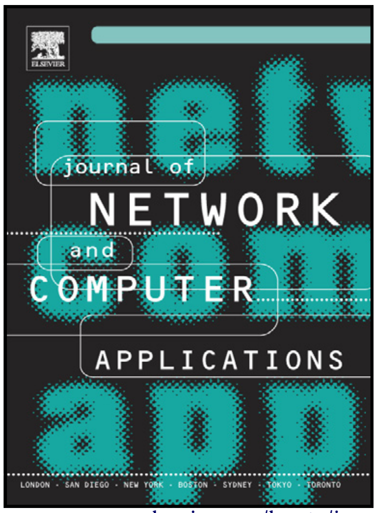


A Petri net-based decision-making framework for assessing cloud services adoption: The use of spot instances for cost reduction

Maristella Ribas, C.G. Furtado, Neuman Souza, Giovanni Barroso, Antão Moura, Alberto S. Lima, Flávio R.C. Sousa

PII: S1084-8045(15)00156-3
DOI: http://dx.doi.org/10.1016/j.jnca.2015.07.002
Reference: YJNCA1429

To appear in: *Journal of Network and Computer Applications*

A Petri net-based decision-making framework for assessing cloud services adoption: The use of spot instances for cost reduction

Author1 (corresponding author):

Name: Maristella Ribas

Address: Rua Francisco Teixeira Alcantara, 500 Fortaleza, Brasil CEP 60182360

Email: mari@techne.com.br

Author 2

Name: Corneli Gomes Furtado Júnior

Address: Rua Juvenal Galeno, 722 - Benfica - Cep 60015-340

Email: cjunior@ifce.edu.br

Author 3

Name: José Neuman de Souza

Address: UFC-DC-Campus do Pici - Bloco 910 - 60440-504 - Fortaleza - CE

Email: neuman@ufc.br

Author 4

Name: Giovani Cordeiro Barroso

Address: Rua Princesa Isabel, 1618/101 CEP: 60.015-061 Fortaleza-Ce, BR

Email: gcb@fisica.ufc.br

Author 5

Name: J. Antao B. Moura

Address: Av. Aprigio Velloso 882 Bloco CN Sala 210 CEP 58429-140 Campina Grande, PB, Brasil

Email: antao@dsc.ufcg.edu.br

Author 6: Alberto Sampaio Lima

Address: Rua Bento Albuquerque, 550 Apto 300 - Coco - Cep 60192-060 Fortaleza - Ceara - Brazil

Email: albertosampaio@ufc.br

Author 7

Name: Flávio R. C. Sousa

Address: Rua Bento Albuquerque, 550 Apto 550 - Coco - Cep 60192-060 Fortaleza - Ceara - Brazil

Email: flaviosousa@ufc.br

A Petri net-based decision-making framework for assessing cloud services adoption: The use of spot instances for cost reduction

Maristella Ribas[a], C. G. Furtado[b], Neuman Souza[c], Giovanni Barroso[c], Antão Moura[d], Alberto S. Lima[c], Flávio R. C. Sousa[c]

[a]Techne Engenharia e Sistemas, São Paulo, Brazil
[b]Federal Institute of Ceará (IFCE), Fortaleza, Brazil
[c]Federal University of Ceará, Fortaleza, Brazil
[d]Federal University of Campina Grande, C. Grande, Brazil
mari@techne.com.br, neuman@ufc.br, cjunior@ifce.edu.br, gcb@fisica.ufc.br, antao@dsc.ufcg.edu.br, albertosampaio@ufc.br, flaviosousa@ufc.br

Abstract
Cloud services are widely used nowadays, especially in Infrastructure as a service (IaaS), with vendors offering several purchasing options and expanding the range of services offered on almost a daily basis. Cost reduction is a major factor promoting the adoption of cloud services among enterprises. However, qualitative factors need to be evaluated as well, thus rendering the decision regarding the adoption of cloud services among enterprises a non-trivial task for Information Technology (IT) managers. In this paper, we propose a place/transition or Petri net-based multi-criteria decision-making (MCDM) framework to assess a cloud service in comparison with a similar on-premises service. The framework helps IT managers choose between two such options, and can be used for any type of cloud service: Infrastructure as a Service (IaaS), Platform as a service (PaaS), Software as a service (SaaS), etc. Because its low cost is among the most important reasons for adopting cloud services, we also propose a Petri net to model cost savings using the spot instances purchasing option in public clouds. Through simulation of several scenarios, we conclude that spot instances present a very interesting cost-saving option in the auto-scaling process, even for simple business applications using few servers.

Keywords—*Cloud computing, Spot instances, BDIM, AHP, Petri nets.*

1. Introduction

Cloud computing is defined by the National Institute of Standard and Technology (National Institute of Standards and Technology, 2009) as a "model for enabling 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."

Infrastructure as a service (IaaS), Platform as a service (PaaS), and Software as a service (SaaS) are the classical categories of cloud services, although there are other proposed categories such as Database as a Service (DBaaS), Cache as a Service (CaaS), Unified Communications as a Service (UCaaS). In fact, cloud services are becoming so popular that some authors mention the category Everything as a Service (XaaS).

One of the many difficult decisions facing Information Technology (IT) managers nowadays is to choose between adopting an IT service in its cloud model and having the service hosted according to the conventional, on-premises model. The manager has to consider a large number of criteria, with cost being possibly the most important one. Cloud services are attractive because of the low-initial investment involved. However, if a cloud service is adopted by an organization, its IT department will no longer have any control over the infrastructure, and will completely rely upon the fulfillment of the Service-level Agreement (SLA) clauses in the contract to obtain the required service.

In this paper, we aim to shed some light on the business management problem of identifying the factors and parameters to determine the advantages and disadvantages of using cloud services in comparison with on-premises solutions for enterprise business applications. Furthermore, we focus on modeling and reducing the cost of the elasticity of cloud services. Also known as dynamic provisioning, elasticity "has become one of the most important features of a cloud computing platform" (Han et al., 2014). By using this feature, application owners can scale up or scale down resources used based on the computational demands of their applications, and need to pay only for the resources they actually use. Elasticity poses new challenges to resource management, as pointed in , and makes it more difficult to estimate cost, thus contributing to the greater complexity of the decision-making process.

As an example of the above-mentioned decision problem, consider a company that is developing a new platform to offer cloud services for developers of enterprise applications in a PaaS model.

Developers will use PaaS to build and publish their applications. The PaaS provider will be required to host all applications, and will need the corresponding infrastructure. This infrastructure can be leased from public clouds, fully hosted on premises, or supplied in a hybrid manner, partly on-premises and partly rented from public clouds. The IT manager must determine the services that the company will host internally and those that will be hosted on public clouds in the most cost-efficient manner, all the while ensuring an acceptable quality of service for PaaS clients. There are instances of commercial PaaS that internally host their infrastructure (Google, AWS) as well as others that lease public IaaS (Heroku). Hence, the choice of service can be a very difficult decision for the IT manager.

In a previous study (Ribas et al., 2014), we proposed an initial version of a framework to support decisions on whether to lease cloud services by considering factors related only to SaaS. This initial framework combined several of the most relevant factors for decisions of this type (according to the literature) and provided an assessment of both options (public clouds and on-premises hosting). The framework was designed using Petri nets (PNs) in order to benefit from their formal description and provide a visual interface that is simple, yet powerful enough to perform simulations for different purposes. In this paper, we extend the framework to capture cost elements for any type of cloud service, and propose a Petri net model to simulate IaaS utilization and compute cost savings in several scenarios. The PN model used to estimate cost savings uses a particular purchasing option for virtual machines, called spot instances . This purchasing option is currently supported by Amazon Web Services (AWS), the leader in the public IaaS market according to the Gartner Group . Spot instances work in exactly the same manner as any other Amazon Elastic Compute Cloud (EC2) virtual machine. The difference lies only in the price scheme: the hourly price for spot instances is not fixed, and clients bid on how much they are willing to pay. AWS dynamically defines spot price, which varies in real-time based on supply and demand. If a client's bid is higher than the spot price, the spot instance commences. If the spot price changes and rises above the client's bid, the instance is terminated by AWS. In this paper, we refer to any type of virtual machine that can be rented in a public cloud as an "instance." Our contributions to the literature are constituted by: 1) Our proposed framework, where we selected, using recent literature as well as practical experience, important factors to be considered as well as a reliable method to assess all factors and provide a simple final score. Moreover, the diagram of the Petri net makes it very easy to understand the underlying methodology. 2) The results of simulation scenarios executed while investigating cost reduction through spot prices, where a discount of up to 60% can be obtained without a significant effort, simply by efficiently using cloud resource purchasing options.

The rest of this paper is organized as follows: Section 2 is devoted to a review of related work in the literature, which provides the parameters need to build the decision-making model. In Section 3, we present and discuss the PN model that we constructed to implement our decision-making framework, and provide an example in Section 4 to explicate our decision-making model. Section 5 contains an introduction to our PN models built to investigate cost reduction using spot prices, whereas Section 6 contains a description of our simulation scenarios, their results, and our discussion. We offer our conclusions and directions for future research in Section 7.

2. Related work

The literature on cloud computing is growing as cloud services are becoming more popular. For a systematic literature review, we searched for the terms "SaaS" and "evaluation," "evaluating," or "evaluate" among publications after 2011 on IEEE explore, the ACM digital library, and Science Direct (Elsevier). From the 228 items yielded by the search, we selected 32 papers that seemed relevant to our study. Many of these focused on the evaluation of SaaS strictly from the perspective of technical performance. However, our work follows research on business-driven IT management (BDIM) , which includes technical as well as business-centric views.

A few studies have assessed cloud solutions from the point of view of BDIM. The framework proposed in is useful for comparing the cost of IaaS with that of on-premises datacenters. The primitive cognitive network process approach presented in is useful for selecting an offer of service (SaaS) from a list of providers of the same software. The survey in revealed that cost advantages were the strongest and the most consistent opportunity factor significantly affecting perceived opportunities in SaaS adoption, whereas security risks were the dominant risk factors, followed by performance and economic risks.

The study in presented results of a survey conducted in Korea to assess the adoption of SaaS and its related benefit to business, thus confirming the premises of the balanced scorecard (BSC) . The research in proposed a process based on goal-oriented requirements engineering (GORE) to provide a systematic approach to evaluate a cloud provider. In , the authors have provided SWOT analysis for the cloud computing industry, as well as various issues that will affect stakeholders.

Wu (2011) attempted to develop an explorative model that examined important factors affecting SaaS adoption, such as integrating Technology Acceptance Model (TAM)-related theories with additional imperative constructs, such as marketing effort, security, and trust. Security is the major risk affecting SaaS adoption according to most researchers (Benlian & Hess, 2011; Wu, 2011a and b; Wu, Lan, & Lee, 2011; Bayrak, 2013), whereas cost reduction is the major expected benefit (Benlian & Hess, 2011; Wu, 2011; Gupta, Seetharamana, & Raj, 2013; Bayrak, 2013).

Other interesting studies of cloud evaluation and adoption relate to pricing schemes , facets of security in the cloud , selection of cloud providers based on security, and privacy requirements . The cost analysis of on-premises solutions against SaaS solutions was conducted in detail in . In this study, unlike in our framework, the authors did not include qualitative benefits and risks to obtain a final score. Garg, Versteeg, and Buyya (2013) used the analytic hierarchy process (AHP) to combine quality of service (QoS) attributes in order to address the problem of selecting a cloud provider, which differs from our problem of comparing on-premises solutions with cloud solutions.

We also conducted a search for “cloud cost model” and selected 43 papers relevant to our study. An interesting comparison of on-premises services with cloud services was conducted by McGougha et al. (2014). They compared cost and overhead for high-throughput computing (HTC) jobs in two environments: a public cloud and a desktop cluster of non-dedicated resources. Their cloud cost model considered hours of use of instances and upload/download data. They noted that the start of the billing period varied among providers. Some, including AWS, charged from the start of the hour within which an instance was initiated — e.g., billing from 7 pm for an instance initiated at 7:59 pm — whereas others charged from the exact time at which the instance was initiated. Their on-premise cost model considered factors such as cost of hardware acquisition, cost of technical support for the desktop cluster, charges incurred for carbon emission, and energy cost (per kWh). They proposed six different cost saving policies in the cloud: P1, limiting the maximum number of cloud instances, P2, merging different users’ jobs, P3, instance keep-alive to avoid initialization time, P4, delaying the beginning of instances, P5, removing the delay on starting an instance, and P6, waiting for the start of the next hour.

Alfonso et al. (2013) compared the cost of HPC using on-premises services with that using cloud. The simplified cost of the on-premises cluster depended primarily on the purchase of the hardware, the maintenance and operation of the cluster, and its energy consumption (which can be reduced by turning off idle nodes). For the cloud cost model, they focused on hours of instance use and analyzed factors such as purchasing options. They proposed an equation to help decide if the option of reserving instances was preferable for a particular case, assuming one had sufficient information regarding cluster usage rate. Reservation is a purchase option offered by AWS, where one can pay a flat fee in advance and obtain a lower hourly rate for instance use. However, as of December 2014, AWS changed the reserved instance price model, and hence the equation will need to be updated.

Elasticity in multi-tier cloud applications was analyzed by Han et al. (2014). They proposed an algorithm that relied on online monitors to detect changes in workloads and perform corresponding scaling in each tier. The algorithm was designed to measure the cost of adding a server divided by the reduced response time due to this addition. Hence, this criterion is called the consumed cost/decreased response time (CC/DRT) ratio.

Some interesting studies investigated cost optimization using linear programming techniques (Malawski et al., 2013) by using Cache as a service (CaaS) to reduce input/output (I/O) costs and improve performance (Han et al., 2012). Baars et al. (2014) explored factors that affect chargeback for cloud services, mainly acceptability and effectiveness, and presented interesting insights on qualitative issues in cloud service use.

Spot instances have been studied (Javadi et al., 2013) through statistical models to characterize their behavior. The authors proposed probability density functions (pdfs) for the calculation of spot price and the interval of price spot change. Another study on spot prices (Tang et al., 2014) proposed a framework for bidding on spot prices in order to achieve monetary advantages, and still comply with SLA regulations.

Petri nets were used in (Sousa et al., 2014) as a tool for stochastic generation of dependability and cost models to represent cloud infrastructures.

To the best of our knowledge, no study provides a final ranking of cloud services in comparison with on-premises solutions considering both cost and other qualitative attributes, including security, which is a major concern. Furthermore, our search of the literature did not yield any study that proposed models to estimate cost reduction using spot instances.

3. The proposed decision-making framework

We propose a novel framework that combines cost and qualitative issues to produce a final score. We aim to employ a methodology that is simple for managers to visualize and understand (the PN visual graph is very helpful for that), and one that can be easily adapted to different scenarios. The mechanism of the framework can be divided into three steps:

1. Estimate the cost advantage (or disadvantage) of a cloud service over on-premise services
2. Evaluate the benefits and risks in a qualitative manner using a multi-criteria decision-making (MCDM) framework, such as AHP.
3. Compute the cost/benefit ratio. The option with the highest ratio is better.

Figure 1 shows our framework. The left part shows Step 1, cost comparison, whereas the upper right part shows Step 2, qualitative evaluation. Step 3 is shown in the lower right part, and contains the final output of the model.

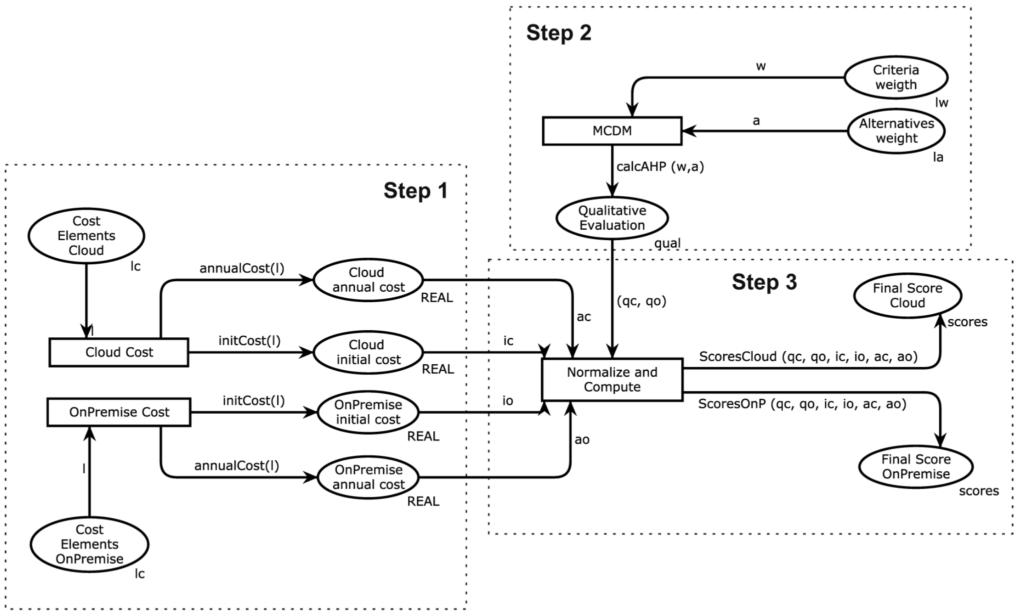


Figure 1: Proposed CPN multi-criteria decision-making (MCDM) framework.

The framework uses colored Petri nets (CPN), an extension of Place-Transition Petri nets, to model complex data types (named colors) and makes use of ML programming. The CPN model contains *places*, drawn as ellipses or circles, *transitions* drawn as rectangular boxes, directed arcs connecting places and transitions, and some textual inscriptions next to the places, transitions, and arcs. In this manner, our model can be summarized in one main graph. The model admits four input parameters — "cost elements cloud," "cost elements on premises," "criteria weight," and "alternatives weight" — and produces two outputs: "final score cloud" and "final score on premises." However, in order to completely understand the model, one needs to understand the colors and functions used. These elements will be discussed as we present each step of the framework.

3.1 Step 1: Cost estimates

For cost estimation, the colored Petri nets (CPN) model uses the following elements:

- Color *lc*: a list of cost elements. Each cost element is represented by a triplet consisting of three basic items of information: type of cost (Initial or Annual), name of cost element (e.g., labor, hardware, software), and value in financial terms. As an example, an annual software subscription of $500 can be modeled as (Annual, Subscription, $500). This color is used in the "cost elements cloud" and the "cost elements on premises" parameters

- Place *Cost Elements Cloud*: it contains a list of cost elements to estimate the total cost of the cloud service. This list is one of the input parameters of the model, since it is very specific to the service. As an example, a list for an SaaS cloud service can be [(Initial, Training, $19.20), (Annual, Salary, $60.00), (Annual, Connectivity, $36.0), (Annual, Subscription, $99.00)].
- Place *Cost Elements OnPremises*: it contains a list of cost elements to estimate the total cost for the service when it is locally operated on premises. This is an input parameter of the model, since it is specific to the service. The "OnPremises" list of cost elements is usually larger, as will be shown in Section 4.
- Function *initCost*(list): it admits a list of costs and returns the sum of all cost elements of type Initial in order to compute the initial investment necessary to operate the service. This function is used in transitions Cloud Cost and OnPremise Cost.
- Function *annualCost*(list): it admits a list of costs and returns the sum of all cost elements of type Annual in order to compute the annual expenditure required to operate the service. This function is used in transitions Cloud Cost and OnPremise Cost.
- Transition *Cloud Cost*: it computes the initial investment and annual expenditure required to operate the cloud service. Following computation, the function will set the computed values as markings of places "Cloud initial cost" and "Cloud annual cost."
- Transition *OnPremise Cost*: it computes the initial investment and annual expenditure required to operate the cloud service. These costs then become the marking of places "OnPremise initial cost" and "OnPremise annual cost."

Cost estimates for IT services, both on-premises and cloud services, can vary significantly depending upon the application, the size of the enterprise, and the complexity of the enterprise's business processes . In our previous work (Ribas et al., 2014), we followed Bibi, Katsaros, and Bozanis (2012). Their proposed model addressed initial costs, which are one-time costs, as well as the expected annual divestment and operational costs. To extend this model to capture all possible cost elements, we introduced the color set *lc*, as described previously. Altmann and Kashef (2014) researched cost elements in recent literature for IaaS cloud services. Table 1 shows a shorter version of the cost factors explored by them. Since their work is related to hybrid clouds, a few cost factors listed are applicable to public cloud services, a few are applicable to private cloud services, and some are applicable to both.

Table 1: List of cost elements adapted from

Cost Type	Cost Elements
(a) Electricity	(a1) Cooling private cloud
	(a2) Electronic devices (idle)
	(a3) Electronic devices (in use)
(b) Hardware	(b1) Server
	(b2) Network device
(c) Software	(c1) Basic server software license
	(c2) Middleware license
	(c3) Application software license
(d) Labor	(d1) Software maintenance
	(d2) Hardware maintenance
	(d3) Other support
(e) Business Premises	(e1) Rack, air conditioner
	(e2) Cabling
	(e3) Facility
(f) Cloud Service	(f1) Internet connectivity
	(f2) Cloud service use
	(f3) Data transfer to cloud
	(f4) Data transfer from cloud
	(f5) Cloud storage

	(f6) Data transfer between clouds
(g) Deployment	(g1) Number of deployments

Our framework is built in order to easily accommodate any type of pre-computed cost element for any type of cloud service.

Step 2: Evaluating benefits and risks

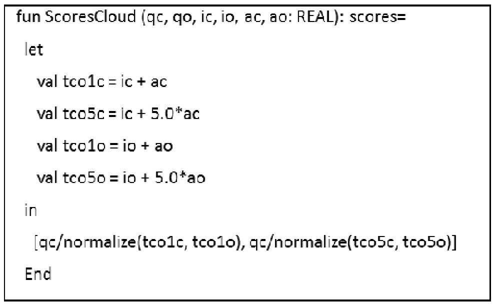
In order to assess the benefits and risks, we use AHP , as a classical MCDM method, since qualitative factors need to be analyzed and AHP has been often used in the literature to evaluate IT and cloud services (Garg, Versteeg, & Buyya, 2013; Yuen, 2012). There are two competing services to be ranked: cloud and on-premises. For benefits and risk evaluation, the CPN model uses the following elements:

- Color *lw*: a list of tuples of a string and a real number (criterion, weight), where "criterion" represents the name of the criterion and "weight" is the value assigned to that criterion (e.g., the list could be [("strategy," 0.3), ("quality," 0.4), ("risk," 0.3)].
- Color *la*: a list of triples of two strings and a real number (criterion, alternative, weight) where "criterion" represents the name of the criterion, "alternative" the name of the alternative, and "weight" is the value assigned to that criterion for that alternative. (e.g., the list could be [("strategy," "cloud," 0.5), ("strategy," "onP," 0.5), ("quality," "cloud," 0.5), ("quality," "onP," 0.5), ("risk," "cloud," 0.5), ("risk," "onP," 0.5)]
- Color *qual*: a pair of real numbers *(qc, qo)* representing the qualitative evaluation of the cloud service *(qc)* and the on-premise service *(qo)*. Place *Qualitative Evaluation* is of this color.
- Place *Criteria weight* holds a list of criteria and their precomputed weights. This is an input parameter to the framework. In Section 4, we will exemplify qualitative criteria commonly mentioned in the literature.
- Place *Alternative weight:* holds a list of weights assigned to the criterion in question for each alternative to be used in AHP methods. This is also an input parameter to the framework.
- Transition *MCDM:* computes the final ranking based on the weight of each criterion using a function (*calcAHP*) to perform the necessary calculations.
- Function *calcAHP (lw, la):* takes the list of criteria weights and alternative weights, and computes final scores for each alternative. In our framework, this function implements the AHP method, but it can be customized to use another method for multi-criteria decision making, e.g., the primitive cognitive network , as long as the function returns a pair of real numbers *(qc, qo)* to represent the qualitative evaluation of cloud and on-premises services.
- Place *Qualitative Evaluation:* contains the result of the qualitative evaluation returned by function *calcAHP*.

Step 3: Compute cost/benefit ratio

The last step of the framework normalizes the costs computed in Step 1 and computes a benefit/cost ratio that ranks the alternatives. For this step, the CPN model uses the following elements:

- Color *scores*: a list of real numbers to hold final scores for each alternative. The use of the list allows the framework to produce several evaluations for each alternative. In our example in Section 4, we will compute scores for one-year and five-year analyses for each alternative.
- Transition *Normalize and Compute*: takes cost estimates (initial and annual) for cloud services and on-premises services as well as the qualitative evaluation for both, and generates the final score. For this, it uses two customizable functions, *ScoresCloud* and *ScoresOnP*.
- Function *ScoresCloud*: computes a score based on user-defined techniques. In our example in Section 4, we implement these functions to compute the total cost of ownership (TCO) for one-year use and five-year use following . We then normalize to obtain values between 0 and 1, and compute the cost/benefit ratio of using each service for one and five years, as shown in Figure 2. This example function can be replaced by methods that include more complex financial calculations to handle depreciation costs, for instance.
- Function *ScoresOnP*: similar to *ScoresCloud*, and is used to compute cost/benefit ratio for on-premise services



```
fun ScoresCloud (qc, qo, ic, io, ac, ao: REAL): scores=
 let
   val tco1c = ic + ac
   val tco5c = ic + 5.0*ac
   val tco1o = io + ao
   val tco5o = io + 5.0*ao
 in
  [qc/normalize(tco1c, tco1o), qc/normalize(tco5c, tco5o)]
 End
```

Figure 2: User-defined ScoresCloud function.

4. An Illustrative example

In order to elaborate on the decision-making scenario, we used the problem presented in (Ribas et al., 2014), where a private school with 2,000 students needs to upgrade its academic management software. The current software was purchased when the school was much smaller, and has been rendered obsolete with the advent of new mobile technologies. The current software uses two servers, one for the Web and the other for the database. These two servers also need to be replaced due to technical problems. The IT manager, after careful research, decides to buy a new software, SCHOOL1, with two deployment options:

1) On-premises option: Perpetual license, whereby the school will have to pay an initial licensing fee of R$145,000 (Brazilian reais - value for up to 2,500 students). There is also an optional annual maintenance fee of R$29,000 for technical support and software upgrades. An IT infrastructure is needed to install and run the software.

2) SaaS option: Annual subscription of R$99,000 (value for up to 2,500 students, which includes technical support and software upgrades), whereby the school will be able to access the software using a Web browser. Moreover, the vendor will sign an SLA to ensure a certain level of service for availability, performance, and security.

In both the two above options, the vendor recommends a training package that enables IT personnel and power users to customize the software to meet the school's requirements in order to ensure that users can operate the software in the most adequate and efficient manner. In the on-premises option, it is necessary to purchase two physical hosts (or a corresponding virtual solution) for the database and the Web servers. Software licenses for a Windows Server and Microsoft SQL Server would be required as well. We estimated the annual salary of two IT professionals to be R$60,000 each. Internet costs were estimated to be R$24,000 and energy at R$3,600 per year. For the SaaS option, the training prices are the same, but the cost of service is 10% lower because there is no installation cost. Internet costs are higher than the on-premise option at R$36,000. However, expenditure on salaries will decrease because only one IT person will be required.

Table 2: Lists of cost elements (values in thousands)

On-premises			
Initial	Training	R$19.20	TCO 1st year (initial + annual)
Initial	Services	R$38.40	
Initial	Hardware	R$8.00	
Initial	Software	R$145.00	
Initial	Middleware	R$11.00	R$399.16
Initial	TOTAL	R$221.60	
Annual	Salary	R$120.00	TCO 5 years (initial + 5 × annual)
Annual	Connectivity	R$24.00	
Annual	Software	R$29.00	
Annual	Hardware	R$0.96	
Annual	Energy	R$3.60	R$1,109.40

Cloud			
Initial	Training	R$19.20	TCO 1st year (initial + annual)
Initial	Services	R$34.56	
			R$248.76
Initial	TOTAL	R$53.76	
Annual	Salary	R$60.00	TCO 5 years 5 (initial + 5 × annual)
Annual	Connectivity	R$36.00	
Annual	Subscription	R$99.00	
			R$1,028.76

Annual	TOTAL	R$177.56	

Annual	TOTAL	R$195.00	

Our model requires two input parameters for cost: "Cost Elements Cloud" and "Cost Elements On Premises." By using the estimates as described above, we built the cost elements table for both alternatives, as shown in Table 2. We also computed the TCO for the one-year and five-year periods.

We thus obtained the following costs for the on-premises model: R$399.16 for the first year and R$1,109.4 for five years. The SaaS option would cost R$248.76 for the first year and R$1,028.76 for five years. Figure 3 shows the evolution of cost for both models. We can see that costs are initially lower for the cloud option but, after five years, they become similar to the cost of the on-premise service. Since the cost function is linear, we expect that cloud costs will be higher than on-premises costs in the long term. However, depreciation costs were not considered in this instance of our framework. Had we chosen a different financial method in the *ScoresCloud* function, for instance net present value (NPV), which was also used in the literature for cloud cost comparison , we might have obtained a result whereby cloud services always cost less than on-premises services.

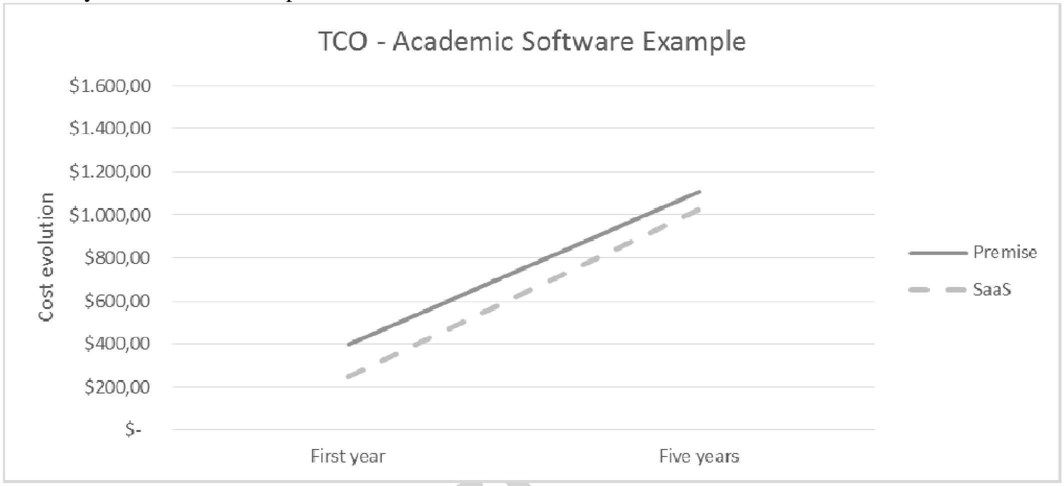


Figure 3: Cost evolution in the example.

Following this, we performed the qualitative evaluation to obtain the parameters of the model for Step 2. We used previous studies [, and] to select the most important criteria for evaluation:

- **Strategy**: refers to strategic issues, such as: 1) flexibility in switching IT provider, 2) scalability, 3) quicker implementation of applications, 4) reduction of vendor lock-in due to lower initial costs, and 5) concentrating efforts on the specific core competencies of the enterprise.
- **Quality**: refers to the efficiency and effectiveness of processes supported by the application services.
- **Performance risks**: refer to the possibility that the alternative might not deliver the expected level of service.
- **Security risks**: refer to the possibility of data corruption, data leakage, errors in authentication, and other threats to security. This is by far the most important inhibitor of SaaS adoption (Lee et al., 2013).
- **Economic risks**: refer to the possibility that a client may have to pay more to obtain the expected level of service than initially anticipated — the so-called "hidden costs" .

We deliberately left cost out of this part because "discussing costs together with benefits can sometimes bring forth many political and emotional responses" . Although AHP method could summarize all criteria, including costs, we believe pairwise comparison between cost and other qualitative criteria may lead to inappropriate weight assignment for qualitative issues. For simplicity, we also excluded criteria that were not considered significant in previous studies, such as 1) access to specialized resources (human and technological) internally unavailable, and 2) managerial risks, which constitute the possibility that the personal reputation and career of the manager responsible for the application may be affected if the software is sourced to an external service provider .

To obtain a pairwise comparison, we asked experienced practitioners to assign reasonable values to the parameters in our example, whereby the consistency indices CI and ratios CR were checked, yielding a value of CR < 0.1, which is acceptable. Tables 3 and 4 show weights obtained in our pairwise comparisons.

Table 3: Weights for criteria and alternatives

Criterion	Alternative	Weight
Strategy	OnPremises	0.357
Strategy	Cloud	0.643
Quality	OnPremises	0.42
Quality	Cloud	0.58
Performance Risks	OnPremises	0.643
Performance Risks	Cloud	0.357
Security Risks	OnPremises	0.75
Security Risks	Cloud	0.25
Economic Risks	OnPremises	0.9
Economic Risks	Cloud	0.1

Table 4: Weights for criteria

Criterion	Weight
Strategy	0.15
Quality	0.2182
Performance Risks	0.2417
Security Risks	0.2077
Economic Risks	0.1821

Using these weights, transition MCDM (Figure 1) will use AHP method (calcAHP function) and will compute qualitative evaluation as (0.62, 0.38), meaning that OnPremises option was ranked first, with value 0.62, and Cloud option was ranked second, with values 0.38. Finally, we calculated the cost/benefit ratio in Step 3. Table 5 shows the results. The on-premises option has the best ratio for the first year as well as for the five-year analysis. This is because of the poor evaluation of SaaS in Step 2. The cloud option, even with lower cost, did not convince our decision makers.

Table 5: Final scores

	On-premises	**SaaS**
First year	1.028	0.956
Five years	1.210	0.778

The ratio of the cloud service was closer to that of the on-premises service in the first year because the latter recorded higher initial costs. However, in the five-year analysis, the on-premises option fared much better. These conclusions are valid only for our example, which was based upon a scenario in Brazil, which has high Internet costs, medium income ranges, and a strong perception of security and economic risks. Practitioners fear for internet hackers and changes in pricing of cloud services. In fact, the qualitative analysis was the major hindrance to choosing the cloud service, and its advantage in terms of cost was not sufficiently strong to revert to this situation. In our example, in order to appeal more to customers, cloud costs would have to be less than 61% of on-premises costs to compensate for the qualitative ranking of 0.62 assigned to on premises option.

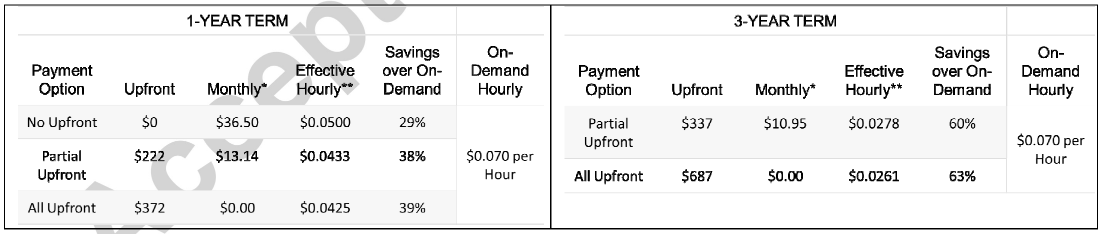
We presented this example to clarify the use of our framework by assigning values to the inputs of the model and following the calculations step by step. Each step can help users organize ideas and reason through each part of the problem. The model produced as output scores that reflected the restrictions on and the preferences of the decision maker; hence, each output was unique. However, we believe that using our framework, the decision maker will have a better insight of the problem and thus can make a better decision. Complete face validity of this model is listed as a future work.

5. Using spot instances for cost reduction

Our decision-making framework takes as input a list of cost elements, a list of qualitative criteria, and weights, and generates as output scores for the cloud and on-premises services. As seen in the example in Section 4, in some scenarios, the cost of cloud services needs to be very low to convince decision makers to adopt this option. An important part of the cost elements of cloud services for IaaS is the server use cost, i.e., the monthly charge levied by providers for the use of their virtual machines in the cloud. Cloud providers usually charge customers according to a pay-per-use basis, whereby the customer pays for each hour (or minute, or month) for which the machine is turned on. If the machine is off, there is no charge.

Each IaaS provider has its own billing model for virtual machines. In this paper, we investigate the three purchasing options currently available for Amazon Web Services (AWS):

- **On-demand**: In this option, charges are levied for each hour the virtual machines (called “instances”) are turned on. There is no upfront investment and no commitment of use. It is a simple use-and-pay method but usually incurs the most expensive hourly rate.
- **Reserved**: As of December 2014, in this option, customers pay for the period of reservation instead of hours of use. Payment options may be no upfront, partial upfront, or all upfront. Figure 4 shows prices for the eastern region of the US and instance size m3.medium (1 vCPU, 3.75 Gb, 1 × 4 SSD Storage). AWS provides an hourly estimate of cost to help compare prices with the on-demand option. However, reserved instances are paid for according to the relevant period (month, year) and not by number of hours of use. This means that if a reserved instance for a month is purchased, it makes no difference whether it stays on: the cost will be the same. The prices of reserved instances prices can be equivalent to on-premises costs of operating a local server.
- **Spot**: In this option, charges are levied for hours of use, similar to the on-demand option. However, the hourly rate is not fixed. Clients bid on how much they are willing to pay for the hour. AWS dynamically defines spot price, which varies in real time based on supply and demand. If a client’s bid is higher than the current spot price, that particular instance is initiated. If the spot price changes to rises above the client’s bid, the particular instance is terminated by AWS.



1-YEAR TERM					
Payment Option	Upfront	Monthly*	Effective Hourly**	Savings over On-Demand	On-Demand Hourly
No Upfront	$0	$36.50	$0.0500	29%	$0.070 per Hour
Partial Upfront	**$222**	**$13.14**	**$0.0433**	**38%**	
All Upfront	$372	$0.00	$0.0425	39%	

3-YEAR TERM					
Payment Option	Upfront	Monthly*	Effective Hourly**	Savings over On-Demand	On-Demand Hourly
Partial Upfront	$337	$10.95	$0.0278	60%	$0.070 per Hour
All Upfront	**$687**	**$0.00**	**$0.0261**	**63%**	

Figure 4: Reserved instances prices for eastern US with instance size m3.medium .

To investigate how the use of spot instances can help cost reduction, we created an additional CPN model hierarchically organized in modules that compute 1) the monthly cost of all running instances, and 2) savings by using spot instances. In our model, there is one (could be more, if necessary) reserved instance that is always switched on to guarantee the service at all times. Since the instance is always switched on, the reserved option is the most cost-effective option. Other instances are turned on and off whenever needed by monitoring the demand for servers by simulating the auto-scaling process. In this manner, the model will simulate the elasticity of server use. Figure 5 shows the main CPN model that is decomposed into four subnets: Monitor Auto-scaling, Scale Up, Scale Down, and Spot Termination. We will discuss these in detail in sections 5.1 to 5.5.

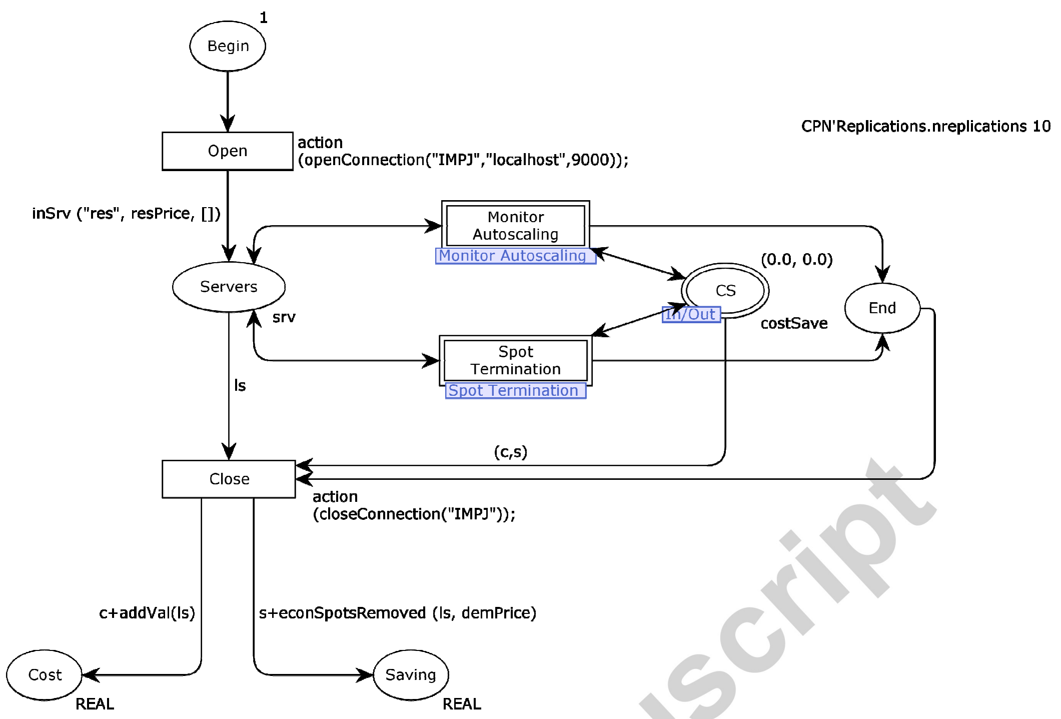


Figure 5: CPN model for instance use simulation.

5.1 CPN Model: Instance Use Simulation

The above model represents our proposed mechanism for using spot instances for cost savings in elasticity where instances will be turned on and off in the auto scaling process. There are four input parameters for the model:

- Hourly price for on-demand instance, represented by the constant *demPrice* in the model.
- Hourly price for the reserved instance, represented by the constant *resPrice* in the model.
- Hourly price for spot instance. These values are obtained dynamically by using the AWS application programming interface (API) during the simulation period. To accomplish this, we used a special programming interface for Java and CPN Tools .
- demand() function: This function represents the demand for servers in the auto-scaling process. It returns the number of servers needed at a given point in time. It must be customized when using the model. Figure 6 shows an example of the use of the demand() function. In this example, the function returns “1” (i.e., one server needed) for nighttime (11 pm to 6 am) and “1 or more” servers for daytime (6 am to 10 pm). To compute the number of extra servers needed, we use a normal distribution with the average value of μ = 3 and standard deviation δ = 0.5, as a hypothetical pattern of use. Our hypothetical extra load is then μ ± 2δ 95% of the time, i.e., during daytime, the number of extra servers needed varies between two and four. This function should be adopted to reflect each scenario involving the need for extra servers in auto-scaling. For instance, Han et al. (2014) monitored the request queue size to infer the need for extra servers. AWS offers a specific cloud service, called CloudWatch, to monitor computing metrics, such as CPU use, memory use, etc., that can trigger the need for extra servers. The demand() function makes the framework easily adaptable to specific needs.

Figure 6: Example of a customizable demand() function.

There are two output values of the model:

- Cost: total monthly cost of the use of EC2 instances, including charges for all instances (reserved, spot, and on-demand)
- Saving: total savings obtained by using spot instances compared to those using on-demand instances.

The CPN model uses the following elements:

- Color *server*: represents an instance currently turned on. It is a 3-tuple consisting of (type, price, time), where "type" can be *res*, *dem,* or *spot*, to identify the purchasing option (reserved, on-demand, or spot), "price" is the hourly rate of the particular instance, and "time" is the start time of use of the instance.
- Color *srv*: list of active instances representing all servers currently turned on.
- Color *costSave*: a 2-tuple consisting of real numbers (r1, r2), where r1 represents the total monthly cost and r2 represents the total savings.
- Place *Begin*: contains a one-time token to start the simulation process at model time 0.
- Transition *Open*: establishes a connection with the Java programming interface and initializes the list of servers, including a reserved instance.
- Function *inSrv (type, price, list)*: inserts a server of a given type and price to the list of active servers.
- Transition *Monitor Autoscaling*: a substitution transition to model the autoscaling process (turning servers on and off as needed). We discuss this in detail when presenting the corresponding subnet.
- Transition *Spot Termination*: a substitution transition to model the spot termination process (turning servers off due to changes in spot prices), and is discussed in detail when the corresponding subnet is presented.
- Place *cs*: used as a temporary space to add the cost of using a server when it is turned off. One can compute the total hours of use of the server and multiply the result by the hourly price. It is discussed in detail when we present the *Monitor Autoscaling* and *Spot Termination* subnets.
- Place *end*: receives a token when the simulation reaches a predefined age (720 hours = 24 hours × 30 days), representing the end of the month being analyzed.
- Transition *Close*: terminates the connection with the Java programming interface and finalizes the simulation process, modeling the action of turning off all servers.
- Function *addVal(list)*: computes the cost of use of each server in the list, multiplying hours of use by the hourly price and adding up the cost of use of all servers.
- Function *econSpotsRemoved (list)*: computes the savings of using each spot instance in the list, multiplying hours of use by (on-demand hourly price - spot hourly price). It then adds savings for all instances.
- Place *Cost*: at the end of simulation, its marking represents the total cost of server use.
- Place *Saving*: at the end of simulation, its marking represents the total savings using spot instances.

We now discuss the subnets used in the main model.

5.2 Subnet: Monitor Auto-scaling

This subnet models the auto-scaling process. Figure 7 shows the CPN model that implements the transition "Monitor Autoscaling" in the main net. It is executed at each time unit and monitored to determine whether there is a need to turn servers on or off. With regard to our objective of cost comparison, the time unit was one hour. The model would not have been significantly different if we had used smaller time units, such as minutes or seconds. Monitor Autoscaling model compares the needs of the servers at a particular time (given by the demand() function) with the number of active servers (the length of the list of active servers). When demand is greater than the current number of servers in use, it places a token in *High Use*, which drives the *Scale Up* process. When demand is smaller than the current number of servers in use, it places a token in *Low Use*, which drives the *Scale Down* process. If demand is neither greater nor smaller than the current number of servers in use, it waits until the next hour by placing a token in *Wait* that will only be available then. It also checks the end of the simulation and, in this case, places a token in *End*.

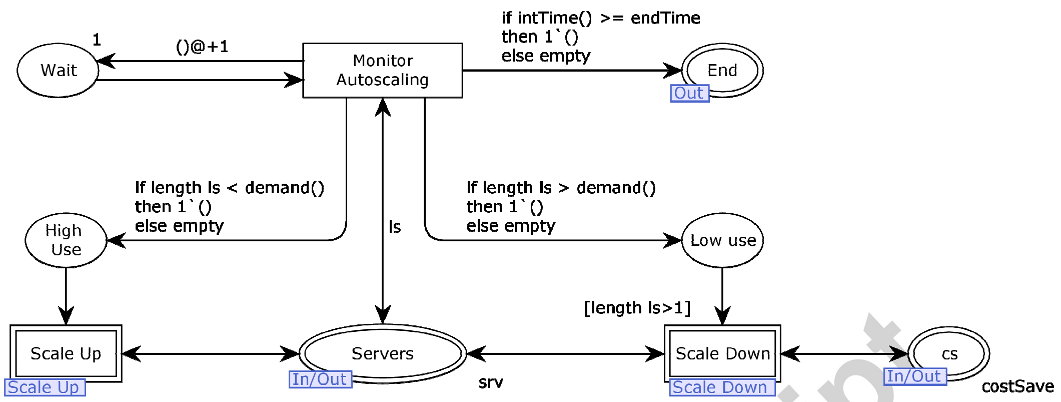
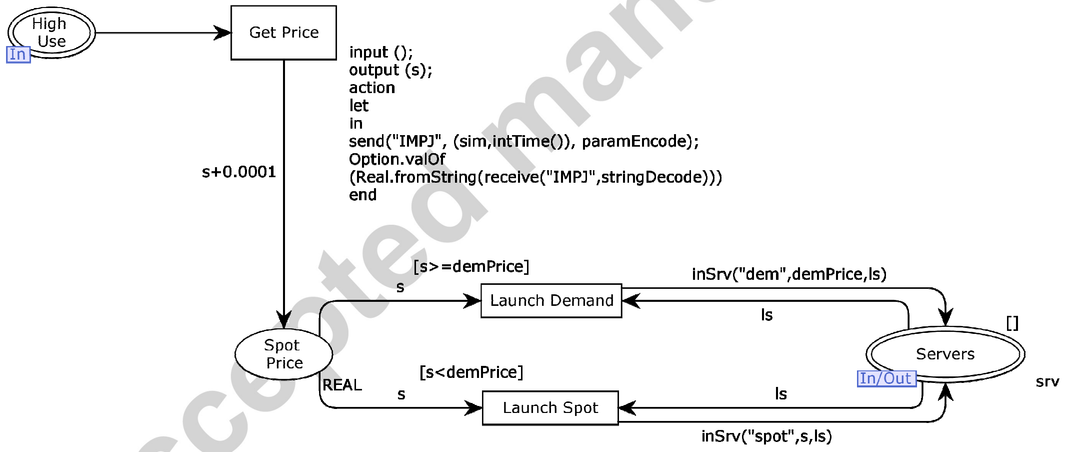


Figure 7: Monitor Auto-scaling subnet.

5.3 Subnet: Scale Up

This subnet models actions needed to add servers to our server list at the least cost possible. Figure 8 shows the subnet *Scale Up*.



```
input ();
output (s);
action
let
in
send("IMPJ", (sim,intTime()), paramEncode);
Option.valOf
(Real.fromString(receive("IMPJ",stringDecode)))
end
```

Figure 8: Scale Up subnet.

The policy implemented by this subnet is to obtain the current spot price, place a slightly higher bid, and verify whether the spot instance or the on-demand instance has the lowest price. It is important to notice that on-demand instances are always available such that this subnet will always add one server to our server list. The CPN model uses the following elements:

- Place *High use*: has a token when scale up is needed
- Transition *GetPrice:* computes an optimal bid for a spot instance. For this, it uses the Java interface to obtain the current spot price for the model time passing parameters *(simulation#, modelTime)* where “simulation#” represents a set of parameters needed to obtain spot prices and “modelTime” represents the day of month and time to obtain the spot price. Tables 6 and 7 list input parameters to the Java interface.
- Place *Spot Price*: holds the bid for the spot instance, which is the spot price (a real number) returned by the Java interface plus $0.0001. In this manner, we ensure that our bid is sufficiently high to obtain a spot instance at the least possible cost.
- Transition *Launch Demand*: fired when the marking in *Spot Price* is higher than that in the on-demand price. In this case, it is not reasonable to use the spot instance because it is more

expensive. Transition launch demand inserts a server purchased using the on-demand option into the list of active servers with its corresponding *demPrice* (input parameter). For this, it uses the *inSrv* function (type, price, list) that saves the model time when the server is inserted into the list and updates the list of active servers.

- Transition *Launch Spot*: fired when the marking in *Spot Price* is lower than that in the on-demand price. In this case, it is reasonable to use the spot instance because it is currently cheaper. Transition launch spot inserts a server purchased using the spot option into the list of active servers with its corresponding price, i.e., the bid. It uses the *inSrv* function.

Table 6: Parameters for obtaining spot prices in AWS

Simulation	Region	OS	Instance Type
1	South America	Windows	m3.medium
2	South America	Windows	m3.2xlarge
3	South America	Linux/UNIX	m3.medium
4	South America	Linux/UNIX	m3.2xlarge
5	US-East	Windows	m3.medium
6	US-East	Windows	m3.2xlarge
7	US-East	Linux/UNIX	m3.medium
8	US-East	Linux/UNIX	m3.2xlarge
…			

Table 7: Model time and corresponding time of day in simulation

Model Time	Day of Month	Hour of day
1	1	0
2	1	1
…	…	…
24	1	23
25	2	0
...	…	…
720	30	23

5.4 Subnet: Scale Down

This subnet models actions needed to remove servers from our server list, selecting first the servers with the greatest possible cost in order to keep using the cheaper ones. Figure 9 shows subnet *Scale Down*.

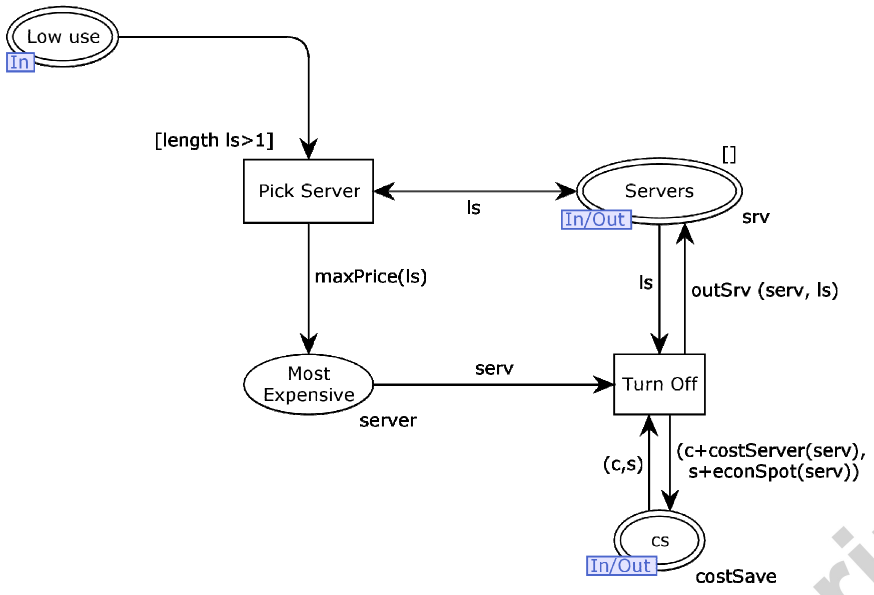


Figure 9: Scale Down subnet.

The policy implemented by this subnet is to select and turn off the most expensive server in terms of hourly price. It also implements the policy of leaving always on the server covered by a reservation (where the client paid an upfront fee to obtain better hourly rates). In this manner, even when there is low utilization, a server will be available. By leaving one server always on, we to model real-world scenarios in business applications available 24 hours a day. It can also model situations where the “reserved” server is actually an internally hosted server, as in hybrid clouds. It is worthy of note that in terms of cost in some scenarios, the cost of a “reserved” server is identical to that of an internally hosted server . The CPN model uses the following elements:

- Place *Low use*: has a token when a scale down is needed
- Transition *Pick server*: picks the most expensive server to be turn off. It uses the function *maxPrice*(list) that selects the server with the highest hourly rate. This function excludes reserved instances because turning these off makes no difference to the final cost, as reserved instance are always charged for the entire period regardless of use.
- Transition *Turn Off*: excludes the selected server from the server list. It also computes the total cost of using the instance in question, as well as savings generated by using spot instances in comparison with those obtained through on-demand instances by using functions *CostServer* and *EconSpot*
- Function *CostServer* computes the cost of using the server using the expression: Cost of use = (hours of use) × (hourly price).
- Function *EconSpot* computes savings using the expression: Savings = (hours of use) × ((hourly price for on-demand) – (hourly price for spot)). It is important to note that savings are be computed when turning off spot instances because they are set to zero when turning off on-demand instances (hourly price for on-demand = hourly price)

5.5 Subnet: Spot Termination

This subnet models the regular verification of the spot price market. As mentioned in the beginning of section 5, spot instances are automatically terminated by AWS when the spot price rises above the rate that the client is currently paying. Figure 10 shows the subnet Spot Termination.

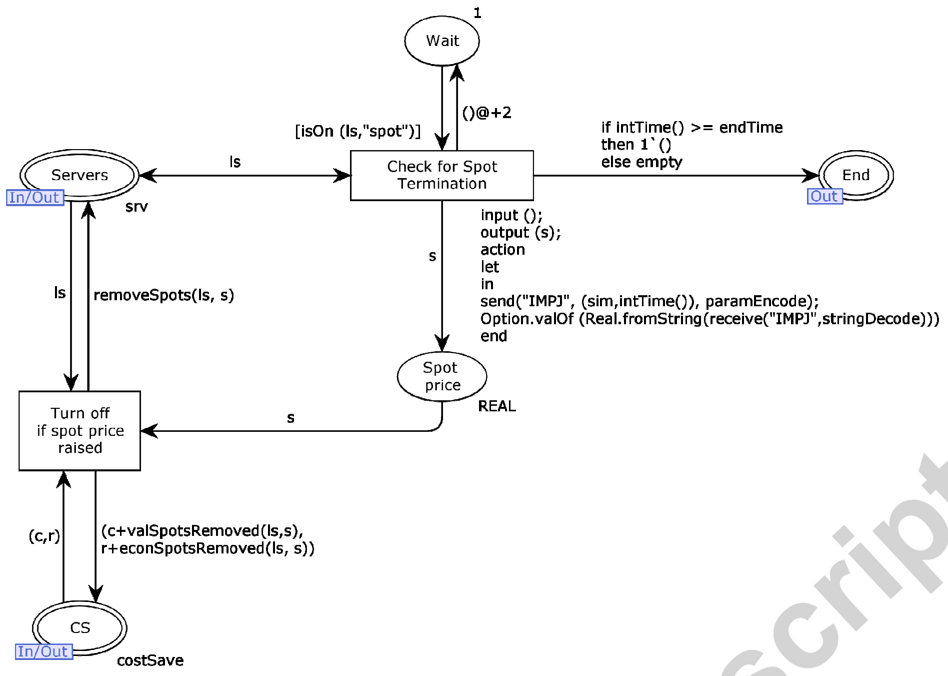

Wait
1
()@+2
[isOn (ls,"spot")]
Servers
In/Out
srv
ls
Check for Spot Termination
if intTime() >= endTime
then 1`()
else empty
End
Out
input ();
output (s);
action
let
in
send("IMPJ", (sim,intTime()), paramEncode);
Option.valOf (Real.fromString(receive("IMPJ",stringDecode)))
end
s
ls
removeSpots(ls, s)
Spot price
REAL
Turn off if spot price raised
s
(c,r)
(c+valSpotsRemoved(ls,s),
r+econSpotsRemoved(ls, s))
CS
In/Out
costSave

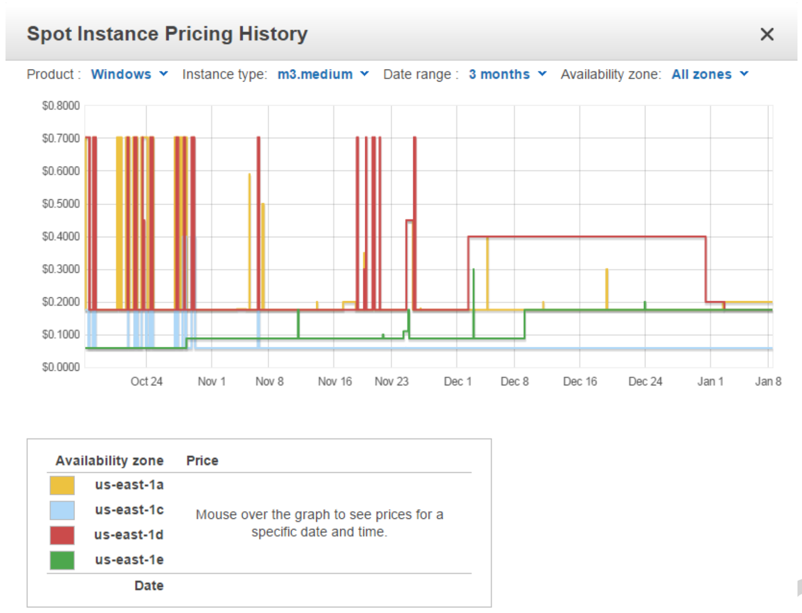
Figure 10: Spot Termination subnet.

The CPN model uses the following elements:

- Transition *Check for Spot Termination (simulation#, modelTime)*: is executed every two hours to monitor spot prices and simulate the termination of spot instances. To this end, it uses the same Java interface used in the transition *GetPrice* in the subnet Scale Up in order to obtain the current spot price. It is only fired when there is at least one spot instance in the list of active servers. Following firing, it waits for two hours by placing a token in *Wait* that will only be available then. It also checks the end of the simulation and, in this case, places a token in *End*.
- Place *Spot Price*: contains the current spot price returned by the Java interface.
- Transition *Turn Off if spot price raised*: checks the list of active servers looking for spot instances with hourly rates lower than current spot prices. These instances are terminated (turned off). It uses *valSpotsRemoved*() and *econSpotsRemoved*() to compute the cost of the use of terminated instances and savings as computed in the Scale Down subnet. The only difference is that *Turn Off if spot price raised* may simultaneously remove several servers from the list of active servers. In our model, the termination of these instances may cause several Scale Up transitions to be fired if utilization levels are still high.

6. Cost reduction simulation results

The experimental design of the simulations followed a fractional 2^k design for each geographic region, where AWS stored their datacenters and the spot instances were available. AWS currently offers services in nine regions: US-East (N. Virginia), US-West (Oregon), US-West (N. California), EU (Ireland), EU (Frankfurt), Asia Pacific (Singapore), Asia Pacific (Tokyo), Asia Pacific (Sydney), and South America (São Paulo). Each region has at least two availability zones (AZs), which are datacenters in different locations connected through low-latency links. Each region has its own pricing table for on-demand and reserved instances. Spot instances price may also vary by AZs in each region. Figure 11 shows spot prices for different availability zones in US-East. One can see that the distribution of prices is random but remains stable (and low) for long periods. Note that for the availability zone US-East-1c, prices were almost constantly low ($0.0591) for two months — November and December. The dynamic graph shown below was available at the AWS console, but an AWS account was needed to access it under the option “EC2 – Spot Requests.”


Spot Instance Pricing History

Product : Windows | Instance type: m3.medium | Date range : 3 months | Availability zone: All zones

Availability zone	Price
🟨 us-east-1a	Mouse over the graph to see prices for a specific date and time.
🟦 us-east-1c	
🟥 us-east-1d	
🟩 us-east-1e	
Date	

Figure 11: Price history for US-East region.

The *GetPrice* function returns the lowest price among all availability zones in a given region. This is to ensure the highest possible savings. Other factors that affect instance prices in AWS are:

- Operating System: AWS has different prices for hours of use of EC2 instances depending, on the OS: Linux/Unix, SUSE Linux, and Windows.
- Instance type: EC2 prices vary with the size of instance. For our experiments, we considered m3.medium (1 vCPU, operating at 3 elastic compute units (ECUs) with 3.75 GB of memory, and a 1HD-type solid-state drive (SSD) with a storage capacity of 4 GB) and m3.2xlarge (8 vCPU, at 26 ECUs, 30 GB memory, and a 2HD-type SSD with a storage capacity of 80 GB)

Table 8 lists factors and levels in the experimental design of our simulations, and Table 9 lists the corresponding AWS instance prices. For each of the selected $9 \times 2^2 = 36$ simulations, we initially ran 10 simulations to obtain relevant statistical information.

Table 8: Factors and levels for our experimental design

Factor	Levels	Selected for Experiment
Region	9	All
OS	3	Windows, Linux/Unix
Instance Type	Over 20	m3.medium, m3.2xlarge

Table 9: Instance prices used in simulations

				Hourly price		
Simulation	Region	OS	Instance Type	On-demand	Reserved (1 year all upfront)	Upfront Investment
1	South America	Windows	m3.medium	0,1580	0,1410	1.235,00
2	South America	Windows	m3.2xlarge	1,2650	1,1205	9.816,00
3	South America	Linux/UNIX	m3.medium	0,0950	0,0509	446,00
4	South America	Linux/UNIX	m3.2xlarge	0,7610	0,4063	3.559,00
5	US-East (N.Virginia)	Windows	m3.medium	0,1330	0,0855	749,00
6	US-East (N.Virginia)	Windows	m3.2xlarge	1,0640	0,6809	5.965,00

7	US-East (N.Virginia)	Linux/UNIX	m3.medium	0,0700	0,0425	372,00
8	US-East (N.Virginia)	Linux/UNIX	m3.2xlarge	0,5600	0,3412	2.989,00
9	US-West (Oregon)	Windows	m3.medium	0,1330	0,0855	749,00
10	US-West (Oregon)	Windows	m3.2xlarge	1,0640	0,6809	5.965,00
11	US-West (Oregon)	Linux/UNIX	m3.medium	0,0700	0,0425	372,00
12	US-West (Oregon)	Linux/UNIX	m3.2xlarge	0,5600	0,3412	2.989,00
13	US-West (California)	Windows	m3.medium	0,1400	0,0983	861,00
14	US-West (California)	Windows	m3.2xlarge	1,1200	0,7829	6.858,00
15	US-West (California)	Linux/UNIX	m3.medium	0,0770	0,0532	466,00
16	US-West (California)	Linux/UNIX	m3.2xlarge	0,6160	0,4216	3.693,00
17	EU (Ireland)	Windows	m3.medium	0,1330	0,0983	861,00
18	EU (Ireland)	Windows	m3.2xlarge	1,0640	0,7829	6.858,00
19	EU (Ireland)	Linux/UNIX	m3.medium	0,0770	0,0522	457,00
20	EU (Ireland)	Linux/UNIX	m3.2xlarge	0,6160	0,4187	3.668,00
21	EU (Frankfurt)	Windows	m3.medium	0,1460	0,0998	874,00
22	EU (Frankfurt)	Windows	m3.2xlarge	1,1690	0,7924	6.941,00
23	EU (Frankfurt)	Linux/UNIX	m3.medium	0,0830	0,0564	494,00
24	EU (Frankfurt)	Linux/UNIX	m3.2xlarge	0,6650	0,4522	3.961,00
25	Asia Pacific (Singapore)	Windows	m3.medium	0,1610	0,0973	852,00
26	Asia Pacific (Singapore)	Windows	m3.2xlarge	1,2880	0,7711	6.755,00
27	Asia Pacific (Singapore)	Linux/UNIX	m3.medium	0,0980	0,0578	506,00
28	Asia Pacific (Singapore)	Linux/UNIX	m3.2xlarge	0,7840	0,4623	4.050,00
29	Asia Pacific (Tokyo)	Windows	m3.medium	0,1510	0,0981	859,00
30	Asia Pacific (Tokyo)	Windows	m3.2xlarge	1,2060	0,7846	6.873,00
31	Asia Pacific (Tokyo)	Linux/UNIX	m3.medium	0,1010	0,0566	496,00
32	Asia Pacific (Tokyo)	Linux/UNIX	m3.2xlarge	0,8100	0,4589	4.020,00
33	Asia Pacific (Sydney)	Windows	m3.medium	0,1610	0,0973	852,00
34	Asia Pacific (Sydney)	Windows	m3.2xlarge	1,2880	0,7711	6.755,00
35	Asia Pacific (Sydney)	Linux/UNIX	m3.medium	0,0980	0,0578	506,00
36	Asia Pacific (Sydney)	Linux/UNIX	m3.2xlarge	0,7840	0,4623	4.050,00

For all our experiments, the variations among the runs were very small. Table 10 presents the performance report obtained from CPN Tools for Simulation 30 — Asia Pacific (Tokyo), OS Windows, and instance size m3.2xlarge, where one can observe standard variation and confidence intervals. We observed that for a 99% confidence level, the average savings were between \$1337.73 and \$1403.95, close to the average of \$1370.84. For this reason, we used 10 runs for each experiment.

Simulations were run using spot prices for December 2014, which were the latest prices available at the time we performed this experiment. Figures 12 and 13 summarize the simulation results. Savings varied from 0 to 70% depending on region, OS, and instance type. The only case where there were no savings was in the South American region for Linux/UNIX OS and instance type m3.2xlarge. This was because during the entire simulation period, spot prices were higher than on-demand prices (\$1.2240 for spot and \$0.7610 for on-demand). In this case, our model did not use spot instances and, hence, there were no savings. In most regions, higher savings were obtained for larger instances (m3.2xlarge) using Windows OS.

Table 10: Performance report for Experiment 30

Statistics							
Name	**Average**	**90% Half Length**	**95% Half Length**	**99% Half Length**	**Std. Dev.**	**Min**	**Max**
Cost							
avrg_iid	641.992290	1.049975	1.295714	1.861658	1.811409	640.220200	646.190700
Savings							
avrg_iid	1370.843310	18.676026	23.047010	33.113521	32.219738	1339.329800	1445.517300

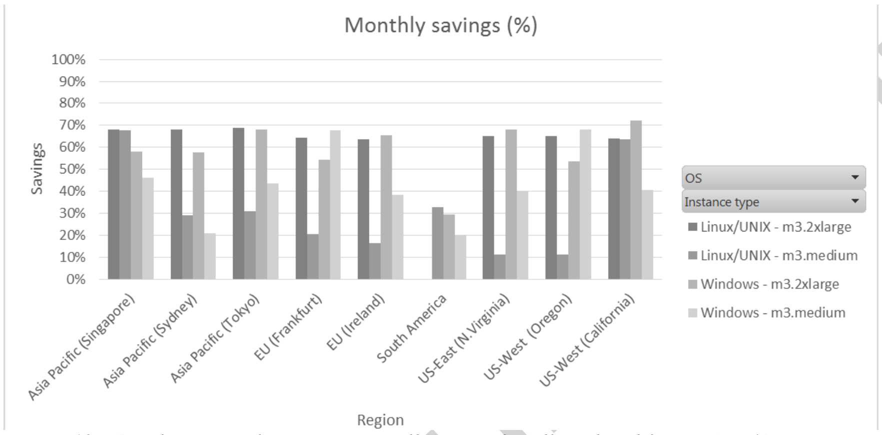


Figure 12: Simulation results – savings in all regions for all analyzed factors (in %).

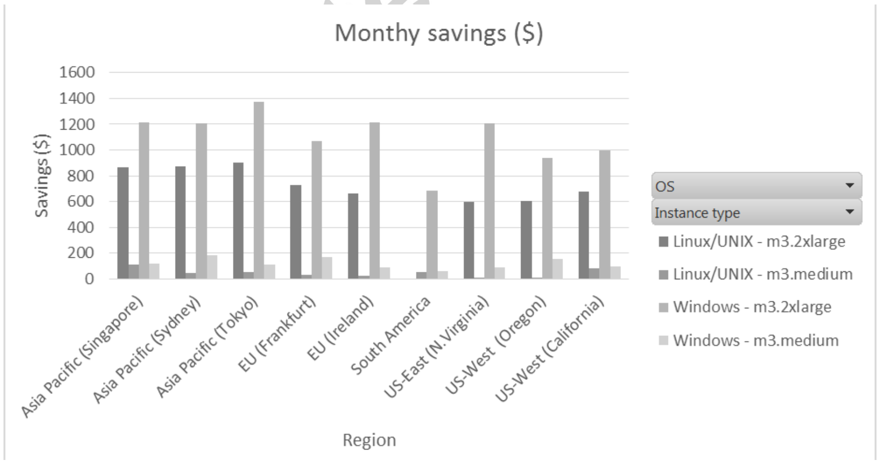


Figure 13: Monthly savings ($).

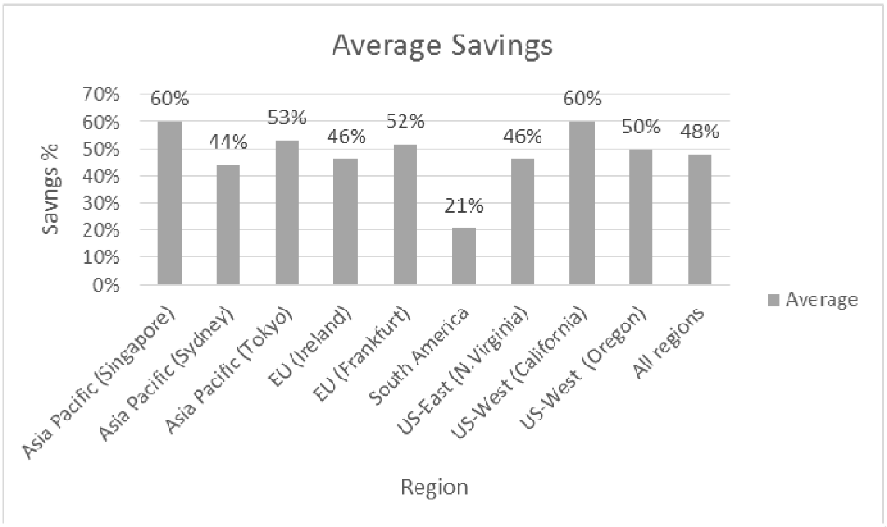
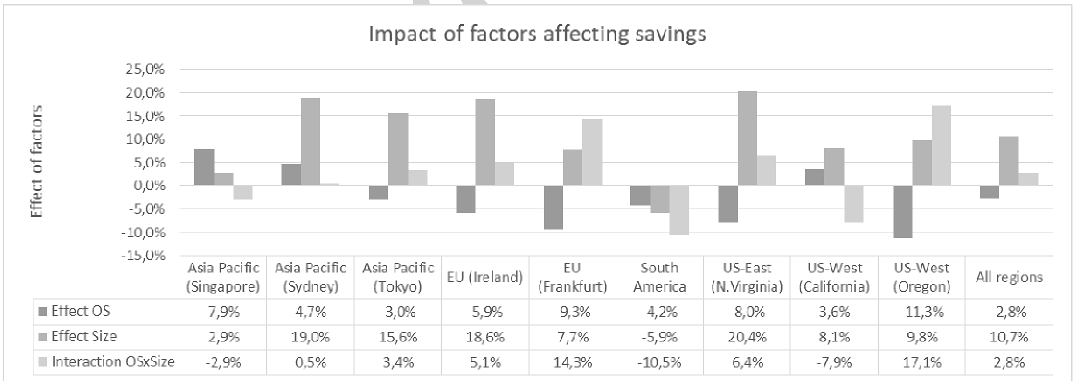


Figure 14: Average savings by region.

Figures 14 shows the 2^k factorial design in each region, including the average savings, and Figure 15 shows the impact of two factors, OS and instance size, on savings. Average savings were above 20% in all regions, and the global average was 48%. We observed a significant positive impact of factor size in most regions, since we obtained higher savings in larger instances. We observed a significant positive impact of factor size in most regions, since we obtained higher savings for larger instances. We also observed a negative impact of the operating system as a factor, since we obtained slightly lower savings on Linux in most regions. The interaction of the two factors produced a slightly positive impact in most regions. In some regions, however, we observed a high value of the effect of the interaction of OSxSize factors, as in the EU (Frankfurt), the Western United States (Oregon), and South America. In these regions, the variations in savings did not follow the most commonly observed pattern. In these regions, we did not observe a significant impact of any specific factor, and the interaction of factors accounted for a greater extent of the variation.



	Asia Pacific (Singapore)	Asia Pacific (Sydney)	Asia Pacific (Tokyo)	EU (Ireland)	EU (Frankfurt)	South America	US-East (N.Virginia)	US-West (California)	US-West (Oregon)	All regions
■ Effect OS	7,9%	4,7%	3,0%	5,9%	9,3%	4,2%	8,0%	3,6%	11,3%	2,8%
■ Effect Size	2,9%	19,0%	15,6%	18,6%	7,7%	-5,9%	20,4%	8,1%	9,8%	10,7%
■ Interaction OSxSize	-2,9%	0,5%	3,4%	5,1%	14,3%	-10,5%	6,4%	-7,9%	17,1%	2,8%

Figure 15: Impact of factors.

We found it interesting to compare the monthly cost of server use in a cloud by implementing our autoscaling policy using spot instances with that of an on-premise scenario. We used AWS charges for reserved instances as a rough estimate for on-premise server cost because these are similar in some situations . We kept in mind, however, that on-premise costs are highly dependent on the efficiency of local IT management. In the on-premise scenario, it was necessary to provide more servers than needed in order to handle the peak load. In our hypothetical *demand*() function, our load varied as a normal distribution with average $\mu = 3$ and standard deviation $\delta = 0.5$, which meant that 95% of values lay in the range $\mu \pm 2\delta$, i.e., during the day, the number of extra servers needed varied between two and four. In this manner, five servers would need to be provided, and would be idle for part of the time. Figure 16 shows our cost comparison for m3.medium instances running Linux.

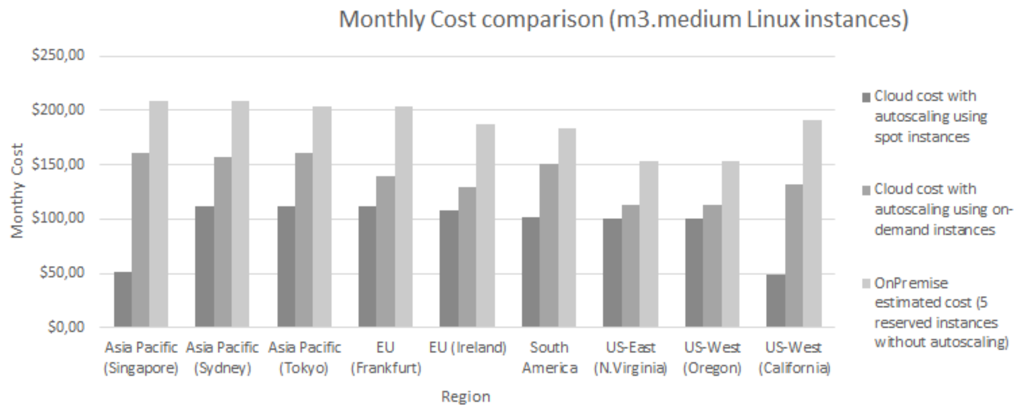


Figure 16: Cost comparison including OnPremises (estimated).

We observed that our estimated on-premise costs were always higher, as expected. However, the amount of savings varied with the policy implemented for using spot instances as well as spot market prices.

In summary, the simulations results lead us to the following conclusions:

- Spot instances are a good option to implement auto-scaling even in a small business scenario. This is not a typical application for spot instances advertised by the vendor, but the savings can reach up to 70% as advertised for other applications.
- There are considerable variation among regions. South America had the least savings, whereas in Asia Pacific regions saving were consistently high.
- Higher savings were obtained using larger instances (m3.2xlarge), however savings using smaller instances were also considerable in many regions.
- Impact of factors affecting savings did not point a specific factor as the main responsible for variation.

Finally, we found a significant difference in cost when taking advantage of possibilities of public clouds. Costs in public clouds may be fixed and easy to compute, by using reserved instances. However, this is never cost-effective unless the server is fully used 100% of the time. Business applications tend to have peak hours and idle periods. Thus, it is necessary to evaluate the scenario workload and plan for rational use of cloud services to obtain the highest economic benefits from cloud use. Implementing auto-scaling policies is the first step. Using spot instances wisely in the auto-scaling process is a step further.

7. Final considerations and future work

In this paper, we outlined a framework to evaluate the adoption of cloud services in comparison with that of on-premises solutions by using a BDIM approach. Since cost is a major factor influencing the decision to adopt cloud services, we also proposed a model for cost reduction in cloud services using spot instances, and carried out extensive simulations to investigate possible savings.

Our framework was intended to capture quantitative and qualitative factors, and to combine them in an organized manner to produce a final ranking of the solutions. We proposed a set of policies in auto-scaling that can help reduce the cost of cloud services by using spot instances and verified through extensive simulations that in many scenarios, these policies can lead to substantive savings of up to 60% more than those obtained from auto-scaling using on-demand instances, and even higher if compared with on-premise scenarios.

Preliminary studies indicated that security threats, performance uncertainty, and economic risks play an important role in the overall ranking of the solutions. Cost advantages must be sufficiently high to overcome these obstacles, or else it would be inefficient to adopt cloud services. We also concluded that using spot instances in auto-scaling significantly reduces the cost of cloud services, and thus can facilitate their adoption.

In future research, we plan to complete the validity of our decision making model as well as the cost reduction model using spot instances. To this end, we might need to develop an automated tool to gather feedback from a wider audience. Evaluating the impact of spot termination on the quality of service is also part of our research roadmap. Extending the framework to handle SLA requirements would be interesting as well. For instance, the cost in terms of material and human resources can differ significantly between a 99.9% or 99.99% availability of an application according to the SLA, especially in the on-premises scenario over a five-year period. Another important extension to the framework would be handling risks in detail to enhance qualitative evaluation.

8. References

Alfonso, C, et al. An economic and energy-aware analysis of the viability of outsourcing cluster computing to a cloud. *Future Gener Comput Syst* 2013; 29(3): 704-712.

Altmann, J., Kashef, M. Cost model based service placement in federated hybrid clouds. *Future Gener Comput Syst* 2014; 41: 79-90.

Amazon. *Amazon EC2 Spot Instances*. Retrieved from Amazon Web Services: http://aws.amazon.com/ec2/purchasing-options/spot-instances. Accessed in December, 2014.

Amazon Web Services. (2014, December). *Amazon EC2 Reserved Instances*. Retrieved from Amazon EC2 pricing: http://aws.amazon.com/ec2/purchasing-options/reserved-instances/?nc2=h_ls

Baars, T. et al. Chargeback for cloud services. *Future Gener Comput Syst* 2014; 41: 91-103, http://dx.doi.org/10.1016/j.future.2014.08.002.

Bayrak, T. A decision framework for SME Information Technology (IT) managers: Factors for evaluating whether to outsource internal applications to Application Service. *Technol Soc* 2013; 35(1): 14-21.

Benlian, A., Hess, T. Opportunities and risks of software-as-a-service: Findings from a survey of IT executives. *Decis Support Syst* 2011; 52:232-246.

Bibi, S., Katsaros, D., Bozanis, P. Business Application Acquisition: On-Premise or SaaS-Based Solutions? *IEEE Software* 2012; 29(3): 86-93.

Boampong, P., Wahsheh, L. Different Facets of Security in the Cloud. *Proceedings of the 15th Communications and Networking Simulation Symposium*, 2012.

CPN Group. *CPN Tools*. Retrieved from http://cpntools.org/.Accessed in November 2013.

E-fiscal. *Computing e-Infrastructure cost estimation and analysis - Pricing and Business Models* 2013. Retrieved from Financial Study for Sustainable Computing e-Infrastructures: http://www.efiscal.eu/files/deliverables/D2%203%20Computing%20e-Infrastructure%20cost%20calculations%20and%20business%20_models_vam1-final.pdf

Furtado Junior, C. G., Soares, J. M., Barroso, G. C. A web cache simulator tool based on coloured Petri nets and Java programming. *Revista IEEE America Latina* 2015; 13.

Garg, S., Versteeg, S., Buyya, R. A framework for ranking of cloud computing services. *Future Gener Comput Syst* 2013; 29: 1012-1023.

Gartner Group. *Magic Quadrant for Cloud Infrastructure as a Service*. Retrieved from http://www.gartner.com/technology/reprints.do?id=1-1UKQQA6&ct=140528&st=sb. Accessed in December, 2014.

Gupta, P., Seetharamana, A., Raj, J. The usage and adoption of cloud computing by small and medium businesses. *Int J Inf Manag* 2013; 33: 861-874.

Haas, R., Meixner, O. *An Illustrated Guide to the Analytic Hierarchy Process*. Retrieved from http://alaskafisheries.noaa. gov/sustainablefisheries/sslmc/july-06/ahptutorial.pdf, Accessed in November 2013

Han, H. et al. Cashing in on the Cache in the Cloud. *IEEE Trans Parallel Distrib Syst* 2012; 23 (1387-1399).

Han, R. et al. Enabling cost-aware and adaptive elasticity of multi-tier cloud applications. *Future Gener Comput Syst* 2014; 32 (82-98).

Ho, W., Xu, X., Dey, P. K. Multi-criteria decision making approaches for supplier evaluation and selection: A literature review. *European Journal of Operational Research* 2010; 202: 16-24.

Jain, R. *Art of Computer Systems Performance Analysis Techniques For Experimental Design Measurements Simulation And Modeling.* Wiley Computer Publishing; 1991.

Javadi, B. et al. Characterizing spot price dynamics in public cloud environments. *Future Gener Comput Syst* 2013; 29 (4), 988-999.

Jensen, K., Kristensen, L. *Coloured Petri Nets — Modelling and Validation of Concurrent Systems,* Springer 2009.

Kaplan, R., Norton, D. *The Balanced ScoredCard.* Harvard Business Review Press 1996.

Lee, S. et al. Drivers and Inhibitors of SaaS adoption in Korea. *Int J Inf Manag* 2013; 33(3): 429-440.

Lee, S., Park, S., Lim, G. Using balanced scorecards for the evaluation of “Software-as-a-service.” *Inf Manag* 2013; 50(7):553-561.

Malawski, M. et al. Cost minimization for computational applications on hybrid cloud infrastructures. *Future Gener Comput Syst* 2013; 29(7):1786-1794.

Manvi, S. S., Shyam, G. K. Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey. *J Netw Comput Appl* 2014; 424-440.

Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J. Cloud computing — The business perspective. *Decis Support Syst* 2011; 51:176-189.

McGougha, A.S. et al. Comparison of a cost-effective virtual Cloud cluster with an existing campus cluster. *Future Gener Comput Syst* 2014;41:65-78.

Merig, J. M., Gil-Lafuente, A. M. New decision-making techniques and their application in the selection of financial products. *Inf Sci* 2010; 180:2085-2094.

Mouratidisa, H., Islam, S., Kalloniatis, C., Gritzal. A framework to support selection of cloud providers based on security and privacy requirements. *J Syst Softw* 2013;86 (2276-2293).

NIST - National Institute of Standards and Technology. *NIST Definition of cloud computing.* Gaithersburg, MD 2009.

Peterson, J.L. *Petri Net Theory and the Modelling of SYSTEMs.* Prentice Hall 1981.

Ribas, M. et al. Assessing cloud computing SaaS adoption for enterprise applications using a Petri net MCDM framework. Proceedings of the 9th workshop on business-driven IT management (BDIM 2014) , *IEEE NOMS* 2014; 1-6.

Rohitratana, J., Altmann, J. Impact of pricing schemes on a market for Software-as-a-Service and perpetual software. *Future Gener Comput Syst* 2012; 28 (1328-1339)

Sauvé, J., Moura, A., Sampaio, A., Jornada, J. An Introductory Overview and Survey of Business-Driven IT Management. *First IEEE/IFIP BDIM* 2006; 1-10.

Sousa, E. et al. A modeling approach for cloud infrastructure planning considering dependability and cost requirements. *IEEE Trans Syst, Man, Cybern, Syst* 2014; 99.

Sripanidkulchai, K., Sujichantararat, S. A Business-Driven Framework for Evaluating Cloud Computing. *Proceedings of 2012 IEEE/IFIP 7th Workshop on Business Driven IT Management* 2012.

Tamanini, I., Pinheiro, P., Santos, M. An hybrid approach of verbal decision analysis and machine learning. *Lect Notes Artif Int* 2012, 7413: 126-131.

Tang, S. et al. A framework for Amazon EC2 bidding strategy under SLA constraints. *IEEE Trans Parallel Distrib Syst* 2014; 25(2-11)

The Yankee Group. *Understanding Total Cost of Ownership of a Hosted vs. Premises-Based CRM Solution* 2013. Retrieved from http://www.avecon.gr/photos/research/yankee%20group_undestanding%20tco%20of%20a%20hosted%20vs.%20premises-based%20csm%20 solution.pdf.

Triantaphyllou, E. Multi-Criteria Decision Making Methods: A Comparative Study. Kluwer Academic Publishers. Available at: http://books.google.com.br/books?id=tuPGe ur-TYC.

Wu, W. (a) Developing an explorative model for SaaS adoption. *Expert Syst Appl* 2011; 38(12): 15057-15064.

Wu, W. (b) Mining significant factors affecting the adoption of SaaS using the rough set approach. *J Syst Softw* 2011; 84(3):435-441.

Wu, W., Lan, L., Lee, Y. Exploring decisive factors affecting an organization’s SaaS adoption: A case study. *Int J Inf Manag* 2011; 31(6), 556–563.

Yuen, K. Software-as-a-Service Evaluation in Cloud Paradigm: Primitive Cognitive Network Process Approach. *Proceedings of the 2012 IEEE International Conference on Signal Processing, Communication and Computing* 2012.

Zardari , S., Bahsoon, R. Cloud Adoption: A Goal-Oriented Requirements Engineering Approach. *Proceedings of ACM SECLOUD* 2011.

Appendix – Petri nets and MCDM

Petri net theory originated in the early work of Carl Adam Petri and has evolved into a useful tool for modeling systems. Analysis of the Petri net can reveal important information concerning the structure and dynamic behavior of the modeled system (Peterson, 1981).

A Petri net structure C is a 4-tuple C = (P, T, I, O). P = $\{p_1, p_2, \ldots, p_n\}$ is a finite set of places, where n >= 0, and T = $\{t_1, t_2, \ldots, t_m\}$ is a finite set of transitions, where m >= 0. The set of places and the set of transitions are disjoint, $P \cap T = \emptyset$. I: $T \rightarrow P^\infty$ is the input function, a mapping from transitions to bags of places, and O: $T \rightarrow P^\infty$ is the output function, also a mapping from transitions to bags of places. A graphical representation of a Petri net structure is useful for illustrating the concepts of the Petri net theory. A Petri net graph G is a bi-partite directed multi-graph, G = (V, A), where V = $\{v_1, v_2, \ldots, v_s\}$ is a set of vertices and A = $\{a_1, a_2, \ldots, a_r\}$ is a bag of directed arcs, $a_i = (v_j, v_k)$, with $v_j, v_k \in V$. The set V can be partitioned into two disjoint sets, P and T, such that $V = P \cup T$, $P \cap T = \emptyset$ and, for each directed arc $a_i \in A$, if $a_i = (v_j, v_k)$, then either $v_j \in P$ and $v_k \in T$, or $v_j \in T$ and $v_k \in P$. A marking μ of a Petri net C = (P, T, I, O) is a function μ: P → N from the set of places P to non-negative integers N (Peterson, 1981).

Colored Petri net (CP-net or CPN) is a graphical language for constructing models. CPN is a discrete-event modeling language combining the capabilities of Petri nets with those of a high-level programming language. The CPN ML programming language, based on the functional programming language Standard ML, provides primitives for the definition of data types, describing data manipulation, and creating compact and parameterizable models .

CPN Tools is a tool for editing, simulating, and analyzing colored Petri nets. CPN tools offer statistical functions that can be used to design stochastic Petri nets. We used CPN Tools to design hierarchical Petri nets in our model. CPN Tools also supports the inclusion of timing information to the model.

The problem of making decisions using several criteria has led to many proposed approaches. MCDM methods can be divided into ones based on Multi-attribute Utility Theory (MAUT) and those based on outranking. The most common MAUT methods are the weighted sum model (WSM), the weighted product model (WPM), and the analytic hierarchy process (AHP). The most common outranking methods are the Elimination Et Choix Traduisant la Realité (ELECTRE) method and the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) method . The WSM method is most commonly used due to its simplicity. However, it applies only when the additive utility assumption applies.. Only criteria that use the same units and scale can be combined. This limits the method severely, although many authors flout this restriction and use WSMs anyway. The WPM method combines criteria by multiplying ratios of metrics for several alternatives being compared. This division of metrics with the same unit cancels out the unit and, for this reason, is called dimensionless analysis. AHP decomposes the decision problem into a hierarchy of criteria and alternatives, and uses pairwise comparisons to express the relative importance of one criterion over another. Using these comparisons, it is possible to build pairwise matrices and calculate the eigenvector in order to rank the criteria. AHP can combine qualitative and quantitative criteria and is commonly used.

AHP is similar to WPM in that it uses ratios of metrics. It is therefore easy to combine criteria that use different units or scales, since only relative values are used to compare two alternatives according to a certain criterion. However, AHP differs from WPM in several important ways. It easily deals with hierarchies of criteria, employs a nine-point scale easily used and understood by decision makers, and allows a check to be performed to identify inconsistent pairwise comparisons between alternatives. Outranking works as follows: Alternative A outranks B if, for a large number of criteria, A performs at least as well as B (concordance condition), while its worse performance is still acceptable for the other criteria (non-discordance condition). After determining, for each pair of alternatives, whether one alternative outranks the other, these pairwise outranking assessments can be combined into a partial or complete ranking of alternatives. An alternative is said to be dominated if there is another alternative that excels it in one or more attributes and equals it in the remaining attributes.

The ELECTRE method uses outranking. With outranking, even when an alternative A does not quantitatively dominate an alternative B, the decision maker may still take the risk of regarding A as almost certainly better than B. The basic concept underlying TOPSIS is that the selected alternative should be the closest to an ideal solution and the farthest from the negative-ideal solution. Euclidean distance is commonly used.

Merig and Gil-Lafuente (2010) developed an approach that used the ordered weighted averaging (OWA) operator in the selection of financial products. Their proposed aggregation operators are useful for decision-making problems because they compare an ideal alternative with available options in order to find the optimal choice. Ho, Xu, and Dey (2010) reviewed literature on multi-criteria decision-making approaches for supplier evaluation and selection. Their results provided evidence that multi-criteria decision-making approaches are better than the traditional cost-based approach. AHP was found to be the most popular integrated approach. The choice of multi-criteria methodology to support the decision process depends directly on the issue in question. Tamanini et al. (2012) claimed that the choice of a method should be the result of an evaluation of the chosen parameters, the type and accuracy of the data, and the decision maker's manner of thinking and his/her knowledge of the problem.

Yuen (2012) used a variation of the AHP method, called the "primitive cognitive network process" (P-CNP) that revised the AHP approach through practical changes. Yuen recommended a pairwise opposite matrix as the ideal alternative to a pairwise reciprocal matrix, as in AHP. However, AHP is largely used in the literature to assess cloud services (Garg, Versteeg, & Buyya, 2013; Yuen, 2012), and was thus selected as the modified condition/decision coverage (MCDC) in our framework.

Highlights

1. A Petri net-based multi-criteria decision making framework to assess a cloud service against a similar on-premises service.
2. Our framework helps IT managers choose between two such options, and can be used for any type of cloud service.
3. We also propose a Petri net to model cost savings using the spot instances purchasing option in public clouds.
4. Simulations showed that spot instances present a promising cost-saving option in the auto-scaling process, even for simple business applications using few servers.