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

SimIC: Designing a new Inter-Cloud Simulation platform for integrating large-scale resource management

Stelios Sotiriadis, Nik Bessis, Nick Antonopoulos, Ashiq Anjum
School of Computing & Maths, University of Derby, Derby, United Kingdom
(s.sotiriadis, n.bessis, n.antonopoulos, a.anjum)@derby.ac.uk

Abstract — **'Simulating the Inter-Cloud' (SimIC) is a discrete event simulation toolkit based on the process oriented simulation package of SimJava. The SimIC aims of replicating an inter-cloud facility wherein multiple clouds collaborate with each other for distributing service requests with regards to the desired simulation setup. The package encompasses the fundamental entities of the inter-cloud meta-scheduling algorithm such as users, meta-brokers, local-brokers, datacenters, hosts, hypervisors and virtual machines (VMs). Additionally, resource discovery and scheduling policies together with VMs allocation, re-scheduling and VM migration strategies are included as well. Using the SimIC a modeler can design a fully dynamic inter-cloud setting wherein collaboration is founded on meta-scheduling inspired characteristics of distributed resource managers that exchange user requirements as driven events in real-time simulations. The SimIC aims of achieving interoperability, flexibility and service elasticity while at the same time introducing the notion of heterogeneity of multiple clouds' configurations. In addition it accepts an optimization of a variety of selected performance criteria for a diversity of entities. The crucial factor of dynamics consideration has implemented by allowing reactive orchestration based on current workload of already executed heterogeneous user specifications. These are in the form of text files that the modeler can load in the toolkit and occurs in real-time at different simulation intervals. Finally, a unique request is scheduled for execution to an internal cloud datacenter host VM that is capable of performing the service contract. This is formally designed in Service Level Agreements (SLAs) based upon user profiling.**

Keywords: Inter-cloud meta-scheduling, inter-cloud simulator, Internet of Things simulator, event-oriented simulation, cloud service and resource management.

I. INTRODUCTION

During the latter years a variety of simulation frameworks and tools have been presented by the academia in order to mimic the behavior of complex systems based on different requirements and use cases [9]. This reproduction is particularly useful for large scale computing systems wherein the utilization of resources for research purposes is difficult to be achieved as involves highly hardware, administrative, licensing and training costs. For the case of distributed computing such as high performance computing and grid a collection of platforms have been developed for either simulating the resource management of local-scheduling queue, or the meta-scheduling concept for large-scale infrastructures based on heuristic decisions.

Recently, cloud computing has been emerged as a promising paradigm wherein elastic services could be delivered to users on a pay-on-demand model. Specifically, services include hardware and software utilized by users in the form of a request for certain resources. These are sandboxed in virtual machines (VMs) that are available for utilization for the desired time duration. However, as the number of users increased the capacity of clouds in terms of offering a diversity of services is analogous with the size and capabilities of the internal cloud datacenter [7]. As the trends show that users become more and more cloud reliant in their every day activities, it is apparent that cloud providers will require to develop a collaboration scheme for allowing exchanging of services and VMs in order to meet high computational demands [8].

This inter-cooperative setting named as inter-cloud [1] allows multiple parties to exchange services for increasing the quality of standards. In advance, the inter-cloud meta-scheduling framework (ICMS), introduced in [8] describes an architectural strategy along with the algorithmic model for achieving such association. 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 criteria, e) execute a VM within this resource, and f) monitor the whole procedure for keeping performance measures. The entire functionality is based upon the meta-computing paradigm [8] that suggests decentralized resource managers are placed on top of a local decision-making system in order to interconnect co-sites that have a corresponding architecture.

In order to design and configure such a topology it is vital to allow service dissemination among clouds and to monitor the performance of the deployed policies. The solution of utilizing real resources from cloud providers e.g. Amazon could lead to a vendor-oriented case. In addition, the scale of the experiment will be limited to the competency of the cloud provider. At last, cost of using resources as well as inter-cloud configuration is beyond of the control of the developer [3]. Therefore, this work presents the 'Simulating the Inter-Cloud' (SimIC) toolkit, a discrete event simulation framework that replicates an inter-cloud service dissemination setting. Specifically, the toolkit is developed using the SimJava package that allows event exchanging among components in terms of messages that are sending among the system entities at different time intervals. Fundamentally performance metrics include service makespan, VM execution times, request turnaround

1550-445X/13 $26.00 © 2013 IEEE
DOI 10.1109/AINA.2013.123

times, throughput of entities, resource utilization, response ratio, energy consumption of datacenters, VMs utilization cost, and service latency figures.

Thus, this paper illustrates the SimIC by initially presented the related cloud simulation toolkits and the motivation in section II as well as the design requirements of proposed framework in section III. The rest of the paper is organized as follows, section IV illustrates the SimIC installation specification and section V the cloud class architecture and the core entities of the toolkit, section VI describes the design of an inter-cloud meta-scheduling case study and section VII presents the experimentation of the SimIC as well as a simulation scenario and performance analysis and results. At last we conclude our work in section VIII by discussing the conclusions and the future directions for adding additional built-in features.

II. THE MOTIVATION IN DEVELOPING SIMIC

Over the years, a variety of simulation toolkits have produced by developers in order to meet the different needs of newly emerging infrastructures. This is because real test bed experiments are difficult due to the large number of difference requirements. The problem gets worse when evaluation of various resource discovery and scheduling policies is required through several different scenarios. In such cases, the actual experiment is limited to the real testbed system scale and capabilities. An alternative is to use simulation tools that allow researchers to evaluate the hypothesis prior to the software development [3] and to implement and test the actual behavior of the system in several scenarios, metrics and criteria. In the case of large scale distributed systems (e.g. grids) a variety of simulation toolkits has been produced for exploring job allocation on high performance systems. Simulators such as GridSim and Alea [4] support the modeling of grid heterogeneous resources together with various job policies.

However, for the case of clouds and inter-cloud none of these solutions can address the arising application level requirements. This is because of the application-oriented cloud emphasis and of the elasticity of services in the pay-on-demand model [2]. For that reason authors in [3] present the CloudSim simulation framework that aims to a pay-on-demand model wherein subscribed services are delivered to the users in an elastic fashion. A different simulation case for clouds is the iCanCloud simulator [5] it's targeted to conduct large experiments and to provide a flexible and fully customizable global hypervisor for integrating any cloud brokering policy. By default, for the inter-cloud case, none of the aforementioned solutions could be able to mimic such functionality without extending the distribution package. This is because our core design elements are meta-computing inspired e.g. the large scale that the system could expand, the decentralization of the distributed resource managers, the dynamic adaptability and the real-time service orchestration.

To this extend, our modeling decision has been concluded to the development of the SimIC simulation framework that is fundamentally inspired by the CloudSim toolkit. By using the SimIC a modeler could 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 clocks 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 policies and VMs migration situations. The inspiration of the design of the core entities of the SimIC came from the CloudSim framework [3], however our classes have been re-designed and considerably extended to include additional features.

III. THE DESIGN REQUIREMENTS

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 as illustrated in [6] 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 policies 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 policies 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 users 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 been 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 include the key aims of our development. Next the installation specification is presented.

IV. INSTALLATION SPECIFICATION

The SimIC (version 1.1) is based on the process event simulation API of the SimJava version 2 distribution available on [10]. The SimIC (version 1.1) has been developed using the Java™ 2 Platform (JDK 1.6) and includes a jFreeChart 1.0.14 library [11] for producing charts and diagrams for a selection of performance metrics for the most of the entities including service makespan, VM execution times, request turnaround times, throughput of entities, resource utilization, response ratio, energy consumption of datacenters, VMs utilization cost, and service latency figures. An extended discussion of selected metrics for achieving inter-cloud performance evaluation is presented in [8].

V. THE SIMIC ARCHITECTURE

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 are 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 we ensure a suitable solution for implementing various simulation cases in order to identify cloud and inter-cloud meta-scheduling benchmarks. This will be 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. Having said that, this section demonstrates the entities of the SimIC that are entitled as follows:

- The *UserCharacteristics* class instantiates the current service information for each user by incorporating hardware and software requirements as defined in two different text files.
- 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 *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)). Parts of this profile are accessible from each or specific components of the SimIC v.1 and accessibility subjects to internal information exposition desired levels.
- The *OpenProfile* class gets 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. This is related with the information exposition levels.
- The *User* class is responsible for forwarding a number of requests for VMs, wherein each request scheduled after a specific processing delay to a dedicated inter-connected cloud interface also named as meta-broker. This relationship is a many (users) to one (meta-broker). Each request could be heterogeneous and send after a specific delay for simulating the internal entity latency of the user. In addition, each request includes information about the job identification, specification, an indication to the user profile, and other relevant to simulation data.
- The *printText* class creates a result file wherein prints each entity submission logs that include 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.
- 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 modeler could expand this functionality to address resource discovery based on P2P chord solution. This solution has being implemented as well. By this way requests for services could be distributed within an inter-cloud from meta-broker to meta-broker in the case that the initially contacted resources are not capable of performing the request or the group of requests 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. At last 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.

- The *Bucket* class represents the termination 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 to be terminated if there is a case of SLA misses matchmaking. It should be mentioned that this class could be instantiated as a terminal to other entities as posed by the use case.
- The *Cloud* class includes an 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 with its local cloud skill. 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).
- The *OpenHost* class imports each host characteristics from a text file to the simulator by allowing the SimIC to access hosts 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.
- 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.
- The *Datacenter* class accepts events for VMs deployment from the cloud that is happened through a hypervisor. By fundamental 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.
- 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 and the second measures the computational power of a host to be shared among current or next VM requests. By default both policies are implemented in a FCFS fashion and their modular structure easily allows the utilization of other scheduling algorithms. The modeler could select among algorithms such as the SJF, EDF and PS that have been developed within the hypervisor. It should be mentioned a request(s) that cannot be executed directly (waiting requests) are placed in the deferred queue that organizes the scheduling according to the desired scheduling algorithm. 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. At last, this component deploys a VM by considering the VM generation delay. This is added to the total delay of simulation.
- The *HyperCall* class generates an internal call thread to the Hyper class in order the last one to release the jobs that have been scheduled in the queue according to a heuristic algorithm. The time interval value is defined by the modeler (static value) or consistent with a probability-generated number. In addition, the modeler could generate various instances of this class according to the desired simulation scenario.
- The *HostCharacteristics* class imports each specific host requirement as defined in a host text file for parsing information of the various SimIC entities e.g. in hyper class.
- 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.
- 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, the VM name, the metabroker name that executes the request, the user name that submits the service, the VM execution time, the makespan of the VM, the energy consumption and the total cost. In addition, user turnaround times (in case of more than one requests), 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.
- 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-

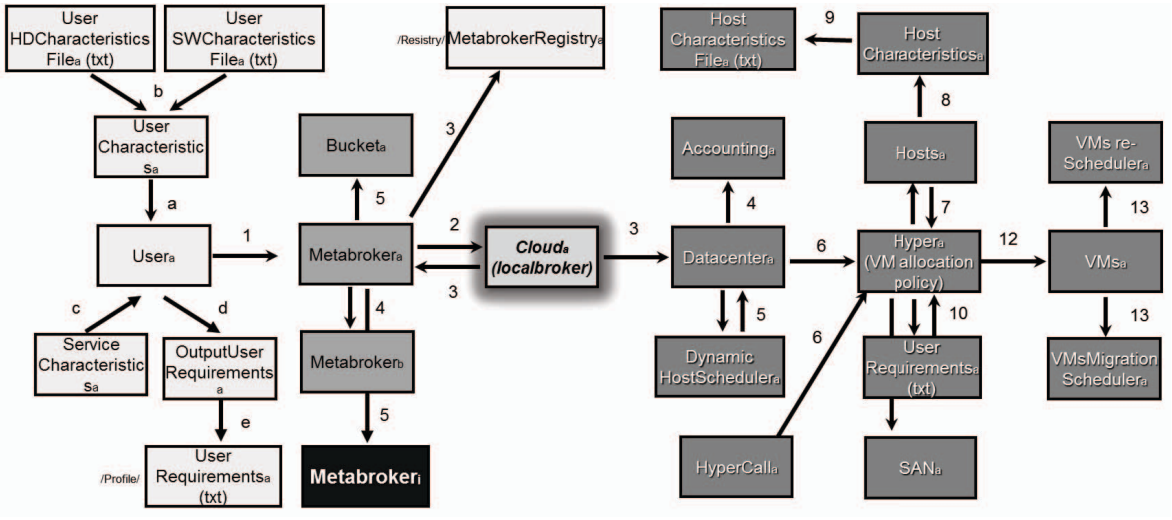

Figure 1: The default message event model for architecting entities of the SimIC framework

instantiate the same profile faster than re-developing the VM from scratch. By this way an overall optimization of the VM performance could be observed.

- The *VMMigrationScheduler* is a 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 case of a sensor indication as a trigger for backing-up reasons or disaster cases).
  - The *SANStorage* class generates an extra hardware space for extending internal hosts storage as well as to be 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.
- 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.
- The *CreateResults* class generates a log file that contains the performance measures of the specific sub-cloud for various criteria. These are the average delay, the turnaround time, the average execution time, the average makespan, average energy, average cost per hour, the throughput, the utilization percentage, the response ratio, and the performance measure presented in [8].
- The *DrawVMs*(*Performance_Metric*) e.g. the *DrawVMsMakespan* class plots a jpg diagram for each of the selected benchmark that are stored in a default directory. For that reason the jFreeChart package is utilized.

To conclude, the aforementioned entities demonstrate the core classes of the SimIC (v1.1) framework. The default relationships of the entities are demonstrated in figure 1. A more detailed discussion on the relationships and the meta-brokering dissemination algorithm is given at the next section. 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 our case we will define an inter-cloud meta-scheduling scenario in order to produce the evaluation study and the experimental results. 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.

VI. Developing A Default SimIC Use Case

This section presents a design case study discussion of the SimIC including a job specification on information processing and retrieval as well as job distribution and allocation. Initially, the SimIC instantiates the user class that contains a number of internal data that are given by the modeler (user name, job number, delay and SLA specification) prior to the simulation started point. The class also loads the user characteristic requirements (presented in table 1) and generates a user profile (with data from table 1 in addition to million of instructions per second (mips), total user job number and submission cloud information).

TABLE 1: User Specification File Formation

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

The default level of SimIC allows full data exposition to the all entities. At last the class sends to the linked meta-broker a request or a set of requests for job allocation(s) along with job data specification (e.g. job id). Each request is forwarder to the meta-broker class that is responsible for dynamically checking its local cloud SLA level and computational capacity. This is to say that the cloud class (representing the local-broker or local resource management system) informs the meta-broker for current resources available in datacenters. If the cloud is unable to execute the job due to SLA miss-matching or limited computing power (e.g. lower CPU than demanding), the requesting meta-broker forwards the job to the next interlinked meta-broker (default solution) or into a group of decentralized meta-brokers with regards to the desired scenario. Each responding meta-broker in the default case checks internal capacity of executing the request, thus it decides to execute the job locally or to forward the job to an inter-connected one. The decision making process is based on dynamic consideration of current workload that being executed or it is already relying in a deferred queue.

In the decentralized case the meta-broker forwards a request for SLA and computational performance to responding meta-brokers that reply with such information. Then the requesting meta-broker sends the job to the first one that responds faster. The resource discovery mean is based on a P2P chord solution [8]. If a job cannot be executed within the inter-cloud then it is forwarded to the bucket class for keeping a log of unfinished (unexecuted) jobs. However, if a job cannot be executed due to SLA mismatching then the bucket terminates the event, however, if the job cannot be executed because of low computational resources the modeler decides whether to re-forward the job when availability occurs (enter a matching deferred queue) or to terminate the job.

In the case that a job is successfully selected for execution is send to the datacenter class for instantiating the accounting, billing, energy logs and regional information. The datacenter class instantiates the class called hypervisor (hyper) and is an entity for managing hosts and generating VMs. The last one offers key functionalities with regards to dynamic and real-time scheduling of jobs in host VMs. This entity generates a deferred queue with incoming events for job executions and releases the queue after an interval (given by the modeler) or after reaching a number of specific jobs. In the first case the simulator generates a trigger event from the CallHyper class for releasing the queue after a specific time interval given by the modeler. This deferred queue offers scheduling of jobs in FCFS, SJF, EDF, or PA algorithm. A modeler can easily develop new heuristic scheduling algorithms within the hypervisor.

In the case that an event is scheduled for execution the hypervisor instantiates the static or the dynamic VM deployment policy. The first case generates a VM from scratch by allocating computational resources, installing operating system and software (involving high latencies), while the second case instantiates a VM relying in a remote SAN storage device by migrating the state of its execution within the host. The dynamic case implies opens a history record profile that matches user names and VM profiles for minimizing the latency of VM deployment. For each case the hyper class allocates host power from the host class for achieving VM execution (sandboxing the job request by the VM class). In addition, the VM class generates logs with performance metrics and plots diagrams for each entity separately.

It should be mentioned that the whole of the entities include delays that postpone the job execution within the VM due to latencies in communication. These values are given either by the developer or generated according to a probability function (discussed in [refs]). Additionally, during the simulation, traces are produced and stored in text files for allowing the monitoring of job execution, as well as the user profile is accessed by various entities. The next section presents a simulation experiment and the output of results as produced by the SimIC.

VII. Experimental Scenario of SimIC

The scenario includes the job distribution of the default SimIC configuration. The assumption is that 4 users request for different requirements (VMs) by each of which is 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 we have setup the delay of all components to be 10ms. and we have utilized a priority algorithm as queuing solution. In addition we make use of a dynamic VM instantiation that implies that there is one VM in the SAN storage for each of the users as they are used the same VMs in the past. This involves lowest delays in VM generation. At last the simulation setting includes that each user enters to a cloud according to its id (user 1 to cloud 1, user 2 to cloud 2 etc.), however whit different SLA specification. 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 happens by the meta-broker in order to demonstrate the allocation of jobs to the first capable for performing execution cloud.

Each request (either from the same or not user) is treated by the SimIC as unique. 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 as follows:

$$ExecutionTime_{VM} = \frac{Instruction}{Program}_{user} \times CPI_{user} \times \frac{1}{CPU_{VM}} \times \frac{1}{CPUCores_{VM}}$$

Thus the result is estimated by the next calculation.

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

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

$$Performance_{VM} = \frac{1}{ExecutionTime_{VM}} = 3.33$$

In addition an analogous metric (mips) as a figure for calculating requesting performance is calculated by the *ServiceCharacteristics* class as follows:

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

The hosts of the experiment are included in a text file that contains the names of each cloud host. Table 2 demonstrates a typical host list along with the first host configuration of the list.

TABLE 2: HOST SPECIFICATION FILES

User specification text files	
Host List 1	***Software Requirements***
Cl.St.Hosta.1 Cl.St.Hosta.2 Cl.St.Hosta.3 Cl.St.Hosta.4	HostName: Cl.St.Hosta.1 HostOS: Linux Platform: Intel Memory(GB): 10 CPU-cores: 1 CPU-speed: 10000 H/D-Controller: CD-DVD Storage-HD: 10000 BW: 10000

The following screenshots demonstrate the output capabilities of the simulation for monitoring performance criteria. Figure 2 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.

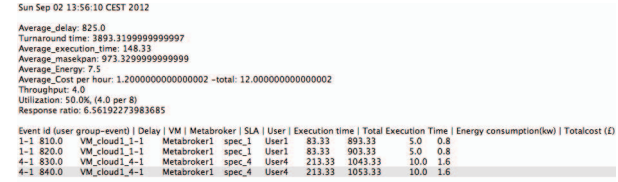

Figure 2: The log of a typical sub-cloud (cloud 1) of the inter-cloud that includes performance metrics values

Figure 3 shows the output of the inter-cloud that involves the collection of all jobs being executed. It should be mentioned the bucket log is 0 (all jobs have been executed). Also, each hyper shows the number of jobs that executes within its queue.

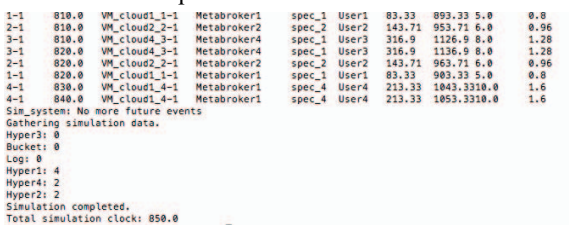

Figure 3: The output of the simulator with regards to the inter-cloud job execution pool.

Figure 4 shows the VM makespan diagram for meta-broker 2. Particularly the makespan represents the productivity of the entity (e.g. the throughput value). Other metrics are also available for generating charts e.g. the turnaround time [8] is calculated as follows:

$$TurnTime_{cloud} = \left\{\frac{instr_{VM} * CPI_{VM}}{CPU_{VM} * CPUCores_{VM} * 10^5}\right\}_{cloud} + CurrentSimTime$$

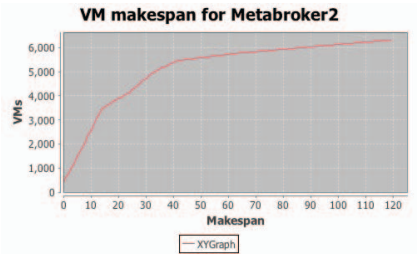

Figure 4: The VM makespan diagram.

Figure 5 shows the SimJava simulation report (a part of the report) that include specific information for each of the entities that include event exchanging during the simulation.

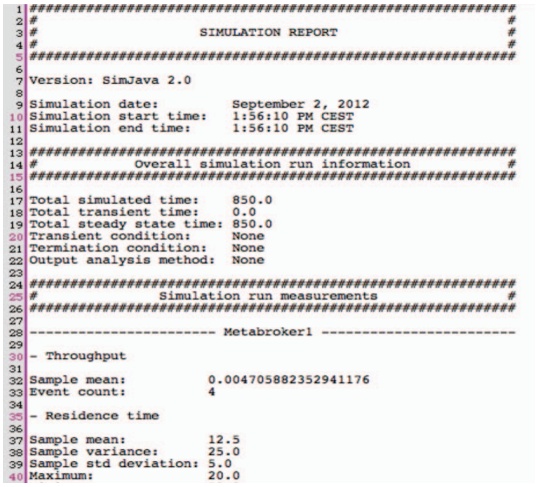

Figure 5: The SimJava simulation report of the SimIC.

At last figure 6 shows a part of the simulation log that includes the job distribution among components including delays and names of entities that exchange events.

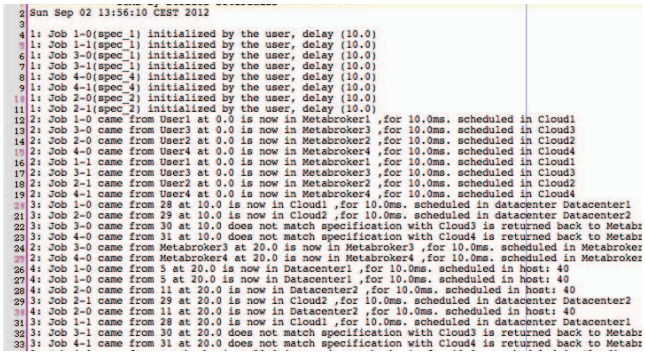

Figure 6: The traces of the events during simulation.

To conclude, this section presented the SimIC capabilities in generating results and outputs. A more detailed discussion of the metrics that are used could be found in [8]. The last work demonstrates in detail the algorithms of job distribution along with sequence diagrams that demonstrate event-exchanging aspects.

VIII. CONCLUSION

This study presented herein introduces the SimIC toolkit (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. 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 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 SimIC implements the ICMS algorithmic structure [8] 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.

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. Finally, the toolkit offers significant advantages by allowing the simulation of users and job submissions that allow decoupling of users and resources based on SLA management cases, dynamic workload decision-making services and real-time scheduling sandboxing of jobs in VMs.

The next development steps include the realization of a collection of requirements for adding built-in SimIC capabilities as follows.

a) Importing energy efficiency measures for optimizing message distribution among entities.
b) Developing meta-scheduling operations on meta-broker level for total decentralized solutions.
c) Increasing the modularity of the system for assisting modelers to define new entities by improving class design.
d) Running various simulation experiments for exploring benchmarks performance based on the CloudSim framework.
e) Adding VM migration cases for scenarios to be available to the modeller. (e.g. disaster case backup).
f) Allowing host scheduling policies to include time-sharing scheduling of multi-cores.
g) Empower simulator to simulate Internet of Things scenarios by implementing a collection of sensors that could utilize the backbone of the inter-cloud infrastructure including more intelligence data management (end users could be everyday people devices e.g. smart city case).

Finally, it should be mentioned that the toolkit is not yet available to for distribution as it still under development. However, modellers that require executing inter-cloud or Internet of Things scenarios are encouraged to communicate with us for collaboratively improving the quality of SimIC.

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., 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.

[3] 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, Pages: 13-31.

[4] Klusáček, D., and Rudová, H. 2010. Alea 2: job scheduling simulator. In Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools '10). ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, , Article 61 , 10 pages.

[5] Núñez, A., Vázquez-Poletti, L., J., Caminero, A. C., Castañé G. G., Carretero, J., and Llorente, M. I., storage networks. iCanCloud: A Flexible and Scalable Cloud Infrastructure Simulator. Journal of Grid Computing, Volume 10, Number 1 (2012).185-209. Springer.

[6] 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.

[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., Kuonen, P. And Antonopoulos, N. (2012). The inter-cloud meta-scheduling framework: Algorithms and performance measures, The 27th IEEE International Conference on Advanced Information Networking and Applications (AINA-2013), Barcelona, Spain, March 25-28, 2013

[9] 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.

[10] SimJava process oriented simulation package, Available at: http://www.icsa.inf.ed.ac.uk/research/groups/hase/simjava/, Accessed 12/09/2012

[11] jFreeChart package, Available at: http://www.jfree.org/jfreechart/, Accessed 12/09/2012