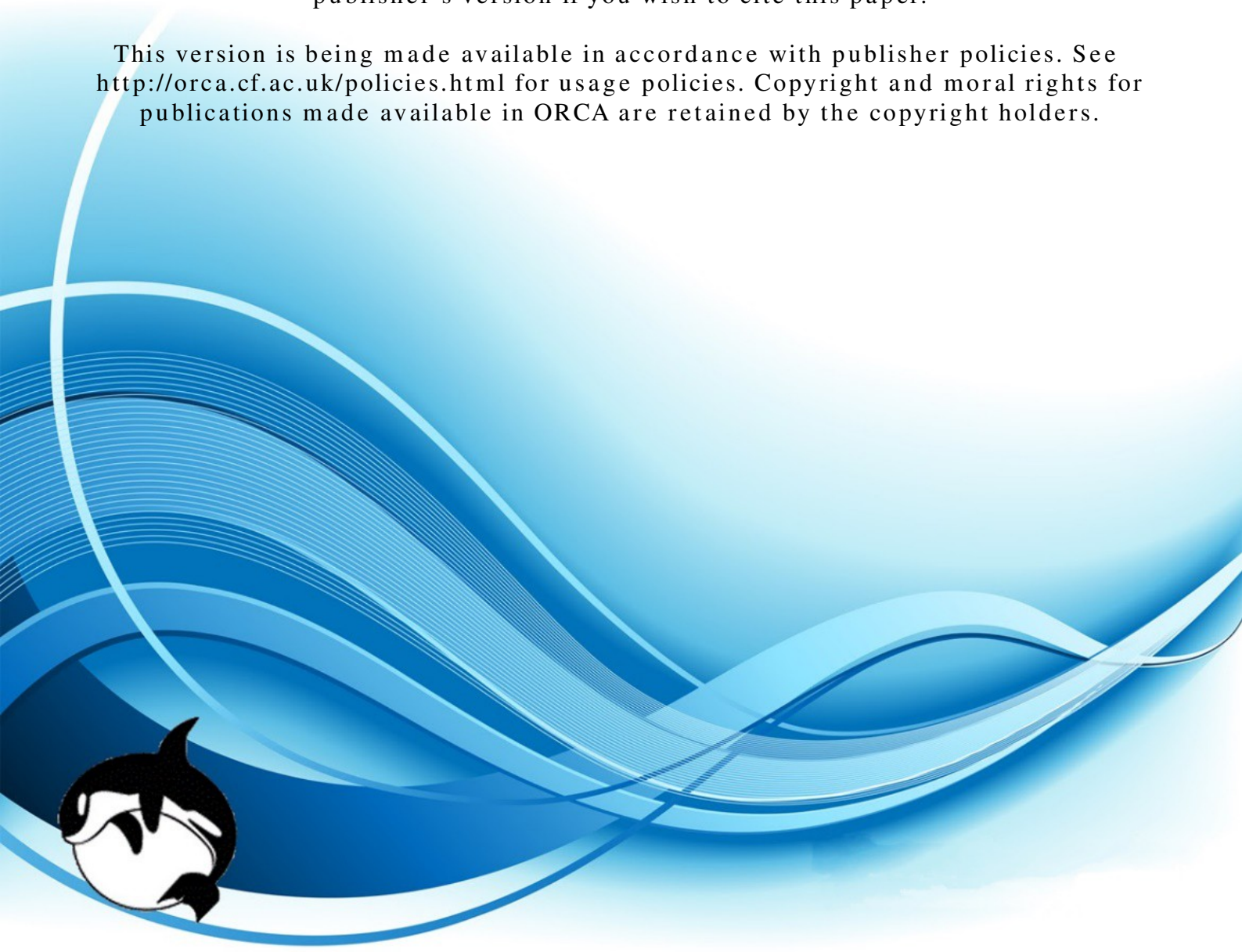
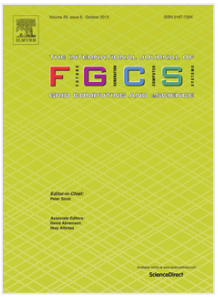


Accepted Manuscript



Risk assessment in service provider communities

Ioan Petri, Omer F. Rana, Gheorghe Cosmin Silaghi, Yacine Rezgui

PII: S0167-739X(14)00161-7
DOI: http://dx.doi.org/10.1016/j.future.2014.08.013
Reference: FUTURE 2607

To appear in: *Future Generation Computer Systems*

Received date: 7 July 2012
Revised date: 16 August 2014
Accepted date: 20 August 2014

Please cite this article as: I. Petri, O.F. Rana, G.C. Silaghi, Y. Rezgui, Risk assessment in service provider communities, *Future Generation Computer Systems* (2014), http://dx.doi.org/10.1016/j.future.2014.08.013

Risk Assessment in Service Provider Communities

Ioan Petri[a,*], Omer F. Rana[a,**], Gheorghe Cosmin Silaghi[c], Yacine Rezgui[b]

[a] *Cardiff University, School of Computer Science & Informatics, Wales, United Kingdom*
[b] *School of Engineering, Cardiff University, United Kingdom*
[c] *Babeş-Bolyai University, Business Information Systems, Cluj-Napoca, Romania*

Abstract

On-line service delivery undertaken between clients and service providers often incurs risks for both the client and the provider, especially when such an exchange takes place in the context of an electronic service market. For the client, the risk involves determining whether the requested service will be delivered on time and based on the previously agreed Service Level Agreement (SLA). Often risk to the client can be mitigated through the use of a penalty clause in an SLA. For the provider, the risk revolves around ensuring that the client will pay the advertised price and more importantly whether the provider will be able to deliver the advertised service to not incur the penalty identified in the SLA. This becomes more significant when the service providers outsource the actual enactment/execution to a data centre – a trend that has become dominant in recent years, with the emergence of infrastructure providers such as Amazon. In this work we investigate the notion of "risk" from a variety of different perspectives and demonstrate how

*Ioan Petri
**Omer F. Rana
Email addresses: petrii@cardiff.ac.uk (Ioan Petri), o.f.rana@cs.cardiff.ac.uk (Omer F. Rana), gheorghe.silaghi@econ.ubbcluj.ro (Gheorghe Cosmin Silaghi), rezguiy@Cardiff.ac.uk (Yacine Rezgui)

risk to a service owner (who uses an external, third party data centre for service hosting) can be managed more effectively. A simulation based approach is used to validate our findings.

Keywords: Risk management, Service Level Agreement, Fault tolerance, Multi-tenancy

1. Introduction

With the emergence of Cloud computing it has become possible to differentiate between a software service owner (responsible for updating and managing a software capability encapsulated as a service) and an infrastructure provider (primarily offering computational, data and network resources that may be used to deploy the software service). A service owner can utilize the capability of one or more such infrastructure providers to offer the capability to clients, whereas an infrastructure provider looks for possible service owners to offer them managed access to resources, often at a pre-advertise price, at multiple capacities (small, medium and large instances in the case of Amazon.com, for instance) and with varying types of Service Level Agreements. Such differentiation between the service owner and infrastructure provider role is useful from a market perspective, as it enables different combinations of price-performance tradeoffs to be made available, thereby reducing the barrier to entry within a marketplace (as service owners no longer need to manage complex infrastructure which often incur significant capital cost) whilst also allowing specialist infrastructure providers to emerge on the market.

Cloud and web applications experience huge and unpredictable variation

in the load over time. Defining the required amount of instances to cope with the load experienced in a given moment can incur risks for both clients and providers. In few cases the load demand is known beforehand thus users could reserve the required amount of instances – a situation which is cheaper than acquiring on-demand instances. However, as loads are unpredictable and variable, users have to combine reserved instances with on-demand instances as well as balance between cost and utilization of the resources. A variance in the pattern of utilization by a client gives the provider an opportunity to offer an on-demand option as a strategy to maximize their profit. Providers generally offer guaranteed availability based on a pre-agreed Service Level Agreement (SLA) [20] with a client.

It is therefore important to understand risk from a financial perspective (expressed as cost and profit) in order to enable service owners to successfully utilize the resources of an infrastructure provider. In addition, the problem of risk assessment and cost becomes increasingly important in the context of open markets where various providers can join and contribute computational capacity and where clients can place requests for various services [21]

The focus of this paper is to determine how a service owner can balance: (i) the loss in revenue incurred due to failure, with (ii) the additional cost of replication needed to prevent SLA violation, in a multi-tenancy environment. We investigate the problem of service outsourcing from a financial perspective in a multi-tenancy environments where a number of services can be combined and deployed over server farms. Determining the number of replicas to support service replication needs to be balanced with the revenue achieved through each service instance and the likely penalty that may arise

due to unavailability (arising from a failure). Section 2 describes the motivation of this work evaluating risks from different perspectives. Section 3 presents the overall methodology we employ to analyse risk for single service outsourcing, extended in section 6 to multiple service outsourcing where deployment can be across multiple server farms. Section 4 presents the simulation framework used for conducting the experiments. Section 5 and 6.1.1 provide the evaluation of the work through a number of experiments carried out with the PeerSim simulator. Section 7 discusses related work in risk management and virtual appliances with a particular emphasis on financial risk. We present our conclusions is section 8.

2. Motivation and approach

Utilizing external infrastructure to deploy services incurs risks for both the service owner and the infrastructure provider. Our focus is primarily on financial risk, invoking the notion of uncertainty and randomness within an exchange between a client and a provider. Significant literature exists about the notion of risk in financial markets, with this term being used synonymously with the "probability of a loss or gain arising from unexpected changes in market conditions" [7]. Although in a financial market risk is often associated with a change in market price of a product or derivative, in the context of this work, we associate risk with the likely financial loss that a service owner or infrastructure provider will incur due to their inability to deliver an advertised capability. It is therefore necessary for the service owner to consider one of the following three options: (i) *trust* the infrastructure provider and assume a certain degree of fault tolerance and

resilience; (ii) *establish a Service Level Agreement (SLA)* to ensure that if a provider is unable to deliver the advertised capability, the infrastructure provider incurs a financial penalty that must be payed to the service owner; (iii) *utilize resilience mechanisms* directly to ensure that any possible faults that may arise can be overcome through a pre-identified strategy, thereby ensuring continued, fault free operation for clients. In (i) when dealing with trusted participants the process is simplified as there are already a number of approaches to ensure correct service provisioning. Trust may be established based on prior interaction with an infrastructure provider or based on the general reputation of the provider within the marketplace. This aspect has been investigated previously by a number of researchers [9, 10]. On the other hand, in the context of untrusted environments ensuring fault free operation can be difficult due to a variety of possible outcomes that may arise during operation. This scenario is particularly prevalent when these parties are unknown to each other and therefore the level of risk associated with the transaction is considerably increased. Expanding on the three considerations identified above:

1. *Using trust mechanisms* – this is applicable when the environment is trusted and either: (i) clients and service providers have already interacted with each and have a history of prior (un)successful interactions; (ii) clients and service providers have access to feedback from other entities they trust – or through an aggregated reputation service they can access. Reputation can either be based solely on prior transactions, or be considered as a multi-dimensional characteristic involving technology, business preferences and usage/business policy – and their combinations [8]. With (ii), the feed-

back data provided by others to calculate the reputation may be misleading and/or sparse – thereby limiting its benefit.

Hence, entities providing feedback can have different types of behaviours (both truth telling and deception), whereby feedback about a particular provider may be influenced by particular incentives that a client may have. By using existing trust mechanisms such malicious intent (based on incorrect feedback) can bias the overall trust establishment within a community of clients and service providers and trust values may change with the number of clients involved in the community and with those providing feedback [11].

2. *Using Service Level Agreements* – this is applicable when the participants are unknown to each other – and therefore untrusted – with the behaviour of the participants being regulated through a previously agreed SLA. Such agreements can be particularly efficacious for mediating business transactions providing a useful reference point for monitoring capability exchanged between a client and provider (given that monitoring is carried out by either a trusted third party or through a pre-trusted component known to the client and the provider). An SLA may be used to specify Quality of Service (QoS) terms, the measurement criteria, reporting criteria and penalty/reward clauses between participants. Within an electronic market, an SLA may be used for: (i) an economic expression/proof of debts as well as credits – debts to the client and credits to the service provider; (ii) as a token of exchange between participants; (iii) as an identification of responsibilities of participants involved (such as the client and service provider). 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 (Bitcoins or other alternative currency) 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) [1].
3. *Using fault tolerance techniques* – this is applicable when dealing with unknown participants whose behaviour cannot be predetermined. Although a client (the service owner) may have an SLA with the provider, the client may still wish to minimise risk by ensuring that suitable fault tolerance strategies are available. For instance, establishing SLAs with entities that may exhibit faulty behaviours may represent a high risk. In order to mitigate these risks we propose a fault tolerance mechanism where various services are replicated among a number of peer-nodes.

In the context of service provision, fault-tolerance has moved from hardware to software, making failure a "normal" event that has to be managed efficiently. Referring to hardware failures within a cluster of 1,800 servers that Google uses as the building block for its infrastructure, Miller (2008) [22] proves that dependability of failures in large scale datacentres can affect significantly the availability of multiple cluster units.

Our approach tries to provide a comprehensive solution by determining how a service owner can control the loss in revenue incurred due to failure and the costs with replication. We focus on the optimisation of costs and profits within the system showing how replicas can be deployed considering the load of demand received from clients and the revenue achieved through each service instance.

Assumptions: In the context of this work we make the following assump-

tions:

- An SLA is a contract between two parties, a client and a provider. The object of the contract can be a single service as identified in "Single service outsourcing" section or a combination of services (defined as a service type) as identified in the "Multiple service outsourcing" section.

- A request of service refers to an event issued by a client for acquiring a resource available at the provider. This resource can be embedded into: (i) a single service or (ii) a generic type of services (which is a combination of multiple individual services). Both of these requests are eventually defined as a functional SLA, the only differentiation factor represents the service based on which the request has been placed.

- The SLA creation process starts when a client (the initiator) sends a request to a potential provider. The provider issues an SLA template, specifying agreement terms and obligations – containing service level objectives, quality terms and business values associated with particular service level objectives. Penalties and rewards are also parts of the SLA template. The initiator fills the template with the required service, asking the provider for a price. The agreement is finalised when the initiator accepts the price from the provider. The underlying protocol can be found in the WS-Agreement specification [23].

3. Single service outsourcing

When establishing an SLA, the service provider agrees to provide a particular capability to the client subject to some QoS constraints – referred to as

Service Level Objectives in the WS-Agreement specification. But when the service providers replicate their services (within one or more infrastructure provider(s) or data centre(s)), there are three important aspects to consider: (1) Risk from the service owner's perspective i.e. how many instances of each service should be replicated taking into account: (i) the cost associated with deploying each replica, and (ii) the penalty that must be paid if a service is unavailable (i.e. no working replica is available when a request is sent by a client).
(2) Risk from the infrastructure provider's perspective, i.e. determine how to optimise service replication in order to reduce deployment (hardware and software) costs and any penalties that may arise due to SLA non-compliance.
(3) Risk from the client's perspective i.e. how to construct/negotiate the SLA considering that a service owner may be unable to deliver the service.

Within these risks, the SLA can be encoded with WS-Agreement standard. After receiving a service request from node A, node B responds with a WS-Agreement template which node A is expected to complete with the following information:

- Name/ID: a unique identifier for the agreement;
- Context: metadata associated with the agreement, such as the agreement initiator, agreement responder, the expiration time etc.
- Service terms: details about the service being provided.
- Guarantee terms: details such as service level objectives, qualifying conditions for the agreement to be valid, penalty terms, etc. The guarantee terms specify which the obligated party is and to which service

this guarantee applies, the service level indicators and service level objectives as an assertion over service descriptions and the business value associated with these objectives.

Consider a server farm SF containing a collection of peer-nodes Pe=$\{pe_1, pe_2, pe_3, ..., pe_n\}$, some of which can be used for replicating a set of services S. S is a collection of services S=$\{s_1, s_2, s_3, ..., s_n\}$ deployed on the server farm SF. A subset $S_k \subseteq S$ defines a collection of services S_k=$\{s_{s1}, s_{s2}, s_{s3}, ..., s_{sm}\}$,$m < n$ owned by a provider Pr_k where each service s_i has a number of replicated instances I_k. The set I_k identifies a set of instances for one service s_i, I_k=$\{i_1, i_2, i_3, ..., i_m\}$. The associated costs of set of instances I_k is the set of costs C_k=$\{c_1, c_2, c_3, ..., c_m\}$; hence pairs (i_i, c_i) are associated with service s_i with c_i representing the cost of deploying instance i_i.

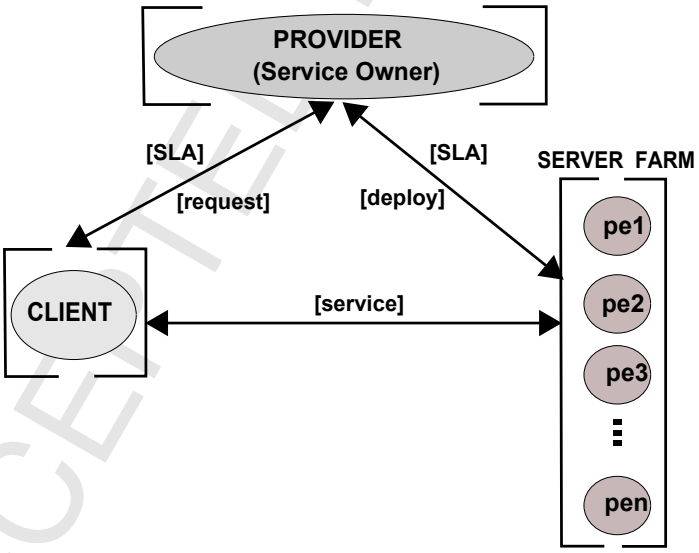


Figure 1: Service Provision Scenario

For defining the cost and profit for clients, providers and farm owners we use the following notations: $Cost$ represents the cost, $Profit$ represents the profit, Cl represents the client, Pr represents the provider and Fo represents the farm owner. From Figure 1, a client sends a request to a provider which then outsources the service execution to a specialised server farm. Based on the provider requirements the server farm will accommodate an amount of replicas for the requested service on the number of available peer-nodes (identified as $p_{e1}, p_{e2}, p_{e3}, ..., p_{en}$ in Figure 1) in the farm.

3.0.1. Clients

A client searches for a service owner able to deliver a required capability, subject to a set of QoS constraints. We use SLA_{Cl}^{Pr} to identify the provider commitment to deliver a service to the client, where the SLA encodes the particular constraints that have been agreed between the two parties. Such an SLA may encode characteristics for a single or multiple services from a provider. It is also necessary to identify the cost $Cost(Cl)$ that a client must pay to the provider Pr in order to acquire a needed service s_i. The cost of the client is calculated as:

$$Cost(Cl) = Price(SLA_{Cl}^{Pr}(s_i)) \tag{1}$$

that is the price paid by a client to the provider for enacting services according to a particular SLA.

3.0.2. Providers

For providers, deploying a single instance of s_i on the server farm incurs a cost c_i. As more instances are deployed, the cost can change based on

the following function: $f(c) : I_k \rightarrow C_k$, where I_k refers to instances i_i which correspond to various replicas of service s_i and C_k=$\{c_1, c_2, c_3, ..., c_k\}$, where c_i is the cost of deploying the i^{th} replica. The cost function $f(c)$ depends on the particular type of infrastructure that is being used in the server farm(i.e. deploying a replica varies from a server farm to another based on certain hardware configuration or software specifications).

A server farm SF_k has a failure rate fr_k calculated as the number of requests successfully processed within a time interval. In particular, we use an SLA (see figure 1) – SLA_{Cl}^{Pr} as an agreement between the client and the provider where the provider Pr seeks to minimise a cost function $f(c)$, to ensure that agreement SLA_{Cl}^{Pr} is complied with, while the client Cl seeks to receive the service capability described in $SLA_{Cl}^{Pr}(s_i)$. In case of non-compliance with SLA_{Cl}^{Pr}, the penalty mechanism is applied.

Cost for providers are represented by: (i) The penalties PEN – based on non-compliance with $SLA_{Cl}^{Pr}(s_i)$; (ii) the price $P(SLA_{Pr}^{Fo})$ paid to the server farm owner (Fo), for deploying replicas. Hence:

$$Cost(Pr) = PEN(SLA_{Cl}^{Pr}(s_i)) + Price(SLA_{Pr}^{Fo}(s_i)) \tag{2}$$

Profit for providers is represented by the difference between the price of carrying out the execution in accordance with the SLA and the cost associated with outsourcing and deploying the service on the server farm. Therefore:

$$Profit(Pr) = Price(SLA_{Cl}^{Pr}(s_i)) - Price(SLA_{Pr}^{Fo}(s_i)) - PEN(SLA_{Cl}^{Pr}(s_i)) \tag{3}$$

3.0.3. Server Farm Owner

Each server farm SF_i has a failure rate such as $fr_i = r_{k'}/r_k$ where $r_{k'}$ defines the rate of unsuccessful requests of SF_i over the interval Δt and r_k represents the number of total requests. We consider that a provider Pr acts as a client for the server farm owner. Between the providers and farm owners we use $SLA^{Fo}_{Pr}(s_i)$ as an agreement which specifies terms and conditions for deploying service replicas on the server farm – hence the cost for server farm owners is based on: (i) the loss with penalties PEN imposed by $SLA^{Fo}_{Pr}(s_i)$; (ii) the cost incurred for deploying n instances based on $SLA^{Fo}_{Pr}(s_i)$. This leads to the relation:

$$Cost(Fo) = PEN(SLA^{Pr}_{Cl}(s_i)) + n * \sum_{i=1}^{n}(c_i(i_i)) \tag{4}$$

Profit for farm owners is represented by the difference between the price paid for the SLA and the cost based on the number of instances needed for replication. Hence,

$$Profit(Fo) = Price(SLA^{Fo}_{Pr}(s_i)) - n * \sum_{i=1}^{n}(c_i(i_i)) - PEN(SLA^{Pr}_{Cl}(s_i)) \tag{5}$$

Latency represents an important factor in the context of service outsourcing. When a failure occurs, either the instance fails completely (fail stop failure), or the server farm operator undertakes some fault tolerance measures to restart the service – we use latency also as a measure the time it takes to have the service available again. According to the value of the latency, a mechanism of penalties is applied leading to additional costs not only for the server farms but also for the providers due to their pre-established SLAs with clients.

Algorithm 1: Latency Calculation

```
1: for i = 0; i < instancesNr; i++ do
2:     SELECT node n_i
3:     for j = 0; j < totalRequests; j++ do
4:         requestIssuingTime = getClientIssuingTime();
5:         requestExecutionTime = getExecutionTime();
6:         requestExecutionLatency = requestExecutionTime -
           requestIssuingTime;
7:         if requestExecutionTime > 0 then
8:             averageRequestExecutionLatency +=requestExecutionLatency;
9:             numberOfExecutedRequests++;
10:        else
11:            numberOfPendingRequests++;
12:        end if
13:    end for
14:    averageRequestExecutionLatency =
       averageRequestExecutionLatency/numberOfExecutedRequests;
15: end for
```

This is therefore significantly important not only because of the additional costs that can occur but especially because of the faults that lead to SLA violations. In our approach we consider that the cost for an individual instance increases with latency. The cost of an instance clr_i is calculated as a product of the price of instance plr_i and the latency of response l_i: $clr_i = plr_i * l_i$. This helps server farm owners to deal with an increasing queue size of service requests. We consider this cost as an incentive for the

server farm to deploy additional replicas, which might be slower due to existing high load, instead of simply rejecting the requests due to inadequate performance or insufficient resources. The methodology for calculating the latency of an instance is presented in algorithm 1.

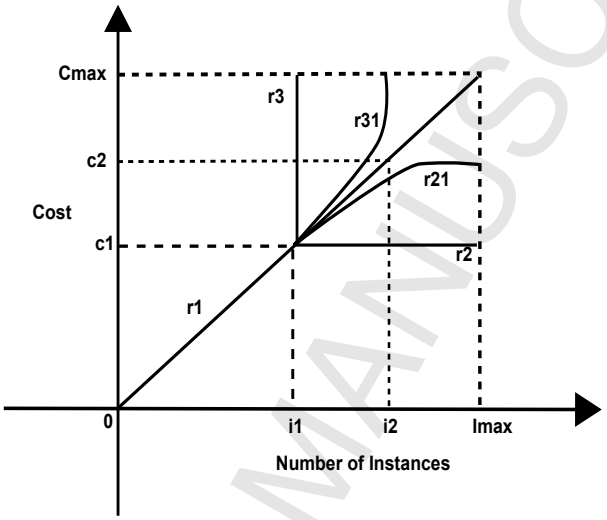


Figure 2: Cost models

3.1. Cost models

The relation between the cost per replica, depicted on the Y-axis of Figure 2, and the amount of instances deployed, depicted on the X-axis, can change depending on the cost model used. The cost is related to the amount of instances being created in the server farm, thus the cost can variate over the capacity interval of the server farm. Hence, the cost per replica can change in accordance with a:

(a) Linear Model – defines a proportional increase in cost for each new instance deployed. In figure 2 this is identified by curve r_1. In the linear

model an increase in the number of instances over intervals $[0, i_1]$ and $[i_1, i_2]$ produces an increase of costs identified as $[0, c_1]$ and $[c_1, c_2]$;

(b) Decay Model – in this instance the cost of deploying new instances decreases (per instance) and eventually becomes stable after a threshold number of instances have been deployed. This implies that as more replicated instances are added for each service (up to the capacity available in the server farm), the system management and deployment costs do not increase. In figure 2 this is identified by curve r_{21}. In the decay model increasing instances over interval $[0, i_1]$ leads to an increase in costs over $[0, c_1]$, with a subsequent reduction of costs $[c_1, c_2]$ for instances $[i_1, i_2]$;

(c) Mixed Model – identifies a mixture between the linear model and the decay model. In figure 2 this is identified by curves r_1 and r_2. The mixed model can identify either an increase of costs $c_i \in [c_1, c_2]$ when a number $i_i \in [i_1, i_2]$ of instances are requested – the case of r_1 or a reduction of costs $c_d \in [c_1, c_2]$ when instances $i_d \in [i_1, i_2]$ are requested– the case of r_2. We consider that r_3 and r_{31} are part of an exponential cost model that we do not consider in the paper. We only consider the decay model, linear model and mixed model. The particular cost model that is applicable depends on the infrastructure being used within a server farm. For instance, in a virtualized environment, adding more virtual machines (VMs) – up to a threshold limit – to each physical machine may not incur any additional cost (especially where replicas are being considered). There is an initial cost of transferring and instantiating a machine image, initiating and deploying the VM, etc. Once this has been done, additional VMs may incur less cost.

3.2. Configuration

In our approach we assume that each $SLA_{Cl}^{Pr}(s_i)$ relates to a *single* service s_i, however a server farm may be hosting multiple types of services (each of which may have multiple replicas). When a service request is submitted at time t_i a number of replicas within the system may fail. We investigate how the system reacts when: (i) a number of services are executed on the server farms generating a load on the system. Service execution starts at a particular time interval defined by a frequency ν; (ii) over the simulation interval, k replicas of a service may fail with probability p; (iii) each provider replicates service s_i based on the cost function $f(c)$.

We use latency as a metric to measure the reaction of the system in the context of a certain load determined by the number of SLAs currently being provisioned. Latency is calculated based on the number of replicated instances and the number of requests submitted to the server farm (as shown in algorithm 1). The number of requests are accommodated on the available instances of the server farm. For each request we calculate a corresponding latency. The average latency is then included in the calculation of the overall cost associated with an instance. The average latency is calculated for each instance deployed on the server farm. The algorithm uses: (i)function results: $requestIssuingTime$, $requestExecutionTime$, where $requestIssuingTime$ identifies the time when an SLA is issued whereas the $requestExecutionTime$ is associated with the actual execution of the service identified in the SLA. The $resultExecutionTime$ value is 0 for all those requests which are pending; and (ii)output parameters: $averageRequestExecutionLatency$, where $averageRequestExecutionLatency$ is then used for obtaining the overall

server farm latency.

4. Simulation framework

Validation of our approach has been carried out through simulation, using a P2P based resource sharing model. P2P systems present two important features: (i) scalability and (ii) dynamism. We make use of PeerSim [4] – a scalable simulation environment that enables the definition of a number of different scenarios. In PeerSim, interaction protocols between peers may either be implemented using a predefined PeerSim API or they can be embedded into a real implementation [3]. PeerSim provides a number of predeveloped modules that can be combined in different ways and provides the flexibility to support a variety of different system configurations. The P2P network is modelled as a collection of nodes, where each node has a list of associated protocols. The overall simulation is regulated through *initializers* and *controllers* – that allow either events to be introduced into the simulation or to enable a particular capability to be added at pre-defined simulation time points.

The issuing of a service request is a process controlled by different configuration parameters within PeerSim. The transition between the issuing of requests and the outsourcing to server farms process is performed based on an *execution* probability.

The *executeprob* is a parameter included in the configuration file. Thus, the number of requests that the system uses during the experiments are controlled with the *executeprob* parameter. At the same time, one SLA is assigned with a *ttl* (time-to-live) parameter defining the time interval over

which an SLA is established between nodes. The operation of the algorithm in terms of issuing and outsourcing is identified in figure 1. These operations are carried out over a duration specified in the time to live (*ttl*) parameter. We consider that penalties are set based on a random distribution with an interval of [1%-10%]. Failure rates are generated as double variables taking random values from the interval [0,1]. The cost models that a server farm may use are assigned based on a random distribution with a set {0,1,2} where each element of the set identifies a cost model (0-linear, 1-decay and 2-mixed).

4.1. Variation in the system

The level of demand introduced into the system can change over time – a configuration parameter in the experiment. It is assumed that the level of demand is based on the number of peers requesting a specific service type – referred to as `view`. This process is triggered when the number of peers requesting service is modified. Our framework is designed to modify the level of demand when new requesting peers are added to the system in the `view` of each participating node.

The price of one specific service type is mapped in accordance with a demand level: $p.type_{c_{i+1}} = p.type_{c_i} * d$ – where c_i represents the i^{th} cycle of the simulation. The demand is based on the `view` parameter assigned to each peer node offering a service of type t_i as a set P_{ssi} consisting of $\{p_{ss1}, p_{ss2}, p_{ss3}, ..., p_{ssk}\}$ indicating the number of peers requesting a specific service. In particular, the level of demand is varied by changing the view parameter assigned to a specific type of service.

In order to validate the hypotheses of demand implication on the status

of the system, PeerSim was chosen as a framework for simulating different scenarios. Within one configuration file, different simulation events are controlled. The PeerSim simulator uses separate source files for programming different needed controllers of the simulation process. In particular, our framework uses three different controllers. The first controller *controller1* defines the number of requests scheduled to be used during the simulation; the second controller *controller2* defines the network variation for each simulation cycle (e.g. how the size changes when injecting new peers) for each round of the simulation. The last controller (*controller3*) is the observer that collects the results for each experiment. The configuration file also contains a number of simulation parameters:

- `cycles`: defines the maximum number of simulation cycles for each experiment.
- `ttl`: defines the time to live for one SLA – i.e. for how long the SLA is valid with reference to the current time.
- `maxnodes`: defines the maximum number of nodes that have been scheduled to issue requests.
- `maxRequests`: defines the maximum number of service requests scheduled to be issued within the system as a whole.

4.2. Demand configuration

Our framework is designed to handle demand as an economic process that can induce fluctuation for the value of exchanged objects. For simulating the

variation of demand within the market, our framework uses several assumptions. Therefore, one specific level of demand is simulated by using one *view* parameter. The *view* is assigned to each node within the system. The *view* of any one peer is independent of other peers in the system. This parameter is adjusted (increased or decreased) during each cycle of simulation. A default value of 1 identifies the case of a regular (stabilised) demand.

The variation of the demand was ensured by PeerSim controller2 (see subsection 4.1), which can inject different numbers of requesting nodes at each simulation cycle. The following configuration parameters are used by this controller:

- control.c1 peersim.dynamics.DynamicNetwork
- control.c1.maxsize v_{max}
- control.c1.add v_{add}
- control.c1.add v_{remove}
- control.c1.step v_{step}
- control.c1.from v_{from}
- control.c1.until v_{until}

The *DynamicNetwork* is a module provided within PeerSim which helps the simulation process to work with a differing number of peer nodes at each simulation cycles. It includes various Java packages initializing a network or modifying it during simulation. It can also be used to model node *churn*. The *maxsize* parameter represents the maximum number of peer nodes that

one simulation process can use; the *add* parameter defines the number of peer nodes injected at each step and the *remove* parameter defines the number of peer nodes removed at each step. The *step* parameter defines the stage of the process (i.e. creation/issuing, outsourcing) for each injected peer node.

Table 1: Configuring the simulation

Parameter	Value
simulation.cycles	200
network.size	2000
network.maxSize	5000
network.minSize	500
protocol.1	fgcs.Risk
protocol.1.ttl	20
protocol.1.issueprob	0.25
protocol.1.outsourceprob	0.25
protocol.1.failureprob	0 (0 if dynamic, 1 if static)
protocol.1.penalty	0 (0 if dynamic, 1 if static)
protocol.1.costmodel	0 (0 if dynamic, 1 if static)
init.3	fgcs.WireKOut
init.3.protocol	1
init.3.maxrequests	10
init.3.reqfrequency	5
init.3.requestnodes	100
init.3.k	25
control.c1	fgcs.DynamicNetwork
control.c1.maxsize	5000
control.c1.add	50
control.c1.remove	50
control.c1.step	1
control.c1.from	2000
control.c1.until	3000
control.ob1	fgcs.Observer
control.ob1.protocol	1
control.ob1.verbosity	1

The parameter *from* specifies the starting number of peer nodes to simulate while the *until* parameter defines the maximum limit on the number of peer nodes that the simulation can use. An example of the configuration file is provided in Table 1.

5. Evaluation & Results

We consider a community where clients, providers and server farm owners can establish SLAs to support service provisioning. We evaluate this community based on the cost incurred by each participant for: (i) acquiring the service – the primary action performed by clients, (ii) outsourcing the service – the primary focus of providers and (iii) deploying virtual instances – the key activity carried out by server farm owners. Each client can request several services and each service has a number of replicated instances. Services are delivered on the basis of pre-established SLAs among peers. In our experiment we consider two metrics:

1. Cost: $C = C(Cl) + C(Pr) + C(Fo)$ where $C(Cl)$ represents the cost incurred by the client, $C(Pr)$ is the cost to the provider and $C(Fo)$ identifies the cost incurred by the server farm owner.
2. Profit: $P = P(Pr) + P(Fo)$ where $P(Pr)$ represents the profit of the provider and $P(Fo)$ represents the profit of the server farm owner.

We carry out a series of experiments to validate how the costs identified above are impacted by different system configurations. Each experiment attempts to evaluate a particular objective. We use a simulation environment with 2000 peer-nodes which are configured as clients, providers or server farm peer-instances.

Experiment 1: In this experiment we investigate how the overall cost and profit within the community is affected when a number of replicas fail – based on a failure rate parameter. In all the experiments the cost and profit are illustrated on the X-axis in a predefined order: first bar is cost, second bar

is profit. Figure 3 illustrates how the cost and profit evolve with different failure rates. We can observe that the cost and the profit in the system are significantly affected when a high number of replicas fail. This experiment considers the failure rate within the following set: {0.01, 0.05, 0.09, 0.5}.

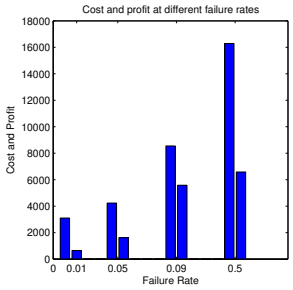


Figure 3: Cost and Profit at different failure rates

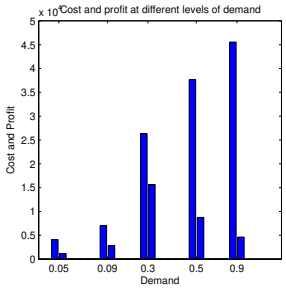


Figure 4: Cost and Profit at different levels of demand

It can be observed that the overall cost increases with fault probability. We observe the difference of cost when considering failure rates over the range [0.09-0.5]. The cost is considerably increased for 0.5 while the profit remains stable. This arises because when multiple service instances fail, the penalties specified in the SLA are applied, leading to an increase in the total cost. In conclusion the impact on cost and profit is determined by the penalty mechanism associated with multiple failures.

Experiment 2 In this experiment we identify how the profit/loss is affected when the demand for services increases. Demand for services in this case is identified based on the number of services executed based on pre-defined SLAs. This experiment demonstrates how demand affects the overall community (service client, owner and server farm owner) in terms of cost and profit.

From figure 4 we observe how the distribution of cost and profit evolves in relation to different levels of demand. When using an initial demand in the system (demand=0.05), the cost and the profit are low. When increasing the demand to approximately 30 service requests per time unit (demand=0.3), there is a significant impact on the cost and profit. This impact is influenced by the penalties imposed when the number of requests increase. From 0.09 to 0.3 an increase in load leads to increased SLA violations and penalties. The greatest difference of costs and profits occurs when using a load of 90 service requests per time unit (demand=0.9). From this experiment we can deduce that there is an increase in the risk associated with service delivery when demand for services increases.

Experiment 3: Previous experiments demonstrate how cost and profit within the system fluctuate when dealing with an increased demand. In this experiment we consider the rate at which new demand can be introduced into the system. This experiment investigates how a variability in the *demand frequency* can affect the cost and the profit.

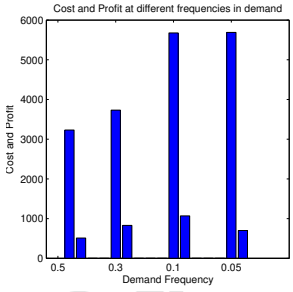


Figure 5: Cost and Profit when varying demand frequency

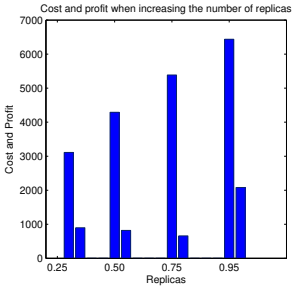


Figure 6: Cost and Profit when varying the number of replicas

Increasing the frequency of service requests over a specific time interval represents another factor which can affect the cost and the profit in the system – illustrated in Figure 5. The difference between demand and demand frequency is determined by the interval when requests arrive. Whereas the demand refers to the overall interest for a service over a interval of time, demand frequency refers to the specific time interval when a request appears. When demand frequency increases, the average latency increases, whereas when service executions are separated by longer intervals, this leads to lower costs. We observe that the impact of demand frequency on profit is lower than on cost. Whereas profit is not significantly affected by frequency, the cost increases in direct proportion to the demand frequency – as the latter implies a higher load on the server farm and depends on the amount of replicas associated with the service request.

Experiment 4: Previous experiments demonstrate how the system changes when providers are dealing with a certain number of replicas (see section 3). In this experiment we extend the number of replicas and observe how the cost and profit are distributed. Figure 6 illustrates the distribution of cost and profit when increasing the number of replicas by 25%. A corresponding impact in the profit value is identified each time we increase the number of replicas. At 75% we observe that the system has an increase in cost while the profit decreases. An increase in profit can be identified when increasing the number of replicas by 95%, indicating that the server farm has enough capacity to deal with these requests. After deploying a number of replicas the increase of costs is reduced leading to an increase of profits.

6. Multiple service outsourcing

In this section we analyse the case where service providers can deploy various combination of services on the server farm in order to reduce cost and increase profitability. While in section 3 we tackle the scenario where isolated services are deployed on a number of server instances, in this particular approach we consider that a service provider can organise capabilities by deploying various combinations of services – with each combination being referred to as a "virtual appliance" (as discussed in section 1). We consider the service delivery scenario of figure 1 where a set of services can be replicated on a server farm. Each service is represented as a service *type* and contains a combination of one or more services. Giving a set of types T=$\{t_1, t_2, t_3, ..., t_k\}$ which can be deployed on a server farm, one type t_i can represent a combination of services such as t_i=$\{s_1, s_2, ..., s_n\}$. Here, each type corresponds to a virtual appliance.

We consider the service owner acting as a SaaS provider, delivering on-demand services to the client. Service delivery between the client and the service owner is regulated by an SLA (identifying the requested QoS properties, the price paid and penalty). In general, the service owner uses a price scheme based on time units: e.g. 10$ per hour. In this model, the service owner uses a server farm to host its services. The server farm might be locally hosted or deployed over a Cloud infrastructure. On the server farm, the service owner can deploy a maximum of n instances(virtual instances) $I = \{i_1, i_2, ..., i_n\}$, with each instance i_i identifying a peer-node p_i on which it is hosted. The infrastructure is unreliable, and each peer node might fail with probability f - named failure rate. f represents the percentage of failed

transactions compared to the total transactions deployed at a given peer.

The server owner is able to deliver k isolated services $(s_1, s_2, ..., s_k)$. The cost of deploying one individual service s_i on one of the n peers is c_i. Instead of deploying a single service s_i on some of the peers, the server farm owner can choose to deploy a combination of services t_i. In this case (multi-tenancy), the equivalent set of costs with such deployment is denoted by $(c_{i_1}, c_{i_2}, ..., c_{i_t})$. For financial reasons, the following relation holds, for any two services s_{i_1} and s_{i_2}:

$$c_{i_1} + c_{i_2} \geq c_{i_1,i_2} \geq c_{i_1} \tag{6}$$

Running services on each peer, the service owner incurs an operating cost c, for each time unit of operation. Based on the incoming request, the server farm owner chooses to deploy only individual service s_i on the n available peers, or whether to deploy some combination of services $(s_{i_1}, s_{i_2}, ..., s_{i_t})$. To deal with the peer failure rate f, a server farm owner can decide to replicate each delivered service m times, but with the constraint that the number of peers n remains fixed. One or several clients create a demand by placing a request R_i from the set $R_T = (R_1, R_2, R_3, ..., R_t)$, where $R_i \in R_T$. A request R_i refers to a combination of services t_i and has a corresponding SLA_{Cl}^{Pr}. By a service type we refer to a combination of individual services. For instance one client may ask for a generic service type which needs to have Microsoft Office, Matlab, a Tomcat Server under a Windows Server operating system. Rather than deploying individual services the provider will choose to customise a virtual instance which can provide all these individual services as a generic service type.

6.1. Protocol

In our framework we consider profit and cost parameters for each individual request $R_i, i = \overline{1,n}$. Therefore, we consider the profit of the service owner in relation to their respective SLA:

$$Profit(R_i) = Price(R_i) - Cost(R_i) - Penalty(R_i) \tag{7}$$

$Price(R_i)$ is calculated as the unitary price p asked by the service owner, times the t duration of the service contract: $Price(R_i) = p * t$. $Cost(R_i)$ is calculated as the cost $c_{i_1,i_2,...,i_t}$ incurred by the service owner for deployment of the requested services on the peers(cost of deployment), plus the unitary costs of operating the peer(cost of execution) c_{p_i} times t, the duration of the contract: $Cost(R_i) = c_{i_j} + c_p * t$. $Penalty(R_i)$ represents the penalty the service owner has to pay if the SLA delivery fails: pen_{SLA}.

Given that the service provider has to deploy services $(s_{i_1}, s_{i_2}, ..., s_{i_t})$ in the server farm based on a request received from a client, an instance is created only if the combination $(s_{i_1}, s_{i_2}, ..., s_{i_t})$ is already deployed somewhere, or several instances which individually compose the requested services (s_{i_1}, s_{i_2},...,s_{i_t}). If one of the requested services is not deployed anywhere, then the server farm can deploy additional instances for accommodating the service request. We assume that the costs of deploying services differ, as suggested by eq. 6. Further, if the service owner requests r_m replicas, then the server farm should identify those peers capable of delivering the requested services, each service being delivered (in isolation or in combination) by p_m peers. In our protocol the SF is created with some instantiation of the services on the existing n peers. We use the replication rate r_m set by the service

owner. The client injects requests for services, extracted out of the $2^p - 1$ possible combinations of services. Clients can generate requests uniformly distributed over the possible combinations, or some combinations might be identified with higher demand and preferred by the client.

6.1.1. Experiments

To validate the multiple service outsourcing approach, we use a PeerSim [4] based simulation. The simulation is based on the architecture in figure 1, where peers can play different roles: clients, service providers or deploying peers. We use the experimental configurations presented in Section 4 with specific adaptations for handling multiple types of services. We carry out a series of experiments to identify how the total profit and average cost evolve in different simulated scenarios. In our experiments we consider two metrics:

1. Average cost per request: $Cost(R_{req}) = \sum_{i=1}^{n}[(c_{i_k} + c_p) * t]/n$
2. Total Profit: $Profit(R_{req}) = \sum_{i=1}^{n}(Price(R_i) - Cost(R_i) - Penalty(R_i))$

where n is the number of requests. We use a simulation environment with 2000 peer-nodes which are configured as clients, providers or server farm peer-instances.

Experiment 1: This experiment explores how the different combination of services that are deployed in the server farm can impact the status of the system. Considering that each type is a combination of individual types of services we look at the relation between the number of services per type (embedded in a service type) and profit.

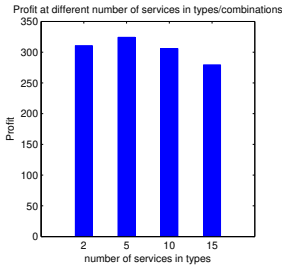


Figure 7: Profit when varying the number of services per type

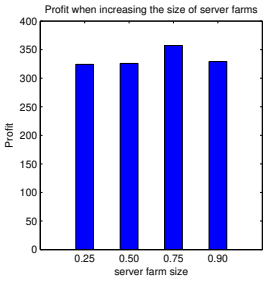


Figure 8: Profit when varying the size of server farms

From figure 7 we observe that from a set [2,5,10,15] of possible service combinations, the highest level of profit occurs when using service types with 2 and 5 services. When using service types with 10 and 15 services in combination, the level of profit starts to decrease. From this experiment we can conclude that there is an optimal number of services to include within each type. The increase of profit for service types 2 and 5 is a result of choosing the right amount of services per type.

Experiment 2: In this experiment we investigate how the profit is affected by expanding the size of the server farm over the range [0.25, 0.50,0.75,0.90] (i.e. 25% bigger, 50% bigger etc). Figure 8 illustrates profits when expanding the number of peers in the server farm. From this experiment we can conclude that additional peers for deploying service types can represent an immediate profit when there is continuous demand (a number of requests uniformly distributed over the simulated execution) within the system. It is interesting to note that when increasing the server farm capacity with 0.25 and 0.50 the profit remains stable. In this case, the profit accumulated from new requests balances the costs associated with deploying new instances. When

increasing the server farm capacity by 0.75 (i.e. 75%), the revenue increases greater than costs determining an increase of profit, unlike the case when the server farm is expanded by 0.90.

Experiment 3a: In this experiment we identify the distribution of profit when varying the demand for all service types. Considering a fixed number of services that can be combined in service types, we analyse how the demand for particular types can influence the distribution of profit. Figure 9 illustrates various stages of profit in relation to the associated demand levels. Investigating various levels of demand within the set [0.25,0.50,0.75,0.90], we observe that the profit increases up to the level of 0.50 (i.e. a 50% increase) demand. When increasing the demand of service types to 0.75 we observe that the profit starts to decrease continuously. This decay in profit is induced by the limited number of peers to deploy services. Even if the number of requests for all service types increases, the capability of the server farm is limited and therefore the process becomes less profitable.

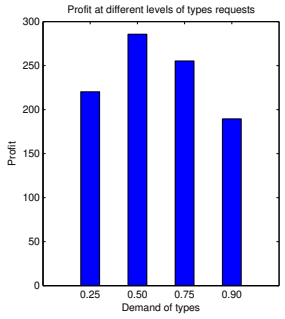


Figure 9: Profit at different levels of demand for types of services

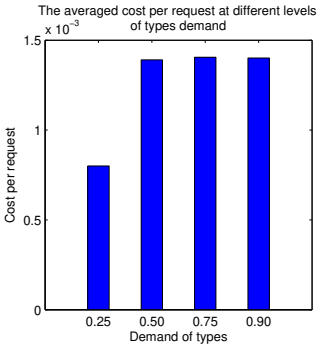


Figure 10: Averaged cost per request at different levels of demand for types of services

Experiment 3b: In this experiment we identify the distribution of the average cost per request when varying the demand for service types. Keeping the same configuration as in experiment 9, we analyse the status of the system from an average cost perspective. The demand for types is varied over the set [0.25,0.50,0.75,0.90]. From figure 10 it can be observed that an increase of demand to 0.50 respectively to 0.75, leads to an increase in the averaged cost per request. This increase of cost is determined by additional penalties that are incurred when increasing the demand. As identified in the previous experiment, the effect of demand for particular service types on cost is related to the limited number of peers to deploy services. In the absence of additional peers to deploy services, there are no other deployment costs incurred, and consequently the averaged cost stagnates.

Experiment 4a: This experiment presents the distribution of profit when varying the failure rate at a fixed demand for service types. Failure is a process that is controlled by a probability distribution, we vary the rate of failure over the set [0.25,0.50,0.75,0.90] and observe how profit is distributed. From figure 11 we observe that profit evolves inversely to the rate of failure. This decrease in profit is due to the increase in penalties associated with not meeting SLA targets.

Experiment 4b: In this experiment we identify the distribution of the average cost per request when varying the rate of failures. Varying the rate of failure for each replica has a direct impact on the average cost of service type requests. Figure 12 presents the distribution of the average cost at various level of failures. As mentioned in the previous experiment, replica failure translates into an immediate cost for providers in being able to handle the

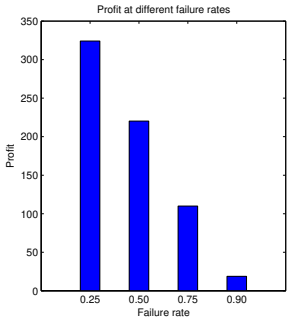


Figure 11: Profit at different failure rate at a constant demand of types

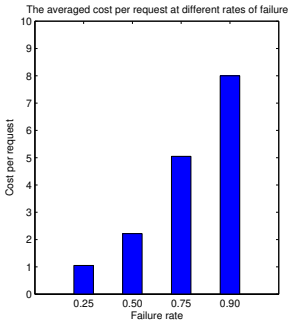


Figure 12: Averaged cost with requests at different failure rates

current demand. When providers are unable to respond to the amount of requests, thereby non-complying with pre-established SLAs, a penalty is applied. With a failure rate varied over the set [0.25, 0.50, 0.75, 0.90], the highest impact on cost is identified when 0.90 of replicas fail.

7. Related work

Risk assessment mechanisms are critical to increase the trust between clients and providers especially in distributed environments. The problem of risk management and associated cost mechanisms within a market of computational resources has been discussed by projects such as AssessGrid [5] and GridEcon [12]. The AssessGrid project proposed the development of a brokering mechanism that enabled *risk-aware* creation of SLAs between Grid service consumers and providers. The focus of the project was to offer a risk-aware decision support system allowing individuals to negotiate and consume Grid resources using SLAs. A utility computing business model was employed for evaluating the emergent *open marketplace*. The architecture of

AssessGrid is divided into three layers, one for each of the actors: end-user, broker and provider. The end-user layer includes a portal which provides a number of abstract Grid applications which can interact with each other through an SLA Broker component (implemented using the WS-Agreement and WSRF (Web Services Resource Framework) specifications) [6]. The broker serves as the central actor of the system and can play the role of a mediator or a contractor on behalf of different participants. By investigating various scenarios and testing different roles that a provider can adopt, AssessGrid provides a risk management framework for supporting reliable service operations, particularly focusing on the concept of probability of failure of a provider.

Multiple computing vendors such as HP, Amazon, Sun and IBM outsource the execution of services on commoditized servers with associated pricing models. These remotely located resources do not necessarily enable the end user to undertake specific analysis or to manage the risk of the system. Li et al. [7] try to predict availability by introducing risk analysis for Grids and propose new means to construct Service Level Agreements (SLAs) by reference to techniques of financial risk analysis. With this theoretical solution, prediction, quantification of risk, and consideration of liability in case of failure can be applied for different provision models specifically, relating to the provision of SLAs through resource brokers, and comparable to markets in other commodities. In addition the model can be applicable to the configuration and management of related architectures such as those of P2P systems and Clouds. For enriching the investigation, an analysis is performed on the potential formulation of a Grid Economy as a commodity

market, and extended towards trading and hedging of risk, options, futures and structured products. This approach involved collecting data about computational resource use within the UK National Grid Service (NGS) and subsequently using this data in combination with approaches from computational finance (in particular the idea of Value at Risk (VaR)) to predict availability of resources and associated insurance against losses (and failures).

Current Cloud provision models typically rely on the use of resource virtualisation in order to enable users to customise their hosting environment and enable multi-tenancy on resources. This is also often undertaken to improve resource utilization by the provider. Salesforce.com is an example of a provider offering multi-tenancy support to customers, offering a variety of on-demand software capability. On the other hand, Amazon.com provides Web-based APIs for controlling virtualized resources, with Amazon EC2 reducing the cost-barrier to the point where it becomes feasible to have each customer of a hosted SaaS solution have their own "virtual appliance" instance(s) rather than forcing them to share common instances. A virtual appliance (popularised by VMWare) is a virtual machine image file with a pre-configured operating system and a single application. The objective is to minimise the operating system capability needed to launch and execute the application, and thereby reduce installation and configuration problems (associated with driver and software library compatibility). A virtual appliance may also involve aggregation of several services to make these available as a single offering (at a single price) to a customer. Where licencing is involved for each service, identifying which services to include in the aggregate bundle is often based on customer demand and can change over time.

Protector [2] is a probabilistic failure detector for cost-effective Peer-to-Peer storage. Protector, based on a SuperPeer overlay creation algorithm provides risk mitigation against transient failures. Protector presents applicability for group replication where all peers host replicas of the same object by detecting the number of remaining replicas in a group, i.e., the number of replicas residing on online peers or peers experiencing transient failures. By using a failure prediction function calculated as the probability that a peer has permanently failed given an observed failure of d time units, Protector performs an aggregation of failure probabilities across all peers in the replica group in order to estimate the number of remaining replicas. Simulation is used to demonstrate that Protector enables the system to maintain objects in the most cost-efficient manner. Implementing a set of methods such as (i) leveraging prior failure statistics, and (ii) making estimates across a group of replicas which balance false positives for some peers against false negatives for others, Protector is validated by deploying a P2P storage system called *AmazingStore* – a storage sharing system that enables trusted users to exchange spare capacity with each other.

Multi-tenancy based virtual appliances are also increasingly used in data centers and offer the benefit of resource consolidation, performance and fault isolation, flexible migration across data center hardware and easy creation of specialised environments. By deploying a secure multi-tenant virtual service provision mechanism, each business unit benefits from the transparency of the virtual environment [15]. Hence, a multi-tenancy architecture enables the provision of new services quickly and cost effectively by using Service Level Agreements (SLAs) [17], [14]. The perspective of multi-tenancy en-

vironments identifies a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. In a multi-tenancy Cloud architecture a software-as-a-service (SaaS) provider, for example, can run one instance of its application on one instance of a database and provide web access to multiple customers. In such a scenario, each tenant's data is (expected to be) isolated and remain invisible to other tenants.

In order to deliver hosted services to customers, SaaS companies have to either maintain their own hardware or rent it from infrastructure providers. Based on this, scalable management of virtual machines residing on distributed hosts has been developed for allowing maximal utilisation of the underlying resources and replacing the traditional "one server, one application" model with a multi-tenant architecture/model of cloud services [16]. This allows SaaS providers to minimize infrastructure cost and SLA violations by mapping customer requests to infrastructure level parameters and handling heterogeneity of virtual machines [17]. Alternatively providers can overcome the situations when they need to serve only a certain amount of requests due to restricted amounts of resources by using a cloud federation technique [18].

Goiri et al. [18] propose a solution to perform a characterisation of providers operating in a federated Cloud environment, by identifying when outsourcing is profitable depending on the operating environment. These include when to outsource to other providers, rent free resources to other providers (i.e.,

in sourcing), or turn off unused nodes to save power. The experimentation evaluated parameters such as the providers incoming workload, the cost of outsourcing additional resources, the ratio of outsourced resources, the ratio of unused resources to be sold, and the cost of keeping the providers resources operative. Their results show that local resources are preferred over outsourced resources, though the latter can enhance the providers profit when the workload cannot be supported locally.

Fito et al. [19] propose a semi-quantitative risk assessment method for Cloud computing regarding the Business-Level Objectives (BLOs) of a given organization. Their approach enables an organization to be aware of risks when using a Cloud system and thereby enable the alignment of the low-level management decisions with high-level (business) objectives of an organisation. The approach classifies risks according to their business level objectives and identifies their potential impact on the overall business context. Through simulation it was demonstrated in their work that a Cloud Service Provider is able to maximise profit by transferring a private Clouds provisioning risks to third-party providers.

The concept of risk has been also specifically applied into a cloud context. The OPTIMIS project [24] considers hybrid clouds as a future commonplace where private clouds can interact with a rich ecosystem of public and other cloud providers. The idea behind OPTIMIS is to allow organisations to automatically externalize services and applications to trustworthy and auditable cloud providers which can greatly optimise operation of services and infrastructures. Our approach instead looks at service deliveries in provider communities from a wider perspective. We consider that risk can be also

seen as an expression of costs and profits within a community for reflecting a status among the interacting parties (organisations). In our paper, services that are outsourced to expert infrastructures (not necessarily clouds) can identify various bulks of resources that a client may request.

8. Conclusion

The emergence of Cloud computing deployment strategies enables us to differentiate between a service owner and an infrastructure provider, where a service owner may utilize the resources of an infrastructure provider to deploy a service. Where the relationship between these two actors (service owner and infrastructure provider) is not based on trust (i.e. based on experience gained in previous interactions), it is often necessary to establish an SLA. Such an agreement protects the service owner if the infrastructure provider is unable to deliver their advertised capability. We consider the financial risk that would be incurred by both the service owner and the infrastructure provider, based on the price paid for the service by a client, the penalty incurred due to non-compliance with the SLA and the deployment cost for running multiple replicas on the infrastructure.

Through simulation we demonstrate that in service provider communities the number of failures and the level of demand can have significant impact on the distribution of cost and profit between the actors. We show that the demand for services represents another factor which can determine the level of profit/loss within the community. When an SLA is associated with each service execution, we demonstrate that the higher the frequency of SLA violations the higher the variation in costs – based on the deployment, penalties

and replication costs present within the system. We increase the number of replicas in the system to increase service availability – focusing on a single server farm. The approach we present can be extended to multiple server farms – operating in different geographical data centres (an approach adopted by Amazon.com as part of their *Availability Zones*).

In the second part of the paper we tackle the problem of multi-tenancy proposing several methods to reduce costs. By simulation we demonstrate that service deployment is influenced by a number of different parameters such as: service combinations (captured through the notion of "service types", demand for types, rate of failure and number of replicas. We show that the choosing the number of services to include in each type represents an important factor which can influence the level of profit/loss. We vary the number of available peers for replication to increase service availability and observe how the system reacts when capabilities for replicas increase. Thus, we validate that the combining services can be an alternative solution for reducing costs with an infrastructure provider (referred to as a server farm owner/operator).

References

[1] I. Petri, O. Rana and G. Cosmin Silaghi, "SLA as a Complementary Currency in Peer-2-Peer Markets", Economics of Grids, Clouds, Systems, and Services, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, pp. 141-152, (2010).

[2] Yang, Zhi and Tian, Jing and Zhao, Ben Y. and Chen, Wei and Dai, Yafei, Protector: A Probabilistic Failure Detector for Cost-Effective

Peer-to-Peer Storage, IEEE Trans. Parallel Distrib. Syst., IEEE Press, 22(9), pp.1514–1527, 2011.

[3] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks". ACM Transactions on Computer Systems, 23(3):219–252, August 2005.

[4] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "Gossip-based peer sampling". ACM Transactions on Computer Systems, 25(3):8, August 2007.

[5] K. Djemame, J. Padgett, I. Gourlay and D. Armstrong, "Brokering of risk-aware service level agreements in Grids", Concurrency and Computation: Practice and Experience, 23(13), John Wiley and Sons Ltd., pp. 1558–1582, 2011.

[6] K. Djemame, I. Gourlay, J. Padgett, G. Birkenheuer, M. Hovestadt, O. Kao and K. Vo, "Introducing risk management into the grid". Proceedings of the Second IEEE International Conference on e-Science and Grid Computing (eScience2006). IEEE Computer Society: Amsterdam, Netherlands, 2006.

[7] B. Li and L. Gillam, "Risk Informed Computer Economics", Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID '09, IEEE Computer Society, pp. 526–531, Washington, DC, USA, 2009.

[8] T. Eymann, S. Konig and R. Matros, "A Framework for Trust and

Reputation in Grid Environments", Journal of Grid Computing, Vol 6, No. 3, pp 225–237, 2008.

[9] Li Xiong and Ling Liu, "PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities", IEEE Transactions On Knowledge and Data Engineering, pp. 843–857, (2004).

[10] A. Elwaer, I. Taylor and O. Rana, "Preference driven Server Selection in Peer-2-Peer Data Sharing Systems", 4^{th} int. workshop on Data-intensive Distributed Computing (DIDC) – alongside the HPDC conference, San Jose, California, USA, 2011.

[11] I. Petri, O. Rana, Y. Rezgui and G. Cosmin Silaghi, "Evaluating Trust in Peer-to-Peer Service Provider Communities", Proceedings of TrustCol workshop, Orlando, Florida, USA, October 2011.

[12] Risch, Marcel and Altmann, Jörn,"Cost Analysis of Current Grids and Its Implications for Future Grid Markets", Proceedings of the 5th international workshop on Grid Economics and Business Models, GECON '08, pp. 13–27, Springer-Verlag, 2008.

[13] US Department of Energy (DOE). The Magellan Report for Cloud Computing in Science, Office of Advanced Scientific Computing Research (ASCR). 2011.

[14] Cisco. Deploying Secure Multi-Tenancy into Virtualized Data Centers. 2011. `http://www.cisco.com/en/US/docs/solutions/Enterprise/DataCenter/Virtualization/securecldeployg.html`, Last accessed: January 2012.

[15] K.Z. Ibrahim, S. Hofmeyr, and C. Iancu. Characterizing the performance of parallel applications on multi-socket virtual machines. In 11th IEEE/ACM International Symposium on Cluster Computing and the Grid. IEEE Press, 2011.

[16] Jianxin Li, Bo Li, Tianyu Wo, Chunming Hu, Jinpeng Huai, Lu Liu, and K. P. Lam. Cyberguarder: A virtualization security assurance architecture for green cloud computing. Future Gener. Comput. Syst., 28:379–390, February 2012.

[17] Linlin Wu, Saurabh Kumar Garg, and Rajkumar Buyya, SLA-based resource allocation for software as a service provider (saas) in cloud computing environments, In 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pages 195–204, 2011.

[18] Goiri, I, Guitart, J, Torres, J, Economic Model of a Cloud Provider Operating in a Federated Cloud, Information Systems Frontiers,pp.1-17, 2012.

[19] Josep Oriol Fit, Mario Macas, Jordi Guitart, Toward business-driven risk management for Cloud computing, International Conference on Network and Service Management CNSM 2010, IEEE Press, 2010.

[20] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimal virtual machine placement across multiple cloud providers," in Proceedings of IEEE Asia-Pacific Services Computing Conference (APSCC'09). Jeju Island: IEEE, Dec. 2009, pp. 103-110.

[21] Ioan Petri, Omer F. Rana, Yacine Rezgui, Gheorghe Cosmin Silaghi: Trust modelling and analysis in peer-to-peer clouds. IJCC 1(2/3): 221-239 (2012)

[22] Miller, R. (2008). Failure rates in google data centers, 30/05/2008. [Blog] http://bit.ly/SJItI3. Last accessed: January 2013.

[23] Antoine Pichot, Oliver Wldrich, Wolfgang Ziegler, and Philipp Wieder, "Dynamic SLA Negotiation Based on WS-Agreement". Proceedings of WEBIST (1) 2008: pp 38-45.

[24] A Risk Assessment Framework for Cloud Computing.. K. Djemame, D. Armstrong, J. Guitart, and M. Macias. IEEE Transactions on Cloud Computing, 2014, to appear.

Ioan Petri holds a PhD in 'Cybernetics and Statistics' from Babes-Bolyai University of Cluj-Napoca(Romania). He has worked in industry, as a software developer at Cybercom Plenware and then as a research assistant on several projects funded by the Romanian the Authority of Research. Starting with 2009, he collaborated with the School of Computer Science & Informatics(Cardiff University) as an internship researcher in Distributed and Parallel Computing. Currently he is working in School of Engineering as an associate researcher in Computational Engineering.

Omer F. Rana is a Professor of Performance Engineering in School of Computer Science & Informatics at Cardiff University and Deputy Director of the Welsh e-Science Centre. He holds a Ph.D. in "Neural Computing and Parallel Architectures'' from Imperial College (University of London). He has worked in industry, as a software developer at Marshall BioTechnology Limited and then as an advisor to Grid Technology Partners. His research interests extend to three main areas within computer science: problem solving environments, high performance agent systems and novel algorithms for data analysis and management.

Gheorghe Cosmin Silaghi is an Associate Professor at the Babes-Bolyai University of Cluj-Napoca, Romania. He received a Bachelor degree in Business Information Systems in 2000 and his Engineering degree in Computer Science in 2002. In 2002, he received his M.Sc. in Artificial Intelligence from Free University of Amsterdam. G.C. Silaghi completed his Ph.D. in 2005. He joined the Babes- Bolyai University in 2000 and currently he is the Head of the Business Information Systems department. His current research interests are focused on resource management techniques in untrusted distributed environments, like peer-to-peer systems.

Professor Yacine Rezgui is a Professor in School of Engineering at Cardiff University and a BRE (Building Research Establishment) Chair in 'Building Systems and Informatics'. He is a qualified architect with an MSc (Diplôme d'Etudes Approfondies) in "Building Sciences" (obtained from Université Jussieu - Paris 6) and a PhD in Computer Science applied to the construction industry, obtained from ENPC (Ecole Nationale des Ponts et Chaussées).
After 4 years of industrial experience in architectural engineering, he joined Derbi Informatique, the IT subsidiary of OTH (Omnium Technique Holding), a French engineering company specialised in all aspects of design and construction engineering. He took part to several of their developments, including their web-based project management system SGTi (www.sgti.fr). He has then worked as a researcher for CSTB (Centre Scientifique et Technique du Bâtiment) and was involved in a number of national and EU research projects in the field of document engineering (DOCCIME), product modelling and Computer Integrated Construction (ATLAS).

author2

Click here to download high resolution image

author1

Click here to download high resolution image

author4

Click here to download high resolution image

author3

Click here to download high resolution image

- We investigate the notion of risk from the perspective of clients and providers
- We determine how a service owner can balance the loss and the cost of replication
- We discover that combining services in types can impact the level of profit/loss
- We show how a demand for types can impact the overall community