment has been increased when the same configuration is taking place in non-IC and IC settings. So IC performs better than or equal (in case that the inter-cloud is comprised by 1 cloud) to a non-IC setting.

6.3.2 The Inter-Cloud vs. ICMS Setting (1 Job Submission per Cloud)

The second experiment aims to demonstrate that ICMS performs better than or equal to the IC setting for the case of the same host configuration. This is achieved due to the origin of the ICMS topology that allows job distribution based on meta-broker decisions. Figure 43 shows the IC topology of 5 clouds where 1 user submission takes place. The proposed meta-brokers are generated by the datacenter that represents the actual inter-cloud communication link. For each user the datacenter binds a meta-broker that is responsible for controlling its requests and SLAs, and monitoring the whole service allocation and execution. Thus, the meta-broker provides an autonomous management component that characterizes the entry point of the cloud from the user viewpoint.

![Diagram of IC topology of 5 clouds and 1 job per user (5 users)](image)

Figure 44: The IC topology of 5 clouds and 1 job per user (5 users)

In addition, each meta-broker has a complete knowledge of the actual cloud infrastructure (e.g. datacenter characteristics, Hosts, VMs) as during job submission it communicates with the cloud broker for information exchange. However, the meta-brokers can have a complete or partial knowledge of other meta-brokers (related to the specific scenario) during service
run-time. This characterizes a setting as centralized or decentralized. This perspective offers a high transparency level for the entire cloud since the users are only mapped to their assigned meta-broker, while the last one spontaneously directs the process. Different from the solution of Buyya et al. (2011), this study realizes ICMS that uses meta-brokering operations. This moves the complexity of handling service requests from the datacenter to the meta-broker. In this way, it could identify available resources more easily and reactively check for service execution opportunities.

The experiment includes an IC of 5 clouds that have the same host specification with the ICMS. For this the study adapts the clouds 1-5 host specification of Appendix B table 5. In this way the comparison is analogous. This experiment demonstrates makespan, turnaround, response ratio and execution times for 1 job submission per user. The delay is set to 2 ms for all components of the simulation. Each user submits a job after the other with an interval of 1 ms. All datacenters can access other datacenters directly in a completely centralized solution. Figure 44 shows the ICMS topology of a decentralized solution. Specifically each cloud meta-broker can access the next in the list. For example, meta-broker 1 sends a job in case of job distribution to meta-broker 2, then meta-broker 2 sends to meta-broker 3, etc.

Figure 45: The ICMS topology of the decentralized meta-brokers

For both cases, the default case is that each job that is submitted cannot be executed in the local cloud so it is always forwarded to remote cloud(s). The availability is set that each job can be executed in the next cloud (e.g. job 1 to cloud 2, job 2 to cloud 3 etc.). In the
centralized case the assumption is that all clouds can access all other clouds directly, while for the decentralized case the topology of figure 44 is assumed. Figure 45 shows the makespan times for 1 job submission per user case with 1 ms interval. It is apparent that the values are decreasing for the ICMS case. The average value is calculated to 519 ms, while the ICMS is measured to 534 ms; that shows an improvement of 15 ms in the average values. The improvement factor for this case is calculated at 3%.

![Makespan times graph](image)

Figure 46: Makespan times for 1 job per user (IC vs. ICMS)

Figure 46 shows the turnaround times for 1 job submission for both cases. The average value for ICMS is 524 and for IC is 539. So, the centralized case increases the turnaround times mainly because of transferring jobs among datacenters (this includes a higher number of internal components). The highest difference of turnaround times is observed in cloud 5 where ICMS sends jobs to the hypervisor in 521 ms, while in IC the same submission requires 548.4 ms with an improvement factor of 5%.

![Turnaround times graph](image)

Figure 47: Turnaround times for 1 job per user (case 2a: IC vs. case 2b: ICMS)

Figure 47 demonstrates the response times of IC and ICMS. It is apparent that for continuous job submissions the response times (the time needed to react to a submission) are decreased for the ICMS case. Also the decreasing rate is significantly improved as new users submit new jobs (e.g. for cloud 1 the difference of response times is 3 while for cloud 5 it has
been increased to 27). The average value of ICMS is 513 ms while for IC is 528 ms so giving an improvement factor of 3%.

Figure 48: Response times for 1 job per user (IC vs. ICMS)

Figure 48 shows the ICMS execution time, energy, cost and utilization levels. It should be noted that IC and ICMS execution times and utilization levels are exactly the same (e.g. 20% for each cloud) as the jobs are allocated the same resources. Also, all jobs (5 in total) are executed in the dedicated clouds that meet the software specification. The figures shown here will be further described and enhanced by the usage of novel optimal schemes in following sections. Figure 48 details that ICMS does not affect the overall cost and energy spend on cloud local hosts for executing requests. It is apparent that the ICMS energy and total costs are increased during job distributions.

Figure 49: ICMS execution time, energy consumption, total costs and utilization levels

Figure 49 shows the performance comparison of both cases in terms of response ratios. The figure shows that the performance increases for ICMS as more users submit more requests in a linear manner. However, as requests are transferred to remote clouds (e.g. cloud 2, cloud 3 etc.) the ICMS performance decreases with regards to the initial ICMS performance. This is because of the increasing makespan time. Nevertheless, ICMS always
outperforms IC for this experimental case. The improvement factor starts from 1% (for cloud 1) to 5% (for cloud 5).

![Response ratio comparison](image)

**Figure 50: Comparison of performance (response ratios) of ICMS vs. IC**

To conclude, the following considerations have been highlighted during this experimental study.

- Makespan times of ICMS have been improved for the collection of the clouds (improvement factor 3%).
- Turnaround times have been decreased for ICMS (improvement factor of 3%).
- Response times have been optimized for ICMS and the response ratio trend line is moving in a decreasing trend compared to the IC solution.
- The clouds for ICMS have the same utilization (20% each) as the IC, thus the allocation does not affect the use of resources.
- The comparison of performance figures shows that ICMS outperforms inter-cloud. The IC performance is worse as it attracts 97% of the performance of the ICMS. In addition, during the remote cloud invocations the ICMS performance increases by a factor of 1%. For example the improvement for cloud 1 is 1% and for cloud 2 is 2% etc.
- The comparison of the performance results for each submission shows that the ICMS optimizes figures each time a new job enters the IC (the average improvement rate is measured at 3%).

### 6.3.3 The Inter-Cloud vs. ICMS Setting (50 job submissions)

This experiment demonstrates the simulation results for high workload submissions (50) per user. As more jobs are exchanged in a busy setting the resource availability and allocation management become more complex. It should be noted that the topology and configuration of IC and ICMS are the same as presented in figures 43 and 44 of section 6.3.2. In order for the results to be comparable the study associates clouds with exactly the same utilization levels
Jobs that cannot be executed due to non-resource availability or SLA mismatching are dropped, as the dynamic workload is inactive.

Figure 51: Makespan times for 50 jobs per user for clouds with same utilization (clouds 3, 4)

Figure 50 shows that the makespan times for 50 jobs per user are significantly optimized results for ICMS and clouds 3 and 4 (same utilization levels). The average makespan time for IC (clouds 3 and 4) is 639706 ms while the same metric value for ICMS (clouds 3 and 4) is 638806 ms (900 ms difference). Figure 51 shows the makespan for clouds with low utilization of 6% (clouds 2, 5). Again, ICMS algorithms offers lower makespan times when compared to IC.

Figure 52: Makespan times for 50 jobs per user for clouds with same utilization of 6% (clouds 2, 5)

Figure 52 demonstrates the response times for clouds 3, 4 (20% utilization). The response times have marginally improved. The observed improvements are 9 ms for cloud 3 and 15 ms for cloud 4. These figures show that ICMS offers slightly better response times compared to IC.
Figure 53: Response times for 50 jobs per user for clouds with same utilization (clouds 3, 4)

Figure 53 shows the turnaround times (50 jobs per user) for ICMS and IC cases. It is presented that the values have been significantly improved (e.g. ICMS average turnaround for clouds 3 and 4 is 638806 ms and IC is 639706 ms).

Figure 54: Turnaround times (50 jobs per user) for clouds with same utilization (clouds 3, 4)

Figure 54 demonstrates various IC and ICMS metrics for all clouds for the cases of IC and ICMS. It is shows that the cost and energy consumption of cloud hosts remain at the same levels for both cases, thus ICMS does not affect these metrics.

Figure 55: ICMS utilization, execution times, energy and cost figures

Figure 55 demonstrates the performance comparison of clouds 3 and 4. The IC performance significantly decreases while ICMS offers better performance for both clouds.
Figure 56: Performance comparison for 50 jobs per user for clouds with same utilization (clouds 3, 4)

Better performance is also observed for ICMS and cloud with low utilization (6%). Figure 56 shows that performance is decreased. Especially for cloud 5 the values have decreased when higher delays are included.

To conclude, the following considerations have been discussed in this experiment.

- ICMS offers optimized makespan times for busy clouds with similar utilization (20% clouds 3, 4 and 6% of clouds 2, 5).
- The response times have decreased for ICMS (both utilization cases).
- The turnaround times have significantly decreased for ICMS (900 ms faster job execution for ICMS).
- The cloud utilization levels vary with regards to job requirements and SLA availability, however remain the same for both cases, thus ICMS does not affect it. The execution times remain at the same levels for both cases, while energy consumption and total costs have been increased for both cases.
The performance comparison shows that ICMS outperforms inter-cloud. The trends show that for clouds 3, 4 and 2, 5 (that have the same utilization for both cases) the ICMS performance is constantly improved.

6.3.4 The ICMS Setting: Low and High Delays and 40% to 100% Resource Availability (Dynamic Workload Management)

This experiment demonstrates the dynamic workload management for an ICMS case. The decentralized ICMS sends the jobs to different clouds by incorporating dynamic distribution. To illustrate the increasing delays the study executes a combination of 1 ms to 4 ms delay and 40% to 100% resource availability. The percentage is related to the ability of a cloud to execute the specific job portion; specifically, the 40% availability is selected as it demonstrates a cloud with low resource availability. The following list shows the mixing of the different combinations.

i. 1-40%: delay 1 ms, availability 40%
ii. *4-40%: delay 4 ms, availability 40% (where * details that delay is 4 times higher than the first case)
iii. 1-100%: delay 1 ms, availability 100%
iv. *4-100%: delay 4 ms, availability 100%

Figure 57 shows the makespan times of ICMS for each of the four cases. It is shown that when the availability is 40% ICMS distributes jobs to all clouds, however in case of 100% availability, cloud 4 executes most of the jobs. This is because of the high number of hosts that are in cloud 4 (21) which increases the computational power.

![Figure 58: Makespan times for ICMS cases](image)

Figure 58 shows the execution times for each of the cases. For 40% availability the execution time is measured at 4 ms which shows a marginally increasing figure. Similarly, for the case of 100% the execution times show slightly decreased numbers for high delays.
The results show that for higher delays ICMS slightly decreases its execution times, thus achieving optimized performance.

Figure 59: Execution times for ICMS cases

Figure 59 shows the overall resource utilization levels of clouds 1 to 5 for cases i to iv. Specifically the highest utilization is found in case 3 that details execution of all the jobs with high resource availability. It is also demonstrated that case 2 involving high delays decreases the utilization levels (due to the increasing delay when continuous submissions occur).

Figure 60: Overall utilization levels for ICMS cases

Figure 60 shows the response ratio (that represents the performance) of the four cases. It is apparent that for the case of 40% the best performance is achieved when ICMS involves low delays. Similarly, for 100% availability the best performance is observed for cloud 4 that encompasses high delays. This shows that ICMS offers better performance for high delays due to the job distribution of the dynamic workload management. The improvement rate is calculated at 13% in case of quadruple delay. The dynamic workload management involves workload calculation that it is related to up-to-date host capacity in terms of CPU, memory, storage and bandwidth. Similar, to real-time solution this involves a re-active workload request management for each job submission to the local-broker. The improvement rate for cloud 5 (case of 40% availability) is 3% for 1 ms delay.
6.3.5 The ICMS Setting: A Mixing of User Submissions and 100% Resource Availability

This experiment demonstrates ICMS performance for a combination of user submissions. Specifically, in the first case 50 jobs are submitted to cloud 1, while in the second case 10 jobs are submitted to each cloud by a user (total of 50 jobs).

Figure 62 shows the successful execution percentages when comparing 1 user per cloud and all users in cloud 1. It is shown that the user distribution in different clouds offers better successful execution percentages. Thus, this shows that in an inter-cloud, the spreading of users in different clouds could assist in order to achieve higher successful execution percentages. Figure 62 shows the makespan and turnaround times. It is shown that for high number of job submissions, ICMS makes a better distribution by allocating resources more efficiently (lower makespan times). In addition, turnaround times for high workload have been sufficiently decreased. For example cloud 2 makespan times shows an improvement rate of 4.9%. The highest turnaround time is the case of all users in cloud 1 that is measured in 1389203 ms.
Figure 63 presents the utilization figures of clouds 1 to 5 for both cases. It is seen that the case of 1 user per cloud increases utilization of resources of clouds 3 and 4 while reducing the utilization level of cloud 5.

Figure 64: Utilization comparison for both cases

This section concludes the ICMS performance evaluation. Overall the section demonstrates that ICMS improves the performance of IC even in the case of decentralized setting for metrics of makespan, turnaround and response times. In addition, ICMS achieves optimized figures for high workloads (busy inter-clouds) that involve high delays due to the sophisticated dynamic workload management technique. To conclude, the following considerations have been discussed in this experiment.

- ICMS improves makespan values and turnaround times.
- For the same configuration ICMS offers identical execution times and it does not affect energy consumption and generated costs.
- The response times have been optimized for ICMS (improvement factor 5%) and the response ratio trend line is decreasing compared to the IC solution.
• For the same configuration ICMS does not affect utilization (the same utilization of 20% for each case is observed).
• The comparison of performance figures shows that ICMS case of user’s submissions to dedicated clouds outperforms the ICMS case of all users to cloud 1. The average response ratio for ICMS (dedicated clouds) is 1778 and for ICMS (all in cloud 1) is 5324 which demonstrates a satisfactory optimization.
• The performance comparison for each submission shows that the ICMS optimizes figures each time new jobs enter the inter-cloud, and especially for distributed submissions with high latencies.

This concludes the comparison of the ICMS experimental cases. Next the study focuses on the service distribution optimal scheme of MEO.

6.4 Service Distribution using MEO

This section presents the experimentation of various inter-cloud simulation cases in order to compare the message models (AllToAll and MEO) and their adaptation to ICMS. The models have been designed and implemented in SimIC and the configuration includes a diversity of inter-clouds in terms of datacenter hosts and software. In this setting a number of users could send single or multiple requests for computational power. This includes CPU cores, CPU, memory, storage, bandwidth, software resources (measured empirically in cycles per instruction and millions of instructions per second) and uptime of Virtual Machines (VM) utilizations. To demonstrate the effectiveness of the approach the focus is on the centralized and decentralized topologies that represent the IC and ICMS respectively as discussed in section 6.3.

6.4.1 The Centralized IC Topology

The single-level message distribution depicts a centralized system (e.g. a sub-cloud of an inter-cloud as presented in 6.3.1 or a local resource management system – LRMS). In such a case a collection of nodes (typically the users) submits jobs to the centralized component (e.g. a local-broker) that forwards requests to internal resources for resource availability. Figure 64 shows the topology of the three cases: AllToAll, MEO and MEO with compression.

By utilizing this topology the study integrates three cases to simulate AllToAll, MEO and MEO with file compression. Specifically, the AllToAll case is selected as the conventional cloud message-exchanging scheme. The ranking criteria define the cost of communicating that is measured from the total delay. This, as presented in sections 6.3.4 and 6.3.5, directly affects makespan and turnaround times, thus performance of the user submission. In addition,
the analysis presents the improvement factor of the MEO model when compared with the AllToAll approach. The MEO model with compression moves the MEO model a step further in order to minimize the cost of the message size during communication.

Figure 65: The three centralized model simulation cases: a) AllToAll, b) message-exchanging optimization and c) message-exchanging optimization with compression

6.4.2 The Decentralized ICMS Topology

The multi-level message distribution shows a decentralized system (represented by the ICMS). Here, the user submits jobs to a node (decentralized meta-broker) that forwards the request to inter-connected meta-brokers for further distribution. The latter redistributes the request to other meta-brokers that could follow the same procedure. The configuration includes a two-level topology where the study implements the three cases as previously. Figure 65 illustrates the topologies of the inter-cloud configuration that includes decentralized meta-broker communication for MEO.

Figure 66: The three decentralized simulation model cases include: a) AllToAll, b) message-exchanging optimization and c) message-exchanging optimization with compression
For each of the cases there are two sub-experiments where the resource availability has standard levels as follows. The first sub-case details that a request that arrives in the execution level (third level) will have 50% resource availability. The third level details a cloud that could execute the job (based on SLA matchmaking and resource availability). For example, the request is forwarded from cloud 1 to cloud 2 and finally to cloud 3. The second sub-case includes that the resource availability (again at third level, e.g. cloud 3) will be 33%. In this way, the study demonstrates how the metrics are affected when the system has medium or low resource availability. This allows the study to optimize the overall message exchanging delays, thus improving the performance of the ICMS.

6.4.3 Experimental Comparison and Analysis

This section presents the configuration of the setting and the inter-cloud cases that are included in the simulation toolkit (SimIC) that implements the algorithms and produces the experimental results. Figure 66 shows the flow of the experiment by an the experimental map that includes evaluation of centralized and decentralized simulations that is turned to compare IC with ICMS.

![Experimental map in SimIC](image)

Figure 67: The experimental map in SimIC

It is expected that the combination of single- and multi-distribution level along with variations of resource availability will demonstrate the effectiveness of the model. Table 6 shows the centralized topology specification that characterizes the experimental case of a typical cloud. For this case the job length is given by the MIPS value. The use of MIPS allows the study to define a job size metric. Also, the compression rate for both experiments is set to 29%, while the compression delay is 1.8 ms and the decompression delay is 3.8 ms.
Table 6: The centralized use case specification

<table>
<thead>
<tr>
<th>Clouds</th>
<th>Users</th>
<th>Average Delay</th>
<th>Number of jobs</th>
<th>Size of each message</th>
<th>MIPS</th>
<th>CPI</th>
<th>Bandwidth</th>
<th>Power (Watts)</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1-5</td>
<td>10 ms</td>
<td>10</td>
<td>5 kb</td>
<td>100</td>
<td>2</td>
<td>3</td>
<td>512 kbps</td>
<td>300-500</td>
</tr>
</tbody>
</table>

Table 7 shows the centralized topology specification that characterizes the experimental case of a typical cloud. The availability is set to 0% for the second level of resources (demonstrating resource unavailability or SLA mismatching). In contrast the availability is set to 50% (sub-case a) and 33% (sub-case b) for all resources of the third level.

Table 7: The decentralized use case specification

<table>
<thead>
<tr>
<th>Clouds</th>
<th>Users</th>
<th>Average Delay 1st levels</th>
<th>Average Delay 2nd levels</th>
<th>Average Delay 3rd levels</th>
<th>Number of jobs</th>
<th>Size of each message</th>
<th>MIPS</th>
<th>CPI</th>
<th>Bandwidth</th>
<th>Power (Watts)</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1-10</td>
<td>10 ms</td>
<td>30 ms</td>
<td>20 ms</td>
<td>2</td>
<td>10 kb</td>
<td>1000</td>
<td>2</td>
<td>4</td>
<td>512 kbps</td>
<td>100</td>
</tr>
</tbody>
</table>

Figure 67 demonstrates makespan and turnaround times of the AllToAll approach and of the MEO model for the centralized experiment. Although the two vertical axes have scales which vary by a factor of 10, the difference in the slope of the trend lines is very clear; the trend lines show that MEO increases at a lower rate compared to AllToAll.

Figure 68: Comparison of the energy consumption rates for AllToAll and MEO

It is shown that the makespan and turnaround times are well optimized for the case of the MEO model. Figure 68 shows the up/down bars that indicate the optimized degree of the MEO approach for the makespan value. This is because 10 messages are transferred for each
new user that submits 10 jobs. In contrast the number of messages is lower, so MEO indicates improved rates with a low-increased trend line.

![Makespan times graph](image1)

**Figure 69**: Comparison of the makespan for AllToAll and MEO and up/down bars

Figure 69 shows the total delay of the AllToAll and MEO models for both cases. It is apparent that the increasing trend lines indicates lower time delays for MEO as depicted in second y-axis (the scale for the MEO model). In addition, the increasing slope of the MEO trend line is lower than the AllToAll approach.

![Total delay variation graph](image2)

**Figure 70**: Comparison of total delay times and polynomial trend lines for AllToAll and MEO

Figure 70 illustrates the energy consumption rates for the MEO with file compression case. The same rates are selected for the AllToAll experiments as well.
Figure 71: Comparison of energy consumption rates of MEO with regards to size cost

The study compares non-compression against compression cases in a centralized system. The conclusion is that as the number of users increases, the number of messages that contain compressed lists of job descriptions is optimized. The file size factor (y axis) describes how the list size (of collected job descriptions in kB) affects the energy consumption of the communication channel in terms of available bandwidth. For the decentralized case, Figure 71 demonstrates the MEO delay times in multilevel job distributions. It can be seen that as the number of users increases (thus their job submissions e.g. 2 per user) the total delays for low resource availability (33%) has been optimized.

![Energy Consumption Rates](image)

Figure 72: MEO total delay times in decentralized multilevel submissions (up/down bars for 33%, 50%, 100%)

Based on these results, the study concludes that MEO model optimizes its delay rates for low resource availability cases. This is particular useful for large-scale dynamic systems where multiple users request job allocations. So as the allocation number increases, MEO will offer low delays to remaining job submissions. Figures 72 and 73 show the comparison of the 50% and 33% resource availability energy consumption improvement rates of MEO.
Specifically, for the case of 50% availability, the polynomial trend line decreases over time; however it remains in the range of 0.49 to 0.50. However, the second sub-case of 33% availability shows increasing rates for high number of users, but remains under the 50% rate (lower than 0.251), so offers improved results. It should be noted that this particular output is for the inter-cloud of 7 sub-clouds where 10 users submit 2 services one after the other with a delay of 10 ms. Figure 74 illustrates the comparison of the value in kilobytes (kB) that are transferred in the multilevel decentralized inter-cloud.
It is shown that the AllToAll approach includes the highest number of data transfers, while MEO with 33% availability offers the lowest. In other words, in a busy inter-cloud the MEO model does not increase the transfer rates and the load of the channel. To sum up, the experimental analysis presents the MEO model in both centralized and decentralized topologies. By implementing the solution in an inter-cloud system, the study demonstrates that MEO outperforms the traditional AllToAll message exchanging in terms of makespan, turnaround times and energy efficiency rates (delay, power and size cost). In order to present the optimized results of previous experiments (section 6.3), figure 75 demonstrates the results as derived by the adoption of the MEO model in 6.3.3. It is shown that the MEO solution offers well-improved results for both IC and ICMS cases.

Figure 76: Comparison of makespan values for IC and ICMS for case 6.3.3

Further, figure 76 demonstrates the performance comparison for clouds 3 and 4 (with 20% utilization-section 6.3.3). In this case MEO outperforms the standard AllToAll solution.

Figure 77: Comparison of makespan values for IC and ICMS for case 6.3.3
Figure 77 shows the comparison of ICMS cases for section 6.3.4, while figure 78 shows the comparison of makespan and turnaround times for section 6.3.5. It is shown that again MEO outperforms AllToAll. Both figures illustrate the effectiveness of the MEO approach.

Figure 78: The comparison of makespan times for ICMS cases of section 6.3.4

Specifically, the makespan values for clouds 1 to 5 have been minimized for the MEO case. The best values (lower makespan) are achieved for 100% resource availability. Similarly, best turnaround times are achieved for the ICMS case of users that submit jobs to dedicated clouds. For this case cloud 1 shows the best turnaround and makespan times, while cloud 3 shows the worst.

Figure 79: The comparison of makespan and turnaround times for the collection of ICMS cases of section 6.3.5
The experiment has focused on the centralized and decentralized topologies. For the second case the study integrates two sub-cases where the inter-cloud offers medium (50%) and low (33%) resource availability. The aim is to show that the MEO model offers optimized results in a highly demanding inter-cloud setting. That is to say that the comparison of the 50% and 33% resource availability energy consumption rates shows improvement for the second case. Finally, the focus is to compare the total kilobytes transferred for the experimental case of AllToAll (33% and 50%), message-exchanging 50% and message-exchanging 33%. Results show that the MEO model does not add load to the communication channel. The simulation experiments draw a number of considerations as follows.

a) The use of MEO offers well-optimized makespan and turnaround times for an inter-cloud setting.

b) The diversity of message exchanging latencies shows increased performance in terms of energy consumption rates.

c) The collective model (operating in synchronous standards) optimizes the number of messages performance (e.g. for the configuration of the centralized experimental case the improvement factor is 3.5).

d) The ranking procedure is considered as first come first served fashion, and for this case the energy consumption levels are improved as well.

e) Both experimental cases show high adaptive-ness to various workloads and topologies.

f) The decentralization offers high dynamic-ness (e.g. for cases of low resource availability) by slightly affecting performance due to meta-brokering message exchanging delays.

This concludes the comparison of the MEO model cases. Next the study focuses on the VM deployment optimal schemes.

6.5 VM Deployment Optimal Schemes

This section presents the VM deployment optimal scheme that allows VMs to be generated on demand and with respect to user requests. A key feature in optimizing cloud performance is the level of gratification that the setting could offer to user job submissions. The study’s view is on user by exploring the VM instantiation that is related to the job requirements. Moreover the terms of static and reactive dynamic deployment for VM instantiation are defined as follows:

1. **Static** is defined as the deployment of VMs in which there is a fixed number of VMs
that are instantiated by the hosts. Specifically, static VMs are established by the cloud administrator and are not drawn up from user queries and requests for service executions. For the experimental case the static VM optimal scheme defines the number of VMs according to the number of requests by taking an on-the-fly decision (for each request a new VM is generated instantly).

II. **Dynamic** is defined as the deployment of VMs in which instantiation is based on different criteria and may vary with time. This is to say that VMs are generated based on the number of jobs entering the environment. In addition, VMs could be migrated as a way of further enhancing the cloud performance. The experiment is based on the investigation of a static cloud and the extraction of performance results by training the setting for certain job variations and various VM numbers. Thus, the study first analyzes the VM instantiation process by implementing the algorithms of chapter 5. The dynamic VMs are reactively adapted to auto-migration utilizing the VM forking concept. For the experimental case the dynamic VM optimal scheme defines the number of VMs according to the past service experiences and by taking an on-the-fly decision (for each request a VM is generated based on the method of forking).

The simulation environment is developed using SimIC and CloudSim version 3.0 as a framework for modeling and simulating clouds and user services. Specifically, CloudSim allows control of a) large scale clouds, b) datacenters, brokers and scheduling optimal schemes in a self-contained fashion, c) adaptability of the virtualization technology for creating multiple virtualization services, and d) flexibility of the processing cores to switch between time-and space-shared allocation optimal schemes. This is especially related to the origin of this experiment (to investigate cloud VMs instantiations); thus the study illustrates the dynamic VM deployment in CloudSim. Based on that, the experiment is configured according to data in table 8.

<table>
<thead>
<tr>
<th>Table 8: The cloud experimental configuration of hosts and VMs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Host</strong></td>
</tr>
<tr>
<td>CPU Cores: 1</td>
</tr>
<tr>
<td>RAM: 2048 (MB)</td>
</tr>
<tr>
<td>Storage: 1000000 (MB)</td>
</tr>
<tr>
<td>MIPS: 1000</td>
</tr>
<tr>
<td>Bandwidth: 10000 MB/s</td>
</tr>
</tbody>
</table>

The selected metric is the total simulation time that represents the time that the cloud requires to execute a collection of jobs as given by formula (33). The variable $i$ represents the job number while the $n$ is the total number of jobs.
Next, the study explores the VM instantiation performance, which is analogous to a combination of the number of jobs, VMs, and hosts. Then the dynamic case is presented that contains the VM migration specification and the experimental analysis. Both solutions are implemented in CloudSim as an alternative for utilizing resources and evaluating static and dynamic hypothesis. Specifically, static presents the experimental analysis of the exploration of the testbed for identifying the performance of the hosts, VMs, and jobs for certain job variations.

6.5.1 The Static VM Instantiation

The study explores variation of the job input number. The VM allocation optimal scheme includes the scheduling of jobs to VMs in a twofold means as presented in Calheiros et al. (2011). Firstly, the space-sharing algorithm in which jobs are placed in the queue when there are free PEs (number of cores) available. Secondly, the time-sharing algorithm indicates that at any given time multiple jobs could be allocated within the cores of a VM. The same could be applied in the case of the VM to hosts’ allocation. This means that VMs can be queued in either space or time with respect to the cores installed in the host. In this experiment, the time-sharing algorithm has been used for both jobs and VM scheduling, thus allowing multi-VM instantiation within the host cores. It should be mentioned that queuing in hosts and VM allocation happens on a first-come-first-serve scheduling.

With respect to the utilization model, the static case selects VMs in a stochastic manner. That means that jobs are submitted to VMs randomly and aim to reach a 100% utilization level. Finally, the provisioning schemes of physical resources (CPU, RAM, etc.) are executed in order to provide the best guaranteed service; this is to allocate a resource whenever it is available. Specifically, the job number is varied from 1 to 250, while the VM number is fixed at 50. The whole setting runs within 50 hosts of one datacenter.

The results show that the average execution and the simulation times are linearly increasing. In addition, the average execution time increases at a lower rate than the simulation runtime. This means that in the case of a high peak workload the average execution time will stay within reasonable levels, however, the whole simulation time that represents the complete service time will be significant. This is because of the operations occurring within the simulated environment, e.g. due to communication latencies.
Specifically, figure 79 illustrates that for a large job submission the total simulation time rises at a higher rate than the average execution time. This could cause significant problems in the case of a massive workload submission.

![Figure 80: Static deployment of VM instantiation performance for input submission of 1000 jobs in 10 to 100 VMs.](image)

A different experiment includes the execution of 10 to 100 VMs with a fixed number of 1000 jobs within an environment of 50 hosts in one datacenter. Figure 80 demonstrates the performance of the simulator when the specification is set to the values in table 8. It is apparent that as the number of VMs increases with constant values of hosts and jobs the average execution and simulation times decrease significantly. Especially, for VM numbers greater than 50 the values increase at a lower rate. In the case of 100 VMs the total execution time decreases dramatically and almost becomes identical to the average execution time. This means that for the specific configuration, and with VM numbers greater than 100 the system reaches a steady state. In the next experiment the study monitors the performance of one VM when executed in 1 to 20 hosts.

![Figure 81: Static deployment of VM instantiation performance for input submission of 1000 jobs in 1 to 10 hosts.](image)
As the hosts number increases the average execution and total execution times decrease. Especially, after 5 hosts, both show a steady rate. To conclude, the static setting presents the experimental analysis of the exploration of the testbed for identifying the performance of the hosts, VMs and jobs for certain job variations. Figure 81 combines all static case results in a way that can be contrasted with the results from the dynamic case to be discussed in the next section.

![Static benchmark results](image)

**Figure 82:** The static benchmark results (average simulation and total execution time)

The specification includes the allocation of 10, 25, 50, 75 and 100 VMs within a fixed environment of 50 hosts. The job input includes the allocation of 1000 jobs.

### 6.5.2 The Dynamic VM Instantiation

This section presents the dynamic workload deployment case that generates VMs based on VM migrations from a pool of available VMs. For modeling this functionality the assumption is that the cloud administrator has previously configured a set of VMs. Utilization is based on migration of VMs within a physical space of the same host. Specifically, migration utilization scheme selects a host with the least computational power due to utilization increase caused by the VM allocation.

Thus, in the experiment the study sets the utilization threshold to 80%, so the system tries to keep the host utilization (CPU) under the specific utilization threshold. The remaining 20% of the utilization is consumed by the migration operations. In this case the migration includes a duplication of the actual VM by using the forking method (Lagar-Cavilla et al. 2009, Lagar-Cavilla et al. 2011) as presented in chapter 5. Specifically, each time a job is submitted to the broker for execution, the datacenter offers an additional functionality that allows VMs to be migrated rather than created from scratch. For experimental purposes the simulator does not
implement the VM forking solution for VM duplication, however the study defines the delay of migration by formula (34).

\[
\text{MigrationDelay}_{VM_i} = \frac{VM_{RAM_i}}{VM_{BW_i}} + (f_{VM_i} \times \text{const}_{VM_i}) \quad (34)
\]

The extension of the formula of Lagar-Cavilla et al. (2009) allows measurement of the delay of the division of the VM\text{RAM} by the bandwidth speed VM\text{BW} in addition to the result of a coefficient value that represents the extra delay. This corresponds to forking latency time \(f_{VM_i}\) multiplied by a constant variable \(\text{const}_{VM_i}\) to control the rate of latency. For example in this experiment the study sets \(f_i\) to 1000 ms and \(\text{const}_i\) to 10, which means that the delay is actually 10 times greater. In this way the experiment becomes a worst-case scenario. By performing migration of tasks in a simulated forking environment the study allows VM instantiation in a dynamic case. This is to say that when there is no availability in terms of computational power, new VMs are generated from a virtual resource pool to handle the workload demanded.

Figure 82 presents the performance of the experiment by measuring the average execution and simulation times when dynamic instantiation occurs. The specification includes the execution of 1000 jobs when the VM numbers are varied from 10 to 100.

![Figure 82: Average execution and total simulation times with dynamic migration of VM instantiation](image)

Figure 82 results show that the average execution decreases while the simulation time increases. In addition, when the VM number is greater than 25 the system reaches a stable state with cloudlet execution under 400 ms. However, the simulation time is increased significantly. For VM numbers greater than 25 the testbed offers a stable state in which execution of 1000 jobs happens in less than 1000 ms. Figure 83 demonstrates the results of
the average execution time by comparing the static benchmarks and the dynamic instantiation for the same VM variation (10 to 100). The rest of the configuration parameters remain the same.

Figure 84: Average execution time of 1000 jobs with the dynamic migration of reactive VM instantiation in comparison with the static case

Specifically, figure 82 compares the average execution time with and without migration. It is clear that the dynamic case with migration outperforms the static solution. Specifically, for VM numbers greater than 50 the static solution becomes stable (under 100 ms) while the dynamic case for VM numbers greater than 25 offers a stable state with execution time under 40 ms. A unique situation is the case of 25 VMs, in which the solution offers the same results for both cases. Figure 83 shows the results of the total simulation time by comparing the static and the dynamic instantiation for the same VM variation (10 to 100) with the same configuration.

Figure 85: Total simulation time of 1000 jobs with the dynamic migration of reactive VM instantiation in comparison with the static and dynamic case
Figure 84 demonstrates the testbed performance with regards to the total simulation time. In this case the non-migration solution outperforms the dynamic one because of the latencies which happen due to VM migrations. It should be mentioned that the initial appreciation that the delay is set to ten times higher than the non-migration is the reason for the high delay numbers. However, when the number of VMs is greater than 25 the solution reaches steady execution level in under 1000 ms.

Figure 86: Number of migrations, SLA violation and energy consumption for reactive VM instantiation

Figure 85 demonstrates the indicators of the number of VM migrations, the SLA violations and the energy consumption in the reactive dynamic migration case when the VMs are varied from 10 to 100. For experimental purposes the determination of SLA violations is measured by the difference between total requested and allocated MIPS divided by the total requested MIPS. The indicators in figure 85 show that for higher numbers of VMs the number of VM migrations is increased. This happens because the experiment aims always to achieve a better distribution of jobs among VMs.

In addition, the number of SLA violations, which is related to the requested and allocated MIPS, is slightly increased. At last, the energy consumption increases significantly due to the extra computational power needed by the datacenter for migrating VMs. To demonstrate the effectiveness of the dynamic VM model based on the ICMS experiment presented in 6.3.2, the study runs a simulation case based in SimIC. Figure 86 shows the comparison of makespan times for ICMS cases (1, 4 ms delay, 40% availability for static and dynamic VM instantiation. It is observed that the dynamic VM instantiation outperforms the static for the configuration of experiment 6.3.5.
By comparing static and dynamic cases the study concludes that a dynamic VM deployment optimizes the average execution times of job executions. Specifically, the value of the average execution times in the dynamic case is 371 ms, while in the static case it is 926 ms. Finally, figure 87 shows the comparison of the performance rates for IC and ICMS and for static and dynamic instantiations. It should be noted that the first two areas demonstrate the static performance measures for clouds 3 and 4, while the rest show the performance for dynamic VM instantiations. For example for cloud 3 the static ICMS performance is 4.7 while for IC is 4.6 and the dynamic ICMS performance is 7.83 for ICMS and 7.82 for the IC.
6.6 Summary

This chapter presents the performance evaluation of the experiments executed in various simulation settings. This includes three cases, firstly ICMS performance analysis demonstrates an optimization of selected metrics, secondly the MEO performance that further optimizes figures in a decentralized setting and thirdly the dynamic VM instantiation model that offers an efficient approach for VM management. Table 9 summarizes the experimental achievements against the existing techniques.

<table>
<thead>
<tr>
<th>Experimentation</th>
<th>Inter-cloud (Buyya et al. 2010)</th>
<th>Variation of delays</th>
<th>Variation of resource availability</th>
<th>AllToAll</th>
<th>VM deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud vs.</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICMS vs.</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICMS mixing of submissions</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEO centralized</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>MEO decentralized</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Static VM deployment</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Dynamic VM deployment</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

This demonstrates that the experimental analysis includes evaluation of various combinations of ICMS configurations and optimal schemes. The next chapter discusses the conclusion of the work, the limitations and the future directions.
Chapter 7: Conclusions and Future Directions

7.1 Outline
This section discusses the contributions and future directions of this study. The ICMS is implemented in the SimIC that highlights the significant contribution of the work. The experimental analysis highlights the contribution and the productive results. Finally, the future directions present a discussion of extension of this work.

7.2 Major Contributions of the Thesis
One contribution of the thesis is the SimIC toolkit simulates inter-cloud environments according to the ICMS model. The design and implementation of the solution is based on meta-brokers that are responsible for service dissemination by having spontaneous and dynamic information of the environment. The following list represents the significant conclusions.

- Large-scale distribution of job requests among meta-brokers as happens in grid systems. In SimIC meta-brokers decide the sub-cloud to execute services by using wide service dissemination algorithms.

- Decentralized topology of meta-brokers including peer-to-peer (P2P) inspired resource discovery. SimIC allows meta-brokers to transfer information and address resource discovery implementations by allowing hashing of meta-brokers ids in P2P networks.

- Static and dynamic management optimal schemes of current workload for each job submission. The cloud (local-broker) is dynamically aware of the current computational capacity for deciding whether to execute jobs locally or forwarding the request to the personalized meta-broker for further distribution.

- Static and dynamic SLA matchmaking optimal schemes among meta-brokers allow an initial criterion of service execution capability of a cloud.

- Static and dynamic instantiation of VMs with regards to history records. A hypervisor is responsible for deciding whether to generate a new VM (static) or migrate one (dynamic) from a SAN storage device. The decision is based on historical delegation records from previous users submissions to the inter-cloud.

- Queuing of VMs according to selected static schedulers. Default developments include first come first serve (FCFS), shortest job first (SJF), earliest deadline first (EDF) and priority scheduling (PS).

The second contribution of the thesis is the model along with the optimal schemes that are demonstrated in the experimental analysis. This demonstrates the performance of the ICMS
in SimIC. It encompasses a number of cases to show the variation of metrics for different submissions. The following list represents the significant conclusions.

- The following considerations have been raised during the experimental analysis. Initially the makespan times of ICMS have been improved for the collection of the clouds. Similarly, turnaround times have been decreased for ICMS. The response times have been optimized for ICMS and the response ratio trend line is moving in a decreasing trend compared to the IC solution. The clouds for ICMS have the same utilization (20% each) with the IC, thus the allocation does not affect the use of resources. Based on that the comparison of performance figures shows that ICMS outperforms inter-cloud. The IC performance is worst as it attracts 97% of the performance of the ICMS. In addition, during the remote cloud invocations the ICMS performance increases by a factor of 1% (based on the particular experimental configuration). A further experiment with more would have given a better reflection of the performance factor.

- The comparison of the performance results for each submission shows that the ICMS optimizes figures each time a new job enters the IC (the average improvement rate has been measured at 3%). By implementing the solution in an inter-cloud system, the study demonstrates that MEO offers well-optimized makespan and turnaround times for an inter-cloud setting. The diversity of message exchanging latencies shows increased performance in terms of energy consumption rates. The collective model (operating in synchronous standards) optimizes the number of messages performance (e.g. for the configuration of the centralized experimental case the improvement factor is 3.5%). Finally, by comparing static and dynamic cases the study concludes that a dynamic VM deployment optimizes the average execution times of job executions. Specifically, the value of the average execution times in the dynamic case is optimized by 555 ms.

7.3 Future Research Directions

Although the research has reached the proposed aim and objectives, there are still areas of improvements mainly related with the experimental analysis and the development of the simulation toolkit. The next discussion details the recommended work in order to improve the quality of the work.

Although energy consumption was out of the scope of the study, the measurements presented in chapter 4 could be extended. This requires to be validated in various workloads and a variety of topologies for identifying supplementary optimization criteria. Also the
security was treated as out of the scope of the study, exploring the security issues of communication among inter-clouds will enhance the study. This includes the realization of encryption in entities communication and methodologies for trust management.

Future research directions also include the realization of the VM migration strategies for dynamic VM instantiations. This will further improve the initial ICMS algorithms as well as expanding functionalities (e.g. further optimize the service distribution algorithm). In addition, the work aims to use the SimIC in order to further explore benchmarks and results extracted from clouds with heterogeneous specification. This will produce further experimental analysis in order to support our work. An important concept to develop is the sharing of the host computational capacity among the VMs. The default ICMS model implies that VMs are allocated as far as computational power exists.

One other aspect of the thesis is the commercialization direction in order to reduce the cost of service consumptions. In particular, the meta-scheduling model along with the utilization of the optimal schemes offers improved performance in terms of execution times. This includes an improvement in the cost of resource usage for resource providers and consumers. Thus, service providers could offer better resource usage prices, by reducing their operational costs, while at the same time clients could lease a variety of resources in better prices. In addition, the energy efficient model further reduces energy consumptions without affecting the performance of the cloud jobs. So, the potential impact of this work is the reduction of cost as well as the energy consumption of resources

With regards to the SimIC toolkit, a future direction will be to further develop the codes in order to accept more advanced optimal schemes (including the exploration of migration issues). It should be mentioned that the study employs ICMS in the SimIC toolkit for realizing algorithmic structure (specific algorithms). With regards to SimIC, the next development steps include the realization of a collection of requirements for adding built-in SimIC capabilities.

Firstly, a challenge will be to import energy efficiency measures for optimizing message distribution among entities. This will increase the effectiveness of current optimization schemes as well as will offer modularity for assisting modelers to define new entities by improving class design. Another challenge is to add VM migration cases and design new scenarios e.g. disaster scenario backup in order to extend the applicability of the toolkit. In terms of toolkit development, the running of various simulation experiments for exploring benchmark performance based on the CloudSim framework will offer comparison measures with SimIC.
Another aspect that is related with the Internet of Things paradigm is to empower simulator to simulate scenarios that implement a collection of sensors that utilize the backbone of the inter-cloud infrastructure. A novel challenge is the exploration of different ranking techniques (based on job performance measures) for achieving a further optimization of this approach. Finally, the study aims of improving the user interface of the simulator in order to allow developers to create simulations more easily.
References


Huang, Y., (2012) Exploiting scheduling upon decentralized distributed grid systems, PhD Dissertation, University of Fribourg, Switzerland


Appendices

Appendix A: Literature review conclusions (Reference tables)

For identifying the key concepts of each approach, the study presents three tables that summarize approaches, benchmark metrics used in experimentation as well as advantages and shortcomings. It should be noted that different solutions have been developed with different requirements, thus the association of these with the inter-cloud will form the basis for identifying correlations. Specifically, table 1 presents the centralized meta-scheduling solutions; table 2 the hierarchical cases and table 3 the decentralized meta-scheduling approaches. Through a detailed demonstration of essential characteristics of related works, the study aims to map characteristics to the essential key requirements for designing the inter-cloud meta-scheduling. The (+) symbol indicates an advantage, while the (-) symbol a disadvantage.