QoS-Aware Scale Up on IaaS Clouds

Luis Fernando Orleans
Computer Science Department
Universidade Federal Rural do Rio de Janeiro
Rio de Janeiro, Brazil
Email: lforleans@ufrrj.br

Geraldo Zimbrão da Silva
Computer Science Department
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brazil
Email: zimbrao@cos.ufrj.br

***Abstract*—For systems hosted in IaaS clouds that target profit, like blogs and e-commerce systems, the final revenue should be the most important metric. Balancing the QoS experienced by clients as a manner to avoid their drop-out against the cost related to leasing instances forms the basis of a good instance management for those scenarios. In this paper, we demonstrate the feasibility of using a Fuzzy Logic Inference System as a tool for maintaining the QoS. Furthermore, it was created a method for reducing the overall cost related to instances leasing using an incremental algorithm, acquiring one computational unit at a time. Finally, we used hooks to handle unpredictable burst of requests, called here as noises. Our experiments evidenced that those methods can keep the response time of all requests below a deadline, avoiding customer dissatisfaction. At the same time, the total cost of servers lease is reduced, even when the cost-benefit among different configurations is not linear.**

Index Terms*—*scale up;PROFUSE;IaaS clouds

I. Introduction

Elasticity is one of the key foundations of Cloud Computing [1] [2]. The ability to rapidly increase the number of resources without the need of service stopping/restarting plus its pay-per-usage nature opened room for a great number of proposals for minimizing both requests response time and costs with instance leasing [3] [4] [5] [6].

Elasticity became particularly important when Quality of Service (QoS) has turned into a crucial requirement for internet-based business models. If a customer is willing to purchase a product or a service, he or she expects the best treatment possible, which can be partially translated as not having to wait too long for his/her requests to be processed. In fact, [7] states that customers' patience lasts 4 seconds in average for each request. According to that study, when requests takes longer than 4 seconds to be processed a phenomena called customer drop-out emerges [8], i.e., clients begin to abandon the system unsatisfied. Guaranteeing QoS is important for avoiding customer frustration and the consequent revenue loss. Several studies have been done in that direction [5] [6] [9] [10], most of them using some mathematical modeling and targeting efficient workflow execution or minimizing the number of available instances for reducing the power consumption.

In this paper, we propose a novel *elasticity model* (em), called PROFUSE, that uses a Fuzzy Logic Inference System to calculate the necessary computing power needed to keep the QoS for requests processing.

A. Problem characterization

A cloud provider (*cp*) offers virtualized instances for e-commerce system providers (*sp*) to lease. Those instances can be of one out of a total of t different configurations, having each configuration a specific cost per hour of rental. Also, client c uses the system maintained by *sp*.

Instances already leased by *sp* form its *instances array*. The cloud provider can limit the maximum size of the array, hence forcing the client who wants to increase overall system computational power to acquire more expensive types of instances. The strategy a client uses to (re)lease an instance forms its elasticity model *em*. Also, *cp* provides an API that *sp* can use for acquiring new instances, opening room for a dedicated middleware that automates *em*.

The system maintained by *sp* works as follows: while surfing through the application, c issues several requests (product search, other customers comments about a product, providing credit card details, etc.). An incoming request is processed on an idle instance. If all instances are busy upon its arrival, the request is sent to a First Come First Serve (FCFS) queue. Also, requests response times should be kept under a threshold (a *deadline*) otherwise c becomes unsatisfied and leaves the system.

The Client Conversion Rate *ccr* is known and represents the percentage of system visits that actually become purchases and is calculated as $\frac{\#purchases}{\#visits}$. On a visit, c issues n requests in average until he or she leaves the system, regardless of whether a purchase was made. In average, each purchase generates a revenue *rb*. Hence, the mean value v of a request can be computed as

$$\overline{v} = \begin{cases} \overline{rb} \times \overline{ccr} \times \frac{1}{n} & \text{if deadline not reached} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Another way to calculate $\overline{v}$ is

$$\overline{v} = \overline{rb} \times \overline{ccr} \times \frac{1}{n} \times q \tag{2}$$

where q is the probability of missing the deadline for that request.

Consider that the system has h visits per day in average and each visit contains in average r requests, the mean day revenue can be computed as

$$\overline{rb}_{day} = \overline{h} \times \overline{r} \times \overline{v}. \quad (3)$$

Another variable that might affect daily revenue is the cost i related to instances leasing per hour, known as instance-hour. The relationship between q and i occurs as to increase the probability of processing requests within the deadline is sometimes necessary to increase the amount of computational power (instances) in the array. Hence, our goal is to minimize i without affecting q. A possible solution relies on Queueing Theory [11] where the following metrics are important: mean requests arrival rate (λ), mean requests processing time (β) and mean requests queue size (δ). The first two can be used to compute the mean system utilization (ρ) as

$$\rho = \lambda \times \beta. \quad (4)$$

Note that $1/\beta$ is the mean number of processed request per time unit. In this paper, β is the mean time a request takes to be processed by 1 instance core. Thus, (4) becomes

$$\rho = \lambda \times \beta \times \frac{1}{s} \quad (5)$$

s being the total number of cores contained in the instances array. An instance may have 1, 2 or 4 cores each and each configuration has its own price. The computational unit cost is achieved by dividing the instance hour price by the respective number of cores (the computational unit used in this work) of the configuration. A *non-linear* cost-benefit occurs when the computational unit cost varies for different configurations.

Finally, with probability *z*, an unpredictable burst of requests can increase λ in Z_p percent. Those bursts last $\overline{t}$ seconds in average and are referred to in this work as *noises*.

B. Contributions

The main contributions of this paper are:

1) A detailed guide for building a Fuzzy Logic-based Inference System that can predict workload changes and detect variations;
2) An efficient strategy for lease instances that aims to minimize the related cost;
3) A strategy that rapidly detects noises on the workload preventing deadline miss rate (DMR) increases.

C. Paper organization

The remainder of this paper is structured as follows: Section II lists the related work, while Sections III, IV and V present the several proposals of this paper. Section VI lists the experimental setup, the obtained results and discusses them thoroughly. Finally, Section VII lists the conclusions and points for future directions.

II. Related Work

Due to its novelty, scale up automation for systems hosted on IaaS clouds is an open problem, having several researches been done in that field.

In [3], authors consider geographically distributed datacenters and each hosted system having its own SLA – which establishes a maximum percentile of unattended deadlines. A dynamic ranking algorithm that identifies the most valuable requests, a gi-FIFO scheduling system and a heuristic-based task placement algorithm are presented.

Also, [12] modeled the problem of scale up as a predictive stochastic problem. The proposed approach explores the trade-off between QoS and servers lease cost by categorizing instances according to their configurations/costs and creating a cost function that is minimized using a customized Convex Optimization Solver algorithm, invoked periodically. Meanwhile, the research done in [13] proposes a scheduling algorithm based on jobs hierarchy. Such algorithm differentiates requests tied to an SLA from requests that do not have time-constraints and prioritize the processing of the former kind.

IBM presented the SmartScale tool in [4]. That work proposes a combination of horizontal and vertical scale up flavors to ensure the system is using the most affordable configuration and idle resources. It uses Decision Trees to periodically determine what have to be altered in the array and keep QoS constant.

Compared to this work, none of the previous mentioned researches comprises workload noises, neither did they used Fuzzy Logic to predict workload changes. Also, we focused solely on horizontal scale up as both [1] and [14] states that vertical scale up causes a momentary performance loss.

III. Predictable scale up

For predicting workload changes, a Fuzzy Logic Inference System (FLIS) [15] was created. We chose a FLIS over other inference mechanisms because it uses a set of IF-THEN rules which allows adjustments made by specialists. Prior to FLIS creation, a requests arrival histogram (RAH) is needed - it can be an estimate for systems that are not in production yet. Such histogram describes the number of requests that arrives per time unit (in this work we used *hour* as time unit, though it could be minute, second, etc.). The number of bars presented on the histogram represents the *time window* (a day, a week, a month, a year, or even longer periods). After RAH creation, the difference of workload (DoW) can be easily calculated from a time unit to the next through the simple formula:

$$DoW_h = DoW_{h+1} - DoW_h \quad (6)$$

where h is the current time bar on the histogram and *h+1* is the next one.

A. Linguistic variables definition

In order to determine the linguistic variables should be used in the FLIS, we used the following process: the RAH was used as input for various rounds of simulations, where some system variables were periodically logged: *requests arrival rate* (RAR), *requests processing rate* (RPR), *mean queue waiting time* (MQWT), *queue size* (QS), *system utilization* (SU), *deadline miss rate* (DMR). The log file served as input for an attribute selection/reduction analysis, performed on the WEKA software [16], which is a tool used by Data Mining professionals mostly because it has many Machine Learning algorithms implemented in it. Afterwards, the following attributes remained: SU, DMR and QS. Note that those variables can provide the FLIS its *reactive* behaviour only. As the *proactive* characteristic of the FLIS, we added two more variables: DoW and *time to next interval* (TTN), where the last stands for the number of time units remaining until the next bar of the histogram is reached.

The output variable was defined as the *number of computational power units* (NCPU). In this work we considered a computational power unit as a computing core, i.e. the number of cores denotes the computing capacity of an instance.

Finally, also using the WEKA software, we performed a Cluster Analysis [17] to determine the initial number fuzzy regions for each linguistic variable, a similar step to that performed by Google to define task placement strategies [18]. Near clusters were combined to reduce the total number of rules.

IV. Unpredictable scale up

The FLIS inside PROFUSE mechanism predicts and reacts well to workloads that are similar to previous ones, notably those that were used to build the RAH. However, web systems can incur into some situations where the difference on the expected number of requests and the actual number of requests is very high. In these scenarios, FLIS react speed may not be fast enough for prevent increasing on the deadline miss rate. As an example, consider a promotion widely spread on social networks made by a sales web system. The requests arrival rate explodes as the promotion announcement gets deeper into the social networks and people get interested on it. The miss rate increase is faster than FLIS feedback and should be detected separately. In this work, those abrupt and unexpected changes on workloads are denoted as *noises* and the noise detection system is called *hook*. Please note that noises are unusual and unpredictable events which severely affect workloads. However they do not last long, which means that once they are gone workloads return to previous states and FLIS use is effective again.

Essentially, a hook is a monitoring component and keeps critical units (e.g. miss rate, queue size, etc.) under close surveillance. Whenever one of those units behaves unexpectedly the expansion routine is called and the system is put back into a consistent state. The algorithm described on Figure 1 details how PROFUSE uses hooks.

V. Leasing policy

From a revenue-centric perspective, only instances with the cheapest configuration should fill in the array. However, such a strategy limits the computational power when cloud providers

```
procedure HOOK_MONITORING
    for all incoming request do
        $su$ ←current_system_utilization()
        $qs$ ←current_queue_size()
        $mr$ ←current_miss_rate()
        if $su$, $qs$, $mr$ exceeds threshold then
            $expand()$
        end if
    end for
end procedure
```

Fig. 1. Hooks monitoring algorithm

limit the number of instances and causes DMR to increase as the requests arrival rate increases. In order to assess the DMR impact, the cost of processing a request on each server configuration (t_x) should be added to (2).

Consider ih_x as the instance-hour of a configuration *t* which can process *tr* requests per second. Hence, t_x can be calculated as:

$$t_x = \frac{ih_x}{tr \times 3600} \tag{7}$$

Equation (7) divides the instance-hour cost of configuration *t* by the total number of requests that *t* can handle in one hour. Therefore, equation 2 can be rewritten taking into consideration the cost of processing a request using configuration *t*

$$\overline{v} = \overline{rb} \times \overline{ccr} \times \frac{1}{n} \times q - t_x \tag{8}$$

Equation (8) computes the aggregated value of a request being processed on an instance with configuration *t* and client remains on the system with probability *q*.

Finally, in order to determine whether is more profitable to process the incoming request on an instance with configuration x_0 or with configuration x_1, where x_1 is more expensive and has twice computing power than x_0, the outcome of $v_{x1} \geq v_{x0}$ should be evaluated. Thus, when

$$q \geq \left(\frac{2 \times [c_{x1} - c_{x0}]}{\overline{rb} \times \overline{ccr} \times \frac{1}{n}} \right) \tag{9}$$

is worth the swapping. Note that i_1 has twice the computing power of i_0, hence the ”2×” on (9)

Focusing on system provider financial loss reduction, PROFUSE's instance allocation works as follows: consider an IaaS cloud provider that limits the number of instances *sp* can lease on *MAX* instances. Each instance is of a configuration *c* and each configuration has an associated cost per hour. Therefore, each instance is represented as i_{nk}, where *n* is the position in the array ($1 \geq n \geq MAX$) and *k* is its configuration ($1 \geq k \geq t$). Hence, the initial array of instances can be represented as

$$a = \{i_{11}, i_{21}, ..., i_{n1}\} \tag{10}$$

```
1: procedure EXPANSION
2:     Start with an instance array with the minimal configuration ($k = 1$)
3:     While size(array) <max - 1 acquire instances with minimal configuration
4:     From that point onwards, use formula 9 to decide whether or not to lease an instance with a superior configuration. In case of swapping:
5:     Start an immediately superior configuration instance.
6:     Release an instance with the current configuration using 9
7:     If the array contains only instances of the configuration $k = t$, start a new instance of configuration $t$
8: end procedure
```

Fig. 2. Expansion algorithm

```
1: procedure INSTANCE_RELEASE($k$)
2:     for all instances $i$ in array do
3:         if $i$ is of type $k$ then
4:             Compute remaining time $rt$ until next instance-hour
5:         end if
6:     end for
7:     Release instance with the least $rt$
8: end procedure
```

Fig. 3. Instance release algorithm

In our approach, the initial array has only instances of the most basic type. For each positive outcome Δ computed by either FLIS or Hooks monitoring system, the leasing module keeps acquiring instances of type $k = 1$ until size of array reaches MAX - 1. Note that in case of *cp* does not impose an array size threshold, PROFUSE will lease only instances of that configuration. The remaining spot in the array is used for swapping instances when additional computing power is needed. Instance swapping consists on leasing an instance with a superior configuration and releasing a smaller instance – needed for future swaps.

Worth mentioning instance swapping only occurs when the outcome of (9) is positive, which indicates the $DMR \times profit$ balance was unfavorable. All instance swaps keep a free spot on the array except on the case the array contains only instances of configuration $k = t$ – when another instance of type *c* is acquired. Figure 2 describes the expansion algorithm.

Similarly, Figure 3 details release algorithm, taking into consideration the *release opportunity*, i.e. the instance closest to increase its cost.

In contrast to expansion algorithm, on the shrink algorithm (Figure 4) an instance of a more expensive configuration is released prior acquiring an instance with a simpler configuration.

```
1: procedure SHRINK
2:     if array is full then                ▷ All instances are of type $t$
3:         Release an instance using algorithm ( 3)
4:     else
5:         Find configuration $conf = MAX(k)$
6:         if $conf = 0$ then ▷ Only small instances in array
7:             Release an instance using algorithm ( 3)
8:         else
9:             Start an instance with configuration $conf - 1$
10:            Release an instance using algorithm ( 3)
11:        end if
12:    end if
13: end procedure
```

Fig. 4. Array shrinking algorithm

VI. EXPERIMENTS

In order to assess the robustness of PROFUSE, we conducted an exhaustive set of experiments using a simulator that was built for easily switch among a plethora of environments.

A. Workload types

For defining workloads shapes we used the study presented in [10]. According to the authors, four kinds of workloads are typical for systems hosted in IaaS clouds: (i) *stable*, where requests arrival rate is almost linear; (ii) *normal*, presenting the occurrence of peak situations; (iii) *growing*, where the number of incoming requests does not decrease over time; and (iv) *on-and-off*, representing some background, administrative tasks such as log archiving and compacting. Note that those workloads shapes can be combined and represent different epochs of the same system through time. Since the objective of this work is to find a new elasticity model capable of handle expected and unexpected burst of requests, the experiments were conducted using normal and growing workload shapes. The former was extracted from a real system whereas the later is a synthetic workload.

B. Parameters

Other parameters used on our experiments were extracted from [19] and are shown in Table I (comma-separated values indicates more than 1 value was used).

C. Environments

The experiments were conducted starting from the most basic scenario and then introducing limitations one at a time. To facilitate referencing the environments, letters were assigned as follows:

(A) Unlimited instances, workload without noises and hooks system deactivated;
(B) Limited number of instances, no noises and hooks system deactivated;
(C) Limited number of instances, noises and hooks system deactivated;
(D) Limited number of instances, noises and hooks system activated.

TABLE I
SIMULATOR PARAMETERS

first	second
Initial instances array size	5
Instance leasing mean time	97s
Instance releasing mean time	8s
Instances array max size	∞, 20
Single-core instance-hour cost	\$0.02
Dual-core instance-hour cost	\$0.34
Quad-core instance-hour cost	\$2.00
Deadline	4s
Client conversion rate	0.01, 1
Request value	\$100, \$0.001
Request-to-revenue probability	0.05, 1
Mean-time between FLIS feedback	120s

TABLE II
DMR PROFUSE WITH NORMAL WORLOAD

Environment	DMR
A	0.00014
B	0.00012
C	8.58581
D	0.00000

Note that environment (A) reflects a linear cost-benefit between instances configuration, since all computational units have the same cost.

PROFUSE's performance was compared against a basic elasticity model, that aims to keep the system utilization constant. Such EM is called Fixed Utilization and works by periodically (using same feedback interval used by PROFUSE) gathering data and (re)leasing instances in order to keep system utilization 0.7. Finally, all presented results are the mean value obtained out of 10 simulation rounds.

D. Results

Tables II and III compare PROFUSE's performances for all environments. Note that environment (B) has a slightly better performance over environment (A) because the boot time of new instances – the more instances leased, greater is the time needed to make them available. At environment (B) when the computational capacity should be increased in 4 units, a quad-core instance can be leased. On the other hand, at environment (A) the system has to wait 4 instances to boot up, with small fluctuations on their boot time.

When environment (C) is used there is a strong performance drop: approximately 8.6% and 10.8% of DMR for normal and growing workloads, respectively. Such a poor performance was expected since a FLIS is incapable of detecting noises and react to them efficiently. However, when hooks are turned on (environment(D)), PROFUSE presents an acceptable performance – there were no deadlines misses with the normal workload and only 0.53% of misses with the growing workload. Worth mention that PROFUSE's FLIS was using an *aggressive* configuration, trying to keep a system utilization of 0.8 in average. When we changed to a *conservative* approach and targeted system utilization to 0.6 in the FLIS, there were no misses at all (last line of Table III).

TABLE III
DMR PROFUSE WITH GROWING WORLOAD

Environment	DMR
A	0.01928
B	0.01691
C	10.79001
D	0.52795
D Conservative	0.00000

TABLE IV
ELASTICITY MODELS COMPARISON

EM	Env.A	Env.C	Env.D
FU	2.57830	97.98622	5.47310
PROFUSE	0.00014	8.58581	0,00000

For comparisons purpose, Table IV shows DMR of Fixed Utilization and PROFUSE elasticity models when submitted to the normal workload on environments (A), (C) and (D). As expected, PROFUSE outperforms FU model on all scenarios, being a more secure choice for guaranteeing QoS constraints.

Using equation 3, the total revenue loss can be calculated. Table V presents the results for both scenarios considered here: e-commerce systems and blogs (see section I-A). PROFUSE's robustness is confirmed on those results, such as the usefulness of hooks to handle workload noises.

Client conversion rate was set to 1% which, as suggested in [20]. Also, the minimum number of clicks needed to purchase an item is 7 (initial, item search, add to cart, initiate check-out, provide username and password or register, insert payment data, confirm purchase). To simulate a more real scenario, where users search other items, read opinions, etc., we assumed purchases are done after 20 requests in average. Finally, the mean value for each purchase was \$100.00. Revenue loss on a blog can be calculated in a simpler way, as all requests generate an income (\$0.001).

PROFUSE's instance rental efficiency was compared against both the cheapest case (20 single-core instances, fixed number) and the most expensive case (20 quad-core instances, fixed number), comprising lower and upper bounds. Table VI shows the cumulative cost (revenue loss plus cost with instances rental) for each strategy, where SC, QC, PL and PNL stands for Single-Core, Quad-Core, PROFUSE-Linear and PROFUSE-Non-Linear, respectively. From the 10th hour onwards, the single-core only strategy is incapable of maintain the agreed QoS becoming the most expensive configuration. From the results, it becomes clear that PROFUSE is a cheaper alternative than resource overprovisioning, here denoted as the quad-core only configuration.

VII. CONCLUSIONS AND FUTURE WORKS

IaaS cloud hosted systems administrators usually face the problem of deciding the computational power needed to ac-

TABLE V
REVENUE LOSS

Environment	E-Commerce	Blog
A	\$0.21	\$0.00
B	\$0.18	\$0.00
C	\$18,546.68	\$370.93
D	\$0.00	\$0.00

TABLE VI
CUMULATIVE COST

Time	SC	QC	PL	PNL
2	\$0.80	\$80.00	\$0.16	\$0.28
4	\$3.60	\$160.00	\$0.74	\$4.56
6	\$8.00	\$240.00	\$1.32	\$8.84
8	\$14.00	\$320.00	\$2.54	\$13.76
10	\$20,397.31	\$400.00	\$3.98	\$22.18
12	\$61,533.23	\$480.00	\$6.00	\$34.94
14	\$102,678.31	\$560.00	\$8.42	\$56.34
16	\$143,894.43	\$640.00	\$11.12	\$82.28
18	\$185,064.57	\$720.00	\$14.06	\$109.44
20	\$226,253.46	\$800.00	\$18.04	\$156.60
22	\$267,323.94	\$880.00	\$22.38	\$207.22
24	\$308,412.46	\$960.00	\$27.82	\$267.12

complish the QoS concerns needed to guarantee users satisfaction. Finding the cheapest combination among the number of instances, their configurations and prices is not an easy task. Also, as workloads varies through time resource over-provisioning can be a very expensive strategy – particularly for cases when the number of clients is small.

This paper presented a novel elasticity model called PROFUSE which computes the necessary computing power needed for keeping requests response times below a threshold. PROFUSE has a Fuzzy Logic Inference System that predicts workload changes. The processes used for determining FLIS variables, their fuzzy regions and initial IF-THEN rules were described in great detail. Also, for handling unpredictable huge workload changes PROFUSE provides a monitoring mechanism, called hooks, that keep crucial system metrics under close surveillance. Whenever an outlier is detected on one of those metrics, the computing power expansion routine is called. Finally, PROFUSE also provides a set of algorithms to lease and release instances using a revenue-centric approach where computational unit prices for each instance configuration are taken into consideration.

The experiments were conducted using two types of workloads that are typical for web system and four possible environments, covering from the simplest to the most complete scenario. Analyzing experiments results, PROFUSE's robustness is clear with it being able to keep the QoS even for the most stressful case. Finally, we showed that FLIS feedback time, called latency, is crucial for a good PROFUSE performance and should be set according to the mean time for leasing a new instance from the cloud provider.

A. Future works

As future works, we intend to investigate a way to identify workload patterns at runtime, giving PROFUSE the ability to handle different workloads with different FLIS and compare the approaches (single FLIS PROFUSE x multiple FLIS PROFUSE). In the same direction, we intend to investigate a method for automatize FLIS feedback times. Finally, we intend to build an incremental version of PROFUSE, without the need of historical data to create the FLIS.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, p. 5058, Apr. 2010.

[2] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[3] K. Boloor, R. Chirkova, Y. Viniotis, and T. Salo, "Dynamic request allocation and scheduling for context aware applications subject to a percentile response time SLA in a distributed cloud," in *2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, Dec. 2010, pp. 464 –472.

[4] S. Dutta, S. Gera, A. Verma, and B. Viswanathan, "SmartScale: automatic application scaling in enterprise clouds," in *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, Jun. 2012, pp. 221 –228.

[5] A. L. Freitas, N. Parlavantzas, and J.-L. Pazat, "An integrated approach for specifying and enforcing SLAs for cloud services," in *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, Jun. 2012, pp. 376 –383.

[6] P. Leitner, W. Hummer, B. Satzger, C. Inzinger, and S. Dustdar, "Cost-efficient and application SLA-Aware client side request scheduling in an infrastructure-as-a-service cloud," in *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, Jun. 2012, pp. 213 –220.

[7] G. McGovern. Selfish, mean, impatient customers: New thinking: Gerry McGovern. [Online]. Available: http://www.gerrymcgovern.com/nt/2008/nt-2008-07-14-selfish.htm

[8] M. Mazzucco, D. Dyachuk, and M. Dikaiakos, "Profit-aware server allocation for green internet services," *arXiv:1102.3059*, Feb. 2011, 18th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2010, pp 277-284.

[9] I. Goiri, F. Julií, J. O. Fitó, M. Macías, and J. Guitart, "Supporting cpu-based guarantees in cloud slas via resource-level qos metrics," *Future Gener. Comput. Syst.*, vol. 28, no. 8, pp. 1295–1302, Oct. 2012.

[10] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, Nov. 2011, pp. 1 –12.

[11] P. Bocharov, C. D'Apice, A. Pechinkin, and S. Salerno, *Queueing Theory*. Walter de Gruyter, 2004.

[12] H. Ghanbari, B. Simmons, M. Litoiu, C. Barna, and G. Iszlai, "Optimal autoscaling in a IaaS cloud," in *Proceedings of the 9th international conference on Autonomic computing*, ser. ICAC '12. New York, NY, USA: ACM, 2012, p. 173178. [Online]. Available: http://doi.acm.org/10.1145/2371536.2371567

[13] R. Rajavel and T. Mala, "Achieving service level agreement in cloud environment using job prioritization in hierarchical scheduling," ser. Advances in Intelligent and Soft Computing, S. Satapathy, P. Avadhani, and A. Abraham, Eds. Springer Berlin / Heidelberg, 2012, vol. 132, pp. 547–554.

[14] C. A. Ardagna, E. Damiani, F. Frati, D. Rebeccani, and M. Ughetti, "Scalability patterns for platform-as-a-service," in *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, Jun. 2012, pp. 718 –725.

[15] R. C. Berkan and S. L. Trubatch, *Fuzzy systems design principles: building Fuzzy IF-THEN rule bases*. IEEE Press, Apr. 1997.

[16] U. of Waikato, "WEKA," http://www.cs.waikato.ac.nz/ml/weka, 2016, [Online; accessed 31-May-2016].

[17] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, “An efficient k-means clustering algorithm: Analysis and implementation,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 7, pp. 881–892, 2002.

[18] B. Sharma, V. Chudnovsky, J. L. Hellerstein, R. Rifaat, and C. R. Das, “Modeling and synthesizing task placement constraints in google compute clusters,” in *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 2011, p. 3.

[19] M. Mao and M. Humphrey, “A performance study on the VM startup time in the cloud,” in *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, Jun. 2012, pp. 423 –430.

[20] D. Felipini, “Plano de negcios para empresas da internet,” Jun. 2003.