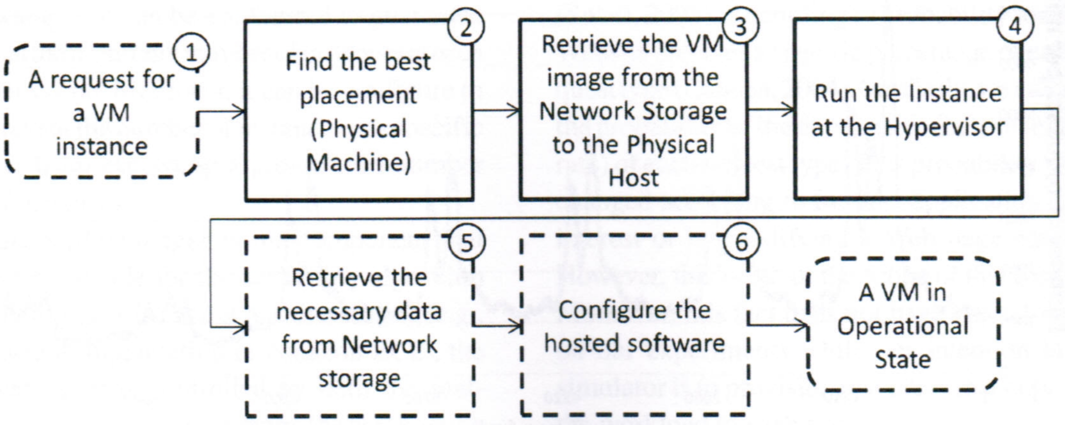
We shed light on other stages where user behavior can contribute to this delay. In Figure 6, we use two types of boxes to refer to two types of stages. The dotted line boxes refer to the stages where users can have some impact on the completion time of the stage. The solid line boxes show stages that are completed totally by the provider and its completion time is totally dependent on the provider algorithms and the current demand on the data centers.

The stages can be explained as follows:

1. The request for a new VM is initiated by the user either manually or by a scalability controller.
2. After receiving the request, the provider runs an algorithm to find the best physical host for hosting the new VM instance. Mao, (2012) has shown that machine size, operating system, time of the day, and number of instances have different weights of impact on the VM's start up times
3. After finding a suitable physical host, the VM image is copied through the network.
4. Once the VM image is copied completely to the physical host, it is run by the hypervisor. Running a VM in the cloud includes: booting the operating system, configuring the network setup (e.g., assigning a private IP and public domain name for a VM running instance), and copying the public key for accessing the VM in case of Linux, or generating a random password in case of Windows operating system.
5. The best practice to assure that the new initiated VM has up to date data is to store it in a networking storage. For instance, to run a Web server with the last version of html pages, at initialization time, the server should be pointed to the repository where a tar ball of html files can be retrieved and extracted to the proper folder on the Web server. The same procedures are applied for the application and database server. This can be completed through scripts that run at the VM start up time. On the other hand, customers should avoid retrieving huge amounts of data that can delay bringing the instance to operational mode. A long delay could make the dynamic scalability non-efficient as we show with the database tier in the next section.
6. Whenever a user gets the domain name of the initiated VM, the user can access it for configuring the hosted software. Users should avoid installing software at VM's start up time, while this can delay moving the VM to the operational mode. The best

Figure 6. Initializing a VM in the cloud


1
A request for a VM instance
2
Find the best placement (Physical Machine)
3
Retrieve the VM image from the Network Storage to the Physical Host
4
Run the Instance at the Hypervisor
5
Retrieve the necessary data from Network storage
6
Configure the hosted software
A VM in Operational State

practice is to pre-install the required software and packages to the VM image and prepare a script that runs automatically at the VM start up time to do the required configurations. These configurations may include passing the acquired private IP (i.e., internal IP) to another VM (e.g., the load balancer).

As seen above, a large part of the delay in VM running in the cloud is attributed to the provider. However, users should avoid any practices that can delay moving initiated instance to an operational state.

5.2. Networking Overhead Impact on Database Scalability

Database scale out is discussed in literature Iqbal et al. (2010) and Ge et al. (2008). Nevertheless, few researchers have considered the overhead of bringing a database VM to operational mode. Some providers offer vertical scalability for relational database (e.g., Amazon RDS, 2012) to cope with the workload increase. Conversely, Amazon RDS implies restarting the VM instance to apply the new assigned capacity. In prior research (Dawoud, 2011-b), we considered scaling database instances (i.e., number of cores) online without restarting the VM. The results are promising but require further investigation to be implemented in production environments.

In this research, we discuss scaling out database tier horizontally by adding more slave instances (i.e., read only instances). For this purpose, we should calculate the time required to bring a new slave database instance to operational mode. In our setup, we consider the typical setup for a scalable database in the cloud infrastructure. Consequently, we assume that an old dump of the whole database is stored in cloud storage (i.e., Amazon S3). Furthermore, an incremental backup of the binary files is also stored periodically to cloud storage, as described in (Dowman, 2009). The non-compressed dump of our database is 1.1GB. To reduce the transferring time from the networking storage to the new VM, the database can be compressed to a lower size (i.e., 153MB). We assume that the slave instance has a new and up to date MySql installation. At the initialization time of the VM, we run a script that copies the compressed dump file from S3 storage to the VM local storage. The dump file is extracted and used to restore the database. Afterwards, we retrieve all binary log files that had been uploaded to S3 during the Master database running time. Theses logs also applied to the new database.

As an example, the time required to run a new slave instance on *m1.small* instance at Amazon EC2 can be estimated as follows:

1. Initializing a small instance in Amazon EC2: 100 seconds
2. Copying the compressed dump of database to the new VM instance: 16 seconds
3. Extracting the database dump and importing it to the new database: 255 seconds
4. Retrieving and applying the incremental binary logs: 130 seconds
5. Get the last updates from master node, restart the slave, and updating the load balancer with the IP of the new slave node: 68 seconds

As shown above, initializing a relatively small slave database instance in the cloud can be done, at best, in 569 seconds, which can be estimated to about 10 minutes. Our measurements are exposed to increase if the size of the dump database was bigger or the number of incremental binary logs was larger. We should remember that our database is considered very small compared to large databases running in a production environment. Large databases require longer time to initialize a Slave VM from scratch, which raises the question about the efficiency of scaling-out a Slave database instances dynamically.

5.3. Evaluation

Current implementation of scalability in Amazon EC2 is reactive, while the trigger of scaling up or down is a result of exceeding the pre-determined thresholds of resources consumption. An example for that is to run a new VM instance when the aggregate CPU utilization of VMs replicas exceeds 70% as a scale out threshold. For that reason, current implementation of *AutoScaleManager*, by Amazon EC2, is considered a reactive controller. The problem with this approach is that it results in periods of under-provisioning, while running a new VM and bringing it into the operational mode does not happen instantly for the reasons explained before.

Unlike database instances initialization, the dominant delay for initializing a new instance in case of Web and application instances is the time required by the provider to run these instances, while the required time for retrieving data and configuring VM instances for both Web and application instances are just a few seconds. Mao et al. (2012) measured the initialization time of Linux instance at Amazon EC2 to be 96.9 seconds in average, while it is measured to be 44.2 seconds in RackSpace in average. In all our simulations, we consider it to be 60 seconds as an average value. This value is exposed to change according to the adopted technology by the provider. So, we keep it as a modifiable parameter, associated with the VM type, in our simulator.

The following run of the simulator considers a one-minute delay to bring a VM instance into operational mode in Web and application tier. Alternatively, the simulator considers 10 minutes to bring a Slave database instance to operational mode. The other parameters are set in Table 3 on the light of the description shown at Table 2. The VMs capacities are equal to *m1.small* instance described by (Amazon EC2, 2012). The workload is the one generated in Section 4.3.

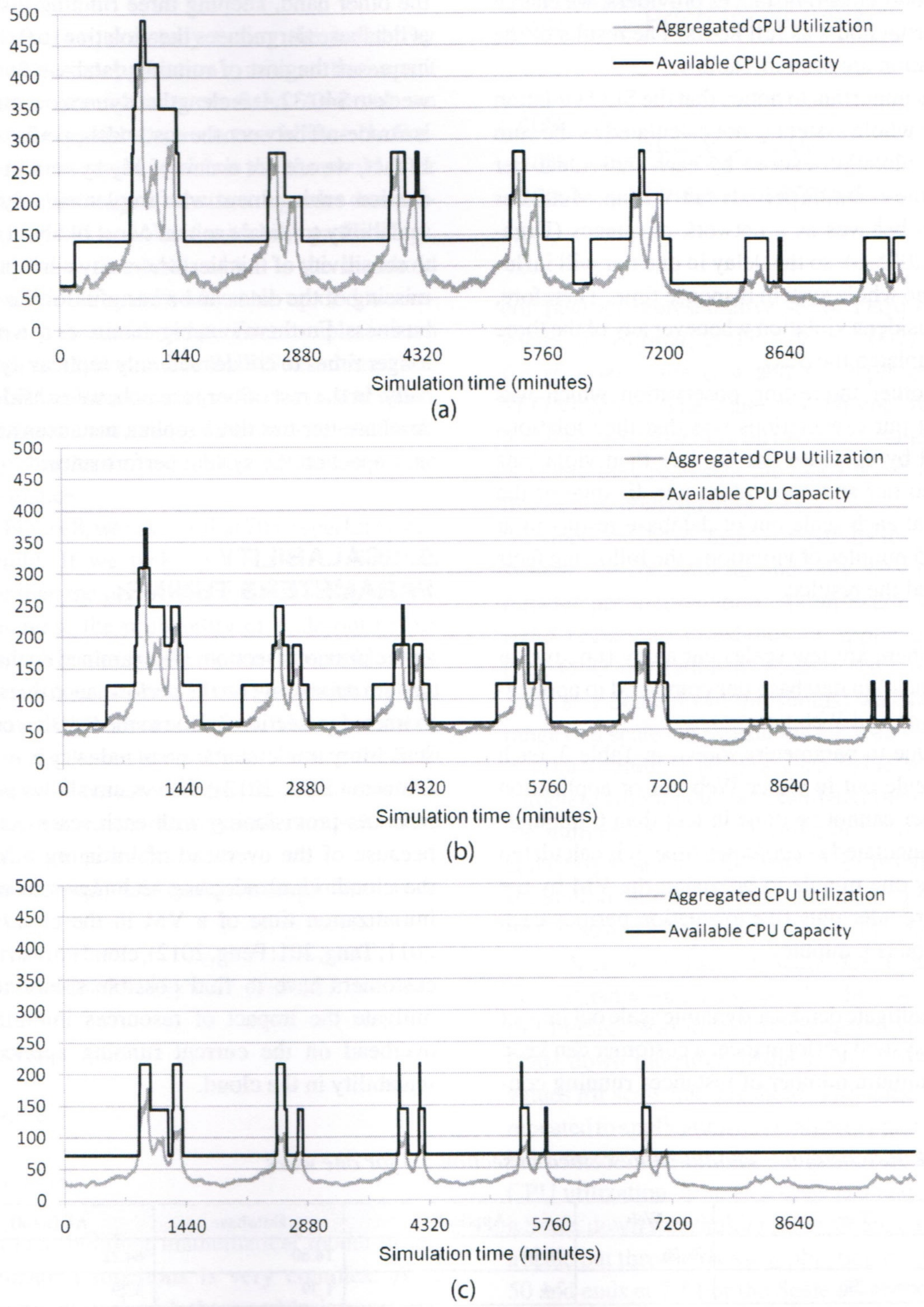
The output when running the simulation with parameters in Table 3 is seen in Figure 10.

SLO violation is recorded when the available CPU capacity at any tier is lower than the aggregated CPU utilization to that tier. For example, at minute 869 of the system run time, according to incoming requests, our simulator calculates that the CPU utilization of the running instance at database tier is 72%. According to scalability policy in Table 3, whenever the CPU utilization at database tier goes over 72%, the system should scale out while there is a high probability to approach an increase in response time (i.e., higher than 100 ms in our case). Scale out will not be triggered before five evaluation periods of CPU utilization, as set in Table 3. After triggering a scale out, the system will go into the sequence described in Section 5.1 and 5.2. So, with current setting, any violation for SLO in database tier will not be removed before 15 minutes. During these 15 minutes, we consider that the system is under-provisioned and unable to fulfill the SLOs.

Table 3. The parameters for running the simulation that results in charts seen in Figure 7

Parameter	Web tier	App tier	Db tier
cooldown	300 seconds after scale out or down	300 seconds after scale out or down	300 seconds after scale out or down
adjustment	1 for scale out -1 for Scale down	1 for scale out -1 for Scale down	1 for scale out -1 for Scale down
metric-name	CPU utilization	CPU utilization	CPU utilization
threshold	70% for scale out 30% for Scale down	62% for scale out 30% for Scale down	72% for scale out 30% for Scale down
period	1 minute	1 minute	1 minute
evaluation-periods	5	5	5

Figure 7. Simulating scalability of the multi-tier system with the parameters in Table 3: (a) web tier's dynamic scalability simulation; (b) application tier's dynamic scalability simulation; (c) database tier's dynamic scalability simulation



Conversely, the cost is calculated by multiplying the instance price per hour (i.e., 0.08$) by the number of running hours. Similar to most of well know on-demand instances providers, we charge the partial hours as full hours. The results of the simulation are seen in Table 4.

It is important to notice that the SLO violation for the whole system is not calculated as the sum of the violation caused by each individual tier while the violation periods can overlap. Multi-tier system behaves as a network of queues (Urgaonka, 2005-b), so the delay in one tier will influence the whole system response time. Therefore, we consider a violation whenever any of the three tiers violated the SLO.

Another interesting observation which was against our expectations was that the violations caused by database tier are less than violations by Web tier and application tier. In spite of the fact that each scale out of database results in at least 15 minutes of violations, the following facts changed the results:

1. There are few scales per a day (i.e., two at most) in database tier compared to application and Web tier.
2. Due to parameters shown in Table 3, each scale out in either Web tier or application tier cannot be done in less than 6 minutes, calculated as *cooldown* time. It is calculated as one minute to provision the VM by the provider plus five evaluation periods each one is a minute.

To mitigate database dynamic scale out impact on the system performance, a customer can keep the minimum number of instances running continuously in database tier. For example, keeping two instances reduces the violation in database tier to 0.23%, but, increases the cost to $27.04. On the other hand, keeping three running instances at database tier reduces the violation to 0.0% but, increases the cost of running database tier per a week to $40.32. It is clear that dynamic scalability is a trade-off between the cost and the performance. In fact, we are not aware of any system in a production environment who implements dynamic scalability to database tier. Most likely, it is due to sensitivity of this tier where any corruption or missing of the data can be harmful for the whole business. Furthermore, big databases can require longer times to create read only replicas dynamically. In the rest of our research, we consider that database tier has three replica instances and has no impact on the system performance.

6. SCALABILITY PARAMETERS TUNING:

In the previous section, we examined scaling the system depending on the performance thresholds extracted in Section 3. Due to the fact that current scalability implementation in industry is reactive (Amazon EC2, 2012); the system shows periods of under-provisioning with each scale out. It is because of the overhead of initiating a VM in the cloud. Until adopting techniques to reduce initialization time of a VM in the cloud (Wu, 2011; Tang, 201; Peng, 2012), cloud infrastructure customers have to find possible solutions that mitigate the impact of resources initialization overhead on the current running applications scalability in the cloud.

Table 4. Results of the simulation described in section 5.3 for one week

Tier	Web	Application	Database	All (total)
Cost ($)	28.56	21.36	14.80	64.72
SLO violations (%)	1.9	2.4	1.39	5.29

6.1. Calculating Scalability Thresholds as an Optimization Problem

In this section, we study tuning the scalability thresholds of an application to mitigate the impact of resource provisioning overhead. As a start, we should distinguish between two idioms: 1- Performance threshold, which is the threshold after which the system performance degrades dramatically (i.e. SLO is violated with high probability). 2- Scalability threshold, which is the threshold after which the system will scale up or down regardless of the performance. For example, as seen in Section 3.3, Web tier in RUBiS benchmark has 70% CPU utilization as a performance threshold, but in this section, we try different values of scalability thresholds to find the optimal cost and performance.

In Figure 8, we assume that 70% is performance threshold. If we pick up a scale out threshed higher than the performance threshold by Δh, we can increase the probability of scale out before approaching the performance threshold. This implies longer periods of over-provisioning and for sure will increase the cost. The question is, what is the best value of Δh that reduce the SLO violation but do not increase the cost so much? It is clear that we are dealing with multi-objective optimization problem.

The formal definition of the problem is as follows:

minimize $G(x,y) = \{Gc(x,y), Gv(x,y)\}$

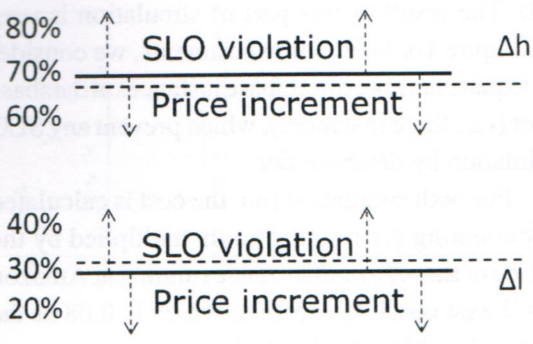
subject to:

$0 < x < 50$

$50 < y < 100$

In fact, building mathematical model of our optimization functions is very complex. As a solution, we deal with them as black boxes and

Figure 8. Scalability threshold tuning



compute all representative set of Petro optimal solutions (Ehrgott, 2005; Coello, 2007), where Gc represents the cost and Gv represents the SLO violation. The value of each goal function is calculated depending on two input values x and y, where they represent the Scale down and out thresholds, consequently.

Since we have different parameters in the system, we fix them all and only vary one parameter per a simulation. As an example, with fixed scalability thresholds of application tier (i.e., CPU utilization 62%, as a scale out, and 30%, as a Scale down thresholds) we evaluate a range of a scale out thresholds for Web tier starts at 60 and ends at 80. For each value we run a complete simulation and calculated the cost and the number of the SLO violations. To be sure about observations consistency, we run individual simulation for the first three day in our generated workload described in Section 4.3. Moreover, we tried different values of Scale down threshold ranging between 20 and 40. So, the x-axis of all sub-Figures 9(a) to 9(f) is the tested values for Scale down threshold while y-axis is the tested values for scale out threshold. The same setup is repeated to application tier scalability parameters. With fixed scalability thresholds of Web tier (i.e., CPU utilization 70%, as a scale out and 30%, as a Scale down thresholds) we evaluate a range of a scale out thresholds for application tier starts at 50 and ends at 70. For the Scale down threshold, we tried different values ranging between 20 and 40. The result of this part of simulation is seen in Figure 10. In all our experiments, we consider adequate number of database instances at database tier (i.e., three instances), which prevent any SLO violation by database tier.

For both simulation run, the cost is calculated by counting the running hours multiplied by the price of the *m1.small* instance running at Amazon EC2 east coast datacenters, which is 0.08 at the time of writing this book. It is important to note that even the partial hours are calculated. Moreover, as we notices in Amazon EC2, whenever it is the time to terminate an instance for Scale down, Amazon terminates the instance with the longer runtime. We use the same election way in our simulator. However, we plan to study optimizing the cost by terminating instances that are more close to the end of the hour.

SLO violation describes the percentage of the time that the response time of the Internet application (95th response time) is probably higher than 100 milliseconds. It is calculated by finding the number of minutes when the CPU utilization is higher than the performance threshold to the number of minutes per day (i.e., 1440 minutes).

From Figure 9 and Figure 10, we have the following observations:

1. **Scale Out Threshold Tuning:**
    a. A scale out threshold higher than the performance threshold decreases the cost slightly, but increases SLO violation.
    b. A scale out threshold lower than the performance threshold increases the cost slightly but reduces the SLO violation strongly.
    c. A very low scale out threshold increases the probability of over-provisioning which increases the cost without remarkable decease in SLO violation. However, if a very low scale out threshold coincides with a high Scale down threshold the probability of oscillating increases as will be shown in Figure 11.
2. **Scale Down Threshold:**
    a. A high Scale down threshold results in a high violation of the SLO. However, it does not result in any reduction of the cost.
    b. A very low Scale down threshold does not show an increase in the cost as expected in the previous section. However, it reduces the SLO violation.

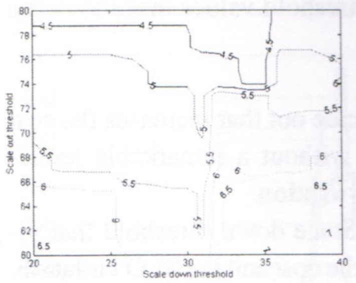
Using MOEA Framework (MOEA, 2011), which is an open source java framework for multi-objective optimization; we calculate the optimal values for scale out and down thresholds. As a multi-objective optimization problem, we have a set of solutions. However, from our observations, we pick up the best solution provided by the optimizer as follows:

- For scale out, picking a value slightly lower than the performance threshold reduces the probability of SLO violation. For example, setting 66 as a scale out threshold for the Web tier and 58 as a scale out threshold for the application tier leads to an optimal solution.
- For the Scale down, any value less that 30 keeps the SLO violation to the lowest if the scale out threshold is set to the optimal value describe before.

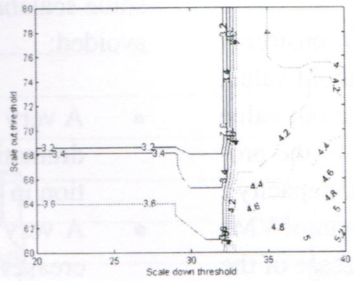
To evaluate scalability parameters tuning on the system performance, we repeat the experiment in Section 5.3, but with the optimized parameters and a continuous running of instances at database tier.

As seen in Table 5, with a little increase in the total running cost (i.e., 3.81%) we achieved a high reduction in the SLO violation (i.e., 72.29%). We appreciate this reduction when we remember that 1% SLO violation means that for each 100 running hour's there is a cumulative one hour where the response time of the system is higher than 100 milliseconds.

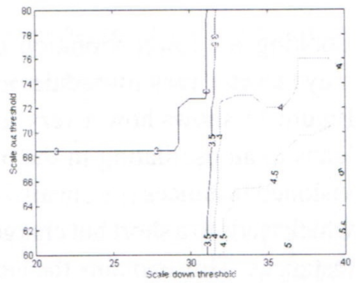
Figure 9. The impact of the scalability thresholds on cost and SLO violations at web tier



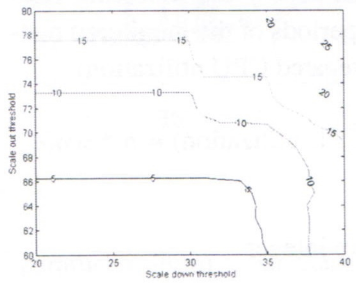
(a) Cost ($) - First day



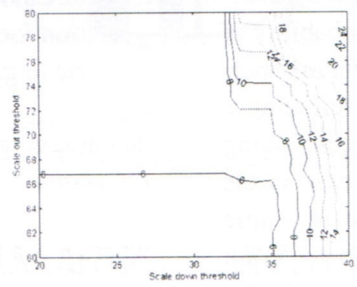
(b) Cost ($) - Second day



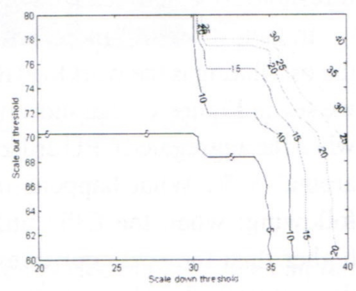
(c) Cost ($) - Third day



(d) SLO violation (%) - First day

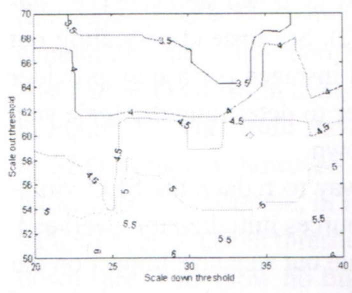


(e) SLO violation (%) - Second day

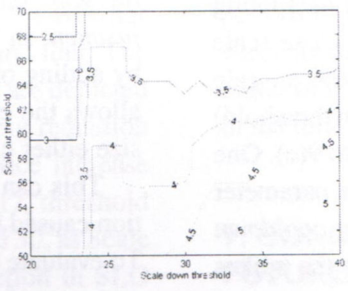


(f) SLO violation (%) - Third day

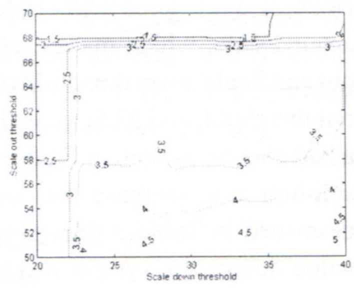
Figure 10. The impact of the scalability thresholds on cost and SLO violations at application tier



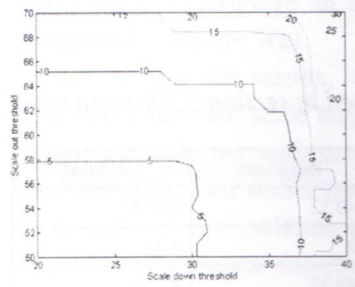
(a) Cost ($) - First day



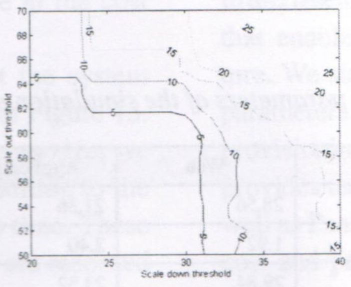
(b) Cost ($) - Second day



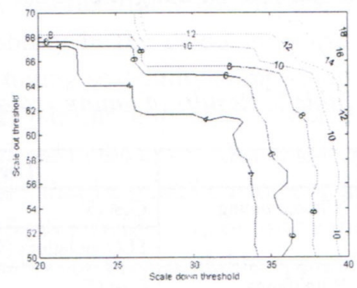
(c) Cost ($) - Third day



(d) SLO violations (%) - First day



(e) SLO violation (%) - Second day



(f) SLO violation (%) - Third day

6.2. Bad Threshold Values

Looking for lower violation of SLO, customer may select a very low scale out threshold value. Figure 11 shows how a very low scale out value leads to an oscillating in the number of the provisioned instances (i.e., available CPU capacity), which leads to a short but chargeable runs of VMs instances. This explains the early increase of the cost seen in Figures 10(a) to 10(c) for scale out threshold 51% and Scale down threshold 30%.

In fact, what also increases the probability of the oscillating is the workload itself. For instance, we see in Figure 11 that most of the oscillating is when the aggregated CPU utilization is oscillating around 51%. What happens in this case is the following: when the CPU utilization is a little higher than the scale out threshold (e.g., 53%); the system will scale out to two instances. At this time, the CPU utilization per an instance will be (53/2 = 26.5) which satisfies the Scale down policy while (26.5% is less than 30%).

Similarly, for Web tier, we observe many periods of aggregated CPU utilization oscillating around 66%, which makes selecting close scale out and Scale down thresholds (e.g., 66%, as scale out threshold, and 33%, as Scale down threshold) a bad choice, as seen in Figures 9(a) to 9(c). One solution is to increase the *cooldown* parameter described in Table 3. However, a big *cooldown* value delays the system response to the spikes in workload and may result in longer periods of under-provisioning (i.e., SLO violation).

According to our observations, we can define some scalability threshold values that should be avoided:

- A very low scale out that increases the cost dramatically without a remarkable reduction in SLO violation.
- A very high Scale down threshold that increases both the cost and the SLO violation.
- A scale out or down thresholds (i.e., *scale_threshold*) that satisfy the following relation for long periods of the monitored metric (e.g., aggregated CPU utilization):

mean(aggregated CPU utilization) = *n* * *scale_threshold,*

where *n* is a positive integer

6.3. Scalability Step Size Impact on Performance

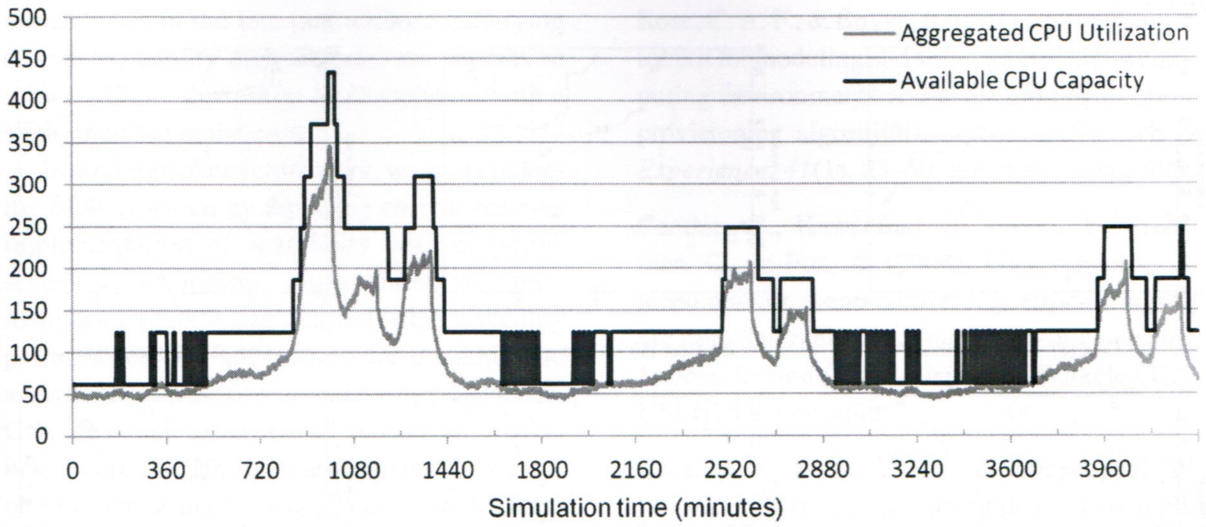
At cloud infrastructure, customers can determine the scale out and Scale down step size (i.e., *adjustment* in Table 2). So, instead of scaling out by adding one VM instance per a step, provider allows the customer to determine the scale step size either up or down.

This can be a way to reduce the SLO violation caused by resources initialization overhead. To evaluate the scale out step size impact on the system performance we repeat last simulation of Web tier but with different values of *adjustment*

Table 5. Results of tuning scalability parameters of the simulation described in section 5.3 for one week

	Tier	Web	Application	Database	All (total)
Without tuning	Cost ($)	28.56	21.36	40.32	90.24
	SLO violations (%)	1.92	2.40	0	4.15
With tuning	Cost ($)	29.84	23.52	40.32	93.68
	SLO violations (%)	0.66	0.51	0	1.15

Figure 11. The impact of bad values of scalability parameters' on the performance of the application tier



parameter. From experiments, we noticed that a fast Scale down has a severe impact on the SLO violation. So, we only examined different values for the scale out step size (i.e., *adjustment*).

Because Figure 9(a) and Figure 9(d) already present Web tier scale out with one VM per scale (i.e., *adjustment = 1*), we only repeat the Web tier simulation with values two, three, and four. The cost and SLO violation of each case are depicted in Figure 12. The result shows no big reduction in SLO violation, however, we notice increase in the cost. For example, in Figure 12, threshold values 66, as scale out threshold, and 30, as Scale down threshold show no big reduction in SLO violation. However, in figures 12(a), 12(b), and 12(c) we can recognize an increase in the cost compared to Figure 9(a).

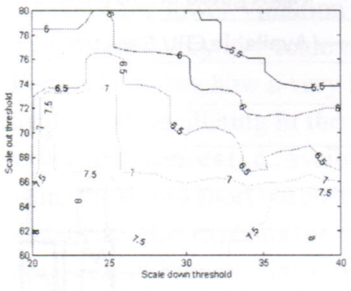
To analyze the results, we plot the system scalability for each scale step size in Figure 13. When we scale out with one VM step size, we can recognize four scales out, in addition to the first scale out at simulation start up time. These periods of the system running time are recorded as violation of the SLO.

In Figure 13(b) the scale step size is increased to two VM instances per scale, which reduced the total SLO violation periods to two. We can recognize periods of over provisioning that explain the increase in the cost seen in Figure 12(b). In Figures 13(c) and 13(d), the step size is increased to three and four, respectively. However, we notice the same number of SLO violations, but extra periods of over-provisioning. This explains why we cannot recognize a real decrease in SLO violation in Figure 12(e) and 12(f), but an increase in the total cost due to over-provisioning.

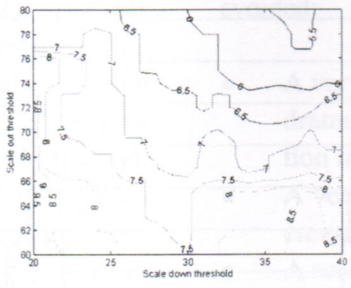
7. CONCLUSION AND FUTURE WORK

In this research, we described the main components that enable scalability in the cloud infrastructure. We studied tuning scalability components' parameters to mitigate the impact of resources provisioning overhead in the cloud. Our research provides techniques that help IaaS customers, as well as PaaS and SaaS providers, to optimize the cost and performance of their scalable applications, and consequently maximize the profit. The analysis depends on measurements from physical environment fed to our developed simulation

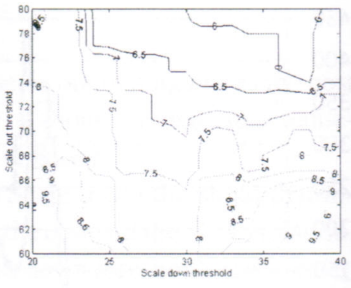
Figure 12. The impact of the scale out step size on the cost and the SLO violation at web tier for the first day



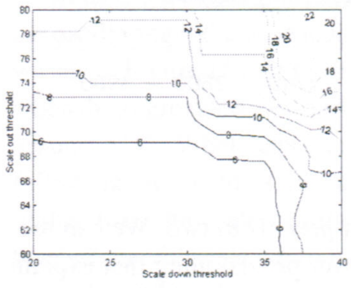
(a) Cost ($) - Two instances



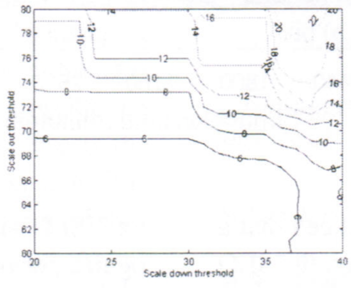
(b) Cost ($) - Three instances



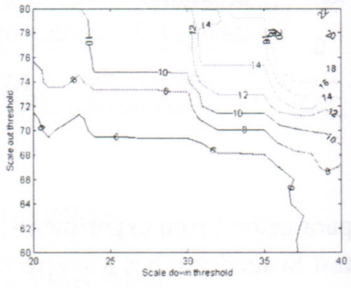
(c) Cost ($) - Four instances



(d) SLO violation (%) - Two instances

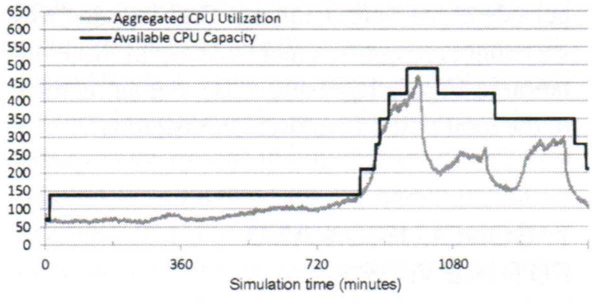


(e) SLO violation (%) - Three instances

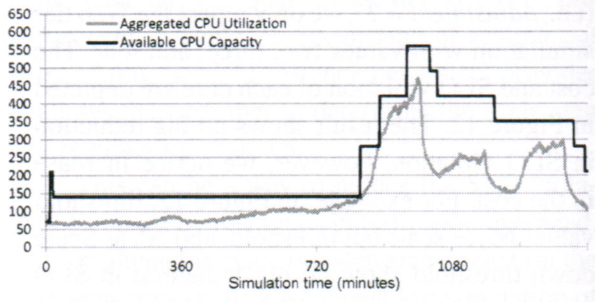


(f) SLO violation (%) - Four instances

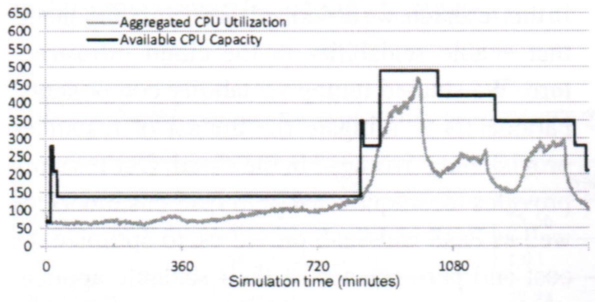
Figure 13. Scale out step size impact on cost and SLO violations at web tier



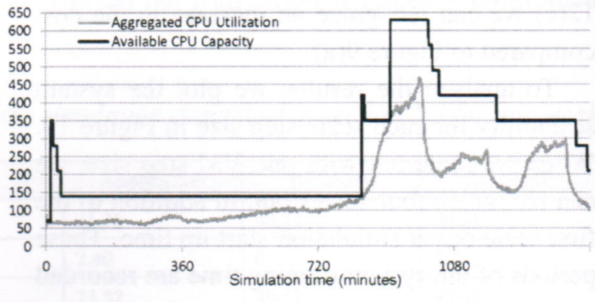
(a) One VM



(b) Two VMs



(c) Three VMs



(d) Four VMs

environment (i.e., ScaleSim). The novelty of our research lies in the fact that without modifying current scalability architectures, we success to achieve 72% reduction in SLO violation with a slight increase in the cost.

In our immediate future work, we study reducing SLO violation by replacing current reactive implementation of scalability with proactive scalability algorithms. Furthermore, we study monitoring window size's impact on the scalability performance. We are looking for dynamic and automatic optimization of scalability parameters. Currently, our experiments depend on RUBiS benchmark; we are looking for generalizing our observation using variety of applications from production environment.

REFERENCES

Agrawal, D., El Abbadi, A., Das, S., & Elmore, A. J. (2011). Database scalability, elasticity, and autonomy in the cloud. In *Proceedings of the 16th international conference on Database systems for advanced applications*. Berlin: Springer-Verlag.

Amazon, R. D. S. (2012). *Amazon relational database service*. Retrieved April 2, 2013 from http://aws.amazon.com/rds/

Amazon EC2. (2012). *Amazon elastic compute cloud*. Retrieved April 2, 2013 from http://aws.amazon.com/ec2/

Arlitt, M., & Jin, T. (1999). Workload characterization of the 1998 world cup web site. *HP Labs Technical Reports*. Retrieved April 2, 2013 from http://www.hpl.hp.com/techreports/1999/HPL-1999-35R1.html

AutoScaling. (2012). *Auto scaling command line tool*. Retrieved April 2, 2013 from http://aws.amazon.com/developertools/2535

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A. F., & Buyya, R. (2011). CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software, Practice & Experience*, *41*(1), 23–50. doi:10.1002/spe.995.

Candea, G., Kawamoto, S., Fujiki, Y., Friedman, G., & Fox, A. (2004). Microreboot --- A technique for cheap recovery. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*. Berkeley, CA: USENIX Association.

Cecchet, E., Marguerite, J., & Zwaenepoel, W. (2002). Performance and scalability of EJB applications. *SIGPLAN Notifications*, *37*(11), 246–261. doi:10.1145/583854.582443.

Chieu, T. C., Mohindra, A., & Karve, A. A. (2011). Scalability and performance of web applications in a compute cloud. In *Proceedings of the IEEE International Conference on E-Business Engineering*, (pp. 317-323). IEEE.

CloudWatch. (2012). *Amazon CloudWatch*. Retrieved April 2, 2013 from http://aws.amazon.com/cloudwatch/

CloudWatch Command Line. (2012). *Amazon CloudWatch command line tool*. Retrieved April 2, 2013 from http://aws.amazon.com/developertools/2534

Coello, C. A., Lamont, G. B., & Van Veldhuisen, D. A. (2007). *Evolutionary algorithms for solving multi-objective problems*. Berlin: Springer.

Curino, C., Jones, E., Zhang, Y., & Madden, S. (2010). Schism: A workload-driven approach to database replication and partitioning. In *Proceedings of VLDB Endowowment*. VLDB.

Dawoud, W., Takouna, I., & Meinel, C. (2011a). Elastic VM for cloud resources provisioning optimization. In *Proceedings of the First International Conference on Advances in Computing and Communications (ACC 2011)*, (vol. 190, pp. 431-445). Springer.

Dawoud, W., Takouna, I., & Meinel, C. (2011b). Elastic virtual machine for fine-grained cloud resource provisioning. In *Proceedings of the 4th International Conference on Recent Trends of Computing, Communication & Information Technologies (ObCom 2011)*. Springer.

Dawoud, W., Takouna, I., & Meinel, C. (2012). Dynamic scalability and contention prediction in public infrastructure using internet application profiling. In *Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science*. Taipei, Taiwan: IEEE.

Dowman, P. (2009). *Backing up your MySQL database to S3*. Retrieved April 2, 2013 from http://pauldowman.com/2009/02/08/mysql-s3-backup/

Dubey, A., Mehrotra, R., Abdelwahed, S., & Tantawi, A. (2009). Performance modeling of distributed multi-tier enterprise systems. *ACM SIGMETRICS Performance Evaluation Review*, *37*(2), 9. doi:10.1145/1639562.1639566.

Ehrgott, M. (2005). *Multicriteria optimization*. Berlin: Springer.

ELB APIs. (2012). *Elastic load balancing API tools*. Retrieved April 2, 2013 from http://aws.amazon.com/developertools/2536

Faban. (2012). Retrieved April 2, 2013 from http://java.net/projects/faban/

Framework, M. O. E. A. (2011). *Version 1.17*. Retrieved April 1, 2013 from http://www.moea-framework.org/

Ge, Y., Wang, C., Shen, X., & Young, H. (2008). A database scale-out solution for emerging write-intensive commercial workloads. *SIGOPS Operating Systems Review, 42*(1).

GoGrid. (2012). Retrieved April 2, 2013 from http://www.gogrid.com/

Heo, J., Zhu, X., Padala, P., & Wang, Z. (2009). Memory overbooking and dynamic control of Xen virtual machines in consolidated environments. In *Proceedings of the 11th IFIP/IEEE international conference on Symposium on Integrated Network Management (IM'09)*. IEEE Press.

Iqbal, W., Dailey, M. N., & Carrera, D. (2010). SLA-driven dynamic resource management for multi-tier web applications in a cloud. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE.

Iqbal, W., Dailey, M. N., & Carrera, D. (2011). Black-box approach to capacity identification for multi-tier applications hosted on virtualized platforms. In *Proceedings of the 2011 International Conference on Cloud and Service Computing (CSC '11)*. IEEE Computer Society.

Jayasinghe, D., Malkowski, S., Wang, Q., Li, J., Xiong, P., & Calton, Pu. (2011). Variations in performance and scalability when migrating n-tier applications to different clouds. In *Proceedings of the IEEE 4th International Conference on Cloud Computing*. IEEE.

Jung, G., Joshi, K. R., Hiltunen, M. A., Schlichting, R. D., & Pu, C. (2008). *Generating adaptation policies for multi-tier applications in consolidated server environments*. IEEE. doi:10.1109/ICAC.2008.21.

Li, J., Wang, Q., Jayasinghe, D., Malkowski, S., Xiong, P., & Pu, C. … Kawaba, M. (2012). Profit-based experimental analysis of IaaS cloud performance: impact of software resource allocation. In *Proceedings of the 2012 IEEE Ninth International Conference on Services Computing* (SCC '12). IEEE Computer Society.

Mao, M., & Humphrey, M. (2012). A performance study on the VM startup time in the cloud. In *Proceedings of IEEE 5th International Conference on Cloud Computing (Cloud 2012)*. IEEE.

Menascé, D. (2002). Load testing of web sites. *IEEE Internet Computing*, *6*(4). doi:10.1109/MIC.2002.1020328.

Miller, R. (2008). *A look inside Wikipedia's infrastructure*. Retrieved April 2, 2013 from http://www.datacenterknowledge.com/archives/2008/Jun/24/a_look_inside_wikipedias_infrastructure.html

Nielsen, J. (1993). *Usability engineering*. San Francisco, CA: Morgan Kaufmann Publishers Inc..

Olio. (2012). Retrieved April 2, 2013 from http://incubator.apache.org/olio/

Peng, C., Kim, M., Zhang, Z., & Lei, H. (2012). VDN: Virtual machine image distribution network for cloud data centers. In *Proceedings of the 31th IEEE International Conference on Computer Communications (INFOCOM 2012)*. Orlando, FL: IEEE.

Rackspace. (2012). Retrieved April 2, 2013 from http://www.rackspace.com/

RightScale. (2012). Retrieved April 2, 2013 from http://www.rightscale.com/

Route 53. (2012). Retrieved April 2, 2013 from http://aws.amazon.com/route53/

Scalr. (2012). Retrieved April 2, 2013 from http://code.google.com/p/scalr/

Sobel, W., Subramanyam, S., Sucharitakul, A., Nguyen, J., Wong, H., Klepchukov, A., … Patterson, D. (2008). *Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0*.

Tang, C. (2011). FVD: A high-performance virtual machine image format for cloud. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*. USENIX.

Urgaonka, B. (2005). Dynamic resource management in Internet hosting platforms. *Electronic Doctoral Dissertations for UMass Amherst*. Paper AAI3193951. Retrieved from http://scholarworks.umass.edu/dissertations/AAI3193951

Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M., & Tantawi, A. (2005). An analytical model for multi-tier Internet services and its applications. *SIGMETRICS Performance Evaluation Review*, *33*(1), 291–302. doi:10.1145/1071690.1064252.

Van Baaren, E. (2009). *WikiBench: A distributed, wikipedia based web application benchmark*. (Master Thesis). VU University, Amsterdam, The Netherlands.

Windows Azure. (2012). Retrieved April 2, 2013 from http://www.windowsazure.com/

Wu, X., Shen, Z., & Lin, Y. (2011). Jump-start cloud: efficient deployment framework for large-scale cloud applications, In *Proceedings of the 7th International Conference on Distributed Computing and Internet Technology ICDCIT 11*. ICDCIT.

Zhang, Q., Cherkasova, L., & Smirni, E. (2007). A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In *Proceedings of the Fourth International Conference on Autonomic Computing*. Washington, DC: IEEE Computer Society.

KEY TERMS AND DEFINITIONS

Scale Out: To scale out (scale horizontally) is to add more nodes to the system. An example might be adding more Web instances to Web tier.

Scale Up: To scale up (scale vertically) is to add resources to the same node in a system. An example might be to add more physical memory (i.e., RAM) to a database node.

Scale Down: To Scale down is to release some acquired resources. If the resources acquired by scale out, Scale down means releasing some nodes. If the resources acquired by scale up, Scale down means removing some of the node's resources.

Scalable Architecture: It is architecture enables rapid, automated, self-balanced, and transparent scalability to the users of the system.

Service Level Agreement (SLA): SLA is an agreement outlining a specific service commitment made between contract parties – a service provider and its customer. The agreement describes the overall service, support details, financial aspects of service delivery, penalties, terms and conditions, and performance metrics that govern service delivery.

Service Level Objective (SLO): SLO is specific measurable characteristic of the SLA such as availability, throughput, response time, or quality. An example of response time as an objective is: "95% of the requests to an Internet application should be answered in less than 100 milliseconds measured over 24 hours."