Exploring Models and Mechanisms for Exchanging Resources in a Federated Cloud

Ioan Petri[1], Tom Beach[1], Mengsong Zou[2], Javier Diaz-Montes[2], Omer Rana[3] and Manish Parashar[2]
[1]School of Engineering, Cardiff University, UK
[2] Rutgers Discovery Informatics Institute, Rutgers University, USA
[3] School of Computer Science & Informatics, Cardiff University, UK
contact author: petrii@cardiff.ac.uk

***Abstract*—One of the key benefits of Cloud systems is their ability to provide elastic, on-demand (seemingly infinite) computing capability and performance for supporting service delivery. With the resource availability in single data centres proving to be limited, the option of obtaining extra-resources from a collection of Cloud providers has appeared as an efficacious solution. The ability to utilize resources from multiple Cloud providers is also often mentioned as a means to: (i) prevent vendor lock in, (ii) to enable in house capacity to be combined with an external Cloud provider; (iii) combine specialist capability from multiple Cloud vendors (especially when one vendor does not offer such capability or where such capability may come at a higher price). Such *federation* of Cloud systems can therefore overcome a limit in capacity and enable providers to dynamically increase the availability of resources to serve requests. We describe and evaluate the establishment of such a federation using a CometCloud based implementation, and consider a number of federation policies with associated scenarios and determine the impact of such policies on the overall status of our system. CometCloud provides an overlay that enables multiple types of Cloud systems (both public and private) to be federated through the use of specialist gateways. We describe how two physical sites, in the UK and the US, can be federated in a seamless way using this system.**

Keywords*—*Cloud Computing, Cloud Federation, CometCloud, Tuple-Space, Task Outsourcing.

I. Introduction

Through the federation of cloud systems it has become possible to connect local infrastructure providers to a global marketplace where participants can transact (buy and sell) capacity on demand. The mechanisms used to support cloud federation can bring substantial benefits for service providers by offering facilities for accessing global services instead of increasing costs associated with building new infrastructure (which may not be fully utilized and may only be needed to support peaks in workload over short time frames). More importantly, organisations with spare capacity in the data centre are now provided with a simple way to monetize that capacity by submitting it to the marketplace for other providers to buy, creating an additional source of revenue [1]. Even if computational infrastructure was made available, it may not be possible to host services or data due to issues associated with licencing and intellectual property. Federation in cloud systems has led to a real *democratisation* of cloud markets – enabling businesses to make use of a variety of different cloud providers in different geographic areas. The federated cloud can also provide to end users the ability to host applications with their cloud provider of choice. Similarly, users can choose a local host with the exact pricing, expertise and support packages while still receiving instant access to local or global computational resources.

In federated clouds, providers aspire to accept as many new requests as possible with the main objective of maximising profit; nevertheless, they must guarantee Quality of Service (QoS) based on the agreed Service Level Agreement (SLA) with customers. Establishing an SLA between two parties (client & service provider) implies that the service provider has agreed to provide a particular capability to the client subject to some QoS constraints. In return, the client must provide a monetary payment (most often) or credit to the provider once the service has been delivered (subject to a penalty, often also monetary, in case the quality of service terms have not been adhered to). In a federation context there are several parameters that need to be considered in order to determine the overall federation workflow. When two or more sites come together, it is important to identify not only the incoming workload of each site but also the cost of outsourcing additional resources, the revenue obtained from outsourcing tasks or the cost of maintaining a reasonable level of utilisation. Identifying a set of such parameters is a challenging task due to the variability in the parameters of a federated environment (such as number of resources allocated to local vs. remote jobs, how many jobs to outsource to another site, the time interval over which access to remote jobs should be allowed, etc) and the fluctuation of resource demand. Depending on their value, a site manager must decide whether to outsource resources, compute tasks locally or reject remote task requests [3].

A federated system may have a number of associated access and management policies (based on the sites involved) to be considered in order to increase the utility of providers contributing resources. Such policies would generally include operational and revenue-related parameters for each site. In this paper we propose a CometCloud based federated system by having as an objective the reduction of delay in task processing and maximisation of profit within each site. We develop two different federation models (i) CometCloud Federation, where different sites interact with each other using direct communication and (ii) aggregated CometCloud Federation, where sites interact with each other using a distributed coordination space. Using these federation models we explore different scenarios where tasks are processed exclusively "in house", using local capabilities, or they can be outsourced to remote federated locations. Using these two cases we show how the completion time of the tasks, site utilization, and costs are affected by the choice of a particular policy. Moreover, we include a third scenario that represents a market. In this case, we show how the profit is affected by the chosen benefit per task and the reputation of the involved site.

The reminder of this paper is organised as follows: Section I, section VIII and section III outline the development and use of federated clouds, providing a key motivation for our research (and analysing several related approaches in this area). Section IV presents the model explaining the methodological details of the CometCloud and the *aggregated* CometCloud federation. The evaluation of our implemented system is presented in section VII. We conclude and identify future work in section IX.

II. CometCloud

Our federation models are built on top of CometCloud [16], an autonomic computing engine based on the Comet [17] decentralized coordination substrate, which supports highly heterogeneous and dynamic cloud/grid/HPC infrastructures. CometCloud enables the integration of public/private clouds and autonomic cloudbursts, i.e., dynamic scale-out to clouds to address extreme requirements such as heterogeneous and dynamic workloads and spikes in demands. CometCloud has been demonstrated to work with Amazon EC2 and enables integration with other IaaS providers using specialist gateway nodes.

Conceptually, CometCloud is composed of a programming layer, service layer and infrastructure layer. The infrastructure layer uses a dynamic self-organizing overlay to interconnect distributed resources of various kind and offer them as a single pool of resources. It enables resources to be added or removed from the infrastructure layer at any time as well as robustly deal with disconnects and failures. This dynamic additional/removal is supported through the use of specialist gateway nodes that interact with workers in CometCloud. An information discovery and content-based routing substrate is built on top of the overlay. This routing engine supports flexible content-based routing and complex querying using partial keywords, wildcards, or ranges. It also guarantees that all peer nodes with data elements that match a query/message will be located. The service layer provides a range of services to support autonomics at the programming and application level. This layer supports a Linda-like [18] tuple space coordination model and provides a virtual shared-space abstraction as well as associative access primitives. Dynamically constructed transient spaces are also supported to allow applications to explicitly exploit context locality to improve system performance. Asynchronous (publish/subscribe) messaging and event services are also provided by this layer. No explicit synchronisation between tasks is supported in CometCloud – instead this is assumed to take place by an external application (e.g. a workflow engine) which interacts with the Comet space.The programming layer provides the basis for application development and management. It supports a range of paradigms including the master/worker/BOT. Masters generate tasks and workers consume them. Masters and workers can communicate via the virtual shared space or using a direct connection. Scheduling and monitoring of tasks are supported by the application framework. The task consistency service handles lost/failed tasks. CometCloud is not restricted to applications that have been developed in a particular programming language (e.g. Java), and has been demonstrated to work as a wrapper for a number of other languages (e.g. C, Matlab, Fortran, etc) [2]

III. Approach

Several scenarios and associated policies in the context of cloud federation are outlined in this section. We consider the following two objectives: (i) evaluate the use of CometCloud in a federated context, between two sites connected over a public network – primarily Rutgers & Cardiff. The scenarios are meant to validate the correct functioning of the capability between the two sites. Although we report on two sites, the approach outlined in this work can scale to multiple sites based on the use of the Comet overlay; (ii) demonstrate the development of a market model for resource sharing, based on the use of the CometCloud system between these sites. The approach identified in this paper involves distribution of application execution requests across federated sites – these requests have been identified as "tasks". The granularity of a task is therefore dependent on the size of executable invoked by a worker, and can range from a complete application to a library call.

The process of outsourcing tasks is the result of a series of decisions that must be made at each site. It is therefore important for each site to decide at what point it is appropriate to offload tasks, how many tasks it should offload in one go and how to distribute the tasks between other sites it is connected to. Given a number of tasks to be processed, one site should also decide to outsource tasks based on:

1. <u>Time constraints</u> associated with task processing. Tasks will be outsourced to external sites only if the deadline for completing the tasks cannot be accomplished within the local site. This estimate is based on past executions of similar tasks on the local site vs. the remote site.

2. Lack of <u>computing capability</u> within the site. Tasks will be outsourced to external sites only if the site does not have the required capabilities (software libraries or data sources) for computing the requested tasks. This could be due to the lack of software licences being available locally or the inability to migrate data available at a remote location.

3. <u>Budget</u> constraints associated with task execution – tasks will be outsourced because the costs predicted by the local site for computing the tasks (i.e. new infrastructure deployment) are higher than the costs of off-loading the tasks to other sites. This is an estimate made on previous task executions or advertised costs.

All three of these requirements are based on the use of historical data from prior executions of "similar" tasks. Where such data is not available, we assume that the site will either randomly or alternatively allocate a job to a local vs. a remote resource.

IV. System model

We consider a set of N tasks in the tuple-space, where each task is characterised by three parameters: (i) execution time per task, (ii) type of computation associated with the task, (iii) price of executing the task. Withing the set of tasks N we

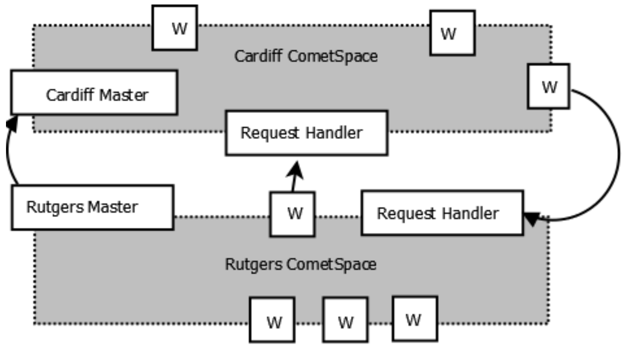


Fig. 1: Federation model

consider two subsets, local tasks (i.e. those executed on local infrastructure) and remote tasks (executed on a federated site). A homogeneous set of tasks is called a "job" when all tasks have the same type of computation.

A. CometCloud Federation Setup

For the CometCloud federation we consider that the tuple-space running at a site has a number of available workers and a master that receives requests: (i) locally – identifying tasks received from users at the same site; (ii) remotely – requests from remote users at the other site – via the use of a request handler.

As illustrated in Figure 1, the Master must decide how many tasks to accept based on a number of policies. We assume that there is one worker per compute/data access node. All workers are assumed to be the same – i.e. they can execute tasks on resources that are identical. The only differentiating factor, therefore, is the number of workers allocated to local vs. external/remote requests. When one site has a high workload and it is unable to process tasks from its local users within their deadlines it negotiate for the outsourcing of tasks to other remote sites. This could range from two cloud systems sharing workload (as in Figure 1) to a cloud outsourcing some of its workload to multiple other cloud systems. Conversely this ability allows systems with a lower workload to utilise spare capacity by accepting outsourced tasks from other cloud systems. Practically, this process of task exchange is undertaken by the master nodes of the two clouds negotiating how many tasks to be exchanged. Once this has been completed the master node on the receiving cloud informs its workers (using CometSpace) about the number of tasks it is taking from a remote site, and the connection details of the request handler from where the task is to be fetched. Subsequently, when a worker comes to execute a task from an external cloud system, it then connects to the request handler of the remote cloud to collect the task and any associated data. Within such a federation model, for each type of task, we use an associated cost such as: (i) *Cost for local jobs* c_l – this is set to a low value – i.e. the worker/resource owner does not gain any substantial financial benefit for processing a local request; (ii) *Cost for remote jobs* c_r – this can be varied over a range – but set to be higher than costs for local jobs. In this instance, the worker benefits financially by processing requests for remote jobs.

B. Establishing CometCloud Federation

The CometCloud federation model is designed to be dynamically updated as it is created in a collaborative way, where each site communicates with others to identify itself, negotiate the terms of interaction, discover available resources, and advertise their own resources and capabilities. In this way, a federated management space is created at runtime and sites can join and leave at any point. This federation model does not have any centralized component and users can access the federation from any site, which increases the fault tolerance of the overall federation, see Figure 2. Another key benefit of this model is that since each site can differentiate itself based on the availability of specialist capability, it is possible to schedule tasks to take advantage of these capabilities.

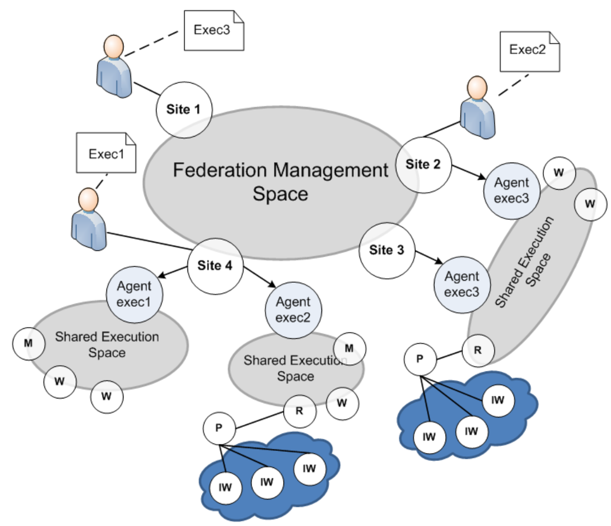


Fig. 2: The overall Federation Management Space, here (M) denotes a master, (W) is a worker, (IW) an isolated worker, (P) a proxy, and (R) is a request handler.

The federation model is based on the Comet [17] coordination "spaces" (an abstraction, based on the availability of a distributed shared memory that all users and providers can access and observe, enabling information sharing by publishing requests/offers to/for information to this shared memory). In particular, we have decided to use two kinds of spaces in the federation. First, we have a single federated management space used to create the actual federation and orchestrate the different resources. This space is used to exchange any operational messages for discovering resources, announcing changes at a site, routing users' request to the appropriate site(s), or initiating negotiations to create ad-hoc execution spaces. On the other hand, we can have multiple shared execution spaces that are created on-demand to satisfy computing needs of the users. Execution spaces can be created in the context of a single site to provision local resources or to support a *cloudburst* to public clouds or external high performance computing systems. Moreover, they can be used to create a private sub-federation across several sites. This case can be useful when several sites have some common interest and they decide to jointly target certain types of tasks as a specialized community.

As shown in Figure 2, each shared execution space is controlled by an agent that initially creates such space and subsequently coordinates access to resources for the execution of a particular set of tasks. Agents can act as a master node within the space to manage task execution, or delegate this role to a dedicated master (M) when some specific functionality is required. Moreover, agents deploy workers to actually compute the tasks. These workers can be in a trusted network and be part of the shared execution space, or they can be part of external resources such as a public cloud and therefore in a non-trusted network. The first type of workers are called secure workers (W) and can pull tasks directly from the space. Meanwhile, the second type of workers are called isolated workers (IW) and cannot interact directly with the shared space. Instead, they have to interact through a proxy (P) and a request handler (R) to be able to pull tasks from the space.

V. Proposed policies

A key policy is that each site must attempt to make maximum revenue from remote tasks without excessively compromising the local tasks running on the system. For developing our policies we use a number of tuning parameters such as: c representing the cost per second per node and t the average execution time per task.

General Policy: When a new job is submitted into the tuple-space, the following action is taken depending on the type of task. For each task, we calculate the time to complete locally – i.e. $ttc = ttc(t) + ttc(j)$, where $ttc(t)$ is the time until the job can start and $ttc(j)$ is the time needed for job to execute.

Policy for Local Tasks: When evaluating a local task the policy is to always accept local tasks, thus local task have a *non-rejection policy*, according to which tasks submitted from local users will never be rejected but queued.

Policy for Remote Tasks: When evaluating a remote task the policy is to accept as many remote tasks as long as $ttc < t$ with an associated price of $p = ttc(j) * c$. Remote tasks have a rejection policy attached according to which remote tasks can be rejected when the site cannot meet the deadline and when accepting remote tasks can affect the processing of local tasks. When outsourcing tasks, a site requests price quotations from all other connected sites and sends as many tasks as possible to the cheapest of these. This policy will be repeated until all tasks have been outsourced. Depending on the number of sites involved, this process could generate significant price information traffic between sites. In our current prototype we consider two sites involved in the federation. To aid scalability, we also foresee the availability of specialist broker services which manage pricing information from multiple sites and can update this information periodically by polling for this information from site manager agents/nodes.

Market Policy: In this policy both local and remote tasks go to a common market for offers from every site interested in executing them. As in the previous cases, tasks are discriminated based on their origin to decide the offered price as well as the resources. Sites only place offers if $ttc < t$.

VI. Implementation details

CometCloud software stack has been developed using Java and it is shipped in a convenient package ready to use. The federation is created ad-hoc where peers talk with each other to negotiate the terms of adhesion. In order for a site to join the federation, it has to support Java, have valid credentials (authorized SSH keys), and configure some parameters (i.e. address, ports, number of workers). Once this configuration is done, we can easily launch the different CometCloud components using simple scripts and the site will join the federation, or it will create it if it is the first one.

Next, we provide additional implementation details within the two federated sites.

A. Cardiff Federation site implementation

We deploy our architecture on a cluster based infrastructure with 12 dedicated cluster machines. Each machine has 12 CPU cores and 3.2 GHz CPU speed. Each physical machine uses a KVM (Kernel-based Virtual Machine) virtualization environment with each entity(master,workers,request-handler) within our system, occupying a single virtual machine. Each virtual machine runs Ubuntu Linux utilising one 3.2GHz core with 1GB of RAM and 10 GB storage capacity. In our system workers, master and request handle are running on a separate virtual instance and using the capability of the virtual instance for their corresponding roles as follows:(i) workers are in charge of computing the actual tasks received from the request handler, (ii) the request handler is responsible with selecting and forwarding tasks to external workers from other federation sites and (iii) the master generates tasks based on the users requests, submits tasks into the tuple-space and collects results.

The networking infrastructure is Gigabit Ethernet with a speed of 1 gigabit per second (Gbps). The measured latency on the network is 0.706ms in average.

B. Rutgers Federation site implementation

The Rutgers federation site is deployed on a cluster based infrastructure with 32 dedicated cluster machines. Each node has 8 cores, 6 GB memory, 146 GB storage and Gigabit Ethernet connection. The measured latency on the network is 0.227ms in average.

In our system, each federation site has a site manager that controls one publisher and several subscribers processes, where the publisher put tuples into the federated space and the subscriber retrieve the desired ones. Tuples can be used to exchange operational messages or to describe tasks that need to be computed. The site manager, including the publisher and subscribers processes are running in a single machine with network access to the rest of the sites. They create the federation upon start and manage the different aspects of the federation. Moreover, behind each site we have a local master/worker setup that will compute the tasks that our particular site has decided to run. Here, the master and workers are all running in independent machines.

VII. Evaluation

Based on the policies identified in the previous section we develop an experimental infrastructure based on CometCloud. This infrastructure includes all the mechanisms needed for the coordination across computational resources which are used by our experiments. Thus, the only simulated part of our experiment is the execution of tasks to evaluate the different policies more accurately. First, we describe simulation settings and performance metrics and then the experimental results we obtained.

A. CometCloud Federation

In this federation scenario, we consider two different experimental setups: (i) non-federated setup – where jobs are being executed exclusively "in house" within the local CometCloud infrastructure and (ii) federation aware outsourcing setup – where jobs are split between the federation sites.

1) Simulation metrics: For both of the setups, we use a number of metrics such as:

- *Average Delay*:$AD = \frac{1}{n}\sum_{i=1}^{n}(Delay)$ identifying the average delay associated with the number of n jobs being executed. $Delay = ttc(e) - ttc(a)$ is measured as the difference between the expected time-to-complete $ttc(e)$ and the actual time-to-complete $ttc(a)$.
- *Profit* – identifying the profit of each site obtained after executing jobs, $Pr = AR * NT * c$, where AR is the average runtime per task, NT represents the number of tasks that one site processes and c represents the cost per core per second (in £) derived from Amazon EC2 cost, $c = 0.0000117697$;
- *Average Utilisation*:$AU = \frac{1}{m}\sum_{i=1}^{m}(Utilisation)$ identifies the percentage utilisation within each of the m workers. This represents the utilisation of each worker given a predefined uniform load distribution for each site.

Tasks processed within the system can have different types such as $TP = \{T_1, T_2, T_3, ...T_r\}$ where each type of task T_i has an associated number of attributes such as $T_i = \{[RT(min : max)], [AR(min + max/2)], [Cost(AR * c)]\}$, where $Runtime(min : max)$ represents an interval associated with prior executions of task type T_i, AR represents the average runtime of task type T_i and $Cost$ is a combination of the average runtime of task type T_i and the price corresponding to a task of type T_i. We use an objective function specifying that each site is trying to minimise the delay in execution and maximise the profit: $f(obj) : [min(delay), max(profit)]$. Associating types with tasks enables us to compare one task with another, especially when considering prior execution history – such types can include: a Matlab script, a graphics rendering operation, a database query operation, etc. Such types can also take into account the size of input data associated with a task, i.e. a Matlab script with 100MB input file and a Matlab script with a 500MB input file being classified as two types. We can therefore associate various semantics with such task types.

In our experiments we consider that each site within the federation experiences a different level of existing workload (i.e. the number of jobs that are currently running at each site). We simulate a varying workload in the context of a varying number and types of tasks and evaluate how delay, profit and utilisation are impacted. As the workload is variable there are three different options that can be applied for each new inserted job according to the federation case being considered: (i) schedule for later processing locally, (ii) reject due to high workload or (iii) outsource to remote site.

2) Experiments: We use two different cloud systems – Cloud A and Cloud B in two different contexts: (a) single cloud context (illustrated as Cloud A and Cloud B) where all the tasks have to be processed locally and (b) federation cloud context (illustrated as Fed. Cloud A and Fed. Cloud B) where the sites have the option of outsourcing tasks to remote sites. When each site deals with an existing high workload, the submission of new tasks from users triggers the process of outsourcing in order to minimise delays and reduce rejection, whereas when each site has a lower workload, there is spare capacity at each site which can be allocated for dealing with new submitted tasks. In our experiments, each site has a total of five workers assigned to process a number of jobs/tasks (10 in our experiment) submitted based on a uniform distribution. In these experiments we consider that Cloud A is the cloud with an existing high workload whereas Cloud B is the cloud with an existing lower workload.

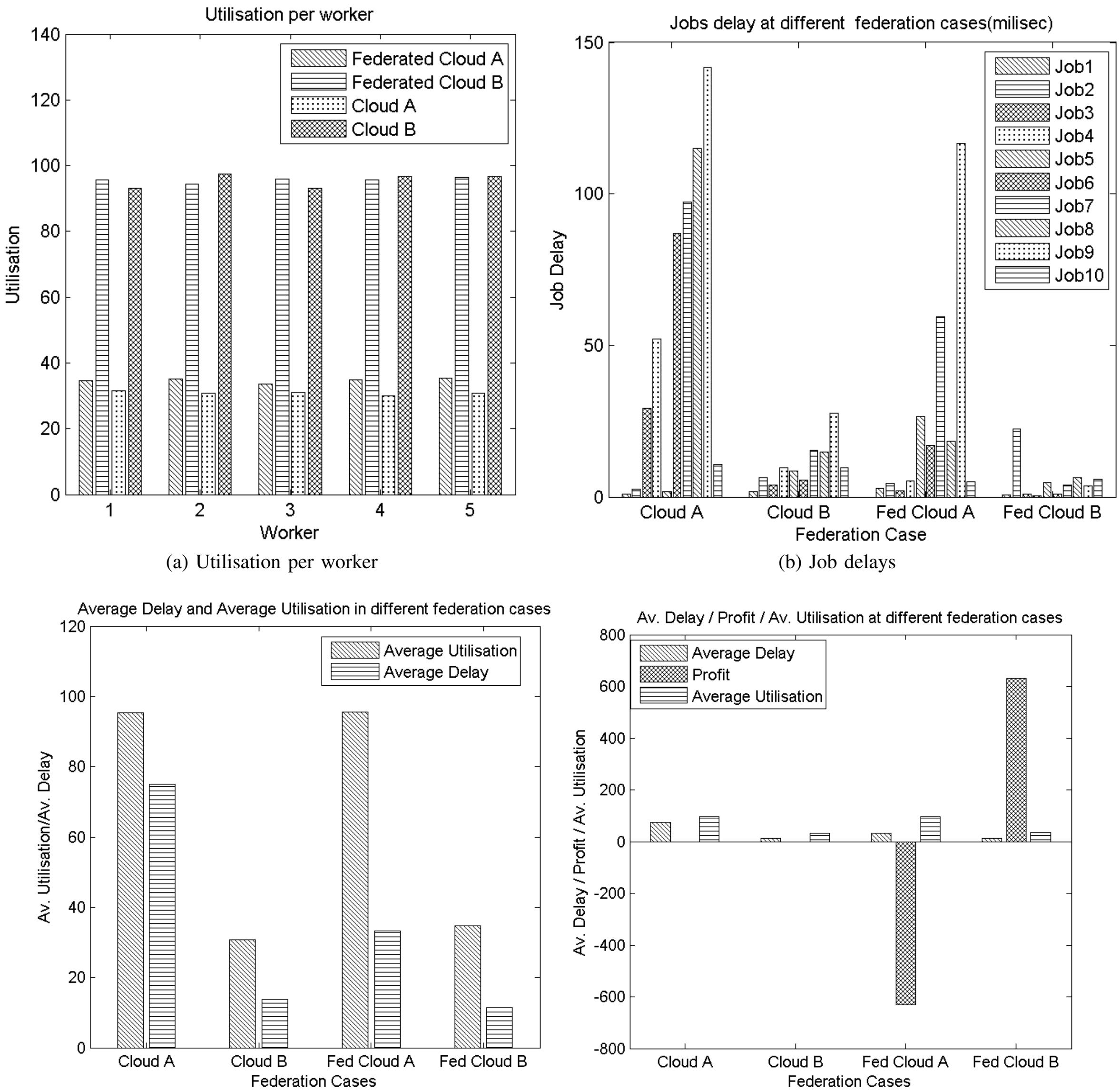
Experiment 1: Utilisation Per Worker:In this experiment we evaluate the level of utilisation per worker given an existing workload load. The experiment illustrated in Figure 3a is presenting a comparison between the case of single cloud (Cloud A and Cloud B) and a federation case (Cloud A and Cloud B federated) with varying sized workloads. It can be identified that in the case of single clouds the utilisation is lower than in the case where the two sites are deployed in a federation. In the context of federation the process of outsourcing is enabled therefore clouds with an increasing load such as Cloud A can decide to offload a percentage of their tasks to remote sites in order to reduce the delays.

Experiment 2: Expected Time-To-Complete(ttc) and Delay: Given a predefined uniform load for each of the sites we investigate what is the impact of this load on the performance of the sites by measuring the delays for each job. From Figure 3b can be observed that the jobs with an increasing delay correspond to the sites dealing with high load. Explicitly presented in Table I, the site with higher load, Cloud A, imposes the highest delay of the processed tasks. When enabling the federation between the sites(Fed. Cloud A and Fed. Cloud B) is observed that the delay corresponding to jobs processed on Cloud A is diminishing. This happens because in the context of federation Cloud A outsources part of its tasks to Cloud B thus improving the overall delay of jobs.

Experiment 3: Average Delay and Average Utilisation: In this experiment we evaluate the performance of the sites from the perspective of average delay and average utilisation. Evaluating the two configurations where sites can either be involved in a federation context or can be single clouds we observe from Figure 3c that the average utilisation and average delay are depending on the level of load existing within each site. Cloud A identifying an increased level of load has a high level of utilisation and delay, in contrast with Cloud B where the load is lower, leading to reduced utilisation and delay. When moving to a federation context we observe that the average utilisation of site A remains high whereas the average delay significantly decreases. This reduction in average delay is caused by the fact that due to an existing high load Cloud A has to outsource part of its tasks to Cloud B, reducing task delay and an increase in the average utilisation of Cloud B.

TABLE I: Time-to-Complete(TTC) and delay for jobs

Clouds/Metric	Job1(TTC-Delay)	Job2	Job3	Job4	Job5	Job6	Job7	Job8	Job9	Job10
Cloud A	60-1.087	80-2.6	50-29.36	150-52.18	10-1.708	90-86.86	80-97.19	50-114.904	180-141.568	10-10.76
Cloud B	210-1.859	80-6.304	20-4.05	70-9.745	60-8.639	180-5.484	80-15.471	300-14.748	100-27.525	40-9.724
Fed. Cloud A	45-2.823	80-4.371	100-1.968	90-5.307	150-26.49	40-16.905	300-59.454	30-18.393	80-116.704	20-4.994
Fed. Cloud B	5-0.648	40-22.486	20-0.942	40-0.53	40-4.846	20-0.97	210-4.006	100-6.434	90-3.542	60-5.827



(a) Utilisation per worker

(b) Job delays

(c) Average delay and average utilisation

(d) Av. Delay/Av. Profit/Av. Utilisation

Fig. 3: Summary of experimental results of CometCloud federation

Experiment 4: Averaged Delay, Profit and Averaged Utilisation: In previous experiments we analysed the system from the perspective of performance metrics such as delay, utilisation and load. In this experiment we measure economic profit in the different federation scenarios. Considering that tasks submitted by local users pay no price and thus generate no profit, we look at the profit exclusively when the sites are involved in a federation. From Figure 3d it can be observed that Cloud A, having a higher level of load, has negative profit (i.e. it costs more to run the site than the revenue that the site generates). On the other hand, Cloud B, having a lower assigned workload, and thus spare capacity can process remote tasks, thereby leading to profit. The oscillation of profit and cost between the two sites is caused by the difference in the level of workload assigned to each site. As Cloud A has an existing high load, when new tasks are submitted it is forced to outsource part of these tasks to Cloud B, paying a corresponding price for each of the tasks but equally significantly increasing the profit of Cloud B. It is therefore necessary to define a policy that balances processing of local vs. remote tasks – to balance fairness of access to local tasks vs. generating revenue from the execution of remote tasks.

B. Aggregated CometCloud Federation

In this experiment, we consider three federated sites using the market policy and study how the profit and reputation of each site varies over time when executing tasks.

1) Simulation metrics: We use the following metrics in this experiment:

- *Profit:* The computation of a task involves a cost, but also provides a benefit. Therefore, the total profit of a site Pr is calculated using the benefit b obtained from each task and the cost c of computing it, $Pr = \sum_{i=1}^{n}(b-c)$. For simplicity we consider that all sites have the same infrastructure cost and therefore we do not include it in the equation.
- *Reputation:* The reputation R of a site is calculated using the scores obtained from the execution of tasks, $R = \frac{1}{n}\sum_{i=1}^{n}(Score)$, where score is a value between 0 and 1. A site obtains a score of 1 if the task is completed within the deadline ($ttc <= deadline$) and 0 if the deadline is missed ($ttc > deadline$).

Tasks executed in the system can be of three types and each one has a different cost, completion time and deadline. The values of these properties are fix for all sites as shown in Table II.

TABLE II: Tasks types properties for all the sites.

Task/Metric	Cost	Completion Time	Deadline
Red	10	9	12
Black	8	7	10
Blue	6	5	8

2) Experiments: In this scenario, the task assignment is done using a blind auction, i.e. a task is put in the federation space and each site places an offer. The client that puts the task into the space decides which site has the best offer and therefore executes the task. Each site has a limited number of resources, which in this case is two workers dedicated to compute local tasks and another two workers for external tasks. Hence, a site only places an offer if the deadline can be met. However, it can happen that one site places multiple offers at the same time before knowing if it won an auction, which could lead to missing deadlines if the site wins many of those tasks. During the experiments each site will submit tasks into the federation using a different Poisson process, where the occurrence of a task is every 4 seconds on average. The distribution of tasks is illustrated in Figure 4.

In these experiments we are going to focus on the effect that the benefit per task and the auction winning criteria have over the profit and reputation of each site:

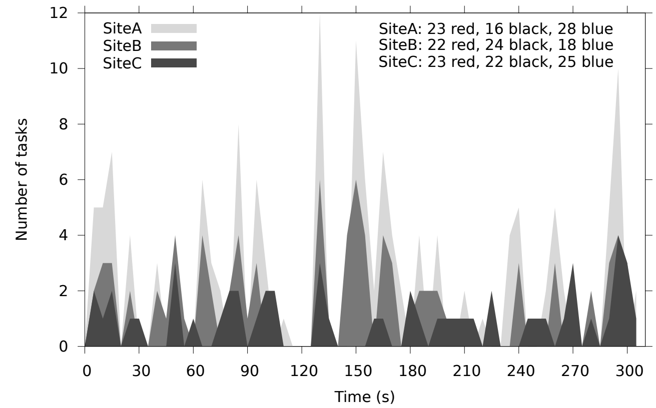


Fig. 4: Task distribution per site following a Poisson process. The type of task was selected randomly for each case.

Benefit per task: This is the benefit that each site wants to obtain from each task. Here it is a percentage of the cost of a task. We will use three different situations:

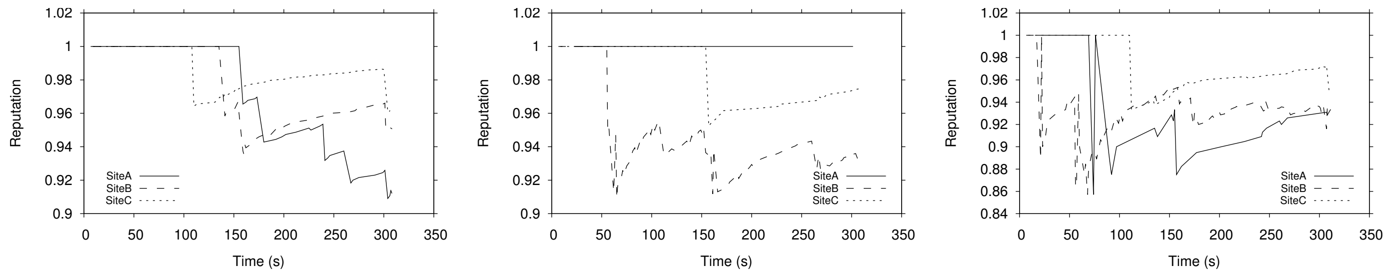
1) All sites get the same benefit per task, which has been fixed to 5% for local tasks and 10% for external tasks (5%/10% abbreviated).
2) One site has an aggressive price policy where it gets only 5%/10% and two sites have very conservative policies where they get 50%/60%.
3) One site has an aggressive price policy where it gets only 5%/10% and two sites have less conservative policies where they get 20%/30%.

Auction winning criteria: it is the criteria to select the best offer of an auction. Here we consider two different criteria:

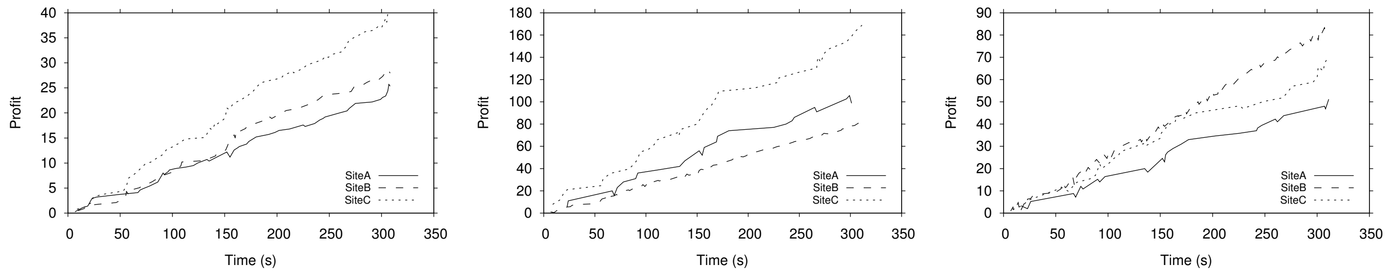
1) We only consider the price that the user has to pay to compute the task and chooses the minimum price among all the offers, $f(obj) = min(p)$.
2) We consider the price and the reputation of the site that has to compute the task, $f(obj) = min(p/R)$. Thus, the best offer is the one that has lower price/reputation ratio.

Experiment 1: Auction criteria based on price only - In these experiments we have considered only the price as criteria to select the auction winner and we have modified the benefit policy of each site to see how both parameters affect the overall profit and reputation of the site. Figures 5a and 5d shows how the reputation and profit evolve when all sites have the same benefit policy. Here, each compute computes a similar number of tasks (A=58,B=63,C=79), which is close to the total number of tasks generated from that site. The reason is that each site has the lower price for its own tasks and they will win local tasks until the resources assigned to those tasks are full. Then they will stop bidding for their own tasks and others sites will have the chance of executing those tasks. This can represent a scenario where local tasks are outsourced to external resources when the load of the site is high.

Figures 5b to 5f collect the results of the seconds and third benefit situations, where site B has an aggressive price policy and the rest has more conservative ones. We can observe that Site B does not always obtain the highest profit, although it

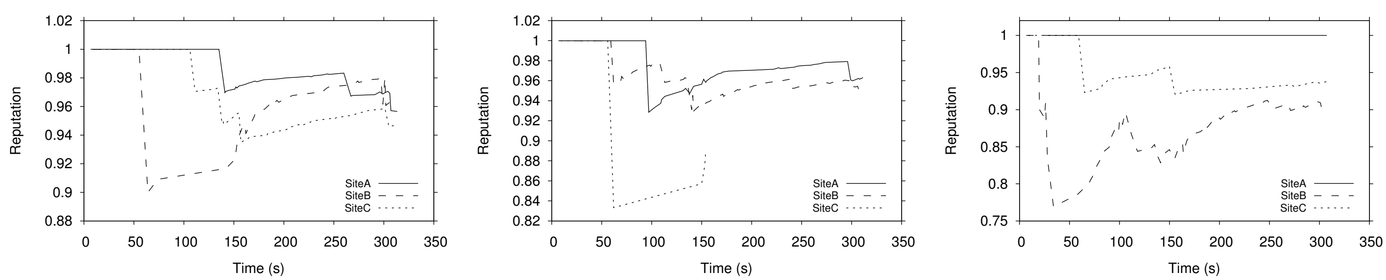


(a) Reputation at ABC:5%/10% (b) Reputation at B:5%/10%;AC:50%/60% (c) Reputation at B:5%/10%;AC:20,30

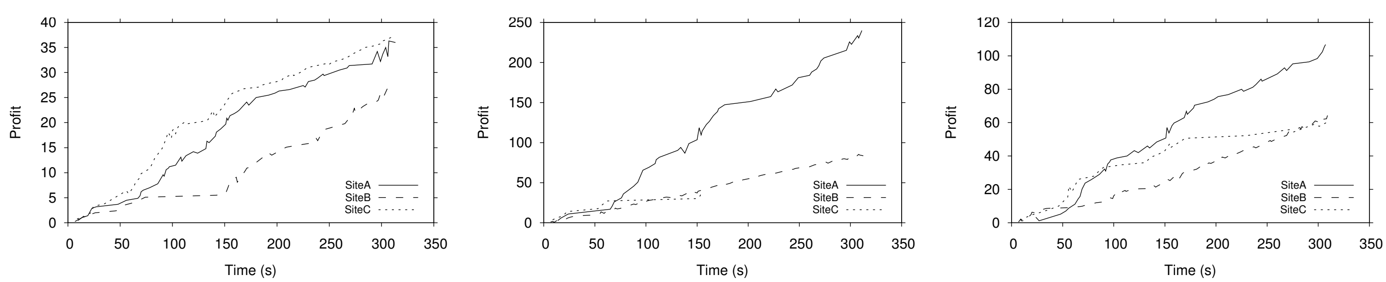


(d) Profit at ABC:5%/10% (e) Profit at B:5%/10%;AC:50%/60% (f) Profit at B:5%/10%;AC:20%/30%

Fig. 5: Summary of experimental results where the criteria to win the auction is the minimum price. There are three set of experiments where each site has a different benefit policy, i.e. B:5,10;AC:20,30 means that site B gets a 5% and 10% of benefit of each local and remote tasks respectively, while sites A and C get 20% and 30%.



(a) Reputation at ABC:5%/10% (b) Reputation at B:5%/10%;AC:50%/60% (c) Reputation at B:5%/10%;AC:20%/30%



(d) Profit at ABC:5%/10% (e) Profit at B:5%/10%;AC:50%/60% (f) Profit at B:5%/10%;AC:20%/30%

Fig. 6: Summary of experimental results where the criteria to win the auction is the minimum price/reputation ratio. There are three set of experiments where each site has a different benefit policy, i.e. B:5,10;AC:20,30 means that site B gets a 5% and 10% of benefit of each local and remote tasks respectively, while sites A and C get 20% and 30%.

is by far the one that executes more tasks. In both cases case B executes around 130 tasks while the other two sites execute around 35 tasks each. This is due to the limited number of resources and shows that when different sites have similar resources, a very aggressive price policy may not be profitable. We can also observe that the reputation of B changes abruptly because it places many auctions offers before it knows the results of those actions and it end up winning too many tasks for its limited resources.

Experiment 2: Auction criteria based on price and reputation - We have repeated the same experiments changing the auction criteria to consider not only the price of the task but also the reputation of the site that places the offer. Figures 6a and 6d show that site B looses a lot of reputation at the beginning of the experiment, which has an important effect in the overall profit of this site. Figures 6b and 6e show an interesting situation. Here, site C looses a lot of reputation at the beginning of the experiment, which combined with its high prices turn out in executing only nine tasks. This clearly benefit site A that executes 54 tasks and gets high profit at the end. Finally, Figures 6c and 6f show how site B can deal with a bad reputation thanks to its low price policy. As opposed to Site C in the previous case, here B keep computing tasks and it can even improve its reputation.

VIII. Related work

Several approaches have been reported in literature to support cloud federation. Villegas et al. [5] proposed a composition of cloud providers as an integrated (or federated) cloud environment in a layered service model. In [13], Celesti et al. proposed a cross-federation model based on using a customized cloud manager component placeable inside the cloud architectures. Assuncao et al. [6] described an approach for extending a local cluster to a cloud resource using different scheduling strategies. Along the same lines, Ostermann et al. [7] extended a grid workflow application development and computing infrastructure to include cloud resources and carried out experiments on the Austrian Grid and an academic cloud installation of Eucalyptus using a scientific workflow application. Similarly, Vazquez et al. [8] proposed an architecture for an elastic grid infrastructure using the GridWay meta-scheduler and extended grid resources to Globus Nimbus. Bittencourt et al. [9] proposed an infrastructure to manage the execution of service workflows in a hybrid system composed of both grid and cloud computing resources. Analogously, Riteau et al. [10] proposed a computing model where resources from multiple cloud providers are used to create large-scale distributed virtual clusters. They used resources from two experimental testbeds, FutureGrid in the United States and Grid'5000 in France to achieve this.

The Reservoir project [4] proposes an open service-based environment in which resources and services are transparently provisioned and managed across clouds. In the Reservoir model, the providers of the services and the infrastructure providers are two separated entities, where the service that a service providers commits to provide is actually outsourced to an infrastructure provider. The Reservoir project makes the assumption that federation and interoperability can enable infrastructure providers to take advantage of their aggregated capabilities to provide a seemingly infinite service computing utility. Therefore a provider federates with other providers (i.e., other Reservoir sites) based on its own local preferences within a fully autonomous environment and governed by particular business policies. For optimising the process of federation in Reservoir, a service may be moved to other sites based on economic performance or availability considerations.

Provider revenue in federated systems represents an important aspect especially in the context of outsourcing of services to another, external, entity. Goiri et al. [12] explore federation from the perspective of a profit-driven policy for outsourcing (a required capability) by attempting to minimise the use of the external resource (and thereby the price of resources paid to the external provider). Using a set of monitored parameters, such as the providers incoming workload, the cost of outsourcing additional resources, the ratio of outsourced resources and the ratio of unused resources to be sold, the system can make decisions about the placement of services on the nodes of the provider and the number of resources that must be allocated to each node in order to guarantee that they meet an agreed performance (and to maximise "utility").

More work on the topic of outsourcing policies has been undertaken by Toosi et al. [3]. This work focuses on specifying reliable policies to enable providers to decide which incoming request to select and prioritise, thereby demonstrating that policies can provide a significant impact on the providers' performance. When optimising parameters such as the ratio of spot Virtual Machines (VMs) to the total workload, percentage of persistent spot VMs, number of providers in the federation and their workload, providers are able to maximise profit and to reject less requests, while they keep utilisation at an acceptable level. It has also been identified that running on-demand requests locally is more profitable if the provider has high ratio of spot VMs. Conversely, outsourcing proves to be more profitable when spot VMs are scarce and termination of them may result in discontinuation of using such services by customers.

The possibility of creating federated markets has been extensively explored by researchers [14], [15]. Federation markets provide the mechanisms to mostly promote fairness and ensure mutual benefits for parties involved in the federation. Gomes et al. [14] explore the application of federation mechanisms based on the General Equilibrium Theory to coordinate the sharing of resources between providers in a federated cloud – with an objective of optimising costs and coping with variations in demand.

Although these solutions enable participants to increase capacity and extend an existing market place, these efforts still leave undetermined when a participant should outsource a local request (compared to executing it locally) and the types of policies that may be used to govern such a federation model. In this work we target these research challenges by exploring different scenarios implemented in a real federated Cloud system based on CometCloud. CometCloud uses a distributed coordination space to facilitate task processing across sites. CometCloud has been demonstrated to integrate both public clouds (based on Amazon EC2/S3) and specialist high performance computing environments [19], [20].

IX. Conclusion and Future Work

Federated Cloud systems enable individual, possibly competing, Clouds to cooperate so as to optimize costs and cope with variations in demand. In a community of Cloud providers federation can also enable the process of trading resources, thereby greatly encouraging the emergence of federated markets. We demonstrate the implementation of a federated Cloud system, making use of an aggregated tuple-space and a policy-based resource assignment to tasks. The design and implementation of the system have been described along with evaluation through a number of synthetic workloads (based on our experience of computational science applications) for both individual and federated Clouds. The experimental results demonstrate benefits of federation, for resource utilisation, execution delay, cost and reputation. When evaluating the utilisation and delay in a CometCloud federation we observe that the existing workload of each site determines the level of utilisation for each worker. Utilisation, on the other hand can be the cause of high delay especially in a non-federation context.

The main advantage of aggregated CometCloud federation is the use of a centralized mechanism which enables users to access any site involved in the federation. Task assignment to various sites is influenced by a market mechanism (based on bids and offers), coordinated through Comet space. By using different auction criteria and pricing strategies we demonstrate that when sites are playing aggressive price strategies, the completion of jobs is strongly affected and consequently also the associated benefit and reputation of sites. As the auction strategies and price policies are strongly related to the number (and type) of resources within each site, it has been demonstrated that when different sites have similar resources, a very aggressive price policy may not be profitable.

As ongoing work, we can also consider how sites can specialise and only select tasks which match their particular capability (based on a subscription based model). This would enable sites to evolve over time to select the type of tasks that give them the most profit, reputation, etc. Moreover, sites could also dynamically change their prices to increase benefit or recover reputation as needed. Conversely, policies may include references to a Service Level Agreements (SLAs), enabling sites to determine if they are willing to take the risk of violating and SLA and incurring a penalty. Our current work has also focused on investigating two application scenarios – Energy+ and an image processing pipeline to determine values of ttc identified in section V through benchmarking. Given different data sizes and parameter ranges for these applications, we can estimate values of ttc.

Acknowledgements: The research presented in this work is supported in part by US National Science Foundation (NSF) via grants numbers OCI 1339036, OCI 1310283, DMS 1228203, IIP 0758566 and by an IBM Faculty Award.

REFERENCES

[1] I. Goiri, J. Guitart, and J. Torres. Economic model of a cloud provider operating in a federated cloud. Information Systems Frontiers, pp. 1-17, 2011.

[2] CometCloud: An Autonomic Cloud Engine, H. Kim and M. Parashar, Cloud Computing: Principles and Paradigms, Editors: R. Buyya, J. Broberg, A. Goscinski, John Wiley & Sons, ISBN 978-0-470-88799-8, pp. 275 297, 2011.

[3] Adel Nadjaran Toosi, Rodrigo N. Calheiros, Ruppa K. Thulasiram, and Rajkumar Buyya, Resource Provisioning Policies to Increase IaaS Provider's Profit in a Federated Cloud Environment. In Proceedings of the 2011 IEEE Int. Conf. on High Performance Computing and Communications (HPCC '11). IEEE Computer Society, Washington, DC, USA, pp. 279-287, 2011.

[4] Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M.Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben- Yehuda, W. Emmerich, and F. Galan, The Reservoir model and architecture for open federated Cloud computing, IBM Journal of Research and Development, vol. 53, no. 4, pp. 1-17, Jul. 2009.

[5] D. Villegas, N. Bobroff, I. Rodero, J. Delgado, et al., Cloud federation in a layered service model. J. Comput. Syst. Sci., 78(5):1330-1344, 2012.

[6] M. D. de Assuncao, A. di Costanzo, and R. Buyya, Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters, In Proceedings of the 18th ACM international symposium on High Performance Distributed Computing (HPD, pp. 141-150, 2009.

[7] S. Ostermann, R. Prodan, and T. Fahringer, Extending grids with cloud resource management for scientific computing, In Proceedings of the 10th IEEE/ACM International Conference on Grid Computing, pp. 42-49, 2009.

[8] C. Vazquez, E. Huedo, R. Montero, and I. Llorente, Dynamic provision ofcomputing resources from grid infrastructures and cloud providers, Proceedings of the 2009 Workshops at the Grid and Pervasive Computing Conference, pp. 113-120, 2009.

[9] L. F. Bittencourt, C. R. Senna, and E. R. M. Madeira. Enabling execution of service workflows in grid/cloud hybrid systems. In Network Operations and Management Symp. Workshop, pp. 343-349, 2010.

[10] P. Riteau, M. Tsugawa, A. Matsunaga, J. Fortes, and K. Keahey. Large-scale cloud computing research: Sky computing on futuregrid and grid, In ERCIM News, 2010.

[11] R. Buyya, R. Ranjan, and R. N. Calheiros, InterCloud: Utility-oriented federation of Cloud computing environments for scaling of application services, in Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP10), LNCS vol. 6081. Busan: Springer, pp. 13-31, May 2010.

[12] I. Goiri, J. Guitart, and J. Torres, Characterizing Cloud federation for enhancing providers profit, in Proceedings of the 3rd Int. Conf. on Cloud Computing. Miami: IEEE Computer Society, pp. 123-130, July 2010.

[13] Celesti, Antonio and Tusa, Francesco and Villari, Massimo and Puliafito, Antonio, How to Enhance Cloud Architectures to Enable Cross-Federation, Proceedings of the 2010 IEEE 3rd Int. Conf. on Cloud Computing, CLOUD '10, IEEE Computer Society, pp. 337–345, 2010.

[14] E. Gomes, Q. Vo, and R. Kowalczyk, Pure exchange markets for resource sharing in federated Clouds, Concurrency and Computation: Practice and Experience, vol. 23, 2011.

[15] B. Song, M. M. Hassan, and E.-N. Huh, A novel Cloud market infrastructure for trading service, in Proceedings of the 9th International Conference on Computational Science and Its Applications (ICCSA). Suwon: IEEE Computer Society, pp. 44-50, Jun. 2009.

[16] CometCloud Project. http://www.cometcloud.org/. Last accessed: August 2013.

[17] L. Zhen and M. Parashar, "A computational infrastructure for grid-based asynchronous parallel applications", HPDC, pp. 229-230, 2007.

[18] N. Carriero and D. Gelernter, "Linda in context", Commun. ACM, vol. 32, no. 4, 1989.

[19] H. Kim, Y. el-Khamra, S. Jha, I. Rodero and M. Parashar, "Autonomic Management of Application Workflow on Hybrid Computing Infrastructure", Scientific Programming Journal, Jan. 2011.

[20] M. Parashar, M. Abdelbaky, I. Rodero, A. Devarakonda, "Cloud Paradigms and Practices for Computational and Data-Enabled Science and Engineering", IEEE Computing in Science and Engineering (CiSE) Magazine, to appear.