Dynamic request management algorithms for Web-based services in cloud computing

Riccardo Lancellotti, Mauro Andreolini, Claudia Canali, Michele Colajanni
Department of Information Engineering
University of Modena and Reggio Emilia
Email: {riccardo.lancellotti, mauro.andreolini, claudia.canali, michele.colajanni}@unimore.it

***Abstract*—Providers of Web-based services can take advantage of many convenient features of cloud computing infrastructures, but they still have to implement request management algorithms that are able to face sudden peaks of requests. We consider distributed algorithms implemented by front-end servers to dispatch and redirect requests among application servers. Current solutions based on load-blind algorithms, or considering just server load and thresholds are inadequate to cope with the demand patterns reaching modern Internet application servers. In this paper, we propose and evaluate a request management algorithm, namely Performance Gain Prediction, that combines several pieces of information (server load, computational cost of a request, user session migration and redirection delay) to predict whether the redirection of a request to another server may result in a shorter response time. To the best of our knowledge, no other study combines information about infrastructure status, user request characteristics and redirection overhead for dynamic request management in cloud computing. Our results show that the proposed algorithm is able to reduce the response time with respect to existing request management algorithms operating on the basis of thresholds[1].**

I. Introduction

Service providers have now the alternative to create Web-based services on top of cloud computing infrastructures providing virtualized resources. Even in these instances, sudden and unpredictable modifications of the request patterns may affect the performance experienced by the users and the service providers have to introduce suitable request management algorithms. We consider the most common case where the number of virtual servers acquired by the service provider does not change. Hence, spikes and flash crowds must be managed through distributed request redirection algorithms among virtual servers. When a virtual server receives a request, the management algorithm must decide whether it is convenient to process it or to redirect to another virtual server. Since in modern Web-based services the service of a request typically involves access to user session information, with redirection we mean both the migration of already established user sessions and the actual forwarding of the user request to a different server. Decisions concerning request redirection are usually enforced through the evaluation of the load conditions of the local server and (possibly) of its neighbors, and through the comparison of these load values against a fixed, static threshold [1], [2]. Even if this approach has been applied in several fields of ranging from network resource optimization [3] to performance improvement on a local [4], [2] and geographic scale [5], these proposals do not consider pieces of information that could improve redirection decisions, but rely only on a comparison of the server load with static thresholds. This approach has some limits: (a) it does not consider the redirection overhead, that includes the delays introduced by request forwarding and session migration; (b) it does not take into account the computational demand of a user request and, consequently, the impact of its redirection on the load of other servers.

To address these issues, we propose a novel request management algorithm, namely Performance Gain Prediction, that evaluates the performance gain associated to the redirection of a user request. For each user request, the algorithm predicts and compares the response times that would be obtained through local service and through a redirection, then it chooses the most appropriate server accordingly. This is the first paper proposing an algorithm that operates dynamic request management in cloud data centers by combining information about infrastructure status, user request characteristics and redirection overhead.

With the help of a system simulator, we demonstrate that the proposed Performance Gain Prediction algorithm outperforms existing, static threshold-based request management strategies. Our results show that, thanks to the combination of different information about user requests, infrastructure status and management costs, the proposed algorithm is able to estimate the actual gain achievable through user session migration and request redirection, thus guaranteeing good and stable performance across different scenarios.

The paper is organized as follows. Section II presents the problem of request management in cloud data centers. Section III discusses the proposed Performance Gain Prediction algorithm for request management. Sections IV and V discuss the experimental setup and the results of the experimental evaluation. Section VI concludes the paper with some final remarks.

II. System model

In this section we present the model of the cloud system as seen from the service provider offering Web-based services

[1]The authors acknowledge the support of MIUR-PRIN project DOTS-LCCI "Dependable Off-The-Shelf based middleware systems for Large-scale Complex Critical Infrastructures".

to the end users, and we describe the request management process.

A. Web-based services on a cloud computing infrastructure

Service providers exploit the Infrastructure as a Service paradigm of cloud computing to purchase from the cloud provider the hardware resources needed to support their Web-based services. The infrastructure made available to the service provider consists of multiple virtual servers that can be used to host the logical tiers of the Web-based service. The cloud infrastructure provides also a storage facility that is used as a repository for data and for images of virtual machines.

From the point of view of a service provider that is deploying one Web-based service on the Cloud infrastructure a cloud data center is basically composed of servers organized according to a three-tier structure, with front-end, application layer and back-end levels. The user requests arrive at the servers of the front-end level, that distributes client requests among the set S of the servers of the application layer that are dedicated to the Web-based service. As modern Web-based services involve the dynamic generation of contents through complex operations, we focus our attention on the set S of servers hosting the application layer, that represent the most stressed component of the infrastructure. The number of servers in the set S may change dynamically because the service provider can dynamically deploy new server to cope with the end-user demands. However, as operations such as creation and termination of virtual machines represent an expensive task, dynamic provisioning is typically carried out on a coarse-grained time scale, possibly through capacity planning algorithms that are run periodically [6], [7]. Sudden and unpredictable peaks in the end-user demand (also known as *flash crowds*) may not be addressed only through dynamic provisioning because the rapidly changing service access patterns often can not be captured (let alone, be dealt with) by means of coarse-grained decisions. Over provisioning the infrastructure may represent a solution in the short term, but it introduces additional costs for the service provider that are quickly becoming unacceptable [8].

In our system model, we assume that the cloud provider may guarantee to the service provider the following features:

- The virtual machines composing the cloud infrastructure provide adequate performance insulation, which means that from the point of view of the service provider each virtual machine is like a traditional server that does not change its characteristics over time; we model the servers hosting the Web-based service as a time shared processor.
- The service provider has access to monitoring tools that collect and exchange system information about each server in the set S. Information may be collected by the cloud provider and then made available to the service provider [9], or directly by the service provider [10].
- The data used by the application layer are stored on a cloud-based storage facility that is accessible to each server belonging to the set S.

B. Request management

When a user request r is received by a server of the front-end level, r is dispatched towards the application layer; to this aim, the front-end level selects a server in the application layer, that we call $s_a \in S$, which hosts the required Web-based service. The request dispatching performed at the front-end level follows a sticky session approach to direct all the requests belonging to the same user session to one server in the application layer that stores the session information [11]. If the request r belongs to an existing user session, the dispatching operation selects s_a on the basis of a binding table that maps user sessions to servers. If the request belongs to a new session, the dispatching selects a new server s_b that will host the request user session. We assume that the selection of a new server is carried out through a load-blind algorithm, such as Round Robin.

The task of request redirection is carried out by the servers of the application layer, that are modeled as follows. Each server contains a *Local redirector* module that is responsible for the management of requests at a fine-grained time scale, as shown in Figure 1. Request r is received by the server s_a at time t. The request may be processed locally or redirected to a server $s_b \in S$. We recall that in modern systems the service of a request involves access to data, such as the user session information, that must be migrated at the moment of request redirection. For this reason, the redirection operation involves both the migration of user session information and the actual forwarding of request r to the server s_b. We assume that the amount of data to transfer for these tasks is similar for every request as user session information tends to be rather standard, like the size of a client request message.

We can summarize the functions of the local redirector as follows:

1) **Activation of the request redirection**. That is, the local redirector must decide if request r should be processed locally on server s_a or if it should be migrated to another server.
2) **Selection of the remote server**. That is, if request redirection is activated the local redirector must select the server $s_b \in S$ to which the request should be sent in order to improve the response time.

In literature there are multiple solutions to the second question. The available alternatives range from the simplest Round Robin or Random algorithms to load-aware solutions, such as Weighted Round Robin or K-Least Loaded solutions [12]. In this paper we adopt the K-Least Loaded algorithm, that combines good performance and simplicity as required in a complex real-time environment.

On the other hand, the decision about whether a request redirection can improve the user-perceived performance is not clear. State of the art proposals are typically based on thresholds. For example, many proposals of request management techniques for large data centers are based on a single threshold about the system load [5], [2]. When the load on a server exceeds the threshold, redirection is activated to offload the server. Redirection of incoming client requests

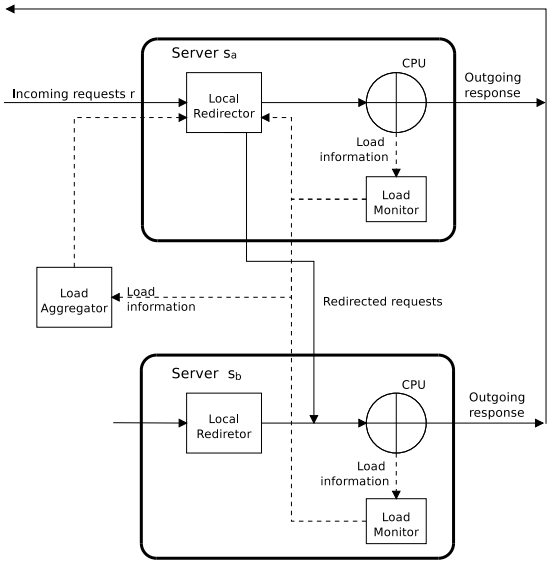


Fig. 1. System model

or sessions remains active until the server load is below the threshold. Other studies, such as [4], [3], exploit a double threshold solution, where redirection is activated by a *High threshold* and remains active until the load is below a *Low threshold*. This latter solution provides a more stable behavior, reducing the number of activation/deactivation of redirection with respect to the single threshold solution, but its higher complexity requires the tuning of the two threshold values to achieve good performance.

While threshold-based algorithms tend to be easy to implement, we believe that considering only the server load in the activation criterion for request redirection may hinder the effectiveness of request management in highly variable scenarios, such as the management of flash crowds in cloud computing data centers. This motivates our choice to propose an algorithm that can exploit multiple information for request management with the goal to improve performance.

III. Performance Gain Prediction algorithm

The Performance Gain Prediction algorithm for request management exploits a combination of multiple information about infrastructure status, user request characteristics and redirection overhead to identify whether it is more convenient to process the request locally or it is better to redirect the request. More specifically, this algorithm aims to estimate the response time that would be achieved on the server s_a receiving the request and on another server s_b in the case of redirection. As a consequence, we can evaluate the *performance gain* that may be achieved by serving the request on server s_b rather than on s_a.

Let $T^r_{s_a}(t), T^r_{s_b}(t)$ be the response times at the data center for the request r, arrived at time t and processed on server s_a and s_b, respectively. As the servers reside in the same data center, we do not consider the network contribution between the client and the data center as a part of the response time. By recalling that s_a is the server that received the request r, and s_b is the remote server (selected through the K-Least Loaded algorithm) to which the request may be redirected, the request redirection is activated only if $T^r_{s_a}(t) > T^r_{s_b}(t)$

Hence, the problem of request redirection requires an estimation of $T^r_s(t)$, for $s \in \{s_a, s_b\}$. To this aim, we consider the following pieces of information:

- The redirection overhead d due to data transfer associated with servicing request r on server s_b
- The service time of the request r
- The load of the local server s_a at time t
- The load of the remote server s_b at time t

The information about the server load status of s_a and s_b is available to the service provider through direct monitoring on the virtual machine hosting the servers or through some monitoring facility offered by the cloud provider.

The redirection overhead, denoted as d, takes into account both the time for the request redirection to the remote server s_b and the time for the migration on server s_b of the user session data. The value of d is calculated through the measurement of the delay introduced by past redirections. We consider a sliding window with the delay of the past k redirection carried out on a server and we apply a moving average filter to determine the value of d that is used in the algorithm.

Let O_r denote the service time of the request r. This information is obtained by classifying requests according to their computational cost. For example, in a Web-based application, we define rules to map URLs into computational cost classes. Let c_r be the computational cost class of the request r. The average service time for a request belonging to that class (that we use as an estimation of O_r) is determined through the statistical analysis method proposed by Pacifici *et al.* [13].

The load of a servers $s \in S$ measured over time denotes a time series $\{Q_s(t), Q_s(t-\Delta t), \ldots, Q_s(t-(k-1)\Delta t)\}$ composed by k samples. As the load measure we consider the process queue length, where Δt is the sampling period of the load monitor and $s \in \{s_a, s_b\}$.

In a highly variable scenario, instantaneous monitored samples are not a good representation of the system load [14] and we need to reduce the variability of the samples through a smoothing function. We define the time series $\hat{Q}_s(t), s \in \{s_a, s_b\}$, as the smoothed data on the process queue length. According to other results available in literature [14], we exploit a linear filtering based on exponential smoothing techniques that provide reliable load representation with low computational cost. We choose the Double Exponential Smoothing (DES), that it is considered a valid solution in highly variable time series characterized by a trend component [15]. As servers are modeled as time-shared processors, a new request entering the generic server s at time t will receive a share of the server computational resources that is $\frac{1}{\hat{Q}_s(t)+1}$. Hence, the response time of request r will be $O_r \cdot (\hat{Q}_s(t)+1)$. We can thus express $T^r_{s_a}$ and $T^r_{s_b}$ as:

$$T^r_{s_a}(t) = O_r \cdot (\hat{Q}_{s_a}(t)+1) \quad (1)$$

$$T^r_{s_b}(t) = O_r \cdot (\hat{Q}_{s_b}(t)+1) + d \quad (2)$$

Equations 1 and 2 represent an approximate prediction of the response time depending on the decision to process locally or to migrate the request. The criterion on performance gain prediction used to activate the request redirection can be expressed as:

$$O_r \cdot (\hat{Q}_{s_b}(t) - \hat{Q}_{s_a}(t)) - d > 0 \quad (3)$$

Hence, we redirect a request if and only if the expected gain on response time due to moving on another server (considering also the redirection overhead d) is greater than 0. It is worth to note that, to avoid ping-pong effects, a request that was already redirected once it is not subject to further redirection.

IV. Case study

In this section we describe the testbed, the workload and the request management algorithms considered for our experimental evaluation.

A. Testbed

To evaluate the performance of the considered request management algorithms, we use a discrete event simulator based on the Omnet++ framework [16]. We assume a cloud data center where 90 servers are dedicated to the deployment of the considered Web-based service, with 50 servers belonging to the application layer and constituting the set S. The CPU utilization and the process queue length on the server are sampled every second among the 50 servers of the set S.

As our request management algorithm is applied to the application layer, we do not detail the model of the front-end and back-end tiers, that are simply considered as additional delays to the response time. A further parameter of interest for the performance evaluation is the redirection overhead d, that we model as a normal distribution, with $\sigma^2/\mu = 0.5$. The model is based on preliminary experiments carried out with a prototype of a Web-based application where user profile information (stored as session variables) is migrated using the RMI mechanism provided by the J2EE platform.

B. Workload

To evaluate the performance of the considered algorithms for request management we use a synthetic workload including user requests belonging to 3 different computational cost classes: the 50% of the requests have a service time ranging between 0.1 and 0.5 seconds, the 25% between 0.6 and 1.9 seconds, and the remaining 25% between 2 and 3.5 seconds [17]. We model the user session duration and the distributions of the inter arrival times of user requests within a session as in [18].

As our goal is to evaluate how the request redirection algorithms can cope with sudden variations in the request patterns, we consider that half of the servers in a building block receive a static load while the remaining 50% of the servers must face a surge in the request intensity that may be double with respect to the static load. This setup represents a scenario where a flash crowd hitting the data center is not distributed evenly over the servers and is consistent with other studies on request management in Web distributed systems [5]. We consider a request pattern, namely *Ramp*, where the request intensity sustained by the server facing the surge experiences a gradual increment passing from 250 clients up to 375 clients at the maximum peak. The peak is followed by a similar gradual decrease. The experiments lasted for 15 minutes.

C. Algorithms

For our analysis we consider three request management strategies: the proposed *Performance Gain Prediction* algorithm, a *Local* algorithm that never triggers request redirection and is used as the worst case scenario, and a *Threshold-based* algorithm that represents a state-of-the-art solution for request management [2], [1]. The Threshold-based algorithm activates redirection on the basis of a local knowledge about the server load. To evaluate the server load metrics, this algorithm relies on the CPU utilization $\rho_{s_a}(t)$ [1], [2], because it is bounded in the $[0, 1]$ interval and is more convenient than process queue length that has no maximum value. For a fair comparison, we apply the DES smoothing techniques also to the CPU utilization to reduce the effect of high variability in samples that could hinder the performance of the Threshold-based algorithm. For every incoming request, the load of the server s_a is evaluated: if it exceeds a given threshold Thr, the request r is redirected and the corresponding user session is migrated.

If the redirection is activated, the remote server is selected through the K-Least Loaded algorithm. To ensure a fair comparison between the Performance Gain Prediction and the Threshold-based algorithms, we consider that the K-Least Loaded algorithm relies on the same load metric, that is the process queue length, to identify the K=3 servers with the lowest load [19]. We performed some experiments with different values of the K parameter, ranging from 2 to 5, but this does not change the results of the comparison between the redirection algorithms.

The condition for request redirection is expressed as: $\hat{\rho}_{s_a}(t) > Thr$, where the value of $Thr = 0.7$ is chosen on the basis of preliminary experiments.

V. Performance results

To compare the performance of the different request management algorithms, we evaluate the response time at the data center for the user requests. Figure 2 shows the 90-percentile of the response time as the average redirection overhead ranges from 0.1 to 2 seconds. The graph demonstrates that the Performance Gain Prediction algorithm outperforms the other alternatives, with a reduction in the response time of more than 25% with respect to the Threshold-based algorithm and close to 100% with respect to the Local algorithm.

Figure 2 allows also to appreciate the impact of redirection overhead on the user-perceived performance. We observe that, with the obvious exception of the Local approach, an increase in the redirection overhead results in a linear growth of the response time. For the Threshold-based algorithm this result is expected because the redirection overhead is not taken into account when performing request direction. The Performance Gain Prediction algorithm experiences a similar behavior for

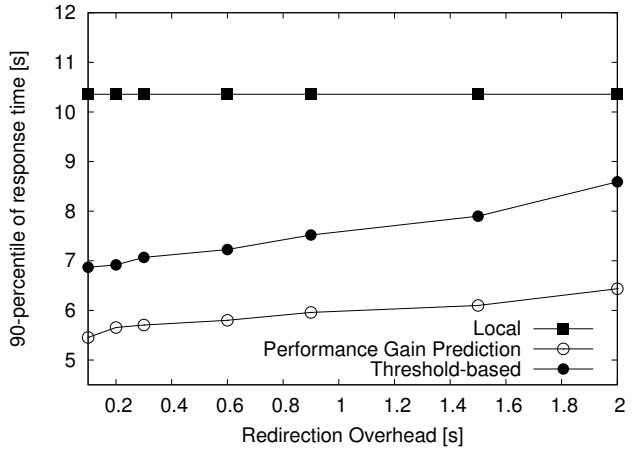


Fig. 2. Response time

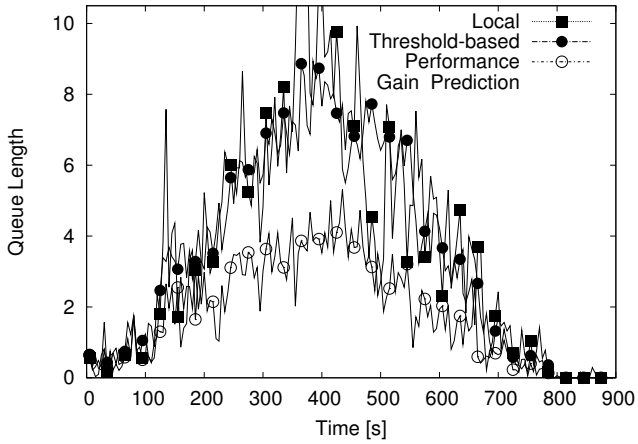


Fig. 3. Process queue length

a different reason. Even if the algorithm takes into account the redirection overhead d, the worst performing 10% of the requests (that determine the 90-percentile of response time) are typically requests subject to redirection, and, as a consequence, the overhead has a direct impact on this metric.

To better understand the reasons for the performance gain of the Performance Gain Prediction algorithm, we study the temporal evolution of the process queue length on the servers throughout the experiments. Figure 3 shows the maximum value of the process queue length among the set of servers S measured at intervals of one second during the experiment. We observe that, as expected, the lack of redirection in the Local algorithm determines high values and oscillations for the process queue length. Also the Threshold-based algorithm is characterized by high values of the queue length. Indeed, the algorithm considers the CPU utilization $\hat{\rho}_s$ for the activation of the redirection, but the non-linear relationship between CPU utilization and process queue length reduces the quality of the Threshold-based solutions when the server load is high. On the other hand, the Performance Gain Prediction algorithm is successful in guaranteeing a queue length almost halved even in the case of a traffic surge, if compared to the alternatives.

The percentage of requests redirected by each algorithm is shown in Table I for two values of the redirection overhead d considered in our experiments. The Threshold-based algorithm presents the same amount of redirected requests for the two values of d. This effect is expected because the algorithm does not consider the value of d. On the other hand, the Performance Gain Prediction algorithm, that considers d as a parameter to decide when redirection is to be activated, shows a significant variation in the amount of redirected requests as a function of the redirection overhead.

These results evidence that the Performance Gain Prediction algorithm is characterized by an amount of redirected requests that is significantly lower (from 3 to 5 times less) when compared to the Threshold-based alternative. The high amount of redirection operations carried out by the Threshold-based algorithms is one of the reasons for the poor performance of this algorithm.

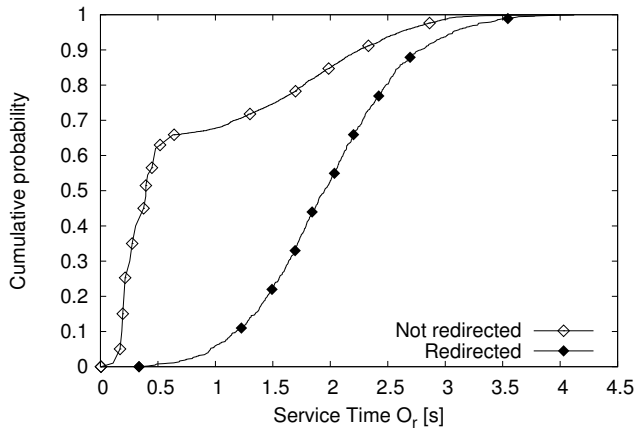
TABLE I
PERCENTAGE OF REDIRECTED REQUESTS

	Redirected requests	
Redirection overhead	**Performance Gain Prediction algorithm**	**Threshold-based algorithm**
$d = 0.1$ s	12%	67%
$d = 2$ s	21%	67%

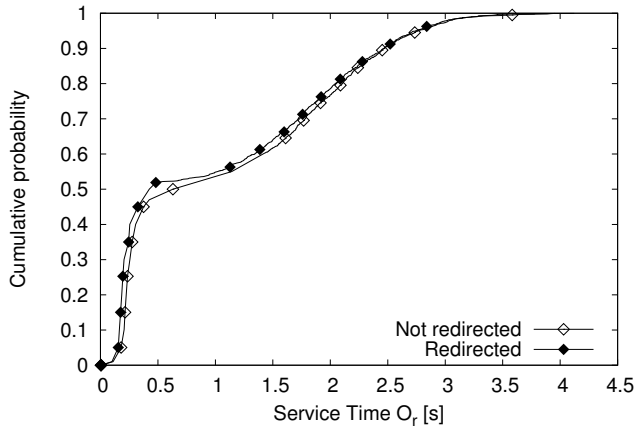
The activation criterion in Equation 3 can be used to explain why such a low number of redirection operations is sufficient for the Performance Gain Prediction algorithm to achieve good results. The algorithm predicts the performance gain due to different load between the servers s_a and s_b and compares it with the redirection overhead. Figure 4 shows the cumulative distribution of the service times O_r for requests that are redirected and processed locally for the Performance Gain Prediction and Threshold-based algorithms. The results refer to the scenario with an average redirection overhead of 0.6 seconds. Figure 4(a) shows that for the Performance Gain Prediction algorithm, the redirected requests are characterized by service times significantly higher with respect to requests that are processed locally (more than 60% of the locally-processed requests have service times in the order of less than 0.5 seconds). This result confirms that the good performance of the Performance Gain Prediction algorithm is motivated by its ability to avoid redirection of requests with low service time, where redirection overhead would hinder the benefit achieved through redirection. On the other hand, the Threshold-based algorithm (shown in Figure 4(b)) does not take into account the computational cost of the requests and the service time of redirected requests follows the same distribution of requests processed locally. As a consequence, a significant number of requests with low service time are redirected, with limited or no performance gain at all from a user point of view.

VI. CONCLUSIONS

Popular Web-based services require cloud data centers to scale to the ever increasing user demand. The mechanisms



(a) Performance Gain Prediction algorithm



(b) Threshold-based algorithm

Fig. 4. Service time

for the location of applications and data over the servers of the data centers, and the subsequent request management algorithms, are typically based on a coarse-grained approach that may be unable to face temporary and unpredictable flash crowds.

In this paper, we propose the novel Performance Gain Prediction algorithm, that evaluates the response time of a request in the cases of local processing and of redirection by exploiting different information about user requests, infrastructure status and management costs. The proposed algorithm is able to guarantee good and stable performance across different workloads and scenarios operating under real-time constraints; our experiments show that the algorithm can provide a performance gain close to 25% on response time if compared to existing solutions for request management based on static thresholds.

The motivation for the good performance of the Performance Gain Prediction algorithm is twofold: first, the global number of redirected requests is reduced; second, the proposed algorithm tends to redirect requests characterized by a high service time. These requests are more likely to guarantee an improvement in user-perceived performance when processed by an alternative server with lower load. Moreover, the impact of the redirection overhead on the response time for these requests is almost negligible.

REFERENCES

[1] T. Wood, P. Shenoy, and Arun, "Black-box and Gray-box Strategies for Virtual Machine Migration," in *Proc. of the 4th USENIX Symposium on Networked Systems Design And Implementation (USENIX'07)*, Santa Clara, CA, Jun. 2007.

[2] A. Singh, M. Korupolu, and D. Mohapatra, "Server-storage virtualization: Integration and load balancing in data centers," in *Proc. of the 2008 ACM/IEEE Conference on Supercomputing*, Nov. 2008.

[3] P. Felber, T. Kaldewey, and S. Weiss, " Proactive hot spot avoidance for Web server dependability," in *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS2004)*, Oct. 2004, pp. 309–318.

[4] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, "Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services," in *Proc. of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI2008)*, Apr. 2008.

[5] M. Andreolini, S. Casolari, and M. Colajanni, "Autonomic request management algorithms for geographically distributed internet-based systems," in *Proc. of the 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO'08)*, Washington, DC, USA, Oct. 2008.

[6] B. Abrahao, V. Almeida, and J. Almeida, "Self-adaptive SLA-driven capacity management for Internet services," in *Proc. of 17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM*, Dublin, Ireland, Oct. 2006.

[7] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. N. Tantawi, "Dynamic placement for clustered Web applications," in *Proc. of the 7th International Conference on World Wide Web*, Edinburgh, Scotland, May 2006.

[8] P. Ranganathan, "Recipe for efficiency: principles of power-aware computing," *Communications of the ACM*, vol. 53, no. 4, pp. 60–67, 2010.

[9] J. Shao, H. Wei, and Q. Wang, "A Runtime Model Based Monitoring Approach for Cloud," in *Proc. of Cloud Computing 2010*, 2010.

[10] A. Ciuffoletti, "Monitoring a virtual network infrastructure: an IaaS perspective," *ACM SIGCOMM Computer Communication Review*, 2010.

[11] Amazon Web Services, "Elastic Load Balancing Developer Guide," technical Documentation, 2010.

[12] V. Cardellini, E. Casalicchio, M. Colajanni, and P. S. Yu, "The state of the art in locally distributed Web-server systems," *ACM Computing Surveys*, vol. 34, no. 2, pp. 263–311, 2002.

[13] G. Pacifici, W. Segmuller, M. Spreitzer, and A. Tantawi, "Dynamic estimation of CPU demand of Web traffic," in *Proc. of the 1st International Conference on Performance evaluation methodolgies and tools (VALUETOOS'06*, Pisa, Italy, Oct. 2006.

[14] M. Andreolini, S. Casolari, and M. Colajanni, "Models and framework for supporting run-time decisions in Web-based systems," *ACM Transactions on the Web*, vol. 2, no. 3, 2008.

[15] S. Everette and J. Gardner, "Exponential smoothing: State of the art," *Journal of Forecasting*, vol. 4, 1985.

[16] "OMNeT++ Discrete Event Simulation System," 2008, – http://www.omnetpp.org.

[17] C. Canali, M. Colajanni, and R. Lancellotti, "Performance impact of future Mobile-Web based services on the server infrastructure," *IEEE Internet Computing*, vol. 13, no. 2, Apr. 2009.

[18] F. Schneider, A. Feldmann, B. Krishnamurthy, and W. Willinger, "Understanding Online Social Network Usage from a Network Perspective," in *Proc. of the 9th ACM SIGCOMM Internet Measurement Conference (IMC'09)*, Chicago, Illinois, USA, Nov. 2009.

[19] V. Subramani, R. Kettimuthu, S. Srinivasan, and S. Sadayappan, "Distributed job scheduling on computational Grids using multiple simultaneous requests," in *Proc. of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC'02)*, Edinburgh, Scotland, Jul. 2002.