Trend-based load balancer for a distributed Web system

Mauro Andreolini, Sara Casolari, Michele Colajanni
Department of Information Engineering
University of Modena and Reggio Emilia
{mauro.andreolini, sara.casolari, colajanni}@unimore.it

Abstract

The unexpected and continuous changes of the workload reaching any Internet-based service make really difficult to guarantee a balanced utilization of the server resources. In this paper, we propose a novel class of state-aware dispatching algorithms that take into account not only the present resource load but also the behavioral trend of the server load, that is, whether it is increasing, decreasing or oscillating. We apply one algorithm of this class to a multi-tier Web-based system and demonstrate that it is able to improve load balancing of the most critical server resources.

1 Introduction

The infrastructures that support Internet-based services are critical systems because they are typically designed and provisioned on the basis of some ideas about the workload reaching them but without any guarantee about the real characteristics and service demands. For these reasons, satisfying high performance, scalability and availability requirements remains a tough issue. An adequate management of these complex systems requires several efficient algorithms for load balancing the server resources [9], for controlling admissions in the case of unexpected hotspots [14, 22], for process dispatching and redirection even at a geographical scale [15]. Most of these algorithms and mechanisms are based on an evaluation of the load conditions of system resources (e.g., CPU, disk, network, memory) through the periodic sampling of *resource load measures* (e.g., CPU utilization, process load, disk throughput, bandwidth). The problem is that the behavior of a resource seen from samples appears extremely variable with several noise components. Hence, these measures are of little help for distinguishing overload conditions from transient peaks, for understanding load trends and for anticipating future conditions, that are of utmost importance for taking correct management decisions.

We propose a novel approach where run-time management algorithms do not consider just the values coming from measurements, but they can take decisions by evaluating also the behavioral trend of the server load, that is, whether it is increasing, decreasing or oscillating. In particular, in this paper we focus on state-aware dispatching algorithms in a locally distributed Web system, but the reader should be aware that other run-time management algorithms could benefit of the proposed approach.

Other reasons for considering behavioral trend instead of simple measures come from the observation that the workload reaching most Internet-based systems is typically non stationary and it is characterized by heavy-tailed distributions [4, 16] and flash crowds [21] that is, sudden peaks of requests that may exceed the capacity of a resource. These characteristics contribute to augment the skew of resource sample distributions. Moreover, raw data tend to become obsolete rather quickly [17], with the consequence that long sample periods reduce the effectiveness of resource load measures as indicators of the real load conditions. These characteristics of server-based measures make really difficult (if not impossible) to deduce a reliable instantaneous load representation and a clear description about the load trend of a resource, for example to find out whether it is offloading, overloading or stabilizing. Hence, it is important that the behavior of system resources supporting Internet-based services are not inferred from raw data, but through some *resource load representation*.

In this paper, we present the main idea of the load trend-aware algorithms and propose an application in the context of a dispatching algorithm for a locally distributed Web-server system. To this purpose, we implement a prototype of a modern multi-tier Web-based system that integrates a load trend-aware dispatching algorithm in the front-end dispatcher of the cluster system. Our experimental results show that the proposed trend-aware algorithm is able to improve server load balance and the overall performance of the distributed Web system with respect to a traditional weighted round-robin algorithm [20] that is one of the preferred load-aware policies because of its simplicity and sta-

bility. We should consider that there is a large literature on load balancing algorithms for Web cluster systems (e.g., [9] and therein citations, [11, 23]), but most of them refer to another generation of Web systems, where requests were mainly for static resources and dispatching was carried out mainly among HTTP servers. In modern Web systems, almost all the responses are dynamically generated and the load balancer forwards the large majority of requests directly to the application servers (additional details are in Section 2). In this context, it is quite innovative to find an algorithm that shows both a better stability of the system node loads and a reduction of the overall response time for a Web resource.

The remainder of this paper is organized as following. Section 2 describes the Web cluster architecture and the nature of the load measures that can be obtained from the system monitors. Section 3 presents the load trend approach and offers a mathematical motivation for its use. Section 4 proposes an example of load trend-aware algorithm that is integrated into the request dispatcher of a modern Web cluster. Section 5 presents the experimental results. Section 6 concludes the paper with some final remarks.

2 Web systems and measures

In modern Web systems, the large majority of responses are dynamically generated. Hence, the traditional multi-tier architecture consisting of the HTTP server layer, the application server layer and the database server layer is being replaced by architectures where the application servers play a more important role.

We cannot enter into the details of the complexity of the modern Web systems consisting of a large set of different servers providing several applications. A simplified, still realistic, version of the architecture that we use as testbed for measurements and for the experimental results, is described in Figure 1. The system is based on the implementation presented in [8]. The application servers are deployed through the Tomcat servlet container, and are connected to MySQL database servers. In our experiments, we exercise the system through real traces; each experiment extents for 30 minutes.

Managing large numbers of requests (possibly hotspots) among these locally distributed systems requires smart policies that must take important decisions at run-time. All these algorithms have the necessity of evaluating the load condition of hardware and software system resources and the ability of recognizing and, possibly, forecasting the critical situations. The main problem is that the view of any resource load that is obtained from system monitors is extremely variable. In Figure 2(a), we report the typical behavior of the CPU utilization in a Web-based system that is reached by a stationary request arrival rate.

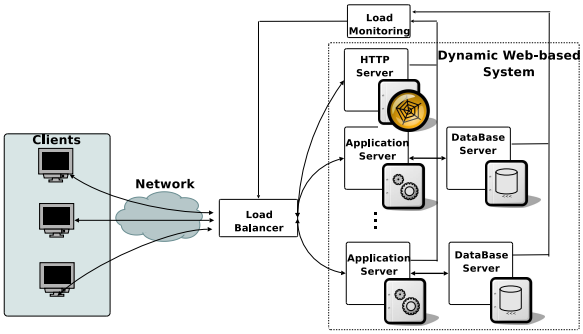
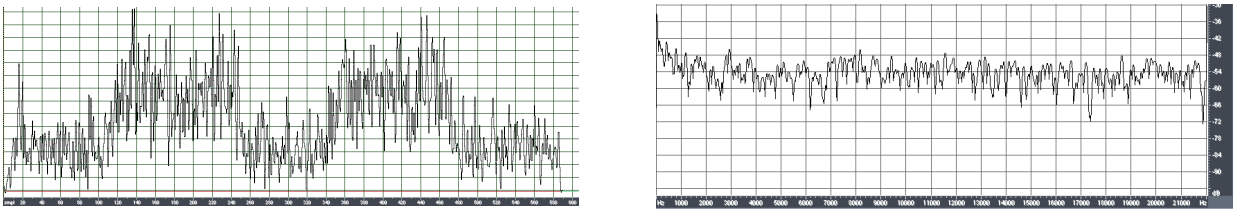


Figure 1. Web system architecture

From Figure 2(a), we can have a first idea that any run-time decision based on these highly variable load values may lead to wrong actions, because, at a certain instant, it is impossible to judge whether and when a resource is really overloaded or not. We give a mathematical confirmation of the visual conclusion by evaluating the spectral analysis of the measurement values shown in Figure 2(b). This figure is obtained by computing the Fast Fourier Transform of the signal coming from the CPU utilization measures [6]. From these results, we can see that the CPU utilization signal has a uniform distribution of the noise at all frequencies, that is typical of a white noise perturbation [10]. A white noise presence means that there is an uncorrelated behavior among all the measured values, hence every signal characterized by white noise appears unpredictable with respect to the previous values.

We should observe that the analysis and conclusions shown for the CPU, are valid for the large majority of the system resources, such as disk utilization and network throughput. Hence, unlike other contexts [5, 13], we claim that for systems supporting Internet-based services it is not convenient to let managing algorithms work directly on resource measures that exhibit white noise perturbations. A first step to address the high variability issue coming from measurements was done in [1]. In that paper, we demonstrated that it is preferable to operate on a "representation" of the load behavior of the system resources instead of working on direct resource measures. To reduce the noise of the resource samples, we proposed the use of aggregation models, such as the exponential moving average. These models work finely even when the offered load is characterized by heavy-tailed distributions, but it is rather stationary. Unfortunately, the load reaching an Internet-based system is stationary just on short terms and, even in absence of flash crowds, it tends to be oscillatory [3].

In this paper, we observe that the aggregation models are useful to reduce the noise of the resource measures, but they



(a) Measurements **(b) Data spectrum**

Figure 2. Spectral analysis of the CPU utilization measures at the database server

offer a limited view of the load condition of a resource, in the sense that, even when they are aggregated, they offer only an instantaneous description. On the other hand, we think that it is important for a run-time dispatching algorithm to know not only an instantaneous load representation, but also some information about how a resource has reached that load state and where it is going: is it stable, increasing, decreasing, oscillatory?

Hence, we propose an algorithm that bases its dispatching decisions both on the resource load representation and on its *behavioral trend*. For this reason, we call it *load trend-aware algorithm*. The idea is that a twofold view should improve the decisions and the performance even when the system is subject to non stationary workloads.

3 Trend-aware strategy

The load representation of a resource is evaluated as a function of multiple resource measures in the past observation interval. These functions should offer to the management algorithm a representative view of the load conditions and any other information that is useful for its objectives. Albeit the load representation offers a view of the load condition better than that achievable by simple measures, we think that it is not sufficient to understand the behavior of the system load when multiple resources are involved. The trend-aware strategy is characterized by three steps:

- the identification of adequate load resource indexes;
- the computation of a *load representation* that gives a fairly reliable interpretation of the load state in the last observation interval;
- the evaluation of the *load trend* that gives information about how the observed resource is behaving with respect to the past load representations: is it increasing, decreasing or oscillating?

The importance of considering the load trend comes from the necessity of comparing the resource load not only in absolute terms. Indeed, a similar approach would lead to wrong management decisions when the workload is not stationary and the resource load measures are extremely variable. Let us consider, for example, the request dispatching problem in a simple distributed system consisting of two servers. Moreover, let us assume that the load representation vectors in the past three intervals are equal to [0.3, 0.4, 0.5] and [0.8, 0.7, 0.6] for server 1 and 2, respectively. A typical dispatching algorithm would probably assign an incoming request to server 1 because in absolute terms is less loaded than server 2. On the other hand, we think that server 2 should receive the request, because its load is decreasing while the load of the server 1 keeps increasing, although in absolute terms the first load is lower than the second load.

For the description of a trend-aware algorithm, we need to compute a reliable load representation of an adequate resource index, and then to evaluate the load trend. To this purpose, for each system resource, we consider the function $Load(s_i, \ldots, s_{i-n+1}) : \mathbb{R}^n \rightarrow \mathbb{R}$ that, at time t_i, takes as its input the resource measures $s_i, \ldots, s_{i-n+1}$ and gives a representation of the resource load conditions (namely, l_i). A continuous application of $Load$ produces a series of load values filtered from outliers and yields a more stable view about the resource load. Typical examples of functions for computing the load representation are the so called moving averages that are widely adopted in econometrics [18]. These models smooth the peaks of the resource measures, but they introduce some delays in the load representation process. We have shown that these models work finely when the amount of load (including the number of users) is almost stationary even if the workload is characterized by heavy-tailed distributions.

When the offered load is variable, as it typically occurs even when there are no hotspots, we find useful to integrate the load representation with a *trend load analyzer* $Trend(l_i, \ldots, l_{i-m+1}) : \mathbb{R}^m \rightarrow \mathbb{R}$. This analyzer takes as its input the last values of the resource load representation $Load$, and extracts a behavioral trend (for example, *ascending*, *descending*, or *oscillating*) out of these values. The load representation and the trend can be used by a *trend-aware algorithm* to compute the *control signals* that drive its decisions. From this approach we may derive a new dispatching algorithm that bases its decisions both on the re-

source load representation and on the load trend behavior.

4 A trend-aware algorithm for load balancing

In this section we illustrate the design of a trend-aware representation process that is used to balance the load across the different nodes of a distributed infrastructure. Although the proposed mechanism has been implemented and integrated into the system presented in Figure 1, we think that the proposed framework can be applied to other distributed systems.

4.1 Load representation

Throughout this paper, we assume that a multi-tier distributed system is characterized by different *computational chains* (briefly, *chains*) $C_1, \ldots, C_C$, where each chain is a set of nodes that cooperate to receive a request, generate a response and send it back to the client. We represent the load of a computational chain through the load of its most representative resource, that corresponds to the system bottleneck. Thus, we will consider the vector $\overrightarrow{R} = (R_1, \ldots, R_C)$ of the representative resources in the system (and the corresponding load representations) as the basis for load balancing. In the considered system, there is no doubt that the representative resources are the CPU utilizations of the database servers [12].

The considered load representation is the Exponential Moving Average that is the weighted mean of the n resource measures of the vector $\overrightarrow{s_i} = (s_i, \ldots, s_{i-n+1})$, where the weights decrease exponentially. An EMA-based load representation, at time t_i, for the representative resource of chain C_c is equal to:

$$l_i^c(\overrightarrow{s}_i) = \alpha * s_i + (1-\alpha) * l_i^c(\overrightarrow{s}_{i-1}) \qquad (1)$$

where the parameter $\alpha = 2/(n+1)$ is the *smoothing factor*. The initial $l_i^c(\overrightarrow{s}_{n-1})$ value is initialized to the arithmetical mean of the first n measures:

$$l_i^c(\overrightarrow{s}_{n-1}) = \frac{\sum_{0 \le j \le n} s_j}{n} \qquad (2)$$

The number of considered resource measures is a parameter of the EMA-based load representation [1]. A preliminary analysis of the statistical characteristics of the considered workload and related resource measures lets us conclude that working on the 60 past measures guarantees an adequate load representation.

4.2 Load trend analyzer

Let $\overrightarrow{l}_i^c = (l_i^c, \ldots, l_{i-m+1}^c)$ be the set of m past values of the load representation concerning the chain C_c. A sequence of m values identifies 3^{m-1} different trends. For example, if we consider two load points, then we have three possible trends: increasing, decreasing and stable. Augmenting the m value, the number of possible combinations of the trends increases and their management becomes problematic. In this paper, we find convenient to consider $m = 3$ in order to solve the trade-off between a simple management and a representative trend. As shown in Figure 3, when $m = 3$, there are 9 possible load trends that can be grouped into 4 classes: increasing, decreasing, oscillating and stable. In particular,

- an "increasing trend" denotes a continuous increment of the resource loads, that is, $l_{i-2}^c < l_{i-1}^c < l_i^c$;
- a "decreasing trend" characterizes a continuous reduction of the resource loads, that is, $l_{i-2}^c > l_{i-1}^c > l_i^c$;
- an "oscillating trend" appears when the past values of load representations have an alternating tendency, for example when $l_{i-2}^c > l_{i-1}^c < l_i^c$ or $l_{i-2}^c < l_{i-1}^c > l_i^c$.
- a "stable trend" is included only for completeness reasons, because it is unlikely that $l_{i-2}^c = l_{i-1}^c = l_i^c$.

Let $\mathcal{EC} = (increasing, decreasing, oscillating, stable)$ be the set of classes associated to the trend of the last 3 values of the load representation. The *load trend analyzer* is a function $Trend(\overrightarrow{l}) : (l_i^c, l_{i-1}^c, l_{i-2}^c) \rightarrow \mathcal{EC}$, that takes the last 3 values of the load representation and maps them into one of the classes $ec_i \in \mathcal{EC}$.

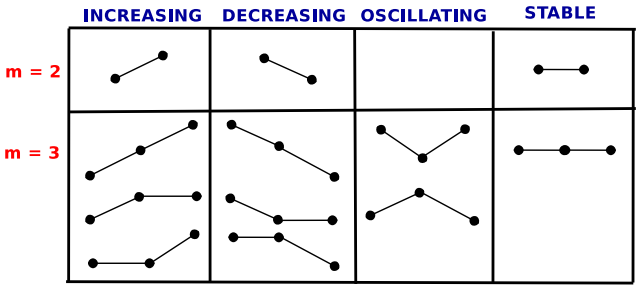


Figure 3. Trend load representation

4.3 Trend-aware algorithm

We introduce a novel trend-aware dispatching algorithm that takes advantages of some resource load representation and information about its trend. The main goal of this algorithm is to improve load balancing especially in the presence of unstable workloads. To this purpose, the algorithm produces control signals that are set as the maximum work capacities of the computational chains. The control signals are continuously adapted to workload changes so that the

load representations of the considered resources converge to the same value, that we call *reference load* (L_i, at the evaluation phase i). The following important operations need to be carried out.

Reference load. There are many possible choices for a reference load of the server nodes. For the scope of this paper, we consider the arithmetical average of the last load representation l_i^c of each computation chain C_c that is, $L_i = \frac{\sum_{c=1}^{C} l_i^c}{C}$ (see Figure 4).

Direction of the control action. The algorithm uses a control signal to drives the quantity of work that should be assigned to each chain. To this purpose, it compares the reference load L_i with the load representation of each chain l_i^c to understand whether its control signal has to be increased or reduced. If the most recent load representation exceeds the reference load ($l_i^c > L_i$), the *control direction* cd_i^c is set to -1 and the algorithm reduces the number of assignments to this chain. Otherwise, if the most recent load representation is below the reference load ($l_i^c < L_i$), then cd_i^c is set to +1 and the algorithm amplifies the control signal.

Strength of the control action. Once known the direction of the control signal, the algorithm has to decide a proper value of each control action. To this purpose, it evaluates the effects of the previously applied control signals. It verifies whether the load across different computational chains is balanced by computing the *load error* ϵ_i^c as the difference between the last load representation l_i^c and the reference load: $\epsilon_i^c = \|l_i^c - L_i\|$. The goal is to choose the control signal strengths so that ϵ_i^c is kept reasonably close to 0 for all the chains.

Choosing the strength of the control signal cs_i^c proportionally just to the value of the last ϵ_i^c has a drawback, because the dispatching algorithm cannot distinguish when the load representation l_i^c is reaching or leaving the reference load L_i. On the other hand, following the idea that the trend and not just the last value should be taken into account, we want to contrast an increasing load error trend through a higher control signal strength, and to reduce the control signal strength when the load error decreases (even to prevent oscillations). For this reason, the proposed algorithm does not evaluate just ϵ_i^c, but it analyzes the trend of the last three load errors that are described by the vector $le_i^c = (\epsilon_i^c, \epsilon_{i-1}^c, \epsilon_{i-2}^c)$ for each chain c. The control signal strengths are computed according to the proper load error trend and stored in the *difference signal* Δcs_i^c variables. The difference signal will be summed to (or subtracted from) the previous control signal for the chain c. The computation of the difference signal is as follows. Its initial value corresponds to the lowest signal and it should never exceed the range of the control signals. If the load error trend is increasing, the previous control signals are not balancing the load. Hence, the algorithm doubles the strength of the difference signal with respect to the previous phase ($\Delta cs_i^c = \Delta cs_{i-1}^c * 2$). If the load error trend is decreasing, the previous control signals are beginning to influence load balancing. In this case, the algorithm reduces the strength of the difference signal: ($\Delta cs_i^c = \Delta cs_{i-1}^c/2$). If the load error trend oscillates, the strength of the difference signal is reduced: ($\Delta cs_i^c = \Delta cs_{i-1}^c/2$).

At each step, the dispatching algorithm computes the new control signals for every chain as $cs_{i+1}^c = cs_i^c + cd_i^c * \Delta cs_i^c$, and scales this value to an appropriate interval. In our implementation, these capacities are used as the weights of a trend-aware Weighted Round Robin algorithm that dispatches requests in a locally distributed Web system.

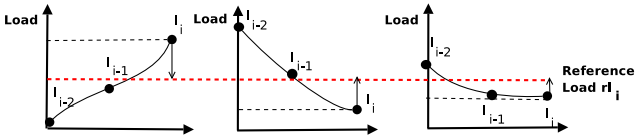


Figure 4. Reference load and trends

5 Experimental results

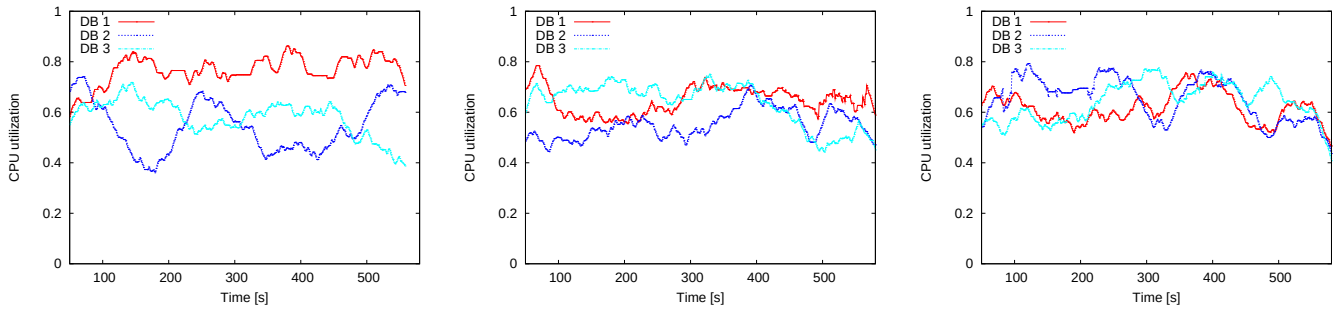
In this section we evaluate the performance of the trend-aware algorithm by considering two workload models:

Stationary scenario: this model emulates a stable workload that is characterized by a population of 120 emulated browsers issuing requests for 10 minutes;

Non stationary scenario: this model denotes a workload characterized by a gradual increment of the population from 80 up to 200 emulated browsers during 5 minutes; the increment is followed by a symmetric decrement of the client population.

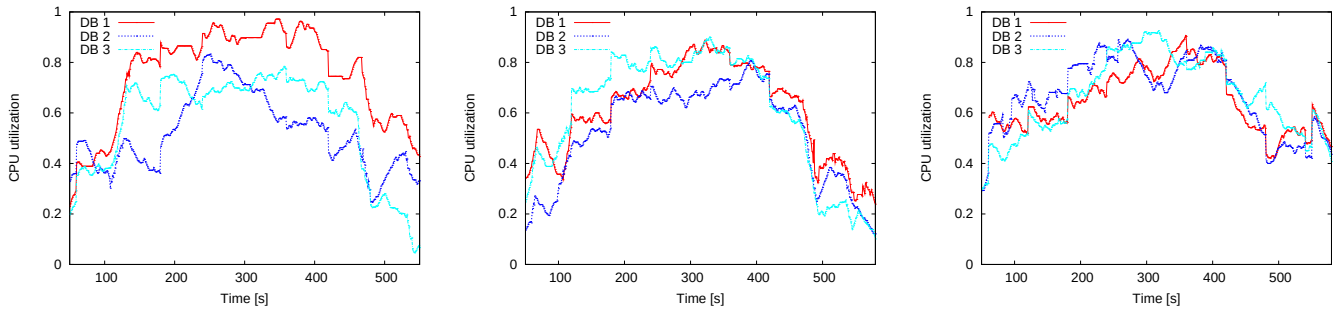
For both scenarios, we evaluate two important performance factors: the quality of load balancing and the impact of dispatching on the response time. As the performance measure, we consider the *90-percentile of the response time* for a Web request. For the quality of load balancing, we consider the *Load Balance Metric* [7] (LBM) that measures the degree of balance across different nodes. Let us define the load of the node j at the i-th observation (of the observation period m) as $load_{j,i}$, and $peak\_load_i$ as the highest node load in the same observation. The LBM is defined as:

$$LBM = \frac{\sum\limits_{1 \le i \le m} peak\_load_i}{(\sum\limits_{1 \le i \le m} \sum\limits_{1 \le j \le n} load_{j,i})/n} \quad (3)$$



(a) Sample-based dispatching **(b) EMA-based dispatching** **(c) Trend-aware dispatching**

Figure 5. Load balancing in a stationary scenario.



(a) Sample-based dispatching **(b) EMA-based dispatching** **(c) Trend-aware dispatching**

Figure 6. Load balancing in a non stationary scenario.

where n is the total number of nodes. Note that the value of the LBM can range from 1 to the number of nodes ($n = 3$ in the considered system). Smaller values of the LBM indicate a better load balance.

We compare three versions of the weighted round robin (WRR) policy [19], where the weights are computed by using three different pieces of load information: the samples of the CPU utilization (this is the traditional dynamic version of the WRR algorithm proposed in literature); the last load representation of the CPU utilization that is obtained through the EMA model on the basis of the last 60 resource load measures (novel algorithm proposed for the sake of comparison); the trend-aware load representation that is proposed in this paper.

The effects on the CPU utilization of the three nodes are reported in Figures 5 and 6. The Figure 5 shows the load, filtered by the noise component, on the three database servers when the load is stationary. We can see that the type of load information (sample, EMA, trend-aware) plays a crucial role in load balancing. Using the resource measures (Figure 5(a)), the system nodes are really unbalanced. On the other hand, the load representation based on the EMA model (Figure 5(b)) improves the quality of dispatching as it is shown by the fairly balanced conditions of the three database servers. An even better result is shown in Figure 5(c), where the dispatcher is based on the proposed trend-aware algorithm.

In the scenario characterized by non stationary workload, the differences among the three dispatching algorithms become more evident. In particular, we can see that in this scenario even the dispatching algorithm based on EMA 6(b) is unable to balance the load. On the other hand, the trend-aware algorithm guarantees the most balanced system (Figure 6 (c)) during the entire experiment.

In Table 1 we give a quantitative summary of the main performance metrics for the three dispatching mechanisms and the two workload models. In the stationary scenario, the results of the EMA-based and the trend-aware dispatching algorithm are similar. They achieve the best load balance in terms of LBM and the best performance in terms of 90-percentile of the response time for a Web request.

However, the real benefits of the trend-aware algorithm can be appreciated in the more realistic non stationary workload. In this case, the difference between the LBM of the sample-based algorithm and that of the trend-aware algorithm increases considerably, ranging from about 10% to more than 20% when passing from the stationary scenario to a non stationary scenario. The reduction of the 90-percentile of the response time (from 450ms for the sample-based algorithm to 370ms for the trend-aware algorithm) confirms the capacity of the trend-aware dispatching algorithm to manage a complex distributed system even in the

Table 1. Performance evaluation of the dispatching algorithms.

	Stationary workload			Non stationary workload		
	Sample-based	**EMA-based**	**Trend-aware**	**Sample-based**	**EMA-based**	**Trend-aware**
LBM	1.61	1.39	1.38	1.85	1.54	1.40
Response time (90-perc.)	175ms	170ms	170ms	450ms	445ms	370ms

presence of a more severe (but even more realistic) non stationary workload.

6 Conclusions

The unknown characteristics of the workload reaching any Web-based system make really difficult to guarantee performance and a satisfactory utilization of the server resources. In this paper, we propose a novel state-aware dispatching algorithm that takes into account the behavioral trend of the server load in the last observation interval. We apply this trend-aware algorithm to dispatch requests in front of a modern multi-tier Web-based system. We demonstrate through several experimental results that the proposed algorithm is able to improve load balance and also system performance in terms of response time. The major improvements with respect to the existing algorithms are achieved in the case of non-stationary workload.

References

[1] M. Andreolini, S. Casolari, and M. Colajanni. Load prediction models in web-based systems. In *In Proc. of VALUETOOLS*, Pisa, Italy, Oct. 2006.

[2] G. Bai and C. Williamson. Time-domain analysis of web cache filter effects. In *Proc. of SPECTS*, San Diego, CA, 2002.

[3] P. Barford and M. E. Crovella. Generating representative Web workloads for network and server performance evaluation. In *Proc. of SIGMETRICS*, Madison, WI, July 1998.

[4] Y. Baryshnikov, E. Coffman, G. Pierre, D. Rubenstein, M. Squillante, and T. Yimwadsana. Predictability of Web server traffic congestion. In *Proc. of WCW*, Sophia Antipolis, FR, Sep. 2005.

[5] N. Brenner and C. Rader. A new principle for fast fourier transformation. *IEEE Acoustics, Speech & Signal Processing*, 24:264–266, Mar. 1976.

[6] R. B. Bunt, D. L. Eager, G. M. Oster, and C. L. Williamson. Achieving load balance and effective caching in clustered Web servers. In *Proc. of WCW*, San Diego, CA, USA, USA, Apr. 1999.

[7] H. W. Cain, R. Rajwar, M. Marden, and M. H. Lipasti. An architectural evaluation of Java TPC-W. In *Proc. of HPCA*, Nuovo Leone, ME, Jan. 2001.

[8] V. Cardellini, E. Casalicchio, M. Colajanni, and P. Yu. The state of the art in locally distributed Web-server system. *ACM Computing Surveys*, pages 263–311, 2002.

[9] T. Carozzi and A. Buckley. Deriving the sampling errors of correlograms for general white noise. *ArvXiv physics e-prints*, May. 2005.

[10] E. Casalicchio and M. Colajanni. A client-aware dispatching algorithm for Web clusters providing multiple services. In *Proc. of WWW*, Hong Kong, May 2001.

[11] E. Cecchet, A. Chanda, S. Elnikety, J. Marguerite, and W. Zwaenepoel. Performance comparison of middleware architectures for generating dynamic Web content. In *Proc. of 4th Middleware Conference*, Rio de Janeiro, BR, June 2003.

[12] X. Chen and J. Heidemann. Flash crowd mitigation via an adaptive admission control based on application-level measurement. Technical Report ISI-TR-557, USC/Information Sciences Institute, May 2002.

[13] L. Cherkasova and P. Phaal. Session-based admission control: a mechanism for peak load management of commercial web sites. *IEEE Trans. Computers*, 51(6):669–685, June 2002.

[14] A. Cohen, S. Rangarajan, and H. Slye. On the performance of TCP splicing for URL-aware redirection. In *Proc. of USENIX*, Boulder, CO, Oct. 1999.

[15] M. E. Crovella, M. S. Taqqu, and A. Bestavros. Heavy-tailed probability distributions in the World Wide Web. In *A Practical Guide To Heavy Tails*, pages 3–26. Chapman and Hall, New York, 1998.

[16] M. Dahlin. Interpreting stale load information. *IEEE Trans. Parallel and Distributed Systems*, 11(10):1033–1047, Oct. 2000.

[17] Holt and C. Charles. Forecasting seasonals and trends by exponential weighted moving averages. *International Journal of Forecasting*, 20(1):5–10.

[18] G. Hunt, G. Goldszmidt, R. King, and R. Mukherjee. Network web switch: A connection router for scalable internet services. In *Proc. of WWW*, Brisbane, Australia, Apr. 1998.

[19] G. S. Hunt, G. D. H. Goldszmidt, R. P. King, and R. Mukherjee. Network Dispatcher: A connection router for scalable Internet services. *Computer Networks*, 30(1-7):347–357, 1998.

[20] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: characterization and implications for CDNs and Web sites. In *Proc. of WWW*, Honolulu, HW, May 2002.

[21] M. Mitzenmacher. How useful is old information. *IEEE Trans. Parallel and Distributed Systems*, 11(1):6–20, Jan. 2000.

[22] Q. Zhang, A. Riska, W. Sun, E. Smirni, and G. Ciardo. Workload-aware Load Balancing for Clustered Web Servers. *tpds*, 16(3):219–233, Mar. 2005.