2013 IEEE 27th International Conference on Advanced Information Networking and Applications

# The inter-cloud meta-scheduling (ICMS) framework

Stelios Sotiriadis[1], Nik Bessis[1], Pierre Kuonen[2], Nick Antonopoulos[1]
[1]School of Computing & Maths, University of Derby, Derby, United Kingdom
[2]Department of Information and Communication Technologies,
University of Applied Sciences of Western Switzerland (Fribourg), Switzerland
[1] (s.sotiriadis, n.bessis, n.antonopoulos)@derby.ac.uk, [2] pierre.kuonen@hefr.ch

*Abstract* — **This work covers the inter-cloud meta-scheduling system that encompasses the essential components of the interoperable cloud setting for wide service dissemination. The study herein illustrates a set of distributed and decentralized operations by highlighting meta-computing characteristics. This is achieved by using meta-brokers that determine a middle-standing component for orchestrating the decision making process in order to select the most appropriate datacenter resource among collaborated clouds. The selection is based on heuristic performance criteria (e.g. the service execution time, latency, energy efficiency etc.). Our solution is more advanced when compared to conventional centralized schemes, as it offers robust real-time scalable, elastic and flexible service scheduling in a fully decentralized and dynamic manner. Similarly, issues related with bottleneck on multiple service requests, heterogeneity, information exposition and consideration of variation of workloads are of prime focus. In view of that, the whole process is based upon random service requests from users that are clients of a sub-cloud of an inter-cloud datacenter and access is done via a meta-broker. The inter-cloud facility distributes the request for service by enclosing each personalized service into a host virtual machine. The study presents a detailed discussion of the algorithmic model for demonstrating the whole service dissemination, allocation, execution and monitoring process along with the preliminary implementation and configuration on a proposed SimIC simulation framework.**

***Keywords: Inter-cloud; service meta-scheduling; cloud service distribution; Meta-brokers, SimIC, cloud metrics***

## I. INTRODUCTION

The Inter-Cloud Meta-Scheduling framework (ICMS) encompasses the architectural constraints of the interoperable cloud functionality for achieving wide distribution and remote invocation of services. Specifically, the study illustrates that the ICMS is characterized as a distributed and decentralized meta-computing operation identical to a meta-scheduling heuristic operation. The last one originally defines that each resource has an internal (local) and an external (meta-) component for processing decision-making [1]. Thus, user requests for services are directly submitted to the meta-component that decides to which local resource to relocate it. Usually these meta-components (meta-schedulers) have the role a broker (or a meta-broker) that arranges transactions among clients and clouds. It is actually a middle-standing element for orchestrating the selection process of the most competent resource based on performance criteria [3] e.g. redirect services to low or under-utilized clouds, achieve minimum execution time or short latency times, or even low energy consumption occasion of datacentres. In the simplest of the cases, meta-brokers query each other on request or in regular intervals so as to collect current computational load data for finding the site for executing the user service. However, in large-scale settings e.g. inter-clouds this progress raises NP-complete problem related matters, thus heuristic evaluation criteria are required to take place [7]. This encompasses methods that used in order to achieve an overall satisfactory solution by optimizing the selected performance criterion. This approach is very advanced and complex as it allows decentralization and control of dynamic-ness compared to the limitation drawn from the centralized and hierarchical schemes.

Specifically, a distributed manager named as the meta-broker is responsible for service dissemination decision-making by having spontaneous information of the environment. This solution is more realistic compared with a complete knowledge setting and it is related to the granularity of the system. For achieving that, the meta-broker profiles the identifiers of other meta-brokers as well as communicates with local resources for information exchanging. In contrast, centralized and hierarchical schedulers require having a complete knowledge of the actual resource meta-actors and pool, thus representing a non-realistic approach for large size settings. This knowledge includes the number of hosts, number of services submitted, the workload of each hosts, the number of virtual machines (VMs) and the topology of the system at any given time. In contrast, the ICMS relies upon the distributed scheme, and assumes that this kind of information is partial 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.

The meta-scheduling decision making process is based on random services request from a user or a set of users that are clients of a sub-cloud datacentre and access it though a meta-broker. The inter-cloud facility distributes the request for service and encloses services into VMs (a procedure that called sandboxing) that belong to an interoperable sub-cloud. The ICMS is composed from a set of sub-scheduling heuristics that aim to a) effectively distribute the service request submissions, b) respect the service level agreements (SLAs) signed by the users and the cloud parties and c) optimize the usage of the internal cloud components (e.g. VMs etc). Identifying the appropriate heuristic scheduling specific is crucial for achieving an efficient scheduling, especially when this includes a sufficient amount of internal policies that required to be controlled. Examples include the way in which VMs share computational power and how hosts computation power is provisioned.

Thus, this work aims of identifying the key policies for achieving an efficient optimized inter-cloud service-exchanging model. To this extend, the study presents the architecture of the ICMS along with the algorithmic pseudo-codes and the meta-scheduling process. The next section II presents the motivation background in the area of inter-clouds and section III the ICMS principals. The rest of the paper is organized as follows, section IV presents the ICMS statement, section V the algorithmic model, section VI the selected metrics, section VII the preliminary simulation strategy and section VIII the drawing remarks. At last in section IX we conclude our study by presenting the future research steps.

## II. The Motivation

The inter-cloud concept has been introduced over the recent years as a logical evolution of the Internet. Instead of a file-system oriented Internet, it could be transformed to a computational collaborated setting with analogous requirements to grids and clouds [11]. Various cloud vendors aimed to an interoperable cloud effort by jointly establishing federations of clouds. However, these vendor-oriented endeavours do not base on future standards and open interfaces. In contrast, the inter-cloud as an inter-cooperative infrastructure has been introduced by [4] yet from a federated perspective. They present a business model of a utility oriented inter-cloud system that includes a centralized coordinator per cloud for service dissemination. The last one acts on behalf of the user that requests for service execution in centralized topology based on service level agreements (SLA) necessities. Authors in [9] discuss that the broker acts as an SLA resource allocator by combining components to achieve the agreed benchmark among users and providers. This is a generic view of brokers that generate questions on how to manage the most effective resource allocation and scheduling.

Specifically, this could produce significant problems when a large bulk of user requests could cause a system bottleneck. This problem is identical with clouds, in which a new model is required to bridge the gap of resource selection, allocation and scheduling. The work of [9] presents the traditional brokering strategies for large scale computing systems usually present a centralized topology of a single broker. Specifically, in a multi-provision setting, a broker compares the SLAs of each provider and selects the most appropriate one on behalf of the user. This implies that the broker requires having a complete knowledge of the whole infrastructure along with current availability and communication quality levels. This centralized framework could be proven to be effective for small scale clouds (e.g. cluster-based), however, when it is extended in large-scale, it will face problems, e.g. single point of failure, bottleneck etc.

In contrast to all the above works, we vision an inclusive design of a total decentralized meta-broker based on our previous inter-cloud model presented in [9]. For this purpose, the study extends the broker functionality by adding a meta-broker on top of the traditional broker for allowing communication with other meta-brokers during service submission. This means that throughout a request for service execution a meta-broker collaborates directly with other meat-brokers similar to a meta-scheduling system. This will offer significant advantages, as it will support highly interoperability; flexibility and heterogeneity while at the same time a job execution in a decentralized fashion.

## III. The Inter-Cloud Meta-Scheduling principals

The ICMS contains the scheduling procedures for controlling the service submissions that performed within an interoperable cloud. As scheduling procedure is defined the management of the functionalities that directly affect the optimization of issues related with the service submission, distribution, allocation and execution. However, in the case of an inter-cloud system this affects the decision making process for directing the virtualization part of resources. Fundamentally, these issues are related with the generic scheduling functions as happened within a traditional batch system. Within this system, the jobs (services in inter-cloud) are submitted by the users to the resources for execution and formed in a queue based on the classic static manner.

However, in the case of inter-cloud, the actual requirements (e.g. computational capacity) are not known in advance and depending on initial conditions and chosen parameters, these are formed during the service submission phase. This is because of the dynamic characteristics of the setting thus a vital requirement is to be orchestrated by a meta-computing component. So the methods for developing a flexible setting should be shelf-adaptive and automated based on current decisions, during the run-time, regarding the given initial requirements and conditions. To this extend, ICMS solution achieves dynamic-ness by considering the decentralized meta-brokering paradigm in real-time decisions, as the employment of effective scheduling techniques is crucial in distributed real-time systems [6].

In general, the meta-brokering functionality aims to form the communication bus that bridges the gap among local and remote resources by (re)-directing user requests and responses to a resource drawn from a criterion. These resources could be considered by several conditions, e.g. aiming to the best performance in terms of computational power or time. Nonetheless, the meta-scheduling systems presented in [1], [3] are directly related with fulfilling requirements of dynamic systems like the inter-cloud. Thus, before continuing with the discussion of the ICMS, the study deliberates the fundamental requirements bellow that forms the motivation principals of the ICMS model.

Initially, interoperability among clouds is not a standard procedure and requires a coordination component for controlling various inter-communication protocols. This is achieved by assigning meta-brokers to act on behalf of cloud datacenters in a mutual agreed inter-cloud collaboration. In addition, the homogeneous pool of resources exists within a cloud system mainly due to the common management administration specifics. However, when the setting is extended to an inter-cloud, heterogeneity becomes an issue that is required to be controlled via the SLA agreements. This is to identify appropriate resources by utilizing service matchmaking processes for matching a host CPU architecture etc.

The dynamic-ness and elasticity of the inter-cloud in terms of large-scale setting and services cannot be controlled by traditional solutions (e.g. static scheduler), and based on chosen parameters and criteria. This will be demonstrated by applying realistic cases of decentralized run-time decision-making processes. Also, geographical distribution of services among sub-cloud pools is a standard case for inter-clouds. However, the decreasing performance due to considerable distances for high workload dissemination is an issue to be resolved during the scheduling system design.

Lastly, rescheduling concepts, advance reservation mechanism and service dissemination priorities could be proven to be realistic scenarios for controlling the level of service executions among collaborative clouds. This will be addressed through the SLA agreement dissemination as well the inter-cloud collaboration level. Profiling of resources from previous service experiences in the form of recorded logs could be considered of adding a future value to the overall ICMS performance. Having discussed that, the next section presents the ICMS that encompasses various inter-cloud capabilities. The aim is to demonstrate the service distribution among interoperable inter-clouds by presenting the statement algorithm, the pseudo-codes and any related service oriented policy.

## IV. The Inter-Cloud Meta-Scheduling Statement

The ICMS contains the complete operations and interfaces for achieving a job distribution within an inter-cloud system. It offers the meta-brokering facility that allows a wide dissemination of user service requests by achieving de-coupling of users and cloud providers while at the same time hides the complexity of the low level infrastructure from the user. This 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 ISPs to access a wide-ranging service pool in which users utilize services offered by different vendors, cloud providers etc.

This model involves a decentralized setting that aims of handling unpredictability in an efficient manner. Accordingly, the ICMS is considered to be fully dynamic as the decision making process happens during the run-time and not based on predefined choices. Fundamentally, the ICMS assumes that various sub-clouds launch communication by utilizing a meta-brokering interface that is placed on the top of a cloud local-brokers (local-decisions) and is empowered with a list of other known meta-brokers. Various such interfaces are distributed in different geographical locations and act similar to distributed management systems. This is to say, undertaking crucial choices on behalf of their local broker datacentre for matchmaking requested and offered resources as well as cloud debts. Following to that, the next section introduces the algorithmic statement of the service (job) dissemination. In advance, the policies regarding the scheduling and allocation concepts of the ICMS are presented as well.

Let assume that there is one inter-cloud setting composed by a number of interoperable sub-clouds each of which is named as $c_a$ where $c_a \in \{c_1, c_2, \ldots, c_k\}$. Individually, clouds comprise a number of datacenters $dc_{ab}$ where $dc_{ab} \in \{dc_1, dc_2, \ldots, dc_l\}$ that constitute the physical location of the cluster of core resources. Further, each datacenter contains a number of physical machines named as $h_{abc}$ hosts where $h_{abc} \in \{h_1, h_2, \ldots, h_m\}$ in such way that the computational capacity of machines integrates the augmented cloud computational competency. In addition, each datacenter $dc_{ab}$ generates a number of local-brokers $lbr_{ad} \in \{lbr_1, lbr_2, \ldots, lbr_p\}$ and assigns one local-broker per user submission for managing the internal service allocation and execution of a cloud $c_a$. At last, each of the hosts generate or instantiate a number of VMs for sandboxing the service requirements. Each $vm_{abce} \in \{vm_1, vm_2, \ldots, vm_r\}$ represents the virtualized part of a datacenter host (machine) that eventually contains and executes user service submission.

Within such system, each datacenter local-broker $lbr_{ad}$ belonging to a datacenter $dca_{ab}$ generates a number of meta-brokers $mbr_{ae} \in \{mbr_1, mbr_2, \ldots, mbr_s\}$ and assigns one meta-broker per user submission and per local-broker for managing the exchanging of information among all the components (users, local and meta-broker). A user submits to one meta-broker of a cloud that could have more than one meta-brokers, depending on the experimental case, however, the default configuration is that one cloud has one meta- and local-broker. Usually the service life-cycle encompasses a user that requests a service allocation that contains a requirements specification. Then, it passes the request to the meta-broker that could be defined as the user's personalized interface. The last one distributes the request to interconnected meta-brokers that exists within a public profile named as meta-registry by always aiming to meet SLA specification. The last one is generated from the requirements specification submission. The job then is directly sent from meta- to local-broker and then to the low-level infrastructure of the cloud datacenter. This is to say that each job $j_t \in \{cl_1, cl_2, \ldots, cl_y\}$ is assigned to a virtual machine $vm_{abce}$ that has been generated to a remote host $h_{abc}$ belonging to a cloud $c_a$ and its datacenter $dc_{ab}$.

For demonstrating that, figure 1 presents the partnership scenario of 3 clouds named as $\text{cloud}_a$, $\text{cloud}_b$, cloud$_c$ that are sub-clouds of the ICMS. Specifically, a user$_1$ requests for resources (job$_a$) by establishing connection with a cloud$_a$. Then, cloud$_a$ assigns to the user a local-broker$_a$ and a meta-broker$_a$. The aim is to differentiate the internal procedures that are handled by the local-broker$_a$ and the external procedures for requesting resources from the inter-cloud that are handled by meta-broker$_a$. Each meta-broker encompasses a meta-registry that is an address book of public meta-brokers trusted and available to receive requests and send responses for resource availability (e.g. meta-broker$_a$ and meta-broker$_b$).

Firstly a request is sending to local resources from meta- to local-broker$_a$. If the SLA is matched and resources exist the local cloud executes the request. In a different the request is send to the meta-broker$_b$ and meta-broker$_c$ respectively, wherein each of which forward it to the internal local-brokers (local-broker$_b$ and local-broker$_c$) for proceeding with matchmaking job$_a$ requirements and SLA$_a$ for resource provisioning specification. If a meta-broker is capable of performing the job only then it sends a request back to the meta-broker$_a$, that sends the job to the first responding meta-broker. After that, the decision-making process happened within the cloud (local-broker) and hypervisor that chooses the resource to be utilized by calling resource allocation and execution policies. The resources are allocated in the form of a VM that generated within a remote host of the selected datacenter. Finally, the user instantiates the VM and executes the job$_a$ in a remote sub-cloud.

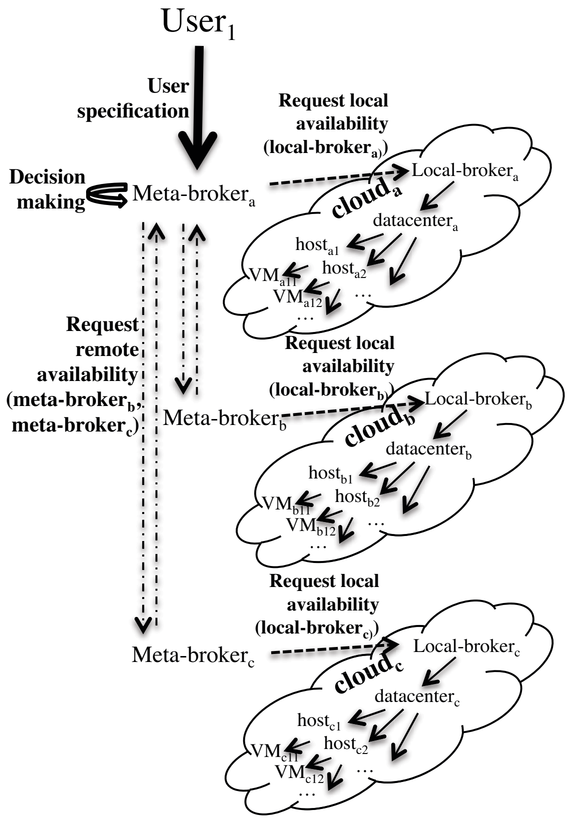

Figure 1: The cloud service request distribution

## V. THE INTER-CLOUD META-SCHEDULING FRAMEWORK: ALGORITHMIC STRUCTURE

The ICMS is composed from a set of statement algorithms that handle the details of the complete service life-cycle and represent the enterprise architecture of the inter-cloud. By splitting the whole development into sub-phases, the study aims to the efficient management of the dynamic and iterative processes. The ICMS is formed by a total of four phases namely as a) the service request, b) the service distribution, c) the service availability, and d) the service allocation. During these phases various components interact with each other (e.g. user, local-broker, meta-brokers, datacenters, hosts, VMs) along with several policies for decision-making processes (e.g. resource availability, utilization models, hosts and VMs scheduling and allocation mechanisms etc). Before that, we clarify the requirements of the service as posed by the user in order to determine the performance measures. Specifically, we assume that at a preliminary stage the user cloud requests for an infrastructure as a service (IaaS) cloud capacity that encompasses software as a service (SaaS) characteristic. Thus, the user requests for cores, CPU power, memory, storage and bandwidth as well as controller (e.g. drives) and platform specification (e.g. operating system).

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 (clock rate of host). This will eventually allow us to determine the performance criteria of the various ICMS entities. Thus, starting with the service submission, we assume that the following scenario is taking place. A user requests for a job (that is sandboxed to a VM) and can get x of 1 (%) of the total augmented host capacity (x is defined by the cloud administrator) of the datacentre and requires executing a set of software (programs) with y instructions, and CPI= z (e.g 300 cycles / 100 instructions = 3) with clock rate w 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 to the overall latency till the service execution started. It should be mentioned that the x% denotes the percentage of the machine to be dedicated to the VMs. The rest will be required by the host in order to operate in highly performance rates.

Formula (1) presents 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.

$$\boldsymbol{ExecTime_{VM}} = \frac{Instruction}{Program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle} \ \mathbf{(1)}$$

Also the CPI represents the cycles per instruction count and given by formula (2).

$$\boldsymbol{CPI} = \frac{Cycles_{user}}{Instructions_{user}} \ (2)$$

Formula (3) presents the performance of the VM for execution time measures by considering a multi-processor request. The *h* parameter demonstrates the time duration of the VM leasing by the cloud user.

$$\mathbf{ExecTime_{VM}} = \frac{\text{Instruction}}{\text{Program}}_{\text{user}} \times \mathbf{CPI_{user}} \times \frac{1}{\text{CPU}_{\text{VM}}} \times \frac{1}{\text{CPUCores}_{\text{VM}}} \times h \ (3)$$

This includes the CPU and cores capacity as required by the user in the user submission. For calculating the CPU burst of the VM in formula (4), we add a coefficient value to the overall user request for controlling further experimental analysis. In addition, we multiply that value with the number of cores for deciding the requested CPU for a multi-processor system as in formula (3). Similarly, we place 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.

$$CPU_{VM} = \frac{CPU_{host} * Coefficient}{VMCount} \ (5)$$

$$Memory_{VM} = \frac{Memory_{host} * Coefficient}{VMCount} \ (6)$$

$$Storage_{VM} = \frac{Storage_{host} * Coefficient}{VMCount} \ (7)$$

$$BW_{VM} = \frac{BW_{host} * Coefficient}{VMCount} \ (8)$$

Finally, the performance measure of a service is given by formula (9) as follows.

$$ExecTime_{service} = Latency_{user-vm} + PerformanceTime_{VM} \ (9)$$

The latency denotes the time that passes till the service execution started. Relevant delays involve meta-brokers, local-brokers, and hypervisors (host and VM allocation) decision-making processes. At last the performance of the service is non-relevant to the execution time of the service as given by formula (10).

$$Performance_{service} = \frac{1}{ExecutionTime_{service}} \ (10)$$

Similarly, we measure the performance of the VM as given by formula (11)

$$Performance_{VM} = \frac{1}{ExecutionTime_{VM}} \ (11)$$

For illustrating the aforementioned mathematical formulas and for measuring the performance of a service submission, we present a brief scenario case. A user requests for a VM with 0.25 (25%) of host performance and executes a set of programs with $100*10^6$ instructions, and CPI= 3 (300 cycles / 100 instructions) with clock rate 1000 MHz (0.25 of 4000MHz of Host with single core). The performance of the VM is calculated as follows:

$$ExecTime_{VM} = 100 * 10^6 (ns.) \times 3 \times \frac{1}{1000} \times \frac{1}{1}$$

$$= 3 \times 10^{5} ns. = 0.3\ ms.$$

Thus, the performance 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:

$$ExecutionTime_{\ service} = 0.3 + 10 = 10.3 \text{ ms.}$$

$$Performance_{service} = \frac{1}{10.3} = 0.097$$

At last, a useful computational metric for measuring performance capacity of a job is the million of instructions per second (mips) that are required by the user. This demonstrates the application requirements and is utilized as an indicator to the required CPU power by a user. In addition, by using this metric we can compare user specifications if required. Formula (12) shows how mips are calculated.

$$mips = \frac{clock\ rate}{cpi} * 10^{-6} \ (12)$$

By having defined the initial requested performance measures, then we present the actual algorithmic structure of the ICMS framework. At last we present the metrics that shows the performance of the algorithms with respect to required as initially calculated by formula (8). The core algorithms represent the service (or job) user submission life-cycle to an inter-cloud setting.

## *A. The Service-Request algorithm*

The algorithmic pseudo-code I demonstrates the job (service) formation and request according to a user specification for a cloud leasing case.

**Algorithm I: 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 |
| | $meta\text{-}broker_n$ | 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, $meta\text{-}broker_n$, 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 ($job_i$, user_delay, user_name, mips)
12: **end for**
```

Specifically, the Service-Request (I) algorithm 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 an preliminary performance measure. At last, 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.

## ***B.*** *The Service-Distribution algorithm*

The Service-Distribution algorithm exemplifies 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 send 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 solution more flexible. The following algorithmic pseudo-code II demonstrates such procedure.

**Algorithm II: Service-Distribution**

| | | |
|---|---|---|
| Require: | meta-broker_name: | meta-broker identity |
| | meta-broker_delay | meta-broker decision latency |
| | meta-registry | list with linked meta-brokers |
| | added_delay | increasing delay figure |
| | meta-broker_delay | meta-broker latency |
| | res_meta-broker | responding meta-broker |
| | req_meta-broker | requesting meta-broker |
| | flag | flag to control termination or re-distribution |
| | count | iteration counter |
| | num | integer variable |
| Method: | get(user_data) | user sent data |
| | update(tag) | update the tag value |
| | send(entity, event data): | send request and data |
| | monitor(job, delay,user,mips) | monitor current data |
| | process(meta-broker_delay) | process of meta-broker decision making time |
| | terminate(messages) | terminate messages after the |
| | re-distribute | selected meta-broker jobs are redistributed |

```
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**
5:    send (cloud, added_delay, current, SLA, tag)
6:    monitor ($job_i$, user_delay, user_name, mips)
7:  **end if**
8:  **if** get(tag) **is** cloud tag **then**
9:    **for all** $meta\text{-}broker_{inter}$ ∈ meta-registry
10:     send($meta\text{-}broker_{inter}$, added_delay, current, SLA, tag)
11:     monitor ($job_i$, user_delay, user_name, mips)
12:   **end for**
13: **end if**
14: $meta\text{-}broker_{inter}$ → res_meta-broker
15: **if** get(tag) is res_meta-broker **then**
16:   send(bucket, added_delay, current, SLA, tag)
17:   monitor ($job_i$, user_delay, user_name, mips)
18: **end if**
19: **if** get(tag) is req_meta-broker **then**
20:   **if** flag → True **then**
21:     update(tag ← user tag)
22:     send(bucket, added_delay, current, SLA, tag)
23:     monitor ($job_i$, user_delay, user_name, mips)
24:     **if** count = num {default: 3} **then**
25:       terminate_messages($meta\text{-}broker_{inter}$)
26:   **else**
27:   terminate_messages($meta\text{-}broker_{inter}$)
278: **end if**
```

The Service-Distribution (II) algorithm is based on tags that are sending from one entity to the other. Initially, the algorithm gets every job came from user(s) and processes it for a delay. Then, it controls a message passing mechanism presented in [2] for distributing jobs depending on the tag value (initialized by various component). This includes the following:

*a)* In case that the tag denotes a message came from the user then it will be forwarded to the local cloud.

*b)* In case that the tag denotes a message came from the local cloud then it will be 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.

c) In case that the tag denotes a message that came from the same meta-broker (after a circulation) then either re-distributes the message or it terminates it and suspends the job.

This completes the distribution case wherein requests are spread among the meta-brokers relying in the meta-registry list. In the case that the flag is set to true then each job will be iteratively re-distributed till executed. However, in the case of a continuously SLA mismatching the algorithm keeps a counter that terminates the job after a certain value of iterations; the default value is 3.

## ***C.*** *The Service-Availability algorithm*

The Service-Availability algorithm is responsible for primarily producing SLA matchmakings and secondly directing a dynamic workload management. This is particularly useful for large scale systems wherein various requests are submitted in different times thus the system requires to decide whether there is computational capacity to execute or not. Thus, the workload management policy implementer herein either allows a job to be forwarded for execution into the low level infrastructure, or it is returned back to the meta-broker for further dissemination (as presented in Service-Distribution algorithm). Algorithm III pseudo-code illustrates the Service-Availability procedure by encompassing dynamic workload management concept.

**Algorithm III: Service-Availability**

| | | |
|---|---|---|
| Require: | meta-broker_name: | meta-broker identity |
| | cloud_delay | cloud decision latency |
| | added_delay | increasing delay figure |
| | cloud_SLA | SLA defined by cloud |
| | user_profile | user formed profile |
| | workload | figure to current workload |
| | $job_i$ | job submitted by user |
| | host_capacity | host augmented capacity |
| 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 (cloud_delay)
3:  added_delay ← get (added_delay) + cloud _delay
4:  if get(user SLA) match cloud SLA and get(host_capacity) exists
5:     for all job_i
6:        get(user_profile(user_CPU, memory, storage, BW)) < workload [i]
7:        workload [i++]← job_i
8:        send (datacenter, added_delay, current, SLA, tag)
9:        monitor (job_i, user_delay, user_name, mips)
10:    end for
11:  else
12:    send (meta-broker, added_delay, current, SLA, tag)
13:    monitor (job_i, user_delay, user_name, mips)
14: end if
```

The workload calculation is related with the current host capacity in terms of CPU, memory, storage and bandwidth. For instance the CPU augmented values is given by formula 12. For the rest resources the augmented value is represented by the sum of the hosts memory, storage and bandwidth.

$$\boldsymbol{AugmentedHostCPU} = \sum_{host=1}^{i} (\boldsymbol{CPU_i} * \boldsymbol{CPUCores_i}) \quad (\mathbf{12})$$

The Service-Availability (III) algorithm demonstrates that jobs came to the cloud systems (into the local-broker) are dynamically controlled for SLA matchmaking and 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 the delay increase the job executing waiting time.

### *D. Service-Allocation*

The Service-Allocation algorithm demonstrates the VM allocation policies and jobs execution that 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 arrive first in the datacenter management component named as hypervisor. The last one takes decision for the following policies:

a) VM generation in a static or dynamic manner. This includes that VMs are either generating from scratch or are relying in an external storage and space are transferred (migrated) to a cloud host for executing a request that has been existed previously (in the form of a recorded VM).

b) Requests for VMs are organized in a deferred queue that releases jobs after a criterion passes (e.g. after a number of jobs, or after an interval).

At last the algorithm allocates a host portion and starts the VM execution. The local-broker that has knowledge of 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 mentioned that the default ICMS static algorithms include the first come first serve, shortest job first, earliest deadline first, and priority algorithms. Algorithm IV pseudo-code demonstrates aforementioned procedure.

**Algorithm IV: Service-Allocation**

| | | |
|---|---|---|
| Require: | select_VM_allocation_policy | parameter selection of sharing policy |
| | select_queue | parameter selection of queue algorithm |
| | FCFS, SJF, EDF, PA | static algorithms |
| | $job_i$ | user job |
| | user_profile | user profile data as formed by the meta-broker |
| | hypervisor_delay | hypervisor decision latency |
| | dc_delay | datacenter decision latency |
| | added_delay | increasing delay figure |
| | key | hash key for queueing jobs |
| | queue_lenght | desired queue size for releasing the queue |
| | interval | desired time for releasing the queue |
| | static, dynamic | VM allocation policies |
| | workload | parameter for current workload |
| Method: | get(user_data) | user sent data |
| | send(entity, event data): | send request and data |
| | monitor(job, delay,user,mips) | monitor current data |
| | process(hyper_delay) | process of meta-broker decision making time |
| | record(VM data) | keep log of executed jobs |
| | sort(algorithm) | scheduling algorithm fashion |
| | host_allocation(data) | allocating host computational resources |
| | exists(VM) | checking whether VM rely in a pool |
| | migrate(VM) | transfer VM from pool to datacenter |
| | process(hyper_delay) | process of hypervisor decision making time |
| | gen(metric) | generate metric results |

```
1:  get (user_name, added_delay, current, SLA, tag)
2:  select_VM_allocation_policy [static, dynamic]
2:  select_queue [FCFS, SJF, EDF, PA]
3:  for all job_i ∈ queue[]
4:     get (user profile)
5:     added_delay ← get (added_delay) + hypervisor _delay+dc_delay
6:     queue [key, job_i]
7:     select_queue.sort(FCFS)
```

```
8:   if queue_lenght>=s or interval=i then
9:      if select_VM_allocation_policy = static
10:        host_allocation(CPU,memory, storage, BW)
11:        process(VM)
12:        record(VM, job_i)
13:     else if exists(VM) then
14:             migrate(VM)
15:          else
16:             goto(line10)
17:     end if
18:  workload ← host_allocation(CPU, memory, storage, BW)
19:  send (local-broker, workload)
20:  monitor (jobi, user_delay, user_name, mips)
21:  gen(throughput, utilization, turnaround, makespan, energy, cost)
22: end for
```

Finally, the Service-Allocation (IV) algorithm allocates host resources and executes the job. At last the monitor operation keeps a log of traces for each of the job exchanging among entities.

## VI. Algorithmic Metrics

The monitoring procedure generates a number of metrics by instantiating data generated from the whole ICMS process as follows. Initially, the throughput value, formula (13), of cloud includes the number of jobs that matched and executed by the cloud.

$$\boldsymbol{Throughput_{cloud}} = \sum_{0}^{i} \boldsymbol{job_i} \ (13)$$

This affects the cloud utilization parameter, formula (14) that calculates the number of jobs (percentage) executed from the whole user(s) input.

$$\boldsymbol{Utilization_{cloud}} = \frac{\boldsymbol{Throughput}}{\sum_{0}^{counter} \boldsymbol{Jobs}} \times \mathbf{100} \ (14)$$

The turnaround time for a job calculated by formula (15) includes the execution time and the current delay on an entity (e.g. turnaround time in meta-broker or VM).

$$\boldsymbol{TurnTime_{cloud}} = \left\{\frac{\boldsymbol{instructions_{VM} * CPI_{VM}}}{\boldsymbol{CPU_{VM} * CPUCores_{VM} * 10^5}}\right\}_{\boldsymbol{cloud}} + \boldsymbol{CurrentTime} \ (15)$$

The makespan formula (16), demonstrates the sum-up of the VM execution time plus the total delay due to service dissemination.

$$\boldsymbol{Makespan_{job}} = \boldsymbol{VMexecutionTime} + \boldsymbol{totalDelayTime} \ (16)$$

Finally, energy efficiency measures are calculated by formulas (17), (18) 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 the initialization of the ICMS.

$$\boldsymbol{HostEnergyConsumption} = \frac{\boldsymbol{WattsHost * UserHours}}{\mathbf{1000}} \ (17)$$

$$\boldsymbol{TotalHostCost} = \boldsymbol{EnergyConsumption} * \boldsymbol{CostKwph} \ (18)$$

## VII. Experimentation and Simulation using SimIC

The ICMS is implemented in the ‘Simulating the Inter-Cloud’ (SimIC) toolkit [10], which is a discrete event simulation framework that replicates an inter-cloud service dissemination setting. Specifically, the ‘Simulating the Inter-Cloud’ (SimIC) is a discrete event simulation toolkit based on the process oriented simulation package of SimJava. We consider our solution an alternative of the CloudSim simulation framework [5] for achieving cloud interoperability. Specifically, we implemented the SimIC toolkit to allow experimentation on event exchanging among components in terms of messages that are sending among the system entities at different time intervals. Further to this, SimIC compliments CloudSim as 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 and others presented in [10]. Specifically, the resource discovery service has been extensively discussed in [8]. The preliminary simulation study illustrates the performance of the algorithm in terms of producing benchmarks of the selected performance measures. For instance, the user requests for a VM with e.g. 0.25 of 1 host performance and executes a set of programs with $100*10^6$ 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 performance of the VM is calculated by formula (10).

For demonstrating the algorithms performance we implement a scenario case of an inter-cloud that contains 4 sub-clouds with different SLA specifications. To this setting we place 4 users that send 2 to 10 service requests. Each time the ICMS algorithm distributed request according to section V strategies. Table 1 shows a simple user specification formation.

Table 1: User Specification Files

| User specification text files | |
|---|---|
| ***Hardware Requirements*** | ***Software Requirements*** |
| Username: u_ste_1<br>HostOS: Linux<br>Platform: Intel<br>Memory(GB): 2<br>CPU-cores: 2<br>CPU-speed: 3000<br>H/D-Controller: SAN<br>Storage-HD: 1000<br>BW: 1000 | Username: u_ste_1<br>SW1: spec_1_cloud_1<br>Instructions: 1000000000<br>CPI: 3<br>Hours: 240<br>Deadline: 555<br>Priority: 4 |

Further to this we assume that user 1 SLA matches with cloud 1 or cloud 4, user 2 with cloud 2, user 3 with cloud 1 or cloud 4, user 4 with cloud 1, cloud 2 and cloud 3. The ICMS schedules and executes VMs by instantiating ready made VMs from a list. Thus in this case we execute the dynamic VM instantiation case. In addition, we run the following simulation configuration: cloud 1 with 4 hosts with different CPU-speed (host 1:1000 MHz, host 3:500 MHz etc.) and same rest resources (e.g. Memory is set to 10). The average delay among components is set to 10 ms. and the counter for releasing jobs is set to 7. This implies that after 7 jobs the queue will start execution of 7 VMs concurrently. If the queue length is greater than 7 then the system generates interval calls after 800ms.

The preliminary experimental analysis of the real-time scheduler is very prospect (in FCFS queue) as the average values of turnaround and makespan show a decreasing trend. This implies that for increasing workloads the algorithm distributes the jobs more efficiently by considering the ICMS resource management criteria (queuing jobs). In addition, the average value of VM execution time, cost and energy slightly change during the service dissemination, thus this does not affect the metrics. Figure 2 demonstrates the performance of the four ICMS algorithms.

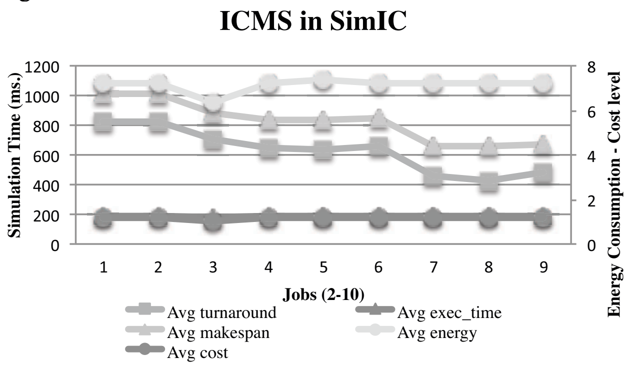

Figure 2: The ICMS four algorithms in the SimIC

Similarly, we execute a dynamic workload management (this time we use SJF algorithm) to illustrate the service distribution when availability is low. Specifically, we decrease the capacity of cloud 1 as we assume that there is a fault case and the host availability is reduced notably. Figure 3 illustrates the workload figures of the ICMS when non-dynamic versus dynamic management is taking place.

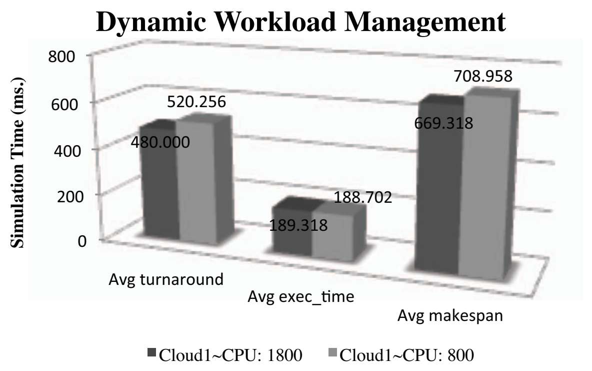

Figure 3: The ICMS for non-dynamic vs. dynamic workload management

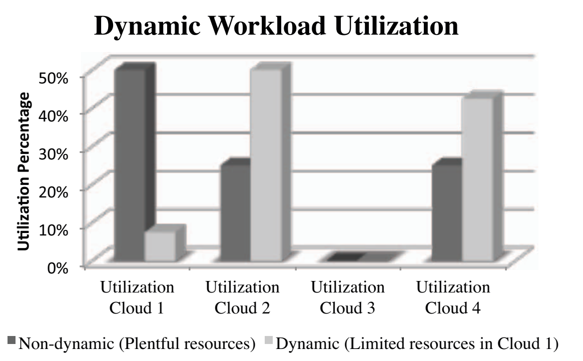

Figure 4: The job utilization for non-dynamic and dynamic workloads

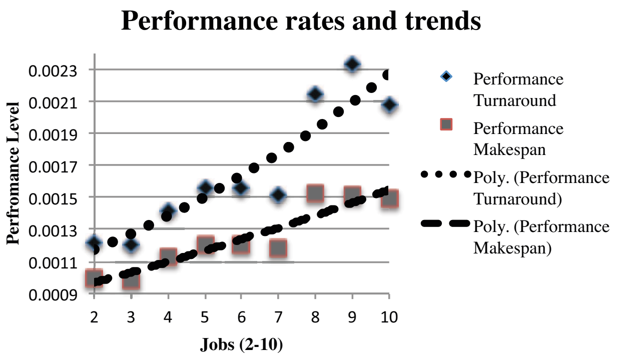

Figure 5: The job performance rates (turnaround and makespan) and trends for submissions 2 to 10.

It is apparent (figure 3) that the average execution time is slightly decreased (from 189.3175 to 188.7017949) as the ICMS allocate more resources on a different cloud. In addition, the average turnaround and makespan are increased as the number of job requests is increased, however without affecting the VM execution performance. Finally, Figure 4 shows the sub-clouds utilization figures for both cases while figure 5 demonstrates the job variation (2-10) performance for the 4 users inter-cloud submission case study. The trend lines (polynomial) show that the performance turnaround and makespan increase tendency as the number of jobs gets bigger.

## VIII. REMARKS

The study presents the ICMS a meta-computing inspired solution for inter-clouds. The algorithmic solution optimizes the submitted workloads by combining multiple brokers into a single aggregated view, identical to distributed resource managers. It should be mentioned that our work is focusing on performance optimization of the IaaS inter-cloud components, thus issues such as security, regional cloud governance, pricing models etc. are considered out of the scope of the study it self. 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. Our solution offers a modular structure of the whole user job submission life-cycle by offering various advantages and it is implemented in SimIC. The inspiration of the design of the core entities of the SimIC came from the CloudSim framework [5], however our classes have been re-designed to compliment additional features as follows.

a) The meta-brokering distribution works on a real-time response model wherein certain data is exchanged during run-time among collaborated meta-brokers. This minimizes the information exposition and increases dynamic-ness.
b) The transparency of the system is increased due to the de-coupled functionality of the system. This includes that users are not de-attached from their desired cloud provider, but the last one could search further to disseminate the service request to interlinked partners. In addition, meta-broker expands the service elasticity

from the capacity of a cloud to the capacity of the inter-cloud, thus offering an almost infinite elasticity.

c) Scalability of service executions increased due to the large number of resources and capabilities. This encompasses various VMs that different clouds could instantiate according to demands. Further to this, heterogeneity is also considered due to the variety of datacenter resources and services.
d) The realization of the ICMS happens in a partial and decentralized knowledge of the resource pool. This is because each meta-broker has a complete knowledge of the local datacenter but a transient and incomplete knowledge of the remote resources.
e) Meta-brokers aim of improving the efficiency of service execution for both service cloud providers and cloud consumers. This could offer an energy aware solution wherein history records of previous successful service executions.
f) Trough static and dynamic instantiation of VMs we aim of adding value with regards to history records. Specifically, ICMS allows the decision on whether to generate a new VM (static) or migrate one (dynamic) from a storage device. The decision is based in historical delegation records.
g) The re-active management of heterogeneous service submissions in the form of VMs extends inter-cloud capabilities. ICMS allows requests for heterogeneous resources to be allocated according to SLAs.

## IX. Conclusions

This work presents the initial study of the ICMS algorithmic structure for defining the most important internal procedures of the inter-cloud. Specifically, the inter-cloud facility disseminates the request for service by sandboxing operations into VMs that belong to an interoperable sub-cloud. The ICMS is composed from a set of sub-scheduling heuristics that aim to a) request and accept connection with remote sites, b) distribute requests within that system, c) search for available resources, d) allocate the resource based on a criterion, e) execute a VM within this resource, and f) monitor the whole procedure for keeping performance measures. The primary experimental analysis shows a prospect performance figure of the workload management.

The future steps of our research include the realization of the VM migration strategy for dynamic VM instantiation. Principally, we will further improve the initial ICMS algorithms as well as we will expand functionalities (e.g. further optimize service distribution algorithm). In addition, we aim to use the SimIC in order to further explore benchmarks and results extracted from a typical cloud and inter-cloud. 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. However, in the future we will further develop our solution in order to accept more advanced policies (including the exploration of migration issues). It should be mentioned that we employ ICMS in the SimIC toolkit for realizing algorithmic structure (four algorithms). In this setting, we are developing the whole solution as well as we have executed the formulas (metrics) of the aforementioned sections. The SimIC is not yet available to the general public as it still under implementation. However, modellers that require executing SimIC scenarios are encouraged to communicate with us.

## References

[1] Bessis, N., Sotiriadis, S., Cristea, V., Pop, F., Modelling Requirements for Enabling Meta-Scheduling in Inter-Clouds and Inter-Enterprises, Third International Conference on Intelligent Networking and Collaborative Systems (INCOS 2011) , Nov 30 - Dec 2 2011, Fukuoka, Japan. pp. 149-156

[2] Bessis, N., Sotiriadis, S., Pop, F. And Cristea, V. (2012). Optimizing the Energy Efficiency of Message Exchanging for Job Distribution in Interoperable Infrastructures, 4th IEEE International Conference on Intelligent Networking and Collaborative Systems (INCoS-2012), September 19-21, Bucharest, pp. 105-112

[3] Bessis, N., Sotiriadis, S., Xhafa, F., Pop, F. and Cristea, V. (2012). Meta-scheduling Issues in Interoperable HPCs, Grids and Clouds, International Journal of Web and Grid Services, Volume 8, Issue 2, Inderscience, pp. 153-172.

[4] Buyya, R., Ranjan, R., and Calheiros, R. N., (2010) InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services, Algorithms and Architectures for Parallel Processing (2010), Volume: 6081/2010, Issue: LNCS 6081, Publisher: Springer, pp. 13-31.

[5] Calheiros, R., N., Ranjan, R., Beloglazov, A., De Rose, C., A., F., and Buyya, R. 2011. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exper.* 41, 1 (January 2011), pp. 23-50.

[6] Stavrinides, G., L., and Karatza, D., H., 2010. Scheduling multiple task graphs with end-to-end deadlines in distributed real-time systems utilizing imprecise computations. J. Syst. Softw. 83, 6 (June 2010), pp. 1004-1014.

[7] Sotiriadis, S., Bessis, N., Xhafa, F., and Antonopoulos, N. 2012. From Meta-computing to Interoperable Infrastructures: A Review of Meta-schedulers for HPC, Grid and Cloud. In *Proceedings of the 2012 IEEE 26th International Conference on Advanced Information Networking and Applications* (AINA '12). IEEE Computer Society, Washington, DC, USA, pp. 874-883.

[8] Sotiriadis, S., Bessis, N. And Kuonen, P. (2012). Advancing Inter-cloud Resource Discovery based on Past Service Experiences of Transient Resource Clustering, 3rd International Conference on Emerging Intelligent Data and Web Technologies (EIDWT-2012), September 19-21, Bucharest pp. 38-45

[9] Sotiriadis, S., Bessis, N. and Antonopoulos, N. (2012). Decentralized Meta-brokers for Inter-Cloud: Modeling Brokering Coordinators for Interoperable Resource Management, 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD'12), May 29-31, Chongqing, May 29 – 31 2012, pp. 2475-2481.

[10] Sotiriadis, S., Bessis, N., Antonopoulos, N. SimIC: Simulating the Inter-Cloud, The 27th IEEE International Conference on Advanced Information Networking and Applications (AINA-2013), Barcelona, Spain, March 25-28, 2013

[11] Sotiriadis, S., Bessis, N., Huang, Y., Sant, P. And Maple, C. (2010). Defining Minimum Requirements of Inter-collaborated Nodes by Measuring the Weight of Node Interactions, 4th International Conference on Complex, Intelligent and Software Intensive Systems (CISIS-2010), 15th-18th February, Krakow, ISBN: 978-0-7695-3967-6/10, pp. 291-298.