2012 IEEE 4th International Conference on Cloud Computing Technology and Science

Dynamic Scalability and Contention Prediction in Public Infrastructure using Internet Application Profiling

Wesam Dawoud
Hasso Plattner Institute
Potsdam University
Potsdam, Germany
wesam.dawoud@hpi.uni-potsdam.de

Ibrahim Takouna
Hasso Plattner Institute
Potsdam University
Potsdam, Germany
ibrahim.takouna@hpi.uni-potsdam.de

Christoph Meinel
Hasso Plattner Institute
Potsdam University
Potsdam, Germany
christoph.meinel@hpi.uni-potsdam.de

***Abstract*—Recently, the advance of cloud computing services has attracted many customers to host their Internet applications in the cloud. Infrastructure as a Service (IaaS) is on top of these services where it gives more control over the provisioned resources. The control is based on online monitoring of specific metrics (e.g., CPU, Memory, and Network). Despite the fact that these metrics guide resources provisioning, the lack of understanding application behavior can lead to wrong decisions. Moreover, current monitored metrics alone do not help in resources contention prediction, which is very common in shared infrastructures like IaaS. Nevertheless, the architecture of Internet applications, as multi-tier systems, makes contention prediction more complex while its influence can migrate from one tier to another.**

In this paper, we propose a pro-active global controller not only for dynamic resources provisioning, but also for predicting and eliminating contentions in multi-tier applications. Our technique combines monitored metrics, which are provided by current IaaS providers, with models that are built depending on the Internet applications profiling. The fitness of the monitored metrics to the application model is used for contention prediction. We examined our technique using RUBiS benchmark. The results express the efficiency of the developed algorithms in maintaining Internet applications performance even in shared infrastructures.

***Keywords*-Public IaaS; Cloud computing; Contention prediction; Dynamic scalability; Application profiling.**

I. Introduction

Typically, Internet applications are implemented using multi-tier architecture as seen in Fig. 1(a). Each tier provides a particular functionality. However, the type of incoming request determines the participating tiers in the request handling. For example, a request for a static page can be handled by a web tier only. On the other hand, a search for items in an online retail store will result in interactions between all tiers including web tier, application tier, and database tier [1].

So, to cope with the incoming workload variation, application at each tier may be replicated into many servers. To keep load balancing, the incoming workload will be distributed among replicas using a dispatcher. The emergence of pay-as-you-go concept in the cloud environment allows customers to specify the number of replicas that cope with workload demand while keeping the total cost to the minimum. To control the number of replicas, IaaS providers (e.g., Amazon EC2) offer the customers an online monitoring of specific metrics utilization (e.g., CPU, Memory, and Network). A simple approach is to determine a static upper threshold (e.g., 70% CPU utilization) as a trigger for increasing the number of Virtual Machines (VM) instances at high workload, and another static lower threshold (e.g., 30% CPU utilization) as a trigger for decreasing the number of VM instances at low workload demand.

In fact, the static threshold approach, without knowledge of application behavior, has many limitations. First, applications are usually exposed to a concurrency limit, so whenever this limit is approached, some requests are dropped [2], [3], which keeps the monitored metrics within a specific limit that does not really reflect the real demand. Second, in a virtualized shared infrastructure, many VMs of different customers can compete on the same physical host resources. This competition leads to resource contention that cannot be expressed by the available monitored metrics, even though contention is often measured as a reduction in resources utilization. We will see this in Section III. Third, according to multi-tier architecture, the influence of contention at one tier can migrate to the other tiers [4], which increases the complexity of system management.

Our contributions in this paper are summarized as follows: First, we built models that describe resources utilization according to workload type and requests rate variation. Second, we designed controllers that employ these models, besides the online monitoring of resources, to dynamically provision resources as well as predict and eliminate contentions in IaaS environments. Finally, we evaluated our approach practically using RUBiS benchmark.

In next section, we present our proposed solution including system architecture, methodology, and models identification. Next, in Section III, we evaluate our system using RUBiS benchmark. In Section IV we give a literature review. Finally, in Section V, we conclude our work and present the intended future work.

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

IEEE computer society

II. Proposed System

In this section, we start with an overview of current application scalability architecture at IaaS, and then we discuss the potential limitations followed by our proposed solution. Afterwards, we provide a detailed description of each component of our system and the running algorithms. Then, we show extraction and evaluation of utilization models. At the end of the section, we discuss some of the technical issues that emerged during our system development.

A. System architecture

Our system assumes that an Internet application is hosted in a public IaaS environment and deployed using multi-tier architecture. A typical multi-tier architecture is illustrated at the upper part of Fig. 1. The rounded rectangles show the running instances at each tier. Whenever the customer submits a request for an instance, the provider finds the best host according to instance type and workload on the hosts. The same host can run instances of different customers with different demands. The number of these instances should be increased or decreased according to workload's variation. Typically, Service Level Agreement (SLA) of IaaS provider describes only the annual up time of the instances, but it does not discuss potential performance degradation caused by contention for resources.

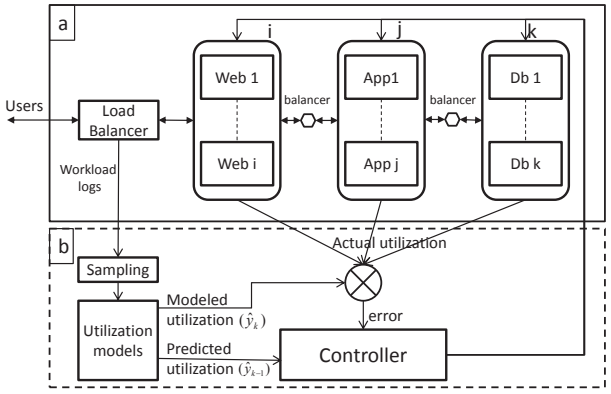

Figure 1. System architecture

IaaS providers delegate the control on the number of instances at each tier to their customers. For this goal, customers are equipped with services which enable them to monitor resources and determine triggers for increasing or decreasing the number of instances at each tier. However, scalability depending on static thresholds of each tier without knowledge of the application model and the real incoming workload is inefficient according to [2] and [3]. Therefore, we propose a global controller that employs the application model to proactively control tiers capacity and resolve application performance anomalies that are caused by contention for resources.

B. Methodology

The architecture of our proposed solution is shown in Fig. 1(b) and explained in details in Fig. 2. It integrates the utilization monitoring with requests logging. Our proposed system is comprised of the following phases: monitoring, sampling, model building and running.

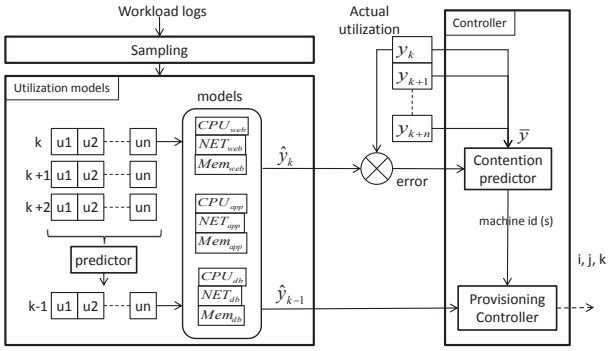

Figure 2. Our proposed system

1) Monitoring phase: Besides the monitored resources that are provided by current IaaS providers we consider the log files that describe the transactions handled by Internet application. Typically, these logs locate at a single point where all traffic passes (e.g., load balancer). However, to avoid single point of failure, some providers offer hardware load balancers or Elastic Load balancers [5] as we will discuss in Section III-C. Even in such cases, the logs collected by different web servers can be synchronized to one log file that describes the demand on the whole Internet application. The access log file can be configured to log access time, response time, and the request URL.

2) Sampling phase: As the transactions continuously arrive to the system, we consider a sampling window that describes the transactions' behavior at specific period of time. Within the sampling window, we classify the transactions into categories, and then determine the rate of each category. The categories are determined depending on the URL. In fact, types of requests depending on URL can be very large in a real Internet application. However, classifying these requests to coarse grained categories: cacheable, non-cacheable, and demanding [6] reduces the number of the categories that really have an impact on the system utilization. For example, Zhang et al. [7] showed that using only 20 types of a requests with a system of total 756 types of requests could lead to accurate resources utilization prediction. In our experiments, we consider 18 types of requests using RUBiS benchmark and ignore the cacheable requests (i.e., static pages and images), while they show a negligible impact on web tier performance. The output of the sampling phase for the sampling period k is a vector describing the number of requests of each category ($k[u^1, u^2, ..., u^n]$). These sampled vectors are used for both

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

IEEE computer society

training and running the system.

3) Modeling phase: In the modeling phase, to avoid any influence from the other tiers, we run plenty of replica instances in all tiers except the modeled one. For instance, to model a web tier, we run a single VM instance at the web tier but many replicas at both application and database tier. The collected traces are CPU utilization, Memory allocation, Network IN/OUT rate, and Disk read/write rate. Afterwards, we synchronize the collected traces with the access logs that are collected in the sampling phase. The rate of each request category vector is an input of our models, while the output is one measured performance metric at a time (e.g. CPU utilization).

4) Running phase: This phase depends on the extracted models in the training phase. These models are employed to do two integrated functions: First, managing the scalability of the system. Second, predict and eliminate contention for resources.

At running time, the *sampling* module, seen in Fig. 2, continuously extracts a raw vector of requests' rate for each sampling period k. The requests vector $k[u^1, u^2, , u^n]$ is passed to the models to predict $\hat{y_k}$, which represents the expected utilization of a resource calculated by models. The online measure utilization y_k, the average of the measured utilization $\overline{y}$, as well as the modeled utilization $\hat{y_k}$ are used in equation 1 to calculate F, which describes the fitness of the measured to the modeled values of resources utilization of VMs instances at each tier.

$$F = 1 - \sqrt{\frac{\sum_{k=1}^{N} |y_k - \hat{y}_k|^2}{\sum_{k=1}^{N} |y_k - \overline{y}_k|^2}} \quad (1)$$

Actually, equation 1 is applied for each resource r of the VM instance n, which runs at tier m. To formalize it, we consider M tiers. Each tier runs N VM instances, while each VM instance has R resources. Practically, we implemented $\hat{y}_k$, $\hat{y}_{k-1}$, $\overline{y}_k$, and y_k using ArrayList object, while each tier can have a different number of instances. To simplify arrays representation, we represent them as three dimensional arrays.

In addition to y_k, $\hat{y}_k$, and $\overline{y}_k$, the inputs of contention prediction algorithm include $ConfWindow$. An integer value determines a threshold value of the acceptable negative fitness occurrence. It is used to increase the system stability. So, before adding a VM into $VMs[\,]$ for replacement, the algorithm allows many occurrence of low fitness. However, if the low fitness occurrence number went higher than the determined $Confwindow$ for a VM instance, the algorithm add that VM to $VMs[\,]$ vector for replacement.

Parallel to the contention prediction algorithm, the provisioning algorithm continuously checks the contented VMs list $VMs[\,]$. The first step to eliminate a contented VM is to run a new VM instance in the same tier. Afterwards, the algorithm continues its run looking for any tier m with a bottleneck. The minimum and maximum thresholds of each resource r at tier m are described by system administrator as $min[m][r]$ and $max[m][r]$, respectively. The bottleneck is determined by comparing predicted utilization $\hat{y}_{k-1}[m][n][r]$ with predetermined maximum thresholds $max[m][r]$. To be sure that each tier is scaled up once per a control loop, we keep a tag array called $scale\_up[m]$ for each tier m. Similarly we have $scale\_down[m]$ tags array for scaling down. At scaling down, the algorithm gives the priority for terminating VMs instances which are tagged as contended instances. However, if a tier does not contain any contended instance, the algorithm picks up a VM instance randomly to terminate.

Algorithm 1 Contention prediction algorithm

Input: $y_k[M][N][R]$, $\hat{y}_k[M][N][R]$, $\overline{y}_k[M][N][R]$, and $ConfWindow$
Output: $VMs[\,]$
Initialization: $temp\_VMs[\,] \leftarrow null$ (Local vector contains candidate VMs for replacement)
loop
  // Find VMs with potential contention
  for $m = 1$ to M **do**
    for $n = 1$ to N **do**
      for $r = 1$ to R **do**
        //Calculate F at equation 1 for each resource r
        if $F < 0$ **then**
          insert VM_id into $temp\_VMs[\,]$
        end if
      end for
    end for
  end for
  if count(VM_id in $temp\_VMs[\,]$) $> ConfWindow$ **then**
    insert VM_id into $VMs[\,]$
    remove VM_ids from $temp\_VMs[\,]$
    pass $VMs[\,]$ to Provisioning controller
  end if
end loop

In our algorithms we assume that each load balancer, seen in Fig.1, routes the same amount of traffic for each replica. In other words, to calculate $\hat{y_k}$ and $\hat{y_{k-1}}$, the model should divide the input vector $[u^1, u^2, , u^n]$ by number of replicas. We will validate this assumption at section III.

C. Models identification

In this section, we start with a single-input, single-output (SISO) AutoRegressive model with eXogenous inputs (ARX) to learn a linear relationship between input u and output y as $y = f(u)$. Later, we will extend the SISO model to a multiple-input, single-output (MISO) model, which is used to model the relation between the multiple inputs (i.e., the rate of each requests category) and a single output (i.e.,

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

IEEE computer society

Algorithm 2 Provisioning algorithm

Input: $\hat{y}_{k-1}[M][N][R]$, $max[M][R]$, $min[M][R]$, $Current[M]$, and $VMs[\,]$
Output: $Next[M]$
Initialization: $scale\_up[M]$ is initialized with false; $scale\_down[M]$ is initialized with false
loop
  for $m = 1$ to M **do**
    if $VMs[\,] \neq null$ **then**
      // Add new VM instances sibling to contented VMs
    end if
    for $n = 1$ to N **do**
      for $r = 1$ to R **do**
        // Check if scaling up is required
        if $\hat{y}_{k-1}[m][n][r] >= max[m][r]$ **and** $scale\_up[m] \neq true$ **then**
          // Add new VM instance to tier m
          $Next[m] \leftarrow Current[m] + 1$
          $scale\_up[m] \leftarrow true$
        end if
        // Check if scaling down is possible
        if $\hat{y}_{k-1}[m][n][r] < min[m][r]$ **and** $scale\_up[m] \neq true$ **and** $scale\_down[m] \neq true$ **then**
          // Turn off a VM instance at tier m
          if $VMs[\,] \neq null$ **then**
            // Schedule terminating the contented VM
          else
            // pickup any VM for scheduled termination
          end if
          $Next[m] \leftarrow Current[m] - 1$
          $scale\_down[m] \leftarrow true$
        end if
      end for
    end for
  end for
end loop

CPU utilization, Memory utilization, Network input/output traffic rate, etc...). The input u and output y are sampled at time k as u_k and y_k respectively. The input-output relationship can be represented by the following difference equation:

$$y_k + a_1 y_{k-1} + ... + a_{n_a} y_{k-n_a} = b_1 u_{k-n_k} + ... + b_{n_b} u_{k-n_k-n_b+1} + \epsilon_k \quad (2)$$

The parameters n_a and n_b reflect how strongly previous steps affect the current output, while n_k represents the delay between the effective input u_k and output y_k. For instance, $n_k = 0$ means a direct coupling between input and output. A compact way to write the difference equation is:

$$A(q)y_k = B(q)u_k + \epsilon_k \quad (3)$$

If we consider q as a delay operator, we can interpret $A(q)$ and $B(q)$ as follows:

$$A(q) = 1 + a_1 q^{-1} + ... + a_{n_a} q^{-n_a} \quad (4)$$

$$B(q) = b_1 q^{-n_k} + ... + b_{n_b} q^{-n_k-n_b+1} \quad (5)$$

The white noise term ϵ_k is usually small for a model with high fitness score, so we can extract the adjustable parameters to the following:

$$\theta = [\ a_1\ a_2\ ....\ a_{n_a}\ b_1\ b_2\ ...\ b_{n_b}\ ]^T \quad (6)$$

If we define a column φ_k as follows:

$$\varphi_k = [\ -y_{k-1}\ ....\ -y_{k-n_{a_n}}\ u_{k-n_k}\ ...\ u_{k-n_k-n_b+1}\ ]^T \quad (7)$$

Then, the estimator $\hat{y}_k$ of y_k can be calculated as follows:

$$\hat{y}_k = \varphi_k^T \theta \quad (8)$$

If we have N measurements of input u_k and output y_k, then the goal is to find θ that results in the lowest quadratic error:

$$\epsilon = \frac{1}{N} \sum_{k=1}^{N} (\hat{y}_k - y_k)^2 \quad (9)$$

Using Least Squares Method (LSM) for θ, we can find $\hat{\theta}_k$ that minimizes the estimated error ϵ as follows:

$$\hat{\theta}_k = \left[\frac{1}{N} \sum_{k=1}^{N} \left(\varphi_k \varphi_k^T \right) \right]^{-1} f(N) \quad (10)$$

where:

$$f(N) = \frac{1}{N} \sum_{k=1}^{N} \varphi_k y_k \quad (11)$$

The SISO model shown in equation 2 could be extended to the MISO model, which considers C categories of requests. Each category rate results in a different consumption of resources. If we refer to category i of requests as u^i, then we should derive the new relationship $y = f(u^1, u^2, ..., u^C)$. In this case, the coefficients of request i could be presented as $\left[b_1^i\ b_2^i\ ...\ b_{n_b}^i\ \right]$. The equation 2 will be updated to consider multiple inputs as follows:

$$\begin{aligned} y_k + a_1 y_{k-1} + ... + a_{n_a} y_{k-n_a} &= b_1^1 u_{k-n_k}^1 + ... + b_{n_b}^1 u_{k-n_k-n_b+1}^1 \\ &+ ... \\ &+ b_1^C u_{k-n_k}^C + ... + b_{n_b}^C u_{k-n_k-n_b+1}^C \end{aligned} \quad (12)$$

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

IEEE computer society

Accordingly, new values of θ and φ that consider multiple inputs are as follows:

$$\theta = [\ a_1\ a_2\ ....\ a_{n_a}\ b_1^1\ b_2^1\ ...\ b_{n_b}^1\ .....\ b_1^C\ b_2^C\ ...\ b_{n_b}^C\ ]^T \quad (13)$$

$$\varphi_k = [\ -y_{k-1}\ ....\ -y_{k-n_{a_n}}\ u_{k-n_k}^1\ ...\ u_{k-n_k-n_b+1}^1\ \ ..... \ u_{k-n_k}^C\ ...\ u_{k-n_k-n_b+1}^C\ ]^T \quad (14)$$

After extracting θ parameter and defining φ_k for multiple inputs model, we can use equations 8 to 11 of SISO model to find $\hat{\theta}_k$ that minimizes the estimated error ϵ of MISO model.

In our experiments, we observed that setting $n_a = 2$ and $n_b = 2$ could reduce the parameters search space without degrading model's accuracy. Moreover, we did not consider any time delay by setting n_k value to zero while the sampling window size is 20 seconds, which is long enough to hide small delays of request impact on each tier. Typically, common least square algorithms have polynomial time complexity $O(u^3v)$ when solving v equations with u variables [7]. However, we solve these equations once at off-line time for each resource. At run time, we calculate $\hat{y}_k$ with a linear complexity $O(n)$, where $n = n_a + n_b * C$.

III. Evaluation

Our approach is evaluated using RUBiS benchmark [8]. It is an online auction site developed at Rice University to model basic functions of ebay.com system. RUBiS is implemented using several Enterprise JavaBeans (EJB) container configurations in order to measure their scalability. The original implementation of RUBiS uses a variety of open source software products including JBoss, JOnAS, Tomcat, and Apache. In this paper we consider a multi-tier system comprised of Apache as a web server, Tomcat as an application server, and MySql as a database.

Our physical infrastructure consists of three Dell OptiPlex 980 servers with Intel Core i5-2400 CPU @ 3.10GHz. Memory size was 8 GB on all machines. These machines are connected using a Gigabit-Ethernet switch. The installed hypervisor is VMware ESX 4.1. To monitor VMs utilization, we implemented a java-based client that consumes the web services of each VMware server to get online measurements of VMs resources utilization. We implemented each tier into a different physical host. Moreover, we depend on the vCPU affinity to isolate performance.

A. Models extraction and validation

To cover a variety of workload intensity, we ran RUBiS client with different values of *number of clients* that range from 100 to 1200. Moreover, we ran RUBiS *default traffic*, which includes both browsing and updating requests. The online measurements are merged and synchronized with the sampled logs from load balancer to build a model for each monitored resource. These models will be used later for contention prediction and system scalability.

For validation, we run the experiment again but with different steps that range from 150 to 1250 to avoid using the same traces of training data. In this experiment we intend to validate extracted models and at the same time measure the scalability impact on models fitness. Therefore, we ran two replicas in both web and application tiers. Figure 3 shows the cumulative distribution function (CDF) of absolute error of CPU utilization of web1, app1, and database. We do not show CDF of web2 and app2 while they are very close to CDF of web1 and app2. The figure shows that 90% of the measured absolute errors are less than or equal 2 for web and application replicas. On the other hand, 90% of the measured absolute errors are less than or equal 4.5 for database. In addition to CDF, we calculate the fitness F, using equation 1, for each VM instance participated in the experiment run.

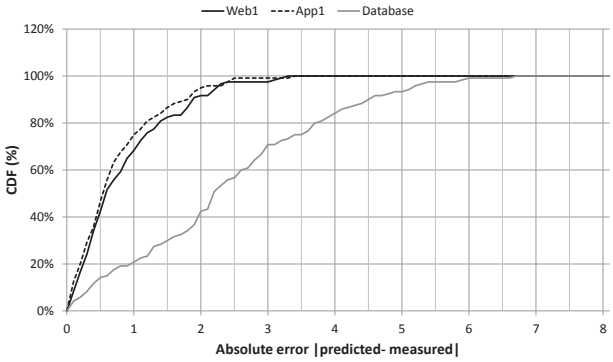

Figure 3. CDF of absolute error of CPU utilization of web1, app1, and database instances

Table I
Fitness of CPU utilization model for two web replicas, two application replicas, and a database calculated by equation 1

Instance:	web1	web2	app1	app2	db
Fitness (%):	95.60	95.75	94.67	94.09	86.85

The fitness values shown at table I, in addition to CDFs at Fig. 3, prove that dividing the measured rate of requests, at equation 14, by the number of replicas results in high accurate models for replicas. Practically, session-based load balancing can result in non-equally distributed traffic to the replicas. However, in all our experiments, even though sticky sessions are enabled at load balancers, the fitness of the models is still high. This confirms the statement of Zhang et al. [4]: "a multi-tier system with a complex session-based workload can be modeled with a transaction-based mix".

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

IEEE computer society

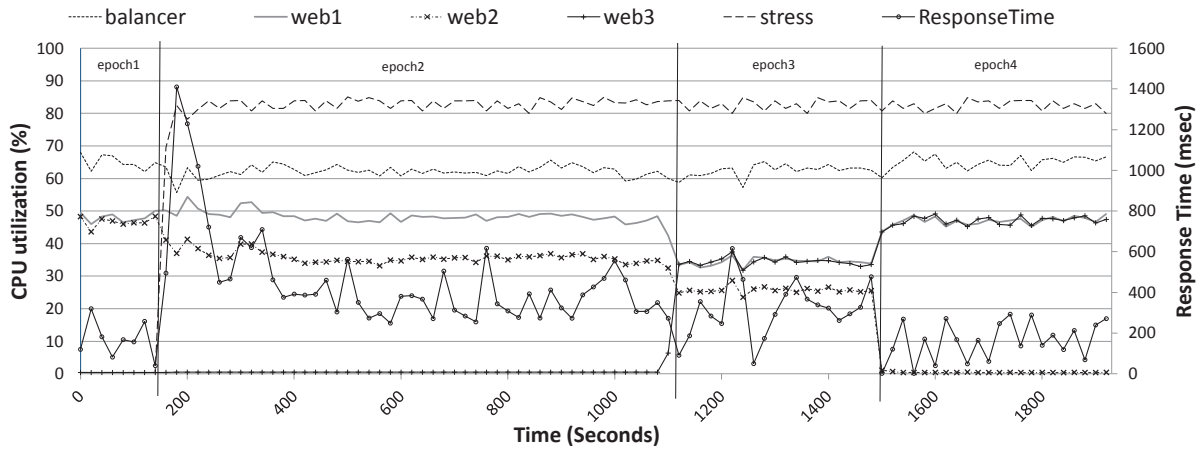

Figure 4. Predicting and eliminating contention that affected web2 VM instance

Another interesting observation was that among many experiments on different system structures, we noticed that the fitness of CPU utilization model of web and application instances is always higher than that of the database. For instance, the fitness of web and application tiers ranges from 90% to 95%, while it ranges from 85% to 90% for the database tier. Zhang et al. [4] and Ahmad et al. [9] have the same observation. Therefore, [9] applied a heuristic approach to increase a model's accuracy. The heuristic approach ignores the difference between the predicted and measured values if it does not have an impact on the system performance. According to [9] experiments, their approach decreases the mean error of CPU model from 12.83 to 11.10. During our experiments, we observed that the rate of miss/hit of database query cache had an impact on measured CPU utilization. Therefore, we are studying the possibility of including cache miss/hit rate into our regression models. However, instead of running additional sensors within each database, we are considering extracting this information from the sequence of the requests. For instance, if "read only " query is followed by "modify" query, then we are quite sure that next instance of any read only query will not hit the cache due to cache invalidation. An example of such database engines is MySql database, which invalidate cache entries for any modified table.

B. Contention prediction

In this section, we demonstrate the system ability to predict and eliminate contention of resources. First we consider running RUBiS client with 1000 simultaneous clients. The experiment run time is divided into four epochs. Each epoch describes a different performance state: epoch1 (0 to 140), epoch2 (140 to 1100), epoch3 (1100 to 1500), and epoch4 (1500 to the end of experiment). Figure 4 shows that both *web1* and *web2* VMs were able to utilize around 48% of the physical core capacity until the end of epoch1. At that moment, a VM called *stress* started competing on the physical core with *web2*. *Stress* is a VM mapped to same physical core with *web2* VM. It runs the command: "stress –cpu 10 –io 8 –vm 2 –vm-bytes 256M". As seen in Fig. 4, *stress* VM has an impact on the whole system performance: First, the response time jumped to 1400 ms second at the moment of starting *stress* VM and stabilized along time interval (380 to 1100 seconds) to be 369 ms, which is 230% higher than the expected response time (i.e., 160 ms). Second, the contention caused by *stress* also influenced *balancer*. Even though they are mapped to different physical CPUs, we noticed a slight decrease in the CPU utilization of *balancer* VM. This decrease is due to the overhead increase at the web tier, which decreased the ability of the multi-tier system to accept more requests. Third, the contention had the most impact on *web2*, where the average CPU utilization dropped down to 35% instead of 48%.

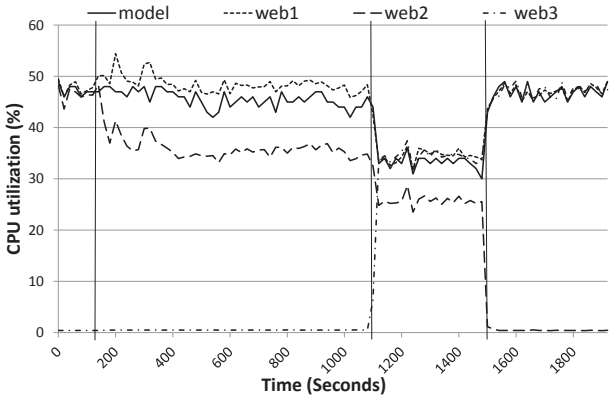

Figure 5. CPU utilization of web tier replicas

Using equation 1, the controller predicts degradation in *web2* performance and starts resolving it at epoch3. As seen in algorithm 2, the first step is to run another replica *web3* to replace *web2*. At that moment, the model of the web tier instances is updated to predict 3 replicas at web

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

IEEE computer society

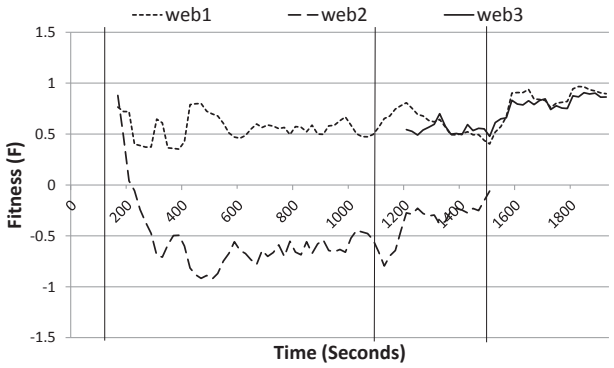

Figure 6. Fitness of web tier replicas calculated by equation 1

tier, as seen in Fig. 5. Also, the fitness is calculated using equation 1 to insure that *web3* has no contention with the other VMs on the physical host. Afterwards, the provisioning controller schedules the contended VM (i.e., *web2*) termination and keeps running *web1* and *web3*. This brings down the response time to 160 ms as it was at the beginning of experiment running at epoch1. Finally, we should note that at epoch3, even thought the system has three web replicas, the VM with contention has influence on response time while a portion of the traffic is still routed to *web2*.

Figure 5 shows both the measured and predicted CPU utilization of each web instance. Epoch 1 and 4 show that the measured CPU utilization closely fits the modeled utilization. However, at Epoch 2, according to contention at web2, the balancer routes more traffic to web1, which results in an increase in measured CPU utilization compared with the modeled CPU utilization. According to model adaptation based on the number of replicas, the average measured values $\overline{y}_k$ in equation 1 is not valid along the whole run of a VM instance. Therefore, at running time of our system, we calculate $\overline{y}_k$ along a specific measurement window (i.e., last 5 measurements). The specified window also helps the *contention predictor*, seen in Fig. 2, to cope rapidly with contentions. Figure 6 shows the fitness of each web instance model. Epoch 4 shows that after replacing the contended instance (i.e., web2) the fitness of web1 and web3 CPU utilization models go again closer to one.

C. Technical discussion

In this section, we discuss some of the technical details that were confronted during our system implementation.

First, *nginx* shows much better performance compared with Apache as a load balancer. During our experiments, we noticed that Apache performance degrades drastically at a high rate of traffic. Moreover, as a process- or thread-driven application, Apache consumes much memory to spawn more processes or threads. On the other hand, as an event-driven application, *nginx* was able to outperform Apache performance even with few processes.

Second, in an IaaS environment, running a new VM instance implies assigning a new IP address to the new instance, which is unknown to load balancer. This requires updating the load balancer with the new IP addresses online. In fact, both Apache and *nginx* enable online reloading configuration file, which contains replications details. However, *nginx* shows no degradation in performance compared with Apache, which interrupts the service temporarily by killing current processes and creating them again.

Third, we implemented our prototype into local infrastructure to have more control over resources during experiments. However, Amazon EC2 [5] has all the tools that support implementing our approach. For example, using Amazon AWS client, a user can monitor, provision, and terminate instances remotely. For load balancing, a customer can either use static load balancers, as seen in our experiments, or a combination of many static balancers and one *Elastic Load Balancer* [10]. *Elastic Load Balancer* is a service available at Amazon EC2 to increase Internet applications reliability. It has many advantages while it allows distributing load among different zones. Moreover, the provider is responsible for its reliability and dynamic scalability to cope with workload demand. Currently, using Amazon CloudWatch, clients can get the number of requests that are manipulated by *Elastic Load Balancer* and the response time of each request. However, there is no information about the request's URL. We hope that Amazon considers such a metric in their development. Until then, our approach remains valid using static load balancer or by collecting and synchronizing access logs from web tier instances.

IV. Related work

Performance evaluation and capacity planning of Internet application have been intensively studied along the last years. To manage applications performance dynamically, systems administrators need models that describe application performance under different workload. In this section, we review two commonly used techniques for modeling Internet applications:

A. Modeling as queues

Queuing theory has been a widely used methodology for modeling system behavior and capacity planning. An early work by Villela et al. [11] only examined the application tier. Each server at application tier is modeled as $M/G/1/PS$ queuing system. Similarly, [12] proposed a $G/G/1$ queuing model for replicated single-tier applications (e.g., clustered web servers). Also [13] modeled the Java application tier of an e-commerce application with N servers as a $G/G/N$ queuing system. Practically speaking, modeling one tier could be useful for specific scenarios, but in reality it neglects crucial features of multi-tier Internet application like caching, bottlenecks shift, and bottlenecks oscillating [3]. This fact limits the ability of single-tier models to

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

IEEE computer society

manage capacity and performance of multi-tier systems. To go over these limitations, Urgaonkar et al. [1] proposed modeling multi-tier system as a closed network of queues, where each queue represents a different tier. Moreover, they consider caching effects and application concurrency limits in their models. Their system is able to predict response time with 95% confidence. However, because they consider a single queue per a tier, their system is able to capture a single resource bottleneck at a time (e.g., CPU utilization or network bandwidth). Furthermore, implementing their approach requires estimating many model parameters such as: arrival rate, service times at each tier, and other parameters related to congestion effects [14]. On the other hand, Stewart, et al. [14] modeled a server as a network of queues representing multiple resources (i.e., CPU, Network, and Disk). This enabled their technique to predict different resources bottlenecks, unlike [1] approach. However, [14] implementation modeled multi-tier system as an open network queue that neglects user thinking time, which is an unrealistic assumption in Internet applications [15], [2], and [1]. Moreover, their approach does not consider the caching effect or application concurrency limit.

B. Statistical modeling

Stewart et al. [16] profiled applications to predict demand on underlying components of online services. Their models account for inter-component communication and placement. The infeasibility of their approach lies in the need for intensive calibration. For instance, to model N components of the system, they should run $O(N^2)$ benchmarks. Moreover, they assume a prior knowledge about components' structure. A later work by Stewart et al. [14] exploits non-stationary workload to obtain the same performance profiles that are extracted in [16] using only lightweight passive measurements (e.g., login, browse, add-to-cart, and checkout rates). Moreover, their improved approach assumes no required knowledge of an internal application structure. Zhang et al. [17] argued that simple categorization approaches, which depend on transaction type to categorize workload [14] [4], result in a fair but not very high accuracy. Therefore, they suggested improving the models' accuracy by iteratively splitting and merging the categories depending on estimated resource usage. Their approach is inspired by [18] approach. The Sharma et al. [18] approach successfully discovered 2 workload categories of PetShop benchmark using a machine learning technique called Independent Component Analysis (ICA). The approach does not require any information about requests URL. Nevertheless, the ICA approach limits the possible number of categories to the number of measurable resources. Zhang et al. [17] overcome this restriction by proposing transaction ICA. So, instead of using ICA solely, [17] start initial categorization of requests using URL, then improve it iteratively by merging homogeneous categories and splitting heterogeneous categories. Ghanbari et al. [19] suggest classifying requests depending on the response time. They validated their approach on two tier system (i.e., web and database server). We are interested in examining the efficiency of such an approach on shared environment where the consolidation can have a strong impact on response time. Sheikh et al. [20] presented an experiment-driven for predicting database response time using Gaussian Process models. A key feature in their approach is its online adaptability to workload change and machine configuration. Currently, their approach targeted databases. However, it is interesting to see it handling http requests of Internet applications, as well as database queries. Jiang et al. [21] profiled services at multi-tier systems looking for constant pair-wise relations between components. These relations (i.e., system invariants) are used for capacity planning and resource optimization. However, authors considered low level metrics (e.g., CPU soft IRQ time and used heap memory size), which increase the system monitoring complexity without crucial contribution to the model accuracy, as we observed in our experiments.

In the light of related work, we can summarize the features of our approach as follows: First, our system is classified as a statistical modeling, while we profiled each application according to incoming requests and the measured resources utilization. Second, unlike [16] and [21] who modeled many components of the system; we modeled only the components that showed high impact on the system's performance to simplify system implementation without decreasing the accuracy. Third, while Urgaonkar et al. [1] consider one resource at a time, we are able to build a model for many resources. Fourth, our approach is considered lightweight while it requires neither internal monitoring as in [12] and [1] nor special knowledge of system components as in [16]. Fifth, just as [14], [16] and [1] we used RUBiS benchmark to build and evaluate our system. Finally, we categorized requests depending on URL and employed prior coarse-grain classification (i.e., cacheable, non-cacheable) to express caching effect on the system performance. Our observations show that prior coarse-grain classification and a careful selection of queries that should be considered in model generation have much impact on model's accuracy. However, we intend to evaluate iterative categorizing techniques suggested by [17] and [19] to exploit the trade-off between the accuracy and the additional implied computation.

V. Conclusion & Future work

In this paper, we proposed a lightweight approach to control system scalability and predict the contention in a shared IaaS environment. Our approach profiles application behavior at each tier of Internet application. We used RUBiS benchmark to build and validate our approach. The extracted models of CPU utilization showed fitness ranges from 90% to 95% for web and application tier, and fitness ranges from 85% to 90% for database tier. Moreover, experiments

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

IEEE computer society

showed the ability of our approach to predict and eliminate the impact of resources contention at shared infrastructure.

Our immediate future work is to improve database tier models and to study the trades-off between the cost of the improvement and the models accuracy. Also, we are going to study the effect of database aging on model's accuracy. Moreover, we plan to develop techniques for automatically categorizing incoming requests. Finally, in our extended research we will consider heterogeneous types of replicas and will examine our approach using more Internet applications.

REFERENCES

[1] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, "Analytic modeling of multitier Internet applications," *ACM Trans. Web*, vol. 1, no. 1, May 2007.

[2] ——, "An analytical model for multi-tier internet services and its applications," *SIGMETRICS Perform. Eval. Rev.*, vol. 33, no. 1, pp. 291–302, Jun. 2005.

[3] S. Malkowski, M. Hedwig, and C. Pu, "Experimental evaluation of N-tier systems: Observation and analysis of multi-bottlenecks," in *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*, ser. IISWC '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 118–127.

[4] Q. Zhang, L. Cherkasova, and E. Smirni, "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, USA: IEEE Computer Society, 2007.

[5] Amazon, "Amazon EC2," 2012. [Online]. Available: http://aws.amazon.com/ec2/, Retrieved: 1st, October 2012

[6] M. Arlitt, D. Krishnamurthy, and J. Rolia, "Characterizing the scalability of a large web-based shopping system," *ACM Trans. Internet Technol.*, vol. 1, no. 1, pp. 44–69, Aug. 2001.

[7] Q. Zhang, L. Cherkasova, N. Mi, and E. Smirni, "A regression-based analytic model for capacity planning of multi-tier applications," *Cluster Computing*, vol. 11, no. 3, pp. 197–211, Sep. 2008.

[8] E. Cecchet, J. Marguerite, and W. Zwaenepoel, "Performance and scalability of EJB applications," *SIGPLAN Not.*, vol. 37, no. 11, pp. 246–261, Nov. 2002.

[9] M. Ahmad and I. T. Bowman, "Predicting system performance for multi-tenant database workloads," in *Proceedings of the Fourth International Workshop on Testing Database Systems*, ser. DBTest '11. New York, NY, USA: ACM, 2011, pp. 6:1–6:6.

[10] M. Tavis, "Web Application Hosting in the AWS Cloud: Best Practices," 2010.

[11] D. Villela, P. Pradhan, and D. Rubenstein, "Provisioning servers in the application tier for e-commerce systems," *ACM Trans. Internet Technol.*, vol. 7, no. 1, Feb. 2007.

[12] B. Urgaonkar and P. Shenoy, "Cataclysm: policing extreme overloads in internet applications," in *Proceedings of the 14th international conference on World Wide Web*, ser. WWW '05. New York, NY, USA: ACM, 2005, pp. 740–749.

[13] S. Ranjan, J. Rolia, H. Fu, and E. Knightly, "QoS-driven Server Migration for Internet Data Centers," in *Proceedings of the 10th IEEE International Workshop on Quality of Service*, Houston, TX, USA, 2002, pp. 3–12.

[14] C. Stewart, T. Kelly, and A. Zhang, "Exploiting nonstationarity for performance prediction," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 31–44, Mar. 2007.

[15] X. Liu, J. Heo, and L. Sha, "Modeling 3-Tiered Web Applications," in *Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, ser. MASCOTS '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 307–310.

[16] C. Stewart and K. Shen, "Performance modeling and system management for multi-component online services," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2*, ser. NSDI'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 71–84.

[17] Z. Zhang and S. Li, "Automatic Fine-Grained Transaction Categorization for Multi-tier Applications," *Cyber-Enabled Distributed Computing and Knowledge Discovery, International Conference on*, vol. 0, pp. 130–138, 2011.

[18] A. B. Sharma, R. Bhagwan, M. Choudhury, L. Golubchik, R. Govindan, and G. M. Voelker, "Automatic request categorization in internet services," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 2, pp. 16–25, Aug. 2008.

[19] H. Ghanbari, C. Barna, M. Litoiu, M. Woodside, T. Zheng, J. Wong, and G. Iszlai, "Tracking adaptive performance models using dynamic clustering of user classes," *SIGMETRICS Perform. Eval. Rev.*, vol. 39, no. 3, p. 15, Dec. 2011.

[20] M. B. Sheikh, U. F. Minhas, O. Z. Khan, A. Aboulnaga, P. Poupart, and D. J. Taylor, "A bayesian approach to online performance modeling for database appliances using gaussian models," in *Proceedings of the 8th ACM international conference on Autonomic computing*, ser. ICAC '11. New York, NY, USA: ACM, 2011, pp. 121–130.

[21] G. Jiang, H. Chen, and K. Yoshihira, "Profiling services for resource optimization and capacity planning in distributed systems," *Cluster Computing*, vol. 11, no. 4, pp. 313–329, Dec. 2008.

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

IEEE computer society