Models and Framework for Supporting Runtime Decisions in Web-Based Systems

MAURO ANDREOLINI, SARA CASOLARI, and MICHELE COLAJANNI
University of Modena and Reggio Emilia

17

Efficient management of distributed Web-based systems requires several mechanisms that decide on request dispatching, load balance, admission control, request redirection. The algorithms behind these mechanisms typically make fast decisions on the basis of the load conditions of the system resources. The architecture complexity and workloads characterizing most Web-based services make it extremely difficult to deduce a representative view of a resource load from collected measures that show extreme variability even at different time scales. Hence, any decision based on instantaneous or average views of the system load may lead to useless or even wrong actions. As an alternative, we propose a two-phase strategy that first aims to obtain a representative view of the load trend from measured system values and then applies this representation to support runtime decision systems. We consider two classical problems behind decisions: how to detect significant and nontransient load changes of a system resource and how to predict its future load behavior. The two-phase strategy is based on stochastic functions that are characterized by a computational complexity that is compatible with runtime decisions. We describe, test, and tune the two-phase strategy by considering as a first example a multitier Web-based system that is subject to different classes of realistic and synthetic workloads. Also, we integrate the proposed strategy into a framework that we validate by applying it to support runtime decisions in a cluster Web system and in a locally distributed Network Intrusion Detection System.

Categories and Subject Descriptors: C.2.4 [**Computer Communication Networks**]: Distributed Systems; C.2.5 [**Computer Communication Networks**]: Local and Wide-Area Networks—*Internet*; C.4 [**Performance of Systems**] —*Design studies, measurement techniques, performance attributes, modeling techniques*

General Terms: Algorithms, Design, Measurement, Performance

Additional Key Words and Phrases: World Wide Web, load prediction, load change detection, distributed systems, load representation

ACM Reference Format:

Andreolini, M., Casolari, S., and Colajanni, M. 2008. Models and framework for supporting runtime decisions in Web-based systems. ACM Trans. Web 2, 3, Article 17 (July 2008), 43 pages. DOI = 10.1145/1377488.1377491 http://doi.acm.org/10.1145/1377488.1377491

Authors' address: M. Andreolini, S. Casolari and M. Colajanni, Department of Information Engineering, University of Modena and Reggio Emilia, Via Vignolese 905, Modena, I-41100, Italy; email: {mauro.andreolini, sara.casolari, colajanni}@unimo.it.

1. INTRODUCTION

The majority of critical Web-based services are supported by distributed infrastructures that are expected to satisfy scalability and availability requirements and to avoid performance degradation and system overload. Managing these systems requires several runtime decisions that are oriented towards load balancing and load sharing [Cardellini et al. 2002; Pai et al. 1998; Andreolini et al. 2003], overload and admission control [Cherkasova and Phaal 2002; Mitzenmacher 2000; Ferrari and Zhou 1987; Menascé and Kephart 2007; Chen and Mohapatra 2002], and job dispatching and redirection even at a geographical scale [Cardellini et al. 2003]. The introduction of self-adaptive systems and autonomic computing [Kephart and Chess 2003; Ganek and Corbi 2003; Wildstrom et al. 2005; Pradhan et al. 2002] will further increase the necessity for management algorithms that take important actions on the basis of present and future load conditions of the system resources.

Most available algorithms and mechanisms for runtime decisions evaluate the load conditions through the periodic sampling of resource load measures obtained from monitors. In different contexts [Baryshnikov et al. 2005; Chen and Heidemann 2005; Abdelzaher et al. 2002], these measures are sufficient to make decisions about present and future system conditions, whether a system resource is offloading, overloading, or stabilizing, and whether it is necessary to activate a management process. On the other hand, these measures are of little value for the systems and workloads that characterize the modern Web and that we consider in this article. We can confirm that the resource measures obtained from load monitors of Internet-based servers are extremely variable even at different time scales and tend to become obsolete rather quickly [Dahlin 2000]. Hence, in the typical heavy-tailed context characterizing the Web workload, a decision system working directly on measures is of little value because they give only a limited and instantaneous view of the resource status and do not capture the behavioral trend.

As an alternative, we propose that the decision systems operate on a continuous representation of the load behavior of the system resources. This idea leads to a two-phase strategy where we separate the problem of achieving a representative view of the resource load conditions from that of using this representation for decision purposes. In this article, we address the main issues related to both phases.

—We first propose and compare different linear and nonlinear functions, called *load trackers*, for the generation of a representative resource load. A load tracker obtains continuous resource measures from the system monitors, evaluates a load representation of one or multiple resources, and passes this representation on to the functions in the second phase.

—In the second phase, we utilize the generated load representation for addressing two important issues that are at the basis of several runtime management decisions: detecting nontransient changes of the load conditions of a system resource (*load change detection*) and predicting future load conditions of a resource (*load prediction*). An initial evaluation of the two-phase approach for load prediction was presented by the authors in Andreolini and Casolari

[2006]. In this article, we extend the idea and propose a general two-phase methodology to support runtime decisions in Web-based contexts.

Unlike the majority of papers focusing on user behavior and characterization, we examine the effects of a heavy-tailed workload from a system point of view. This decision allows us to propose an innovative two-phase strategy that has a general validity because it is independent of the user behavior and can be extended to many different contexts. For example, previous results [Dinda and O'Hallaron 2000; Chen and Heidemann 2005; Baryshnikov et al. 2005; Tran and Reed 2004] suggest the application of linear prediction models directly to resource measures, but this is unsuitable for the workload and system contexts we are considering in this article. However, we show that thanks to the two-phase strategy and the use of adequate load trackers, even a simple linear-based prediction model is able to achieve good predictions.

We compare different linear and nonlinear models for load trackers that are capable of supporting different decision systems and are characterized by a computational complexity that is compatible with the temporal constraints of runtime decisions. All of our results show that the choice of an adequate load tracker is a compromise between the rapidity in signaling a change in the load conditions and the accuracy needed to follow nontransient load changes, but the choice of the best load tracker depends on the objectives and constraints of the application in the second phase.

We validate the proposed two-phase strategy through different real systems. We initially test and tune the models in the context of a multitier Web-based architecture. Then, in order to show that the proposed methodology is not tied to a specific context, we validate the proposed framework with other applications and systems operating in realistic contexts: an admission control and request dispatching mechanism for a cluster supporting Web-based applications and a dynamic load balancer for a locally distributed Network Intrusion Detection System.

The article is structured as follows. Section 2 motivates our work by showing the extreme variability of resource measures at different time scales and for different Web-related workload scenarios; in this section, we also present the two-phase strategy. Section 3 defines the linear and nonlinear models that we use as bases for the load trackers in the first step of the two-phase strategy. Section 4 evaluates the computational costs, the accuracy, and the responsiveness of the considered load trackers. Sections 5 and 6 describe and evaluate two applications of the second phase, that is, the load change detection and the load prediction problems. Section 7 applies the main results of this article to a cluster Web-based system and to a locally distributed Network Intrusion Detection System. Section 8 compares the contribution of this article with respect to the state-of-the-art. Section 9 concludes the article with some final remarks.

2. MOTIVATION AND PROPOSAL

We have carried out a very large set of experiments for analyzing the typical behavior of commonly measured resources. We report on a subset of the results that refer to a specific architecture for eight classes of workload. The reader

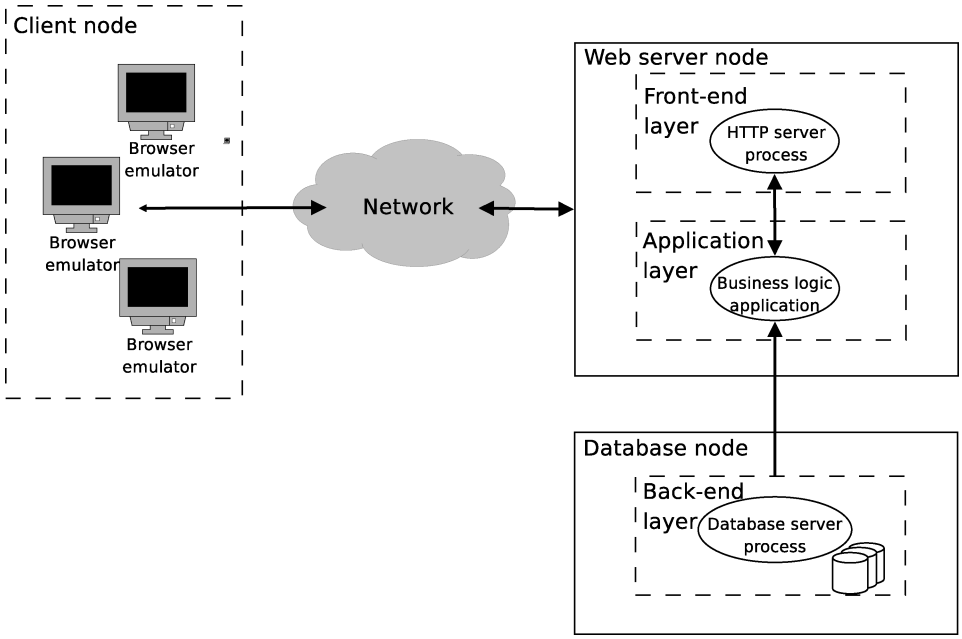


Fig. 1. Architecture of the considered multitier Web-based system.

should be aware that the main observations and conclusions about these results are representative of the typical behavior of the resources of a Web-based system that is subject to realistic workload.

2.1 Workload Models

As a testbed example, we consider a dynamic Web-based system referring to a multitier logical architecture (Figure 1) that follows the implementation presented in Cain et al. [2001].

The first node of the architecture executes the HTTP server and the application server deployed through the Tomcat [2005] servlet container; the second node runs the MySQL [2005] database server. We consider TPC-W as the workload model [TPC-W 2004] because it is becoming the de facto standard for the performance evaluation of Web-based systems providing dynamically generated contents (e.g., Dodge et al. [2001], Cecchet et al. [2003], Cain et al. [2001]). Client requests are generated through a set of *emulated browsers* where each browser is implemented as a Java thread reproducing an entire user session with the Web site. We instrument the TPC-W workload generator to emulate a *light* and a *heavy* service demand that, for the same number of emulated browsers, have low and high impact on system resources, respectively. Table I shows the parameters of the access frequencies of the TPC-W services for these workload models.

For both service demand models, we implement four user scenarios by varying the number of emulated browsers over time. The representative user scenarios for the heavy workload model are shown in Figure 2. (Analogous patterns

Table I. Service Access Frequencies (TPC-W Workload) for Light and Heavy Service Demand Models

	Home	New Prod.	Best Sellers	Prod. Det.	Search	Shop. Cart	Cust. Reg.	Buy	Order	Admin.
Light	55%	14%	14%	9%	7%	0.15%	0.05%	0.41%	0.2%	0.19%
Heavy	29%	11%	11%	21%	23%	2%	0.82%	1.44%	0.55%	0.19%

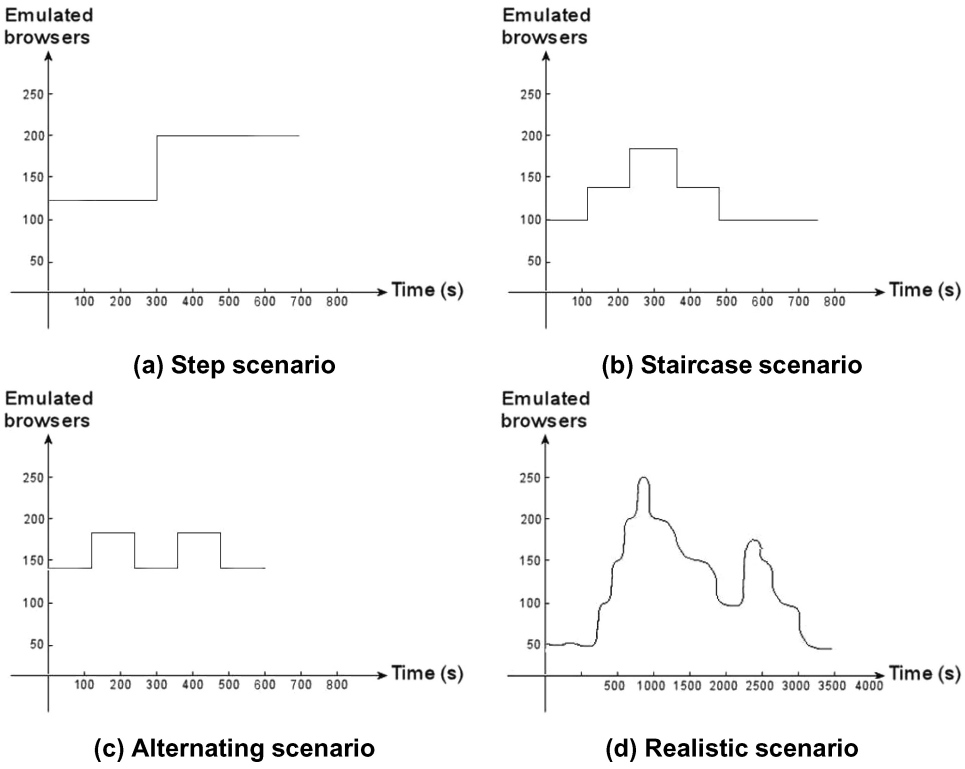


Fig. 2. User scenarios (the number of emulated browsers refers to the heavy service demand).

with different numbers of emulated browsers are created for the light service demand model.)

—*Step scenario.* The scenario in Figure 2(a) describes a sudden load increment from a relatively unloaded to a more loaded system [Satyanarayanan et al. 1997]. For the heavy (light) service demand, the population is kept at 120 (300) emulated browsers for 5 minutes, then it is suddenly increased to 200 (700) emulated browsers for other 5 minutes.

—*Staircase scenario.* The scenario in Figure 2(b) represents a gradual increment of the population up to 180 (600) emulated browsers for the heavy (light) service demand. The increase is followed by a similar gradual decrease.

—*Alternating scenario.* The scenario in Figure 2(c) describes an alternating increase and decrease of the load between 140 (400) and 180 (600) emulated browsers for heavy (light) service demand every two minutes.

—*Realistic scenario.* The scenario in Figure 2(d) reproduces a realistic user pattern (e.g., derived from a subset of data in Baryshnikov et al. [2005]) where load changes are characterized by a continuous and gradual increase or decrease of the number of emulated browsers.

The eight workload models are representative of aggressive Web workloads characterized by heavy-tailed distributions [Barford and Crovella 1998; Crovella et al. 1998; Challenger et al. 2004; Arlitt et al. 2001] and by flash crowds [Jung et al. 2002]. The motivation behind this choice of models is to demonstrate that the two-phase methodology works even in critical scenarios, although the toughest goal of predicting hotspot events remains an open issue beyond the scope of this article.

2.2 Measures and Analysis of Web System Resources

There are many critical resources in any system supporting Web-based services. The resource load or status can be measured through several system monitors (e.g. sysstat, procps, rrdtool) that typically yield instantaneous or average values over short intervals at regular time intervals. We have analyzed the behavior of commonly measured resources that refer by default to the last interval of one second: CPU utilization, disk and network throughput (MB/sec), number of open sockets, number of open files, process load, percentage of utilized memory, each of them considered for different sample periods, workload classes and scenarios. Understanding what is the most critical resource in a complex system is itself a problem that is orthogonal to the issues addressed in this article. We can easily conclude that all our experiments confirm literature results by indicating that the backend node of the multitier architecture in Figure 1 is the most critical system component [Elnikety et al. 2004]. For this reason, we focus on the CPU utilization and disk throughput of the backend node. To give a first qualitative motivation of the difficulties of capturing any clear message from a sequence of resource measures, in Figures 3 and 4 we report the results related to the light and heavy scenario, respectively. In these figures, we consider as examples different intervals, metrics, and scenarios:

—two resource measurement intervals: 1 second (Figures 3(a) and 4(a)), and 5 seconds (Figures 3(b) and 4(b));
—two resource metrics: CPU utilization (Figures 3(c) and 4(c)), and disk throughput as blocks/second (Figures 3(d) and 4(d));
—four user scenarios: step (Figures 3(c), 3(d), 4(c) and 4(d))), staircase (Figures 3(e) and 4(e)), realistic (Figures 3(a), 3(b), 4(a) and 4(b))), alternating (Figures 3(f) and 4(f)).

There are many qualitative messages shown by Figures 3 and 4. The measurement interval does not change the variability impact. Not every resource measure is equally representative of the system load: in general, the CPU follows the input load closer than the disk throughput. On the other hand, all the figures share the common trait that the view of a resource that is obtained from

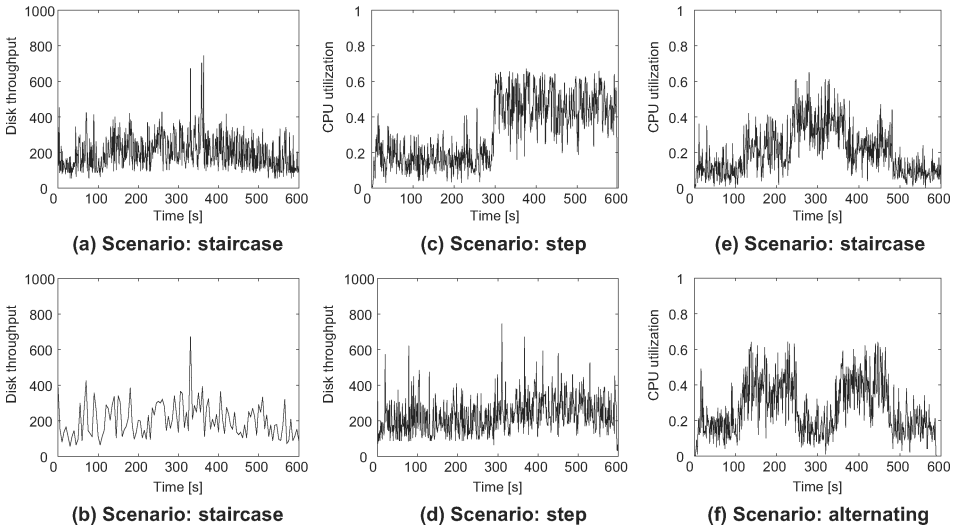


Fig. 3. Resource measurements—light service demand.

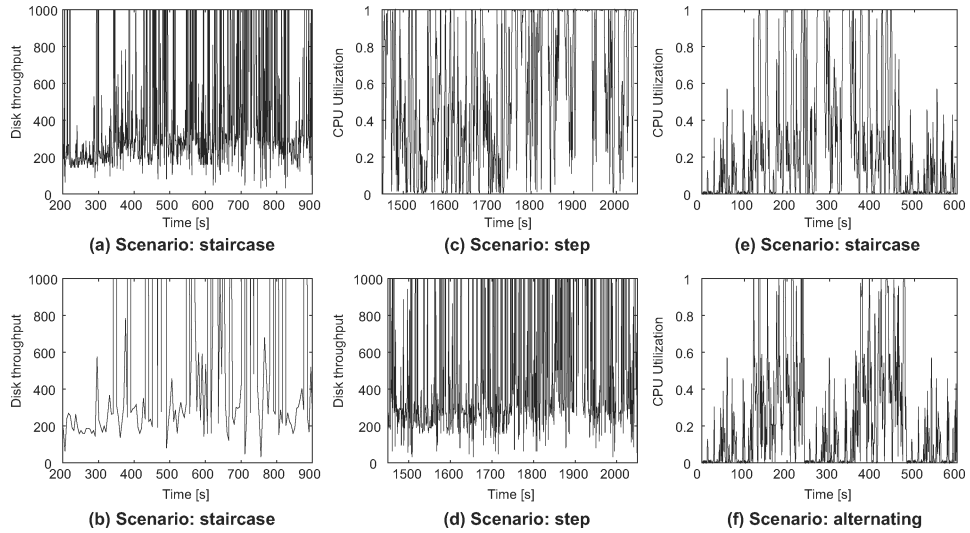


Fig. 4. Resource measurements—heavy service demand.

system monitors is extremely variable to the extent that any runtime decision based on these values may be risky when not completely wrong.

If we compare the two workload classes, Figures 3 and 4 show that heavy service demand causes much higher variability in the resource measures than light service demand. We give a mathematical confirmation of this result by evaluating the mean and the standard deviation of the CPU utilization of the backend node for both workload classes. We consider six stable user scenarios where the number of emulated browsers is kept fixed during the experiment running for one hour. The initial and final ten minutes are considered as

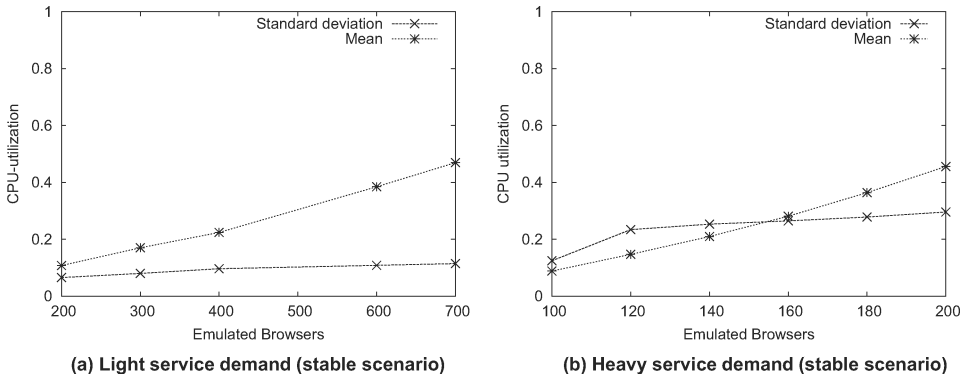


Fig. 5. Statistical analysis of the workloads.

Table II. Statistical Characterization of the Workloads (Mean and Standard Deviation)

	Light service demand		Heavy service demand	
	Mean	**Sd**	**Mean**	**Sd**
Step scenario	0.31	0.18	0.32	0.38
Staircase scenario	0.21	0.13	0.23	0.30
Alternating scenario	0.26	0.15	0.28	0.31
Realistic scenario	0.38	0.15	0.40	0.37

warm-up and cool-down periods, hence they are omitted from the evaluation of the statistics. The average CPU utilization and its standard deviation for light and heavy workload are shown in Figures 5(a) and 5(b).

In Table II, we report the results of the same statistical analysis for the four unstable scenarios: step, staircase, alternating, and realistic. Although in these cases the arithmetic mean is not a good representation of the load behavior, these results confirm the high variability of the resource measures for both workloads. In particular, the standard deviation highlights a twofold dispersion of the resource measures in the case of heavy service demand.

As a final observation, we note that the highly variable nature of the measures occurs for any workload, even when the average load is well below the maximum capacity of a resource. Variability is high to the extent that using direct resource measures for load change detection or load prediction analyses is of little value. For example, let us consider a system expected to make different decisions depending on CPU load. When the CPU utilization measures are similar to those in Figures 3 and 4, any load change detector would alternate frequent on-off alarms, thus making it impossible for a runtime decision system to judge whether a node is really off-loaded or not. On the other hand, a simple average of the resource measures would mitigate the on-off effect, but at the same time it would affect the efficacy of the load change detection algorithm. In short, these preliminary results suggest that a runtime management system should be able to operate on a different representation of the resource load such as that proposed in the following section.

2.3 Two-Phase Strategy

Direct measures have a limited value because they just offer instantaneous views of the load conditions of a resource. Moreover, these measures tend to be useless when they are highly variable as in typical Web workloads. In practice, there is no way to estimate or predict load, to analyze load trend, to forecast overload, to understand where the system is and where the system is going, to decide whether it is necessary to activate some control mechanism and, if it is, to choose the right course of action.

For these reasons, we propose that runtime management systems supporting Web-based services should operate not on resource measures but on a continuous representation of the load behavior of the system resources. This proposal leads to a two-phase strategy where we separate the two main phases behind a runtime decision.

(1) *Generation of representative resource load.* During this phase we obtain a representative view of the resource load.
(2) *Resource state interpretation.* In this phase, we utilize the previous representation as a basis for evaluating the present (e.g., load change detection) or future (e.g., load prediction) resource conditions; these evaluations are then passed on to the runtime decision system.

The two-phase strategy is outlined in Figure 6. In the first phase, a *load tracker* module continuously gets measures from the system monitors and evaluates one load representation of the resource behavior or a different representation for each class of application as shown by the figure. Multiple views from different resource measures may be used to get a global representation of a system component. This issue is, however, out of the scope of this article.

In the second phase, each representation obtained through the load tracker is passed on to an evaluation module that computes the present or future condition of a resource, possibly with respect to its maximum capacity. The final goal is to evaluate the information that is necessary for the runtime decision system to fulfill its goals such as improving the system throughput, avoiding bad request assignments, or refusing additional requests because of overload risks. The idea of a two-phase strategy seems rather straightforward. However, it has never been proposed before in a Web system context. Moreover, it opens several interesting issues that we address in the next sections.

The choice of an adequate load tracker is of utmost importance to the entire runtime management system, and it must be pointed out that no single choice is better than all the others. We implement load trackers based on linear and nonlinear models for different parameters. For the second phase, we consider the problem of detecting nontransient changes of the load conditions of a system resource and of predicting future resource behavior.

Different decision systems may require different representations that can be generated by the underlying load tracker. For example, a valid load change detector should signal to the runtime decision system only significant load changes that require some immediate actions, such as redirecting requests and filtering accesses. On the other hand, a load predictor should provide the runtime

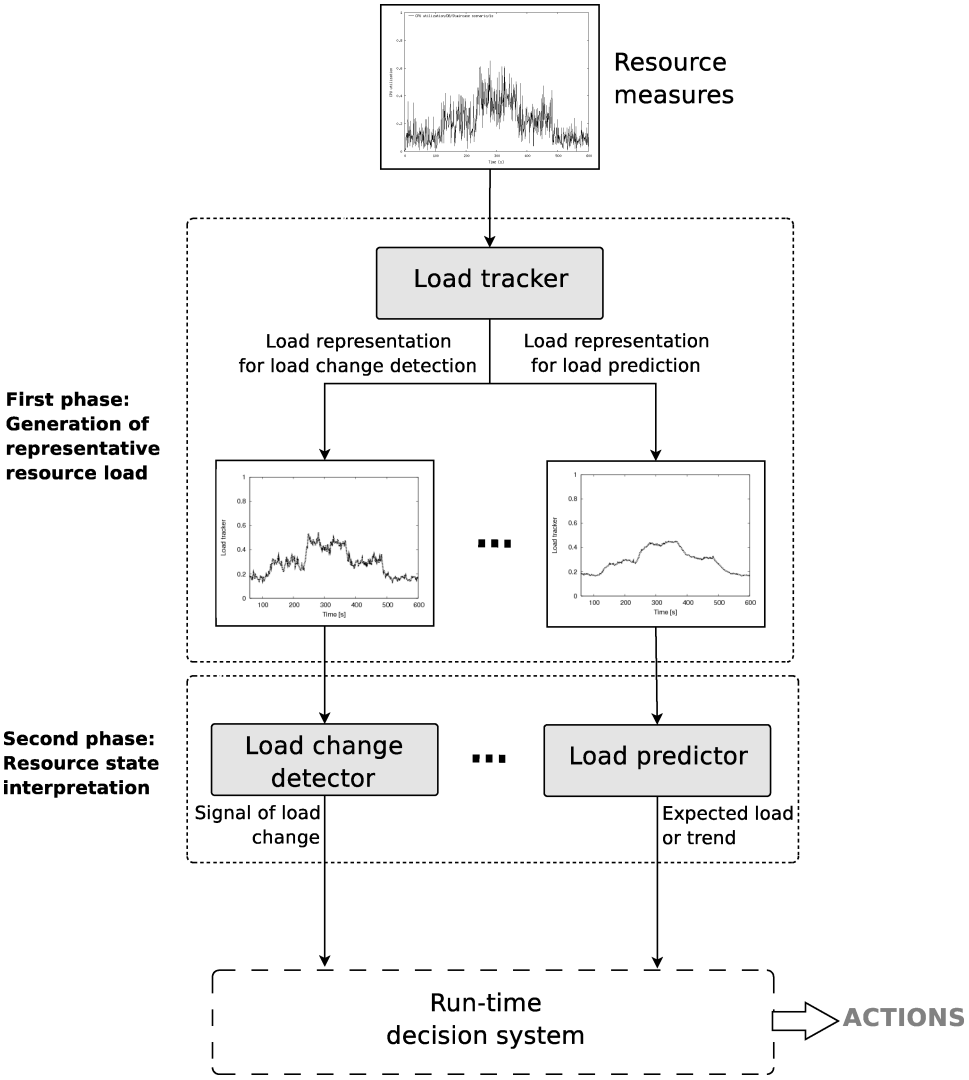


Fig. 6. The proposed two-phase framework for supporting runtime decisions.

decision system with expected future load conditions that are at the basis of different algorithms such as load balancing and request dispatching.

The proposed methodology and framework are modular, hence they can be easily enriched with other models and supports for decision systems. A crucial requirement for all the models in both phases is the capacity to satisfy runtime constraints which, in many Web systems, are on the order of seconds.

3. LOAD TRACKERS DEFINITIONS

In this section, we describe the first phase which aims to obtain a representative view of the load trend from resource measures. Roughly speaking, we consider

a load tracker function that filters out the noises characterizing a sequence of low-correlated and highly variable measures, and then offers a more regular view of the load trend of a resource to the models of the second phase. This problem is not related just to smooth resource measures before using them because an arithmetic mean is greatly smoothed, but it may not be representative of the real load conditions. Different runtime decision systems need different representations and the right compromise between accuracy and responsiveness of a load tracker should be sought.

At time t_i, the load tracker can consider the last measure s_i, and a set of previously collected $n-1$ measures, that is, $\overrightarrow{S_n}(t_i) = [s_{i-(n-1)}, \ldots, s_{i-1}, s_i]$. We define load tracker a function $LT(\overrightarrow{S_n}(t_i)) : \mathbb{R}^n \rightarrow \mathbb{R}$ that, at time t_i, takes as its input $\overrightarrow{S_n}(t_i)$ and gives a representation of the resource load conditions, namely l_i. A continuous application of the load tracker produces a sequence of load values that yields a trend of the resource load conditions by excluding out-of-scale resource measures. For the purposes of this article, we consider and compare some linear and nonlinear load tracker functions.

3.1 Linear Load Trackers

We first consider the class of moving averages because they smooth out resource measures, reduce the effect of out-of-scale values, are fairly easy to compute at runtime, and are commonly used as trend indicators [Lilja 2000]. We focus on two classes of moving averages: the *Simple Moving Average* (SMA) and the *Exponential Moving Average* (EMA), one using uniform and the other nonuniform weighted distributions of the past measures, respectively. We also consider other popular linear autoregressive models [Dinda and O'Hallaron 2000; Tran and Reed 2004], *Auto Regressive* (AR) and *Auto Regressive Integrated Mooving Average* (ARIMA).

Simple Moving Average (SMA). It is the unweighted mean of the n resource measures of the vector $\overrightarrow{S_n}(t_i)$, that is evaluated at time t_i $(i > n)$, that is,

$$SMA(\overrightarrow{S_n}(t_i)) = \frac{\sum_{i-(n-1) \leq j \leq i} s_j}{n}. \tag{1}$$

An SMA-based load tracker evaluates a new $SMA(\overrightarrow{S_n}(t_i))$ for each measure s_i during the observation period. The number of considered resource measures is a parameter of the SMA model, hence hereafter we use SMA_n to denote an SMA load tracker based on n measures. As SMA models assign an equal weight to every resource measure, they tend to introduce a significant delay in the trend representation, especially when the size of the set $\overrightarrow{S_n}(t_i)$ increases. The EMA models are often considered for the purpose of limiting this delay effect.

Exponential Moving Average (EMA). This is the weighted mean of the n resource measures of the vector $\overrightarrow{S_n}(t_i)$, where the weights decrease exponentially. An EMA-based load tracker $LT(\overrightarrow{S_n}(t_i))$, at time t_i, is equal to

$$EMA(\overrightarrow{S_n}(t_i)) = \alpha * s_i + (1-\alpha) * EMA(\overrightarrow{S_n}(t_{i-1})), \tag{2}$$

where the parameter $\alpha = 2/(n+1)$ is the *smoothing factor*. The initial $EMA(\overrightarrow{S_n}(t_n))$ value is initialized to the arithmetic mean of the first n measures:

$$EMA(\overrightarrow{S_n}(t_n)) = \frac{\sum\limits_{0 \le j \le n} s_j}{n}. \tag{3}$$

Similar to the SMA model, the number of considered resource measures is a parameter of the EMA model, hence by EMA_n we denote an EMA load tracker based on n measures.

Autoregressive Model (AR). This is a weighted linear combination of the past p resource measures of the vector $\overrightarrow{S_n}(t_i)$. An AR-based load tracker at time t_i can be written as

$$AR(\overrightarrow{S_n}(t_i)) = \phi_1 * s_{t_i} + \cdots + \phi_p * s_{t_{i-(p-1)}} + e_t, \tag{4}$$

where $e_t \sim WN(0, \sigma^2)$ is an independent and identically distributed sequence (called *residuals sequence*); $s_{t_n}, \ldots, s_{t_{i-(p-1)}}$ are the resources weighted by p linear coefficients; and $\phi_1, \ldots, \phi_p$ are the first p values of the autocorrelation function computed on the $\overrightarrow{S_n}(t_i)$ vector. The p order of the AR process is determined by the lag at which the partial autocorrelation function becomes negligible [Brockwell and Davis 1987; Kendall and Ord 1990]. The number p of considered resource measures is a parameter of the AR model hence by AR(p) we denote an AR load tracker based on p values. Higher-order autoregressive models include more lagged s_{t_i} terms where coefficients are computed on a temporal window of the n resource measures.

Autoregressive Integrated Moving Average Model (ARIMA). An ARIMA model is obtained by differentiating d times a nonstationary sequence and by fitting an ARMA model that is composed by the autoregressive model (AR(p)) and the moving average model (MA(q)). The moving average part is a linear combination of the past q noise terms, $e_{t_i}, \ldots, e_{t_{i-(q-1)}}$ [Brockwell and Davis 1987; Kendall and Ord 1990]. An ARIMA model can be written as

$$ARIMA(\overrightarrow{S_n}(t_i)) = \phi_1 * s_{t_i} + \cdots + \phi_{p+d} * s_{t_{i-(p+d-1)}} + \theta_1 * e_{t_i} + \cdots + \theta_q * e_{t_{i-(q-1)}}, \tag{5}$$

where $\theta_1, \ldots, \theta_q$ are linear coefficients. An ARIMA model is characterized by three parameters, that is, ARIMA(p,d,q), where p is the number of the considered resource measures, q of the residuals values and d of the differentiating values. As an ARIMA model requires frequent updates of their parameters, its implementation takes a nondeterministic amount of time to fit the load tracker values [Dinda and O'Hallaron 2000]. Hence, an ARIMA load tracker seems rather inadequate to support a runtime management system when the underlying infrastructure is subject to variable workloads.

3.2 Nonlinear Load Trackers

Linear models tend to introduce a delay in load trend description when the size of the considered resource measures increases, while they oscillate too much when the set of resource measures is small. The need for a nonlinear tracker is motivated by the goal of addressing in an alternative way the trade-off characterizing linear models. We consider two nonlinear models.

Two Sided Quartile-Weighted Median (QWM). In descriptive statistics, the quartile is a common way of estimating the proportions of the data that should fall above and below a given value. The two-sided quartile-weighted median is considered a robust statistic that is independent of any assumption on the distribution of the resource measures [Duffield and Lo Presti 2000]. The idea is to estimate the center of the distribution of a set of measures through the two-sided quartile-weighted median:

$$QWM(\overrightarrow{S_n}(t_i)) = \frac{Q_{.75}(\overrightarrow{S_n}(t_i)) + 2 * Q_{.5}(\overrightarrow{S_n}(t_i)) + Q_{.25}(\overrightarrow{S_n}(t_i))}{4}, \tag{6}$$

where Q_p denotes the pth quantile of the $\overrightarrow{S_n}(t_i)$.

Cubic Spline (CS). A preliminary analysis leads us to consider the *cubic spline* function [Poirier 1973], in the Forsythe et al. version [1977] as another interesting example of nonlinear load tracker. This decision is also motivated by the observation that lower-order curves (i.e., with a degree lower than 3) do not react quickly enough to load changes, while curves with a degree higher than 3 are considered unnecessarily complex, introduce undesired ripples, and are computationally too expensive to be applied in a runtime context. For the definition of the cubic spline function, let us choose some *control points* (t_j, s_j) in the set of measured load values, where t_j is the measurement time of the measure s_j. A cubic spline function $CS^J(t)$, based on J control points, is a set of $J-1$ piecewise third-order polynomials $p_j(t)$, where $j \in [1, J-1]$, that satisfy the following properties.

Property 1. The control points are connected through third-order polynomials:

$$\begin{cases} CS^J(t_j) = s_j & j = 1, \ldots, J \\ CS^J(t) = p_j(t) & t_j < t < t_{j+1},\ j = 1, \ldots, J-1. \end{cases} \tag{7}$$

Property 2. To guarantee a C^2 behavior at each control point, the first- and second-order derivatives of $p_j(t)$ and $p_{j+1}(t)$ are set equal at time t_j, $\forall j \in \{1, \ldots, J-2\}$:

$$\begin{cases} \frac{dp_j(t_{j+1})}{dt} = \frac{dp_{j+1}(t_{j+1})}{dt}, \\ \frac{d^2p_j(t_{j+1})}{dt^2} = \frac{d^2p_{j+1}(t_{j+1})}{dt^2}. \end{cases} \tag{8}$$

If we combine Properties 1 and 2, we obtain the following definition for $CS^J(t)$:

$$\begin{aligned} CS^J(t) \ = \ & \frac{z_{j+1}(t-t_j)^3 + z_j(t_{j+1}-t)^3}{6h_j} \\ & + \left(\frac{s_{j+1}}{h_j} - \frac{h_j}{6}z_{j+1}\right)(t-t_j) + \left(\frac{s_j}{h_j} - \frac{h_j}{6}z_j\right)(t_{j+1}-t) \\ & \forall j \in \{1, \ldots, J-1\}, \end{aligned} \tag{9}$$

where $h_j = t_{j+1} - t_j$, and s_j are the measured values. The z_j coefficients are solved by the following system of equations:

$$\begin{cases} z_0 = 0 \\ h_{j-1} z_{j-1} + 2(h_{j-1} + h_j) z_j + h_j z_{j+1} = 6 \left(\frac{s_{j+1} - s_j}{h_j} - \frac{s_j - s_{j-1}}{h_{j-1}} \right) \\ z_n = 0 \end{cases} \tag{10}$$

The spline-based load tracker $LT(\overrightarrow{S_n}(t_i))$, at time t_i, is defined as the cubic spline function $CS_n^J(t_i)$, obtained through a subset of J control points from the vector of n load measures.

Although the cubic spline load tracker has two parameters and is computationally more expensive than the SMA and EMA load trackers, it is commonly used in approximation and smoothing contexts [Eubank and Eubank 1999; Wolber and Alfy 1999; Poirier 1973]. The cubic spline has the advantage of being reactive to load changes and is independent of resource metrics and workload characteristics. Its computational complexity is compatible with runtime decision systems especially if we choose a small number of control points J. This reason leads us to prefer the lowest number, that is, $J = 3$.

4. LOAD TRACKER EVALUATION

Load trackers should be evaluated in terms of feasibility and quality. Only the load trackers that have a computational complexity which is compatible with runtime requirements can be considered acceptable. Moreover, it is important to evaluate load tracker accuracy and responsiveness. We will see that these two properties are in conflict, hence the perfect load tracker characterized by optimal accuracy and responsiveness does not exist. We anticipate that this trade-off can be solved by considering the goals of the load tracker application. For example, a runtime decision system epxected to take immediate action may prefer a highly reactive load tracker at the price of some inaccuracy. On the other hand, when an action has to be carefully evaluated, a decision system may prefer an accurate load tracker even if it is less reactive.

4.1 Computational Cost of Load Trackers

In this section, we estimate the computational cost of the load tracker functions in order to assess their feasibility to runtime requirements. We evaluate the CPU time required by each load tracker to compute a new value of the load representation. This time does not include the system and communication times that are necessary to fill the resource measure vector. The results for different measured values (n) are evaluated on an average PC machine and reported in Table III. They refer to the realistic user scenario and heavy service demand, but their costs are representative of any workload. From the table, we can conclude that the computational cost of all considered load tracker functions is compatible with runtime constraints. The majority of load trackers have a CPU time well below 10 ms. The main difference is represented by ARIMA models with a computational cost that is higher by one order of magnitude. Although a cost below 100 ms seems compatible with many runtime decision

Table III. CPU time (ms) for the Computation of a Load Tracker Value

	n = 30	**n = 60**	**n = 90**	**n = 120**	**n = 240**
EMA	0.059	0.059	0.059	0.059	0.059
SMA	0.560	1.039	1.461	1.990	3.785
CS	2.100	3.426	4.242	6.231	12.215
QWM	0.462	0.448	0.456	0.461	0.494
AR(32)	5.752	5.978	5.998	6.070	6.417
ARIMA (1,0,1)	X	67.536	67.765	67.228	72.141

systems, we should consider that behind the choice of the parameters of the AR and ARIMA models there is a complex evaluation. This phase requires the computation of the autocorrelation and partial autocorrelation functions as in Brockwell and Davis [1987] and Kendall and Ord [1990] and concludes that the AR(32) and ARIMA(1,0,1) models are the best for the considered workload. The complexity of this phase more than the CPU time for generating a load tracker value leads us to consider that the AR and ARIMA models are inadequate to support runtime decision systems in highly variable workload scenarios.

4.2 Load Tracker Accuracy and Responsiveness

All the considered load trackers share the common goal of representing at runtime the trend of a set of resource measures obtained from some load monitor. In order to evaluate the accuracy and responsiveness of the load tracker, we need a reference curve that we call *representative load interval*. This is the indicator of the central tendency of the resource measures in specific intervals of the experiment where the generated load is rather stable, although the resource monitors may recognize no stability from the measured values. In real systems, when the control is limited to the server side of the Internet and does not include the client side, it is practically impossible to compute the representative load interval. In our experimental setting, we have the additional advantage of controlling the load generators, and we can compute the representative load offline. Hence, we consider as a reference interval the period of time during which we generate the same number of user requests, that is, we have the same number of active emulated browsers. For example, in the step scenario and light service demand, we have two reference intervals, $T_1 = [0, 300]$ and $T_2 = [301, 600]$. In the staircase and alternating scenarios, there are five reference intervals: [0, 120], [121, 240], [241, 360], [361, 480] and [481, 600]. In the realistic scenario, we consider four intervals: [341, 460], [500, 640], [701, 820] and [821, 1000].

As the skew of the resource measures is severe, the simple mean is not a good indicator of the central tendency of a set of data [Lilja 2000]. Hence, we prefer to evaluate the representative load as the *approximate confidence interval* [Bonett 2006] in each interval. In Figures 7 and 8, we report for six workloads the resource measures (*dots*) referring to the CPU utilization of the database server and the upper (T_I^U) and lower (T_I^L) bounds of the representative load intervals (*horizontal lines*). Even from these figures we can appreciate the higher variability of the workload based on heavy service demand with respect to that based on light service demand. In the former workload, dots are more spread and confidence intervals are larger. For example, the middle

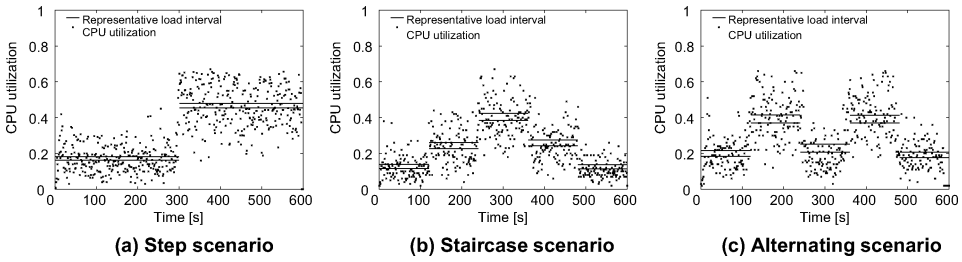


Fig. 7. Representative load intervals for different user scenarios and light service demand.

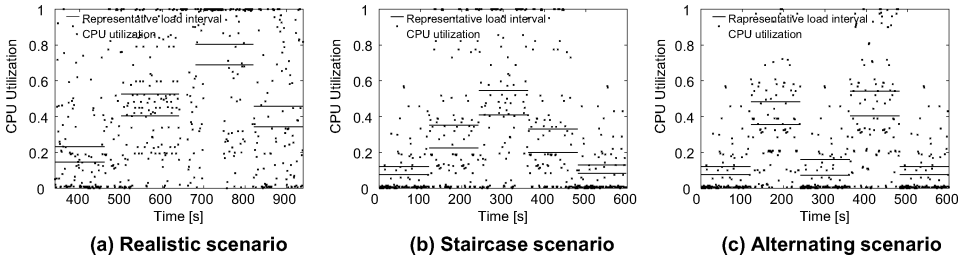


Fig. 8. Representative load intervals for different user scenarios and heavy service demand.

interval of the staircase scenario has $T_3^L = 0.39$ and $T_3^U = 0.42$ for the light service demand, and $T_3^L = 0.42$ and $T_3^U = 0.55$ for the high service demand.

We now evaluate the accuracy and responsiveness of the six load-tracker functions, that is, SMA_n, EMA_n, AR(32), ARIMA(1,0,1), CS_n, QWM_n, in representing the load trend of a set of n resource measures. From a qualitative point of view, responsiveness and accuracy correspond to the capacity of reaching the representative load interval as soon as possible and of having small oscillations around the representative load interval. We now propose a quantitative evaluation for these two parameters.

The *accuracy error* of a load tracker is the sum of the distances between each load tracker value l_i computed at the instant $i \in I, \forall I$ representative load intervals and the corresponding value of the upper bound T_I^U or lower bound T_I^L of the same interval, that is, $\sum_{\forall I>0} \sum_{i \in I} d_i$, where,

$$d_i = \begin{cases} 0 & \text{if } T_I^L \leq l_i \leq T_I^U \\ l_i - T_I^U & \text{if } l_i > T_I^U \\ T_I^L - l_i & \text{if } l_i < T_I^L. \end{cases} \tag{11}$$

The accuracy error corresponds to the sum of the vertical distances between each load tracker point that is out of the representative load interval and the representative load interval bounds for each interval.

For the sake of comparing different load tracker models, we prefer to use a normalized value such as the *relative accuracy error*. As a normalization factor, we consider the accuracy error of the resource measures. The relative accuracy error for any acceptable load tracker lies between 0 and 1; with other values, a load tracker would be considered completely inaccurate and discarded.

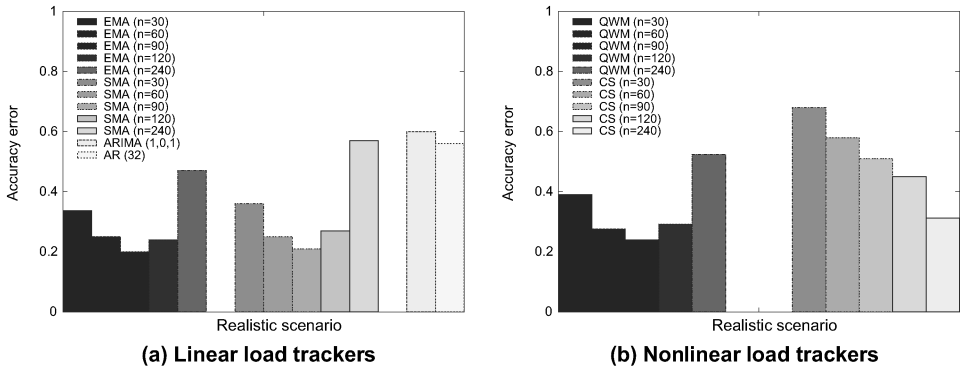


Fig. 9. Accuracy of the load trackers for the realistic user scenario and heavy service demand (n denotes the number of measured values used by a load tracker).

Responsiveness is a temporal requirement that aims to represent the ability of a load tracker to quickly adapt itself to significant load variations. Let t_{I_k} denote the time at which the representative load exhibits a new stable load condition that is associated to a significant change in the number of users. (e.g., in the realistic scenario with heavy service demand shown in Figure 8(a), we have $k = 3$ instants: 500, 680, 820.) A load tracker is more responsive when its curve touches the new representative load interval as soon as possible. Let t_{l_k} denote the instant in which the load tracker value reaches for the first time one of the borders of the representative load interval that is associated with a new load condition. The *responsiveness error* of a load tracker is measured as the sum of the horizontal differences between the initial instant t_{I_k} characterizing the representative load I and the corresponding time t_{l_k} that is necessary for the load tracker to touch this new interval. For comparison, we normalize the sum of the time delays by the total observation period T, thus obtaining a *relative responsiveness error*

$$\frac{\sum_k |t_{I_k} - t_{l_k}|}{T}. \tag{12}$$

In Figures 9 and 10, we report the normalized values of the accuracy and responsiveness errors of some representative load trackers for the workload characterized by a realistic user scenario and heavy service demand. We consider different sets of resource measures where n goes from 30 to 240.

There are some clear results coming from the observation of the histograms in Figure 9 and from all other results of which we report a small subset.

The SMA, EMA, and QWM load trackers are characterized by an interesting trade-off: working on a small ($n \leq 30$) and large ($n \geq 200$) amount of resource measures causes higher accuracy error than that achieved by intermediate size vectors. The reasons for this result are different. For small values of n, the error is caused by excessive oscillations; for large values of n, it is caused by excessive delays. Figures 11(a)–(c) give a visual interpretation of the quantitative results. For example, the SMA_{30} curve touches the representative load intervals quite

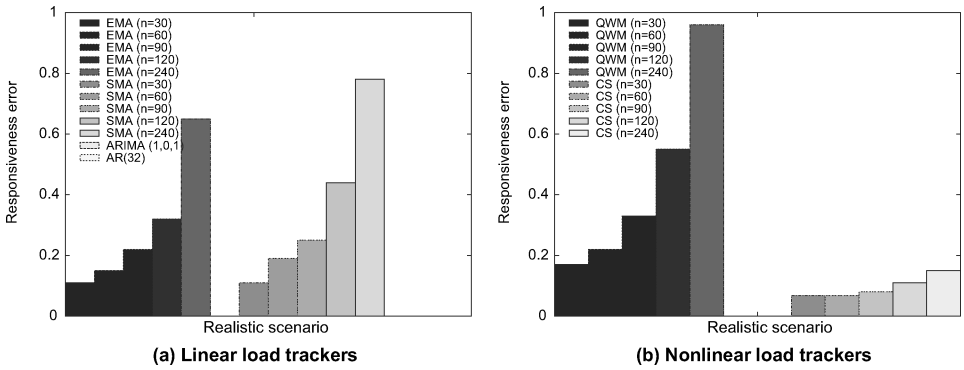


Fig. 10. Responsiveness of the load trackers for the realistic scenario and heavy service demand.

soon, but its accuracy is low because of too many oscillations. On the other hand, the SMA_{240} curve is highly smoothed, but it follows the real load with too much delay, and, even in this case, its accuracy is poor. Similar results are obtained for the EMA_{240} and QWM_{240} load trackers. The best results for $n = 90$ measures are confirmed by the EMA_{90} and QWM_{90} curves that more regularly follow the representative load intervals.

The AR and ARIMA models are characterized by a high accuracy error caused by their extremely jittery nature. Figure 11(d) offers a clear visual interpretation of the quantitative result. The cubic spline model is interesting because larger sets of resource measures lead to a monotonic improvement of the load tracker accuracy. Figures 11(e) and 11(f) show how the curve for $n = 240$ follows the representative load interval much better than the cubic spline for $n = 30$ which is extremely jittery.

A comparison of all results shows that the AR and ARIMA models have the largest accuracy errors. The best results of the EMA, SMA, and QWM models are comparable and all are obtained for a vector of $n = 90$ resource measures. Their accuracy is even higher than that of the best cubic spline model, that is, CS_{240}, although we will see that this function may further improve in accuracy for higher n.

It is interesting to observe that quite similar results are obtained for completely different and stressful workloads, such as the step, the staircase, and the alternating user scenarios, for both light and heavy service demand. Some results shown in Figure 12 refer to the light scenario. They confirm the conclusions about load tracker models, although they are obtained for different values of n. In particular, the SMA, EMA, and QWM load trackers yield their best accuracy for $n = 30$ instead of the previous $n = 90$ case.

Let us now move on to evaluate the responsiveness results that are reported in the histograms of Figures 10 and 13 for the realistic user scenario, and the step, the staircase and the alternating scenarios, respectively. The results shown in these figures and in all other results is a clear confirmation of the intuition that for any load tracker, working on a larger set of resource measures augments the responsiveness error. The most responsive load trackers are the

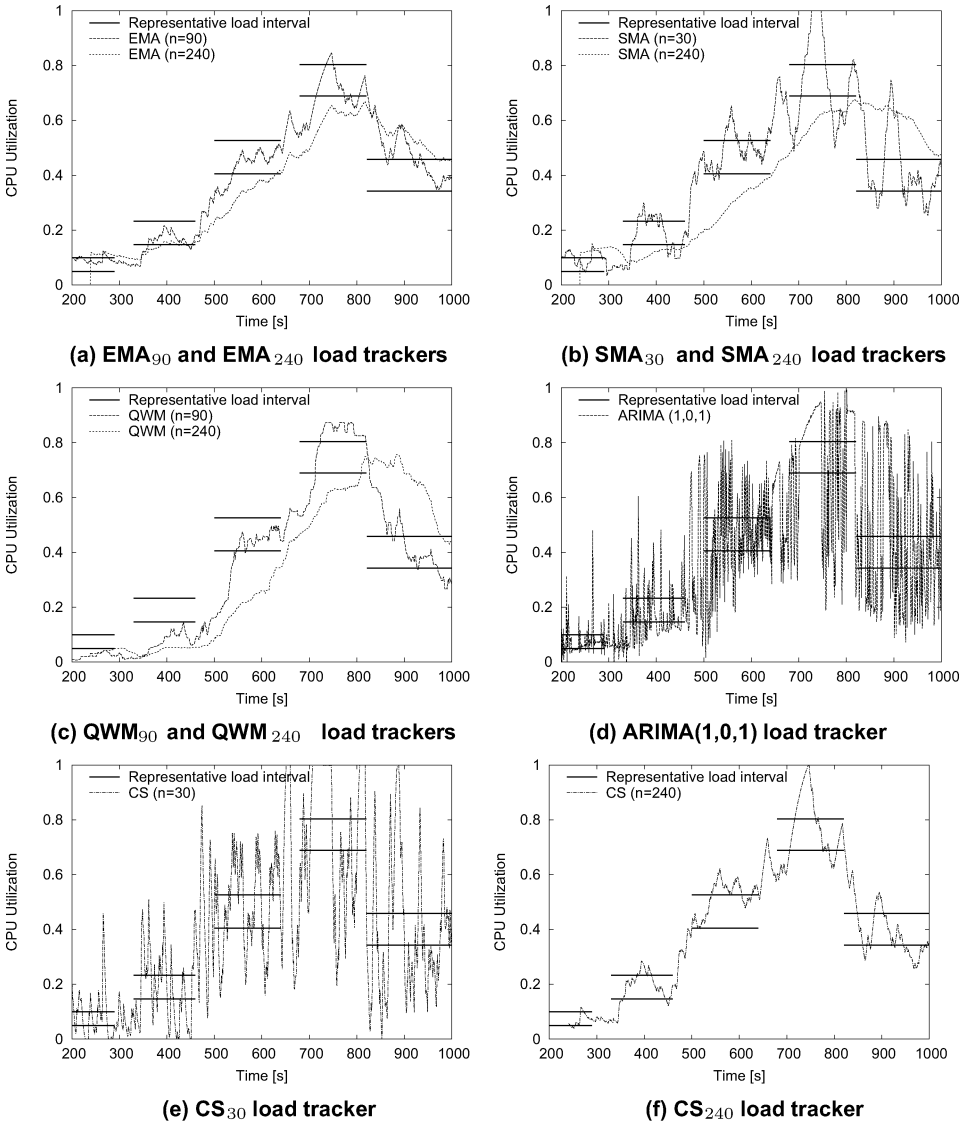


Fig. 11. Load tracker curves with respect to representative load intervals (realistic user scenario and heavy service demand).

AR and ARIMA models that are characterized by a null error. If we exclude these models that are useless for load tracker purposes, the cubic spline functions are the most responsive. Even the stability of their results is remarkable, with an error below 0.1 for any $n < 120$. The load trackers based on EMA and SMA models seem more sensitive to the choice of n; acceptable results are obtained for $n \leq 90$ and for $n \leq 30$ in the light and heavy service demand case, respectively. The QWM model is typically the least responsive and even the range of validity of n is narrower than that of the other load trackers.

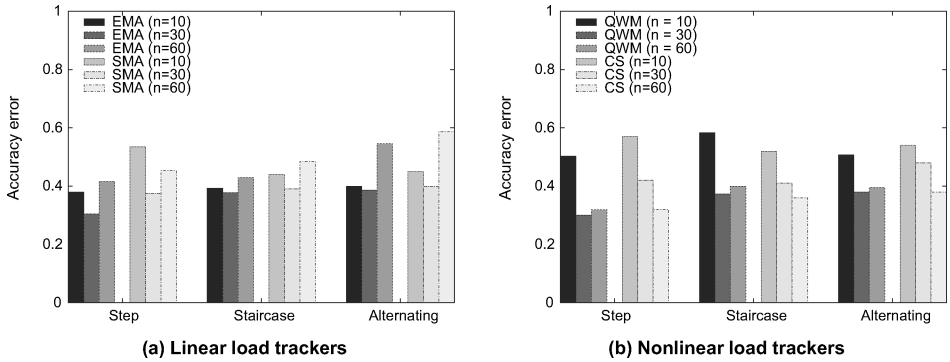


Fig. 12. Accuracy of the load trackers for three user scenarios and light service demand.

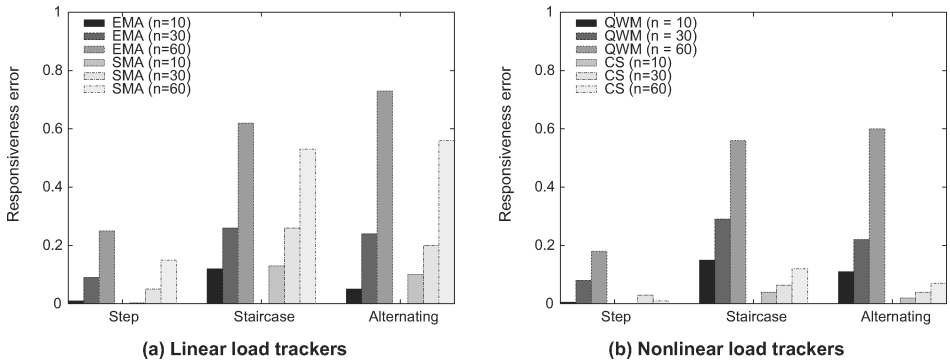


Fig. 13. Responsiveness of the load trackers for three user scenarios and heavy service demand.

4.3 Load Tracker Precision

The choice of the most appropriate load tracker is a compromise between accuracy and responsiveness. Depending on the kind of runtime decision, we can choose a more accurate or a more responsive load tracker. However, we can anticipate that no application can prefer one attribute without considering the others. For this reason, we introduce the *precision* of a load tracker as a combination of accuracy and responsiveness.

The majority of considered load trackers have many parameters, but we have seen that it is possible to relate the solution of the trade-off to the choice of the right number of measured values. For each model, it is necessary to find a value of n that represents a good trade-off between reduced horizontal delays and limited vertical oscillations. For example, the cubic-spline load trackers have the advantage of achieving monotonic and relatively stable results: their accuracy increases rapidly and their responsiveness decreases slowly for higher values of n. The results of the EMA, SMA, and QWM are characterized by a U effect as a function of n.

To evaluate the precision attribute as a trade-off between accuracy and responsiveness, we utilize a scatter plot diagram [Utts 2004]. In Figures 14

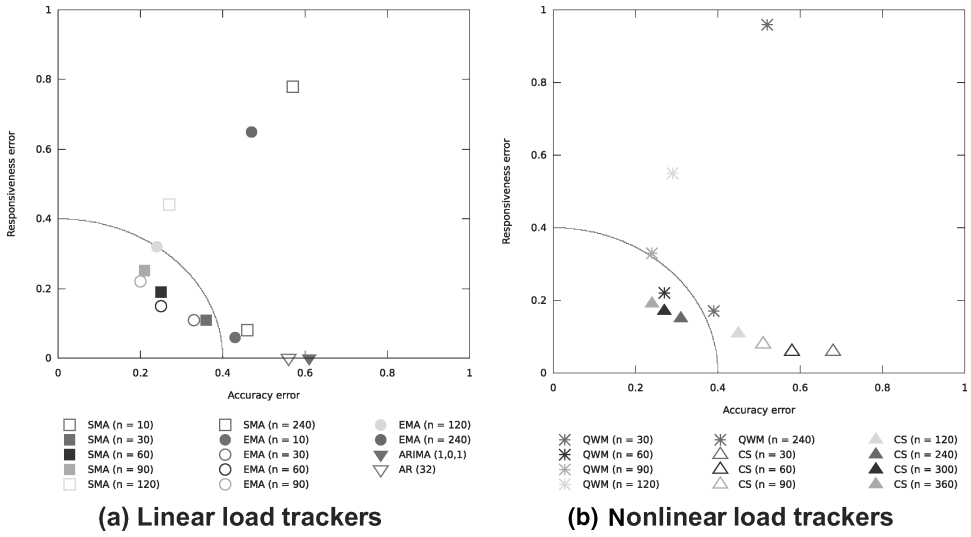


Fig. 14. Scatter plot of the load trackers for the realistic scenario and heavy service demand.

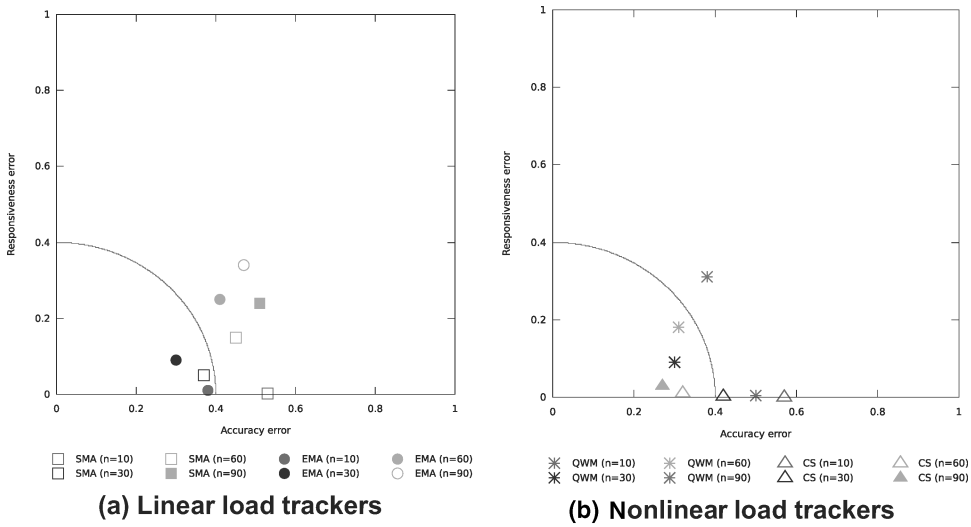


Fig. 15. Scatter plot of the load trackers for the step scenario and light service demand.

and 15, the x-axis reports the accuracy error and the y-axis the responsiveness error. Each point denotes the precision error of a load tracker.

We define the *precision distance* δ_L of a load tracker L as the Euclidean distance between each point and the point with null accuracy error and null responsiveness error (i.e., the origin) of the plot diagram. Moreover, we consider the area of *adequate precision* that delimits the space containing the load trackers that satisfy some precision requirements. In our example, we arbitrarily set the adequate precision range to 0.4; however, we should consider that this limit is typically imposed by the system administrator on the basis of

the application and constraints of the runtime decision system. In Figures 14 and 15, the load trackers having $\delta_L \leq 0.4$ are considered acceptable to solve the trade-off between accuracy and responsiveness.

The SMA, EMA, and QWM load trackers in Figures 14(a) and 15(a) share a similar behavior. For higher values of n they tend to reduce their accuracy error and increase their responsiveness error; at a certain instant, both accuracy and responsiveness degrade, and their points exit from the adequate precision area. We can confirm that the AR and ARIMA models are not valid supports for load trackers because their perfect responsive is achieved at the price of an excessive accuracy error. The cubic spline model confirms its monotonic behavior that can be appreciated by following the ideal line created by the small triangles in Figure 14(b).

Let us summarize the overall significance of this study that comes from the results discussed here and many other unreported experiments that confirm our main conclusions.

—The load tracker models based on EMA, SMA, CS, and QWM have a computational cost that is compatible with runtime constraints.
—For all workloads, the load trackers are characterized by a trade-off between accuracy and responsiveness. This issue can be converted into the problem of chosing the right size of the resource measure vector.
—A clear relationship exists between the dispersion (i.e., standard deviation) of the observed resource measures and the choice of the best resource measure vector size. A high dispersion of the resource measures, such as that of the heavy service demand, requires load trackers working on a larger number of resource measures. On the other hand, the number of needed resource measures to obtain a precise load tracker decreases when the workload causes a minor dispersion of the resource measures.

The proposal of a theoretic methodology to find the best n for any load tracker, any workload, and any application is out of the scope of this article. However, a large set of experimental results points to some interesting empirical evidence.

—A set of feasible values of n exists that guarantees an acceptable precision of the load tracker.
—The range of feasible values for n depends on the standard deviation of the resource measures. For example, in the heavy workload case, EMA is acceptable from $n = 30$ to $n = 120$; in the light workload case, EMA is acceptable from $n = 10$ to $n = 30$.
—The QWM-based load tracker has a limited range of feasibility.
—The EMA-based and the SMA-based load trackers have a larger but still limited range of feasibility.
—The CS-based load trackers have a sufficient precision only for high values of n. However, when this load tracker reaches the adequate precision area, it is feasible for a large range of n values thanks to its monotonic behavior. Although its higher accuracy comes at the price of an increased computational complexity, this does not prevent the application of CS to runtime contexts.

—Once we are in the adequate precision area, all load trackers are feasible. Among them, we can choose the best load tracker on the basis of the requirements of the second phase. In other words, we can give more importance either to the responsiveness or to the accuracy depending on the nature and constraints of the application of the load tracker model.

5. LOAD CHANGE DETECTION

In this section, we consider the *load change detection* problem as an application of the second step of the proposed two-phase strategy. Many runtime management decisions related to Web-based services are activated after a notification that a significant load variation has occurred in some system resource(s). Request redirection, process migration, access control, and limitation are some examples of processes that are activated after the detection of a significant and nontransient load change. Two properties characterize a good load change detector: the rapidity in signaling a significant load change and the ability to discern a steady load change from a transient change. These two properties are conflicting because a detector that is able to quickly signal load changes also has higher chances of mistaking a transient load spike for a steady load change.

The typical load-change detection strategy defines a threshold for a resource load and signals a load variation when the last observed value overcomes that threshold. This model has been widely adopted (just to cite few examples in Ramanathan [1999], Pai et al. [1998], and Pandey and Barnes [1998]), and its oscillatory risks are well known especially in highly variable environments (e.g., vicious cycles in request distribution and replica placement [Canali et al. 2004]). The risks of false alarms can be reduced by using multiple thresholds, by signaling an alarm only when multiple observed values overcome the threshold, by augmenting the observation period, and so on.

In the context of Web-based systems, the use of direct resource measures is quite inappropriate because a load change detector would signal continuous variations between two different states. Let us consider as an example the step-user scenario and the light service demand in Figure 16(a) where the CPU utilization is measured during an observation period of 500 seconds. Let us assume a threshold value set at $\chi = 0.4$ to detect when a change of state occurs. Any time the load change detector observes that the utilization passes over or under the threshold, it signals this event to the runtime decision system that activates or stops some process. This figure evidences the problems that are related to the load change detection when the load is described by resource measures. The representative load interval shows that there is only one significant load change at 300 seconds, but a load change detector based on resource measures signals many other spurious changes of state. For this reason, we think that it is preferable to consider a load representation such as that obtained through a load tracker model. Figures 16(b)–(f) consider the same example when load trackers are based on EMA, SMA, QWM, and CS models. A comparison between Figure 16(a) and any other figure where load change detection is based on a load tracker gives a qualitative motivation of the two-phase strategy. The problem is to find the best load tracker model for this second

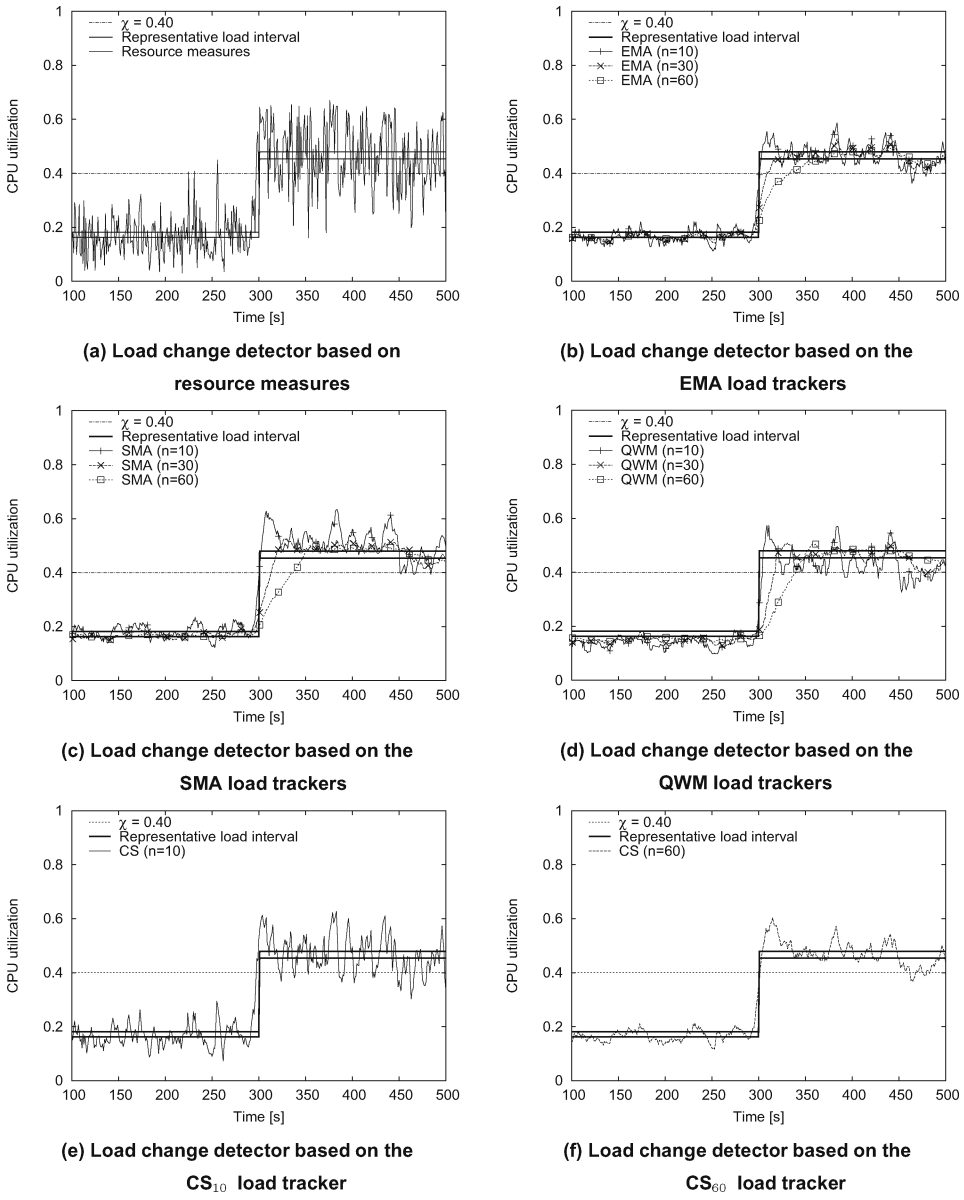


Fig. 16. Load change detection based on different load trackers.

phase. As it corresponds to the load tracker which limits the number of false detections, we observe that there are two possible sources of false detections.

—*Reactivity error*. The excess of oscillations of a load tracker around the threshold value χ causes many false alarms. This type of error is extremely evident in the case of resource measures (Figure 16(a)), but also in the case of a highly responsive load change detector such as CS_{10} (Figure 16(e)).

—*Delay error*. The excess of smoothing of a load tracker may cause a delay in signaling a variation of load conditions. This kind of error is evident in the case of smoothed-load change detectors such as EMA_{60} (Figure 16(b)), SMA_{60} (Figure 16(c)), and QWM_{60} (Figure 16(d)). In our example, a load change detector based on these load trackers signals the load change occurring at $t = 300$ with a delay of about 40 seconds.

The load change detection is a typical problem where we prefer a load tracker that solves the trade-off between too much reactivity causing false alarms and excessive smoothness causing delays. In other words, we do not want a too accurate or too responsive load tracker but one having adequate precision. If there are multiple adequate load trackers, the best choice depends on a preference given to responsiveness or accuracy.

For a quantitative evaluation of the delay and reactivity errors, let us consider the set of observations $\overrightarrow{J_I} = [J_{I_1}, J_{I_2}, \dots]$ at which the representative load interval overcomes the threshold value χ. For example, if $\chi = 0.4$, in the step-user scenario and light service demand (Figures 16), we have just one change, hence $|\overrightarrow{J_I}| = 1$ and $J_{I_1} = 300$. We have two changes ($\overrightarrow{J_I} = [240, 360]$) and four changes ($\overrightarrow{J_I} = [120, 240, 360, 480]$) in the staircase and alternating-user scenarios, respectively. As a further example, if we consider a threshold $\chi = 0.6$ in the realistic-user scenario and heavy service demand, we have two changes ($\overrightarrow{J_I} = [640, 820]$). The function of a load change detector is to detect as soon as possible a change of representative load over or under the threshold χ. Errors are caused when the detector signals an opposite load state l_i that is, when

$$\left(T_I^U < \chi \wedge l_i > \chi\right) \vee \left(T_I^L > \chi \wedge l_i < \chi\right). \tag{13}$$

A delay error is the sum of wrong observations occurring between a change of approximate confidence interval and the first right observation recognizing the load change. This value corresponds to the number of observations that are necessary for the load tracker to touch the threshold value χ after a change of load over or under the threshold.

Once a detector has evidenced a load change, an error due to reactivity occurs every time that an observation signals a change of state that has not really occurred. To compare different detectors, we evaluate the *relative delay error* as the sum of all delay errors normalized by the number of observations M. We also evaluate the *relative reactivity error* as the sum of all relative errors normalized by M. Tables IV and V report the relative delay and reactivity errors for the step, staircase, and alternating-user scenarios and light service demand and for the realistic-user scenario and heavy service demand, respectively. We consider only the load trackers that in Section 4.3 have shown adequate precision. Note that QWM_{60} is inadequate for the alternating and staircase-user scenario.

When the load change detector is based on resource measures, there is a significant number of oscillations around the threshold. In this case, reactivity errors are the only contributions to false detections because there are no delays.

The detectors based on EMA and SMA, and QWM load trackers exhibit a delay error that increases as a function of the number of measures n. This error represents the main contribution to false detections because these linear load

Table IV. False Detections (Light Service Demand)

	Step user scenario		Staircase user scenario		Alternating user scenario	
	Delay Error	Reactivity Error	Delay Error	Reactivity Error	Delay Error	Reactivity Error
Measures	0	17.8%	0	13.8%	0	18.4%
EMA_{10}	0.2%	2.4%	2.0%	0.6%	3.6%	0.7%
EMA_{30}	2.2%	0.2%	3.3%	0	3.9%	0
SMA_{30}	3.5%	0	5.8%	0	6.8%	0
QWM_{30}	2.7%	0	4.2%	0	4.7%	0
QWM_{60}	8.2%	0	X	X	X	X
CS_{30}	0	0.3%	0.5%	1.9%	0.7%	3.8%
CS_{60}	0	0.2%	0.8%	1.5%	0.9%	3.3%

Table V. False Detections (Heavy Service Demand)

	Realistic user scenario	
	Delay Error	Reactivity Error
Measures	0	22.1%
EMA_{60}	2.0%	3.9%
EMA_{90}	3.6%	1.0%
EMA_{120}	9.0%	0
SMA_{30}	1.3%	5.5%
SMA_{60}	4.4%	0.1%
SMA_{90}	8.2%	0
QWM_{60}	6.8%	0
QWM_{90}	9.2%	0
CS_{240}	1.6%	1.8%

trackers seem smoothed enough to avoid reactivity errors except for too small a set of n values. The opposite is true for load change detectors based on CS load trackers: they are not affected by very low delay errors but by high reactivity errors especially for few n values. As shown by Figures 16(e) and 16(f), these load trackers are characterized by a number of oscillations that decrease for higher values of n. These oscillations are the main reason for false detections.

An overall evaluation of the results in Table IV shows that for a light service demand, the best detectors are based on the nonlinear CS_{60} and on the linear EMA_{30} functions. Both of them are characterized by low percentages of false detections that are always lower than 5% even in the most severe alternating-user scenario.

Similar considerations hold true also when we consider the more jittery heavy service demand shown in Table V. This table reports only the load trackers with adequate precision as determined in Section 4.3. The main differences with respect to the user scenarios with a light service demand is the augmented number of errors caused by the higher dispersion of the resource measures. As a consequence, the best load trackers (EMA_{60}, QWM_{60}, and CS_{240}) for supporting load change detection need a larger amount of measured values than required for the user scenarios with a light service demand.

6. LOAD PREDICTION

6.1 Motivations

The possibility of forecasting the future load from a set of past values is another key function for many runtime decision systems that manage Web-based services. We define *load predictor* a function $LP_k(\overrightarrow{L_q}(t_i)) : \mathbb{R}^q \rightarrow \mathbb{R}$ that takes as its input the set of q values $\overrightarrow{L_q}(t_i) = (l_{i-q}, \ldots, l_i)$ at time t_i and returns a real number corresponding to the predicted load value at time t_{i+k}, where $k > 0$.

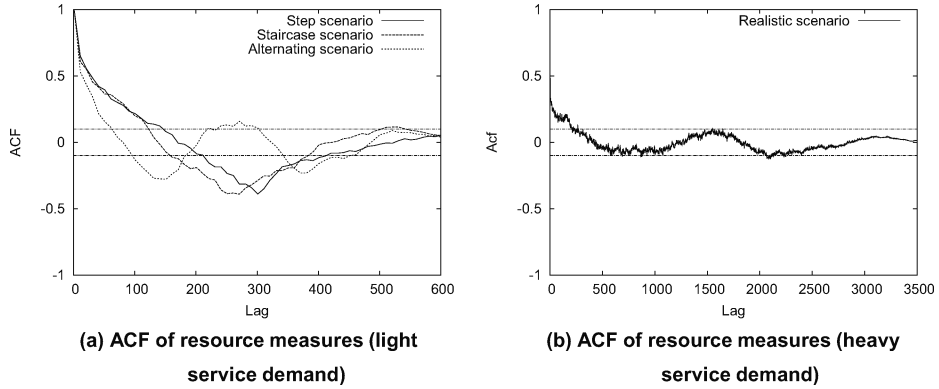
There is an important difference between our load predictor proposal and the state-of-the-art. In previous models, the vector $\overrightarrow{L_q}(t_i)$ consists of resource measures, and the load predictors aim to forecast a future resource measure at time t_{i+k}. Following the two-phase strategy, we propose a load predictor that takes as its input a set of load tracker values and returns a future load tracker value. In a context where the resource measures obtained from the load monitors of the Web-based servers are extremely variable, there are two reasons that justify our choice.

—The behavior of monitored resource loads appears extremely variable to the extent that a prediction of a future resource measure value is useless for taking accurate decisions.

—Many proposed load predictors working on real measures may be unsuitable to support a runtime decision system because of their excessive computational complexity.

We confirm the first motivation through a study of the autocorrelation function of the CPU utilization measures. The accuracy of a prediction algorithm depends on the correlation between consecutive resource measures. When the autocorrelation functions of the set of analyzed data fall rapidly, it is difficult or impossible to have an accurate prediction [Tran and Reed 2004; Baryshnikov et al. 2005]. In these scenarios, even an attractive approach for statistical modeling and forecasting of complex temporal series, such as the Box-Jenkins's Auto-regressive Integrated Moving Average (ARIMA) [Box et al. 1994], tends to provide models that do not adapt well to highly variable changes in the workloads.

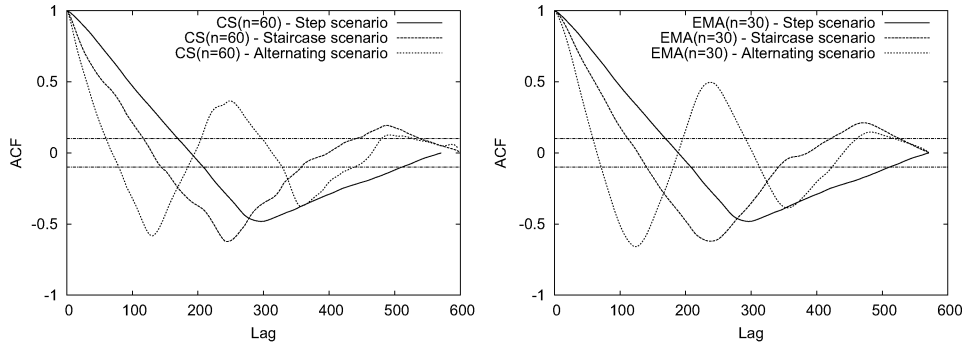
For example, in Figure 17(a), we show the autocorrelation function (ACF) of the CPU utilization for the three stressful user scenarios and light service demand for an observation period of 600 seconds, while, in Figure 17(b), we report the ACF for the realistic-user scenario and heavy service demand for an observation period of 3500 seconds. A point (k, y) in this graph represents the correlation value y between the resource measure s_i at time t_i and the measure s_{i+k} at time t_{i+k}. A positive autocorrelation value denotes a correlation between the two resource measures; consequently, the resource measure at time t_i may be used to predict the load value at time t_{i+k}. On the other hand, a low value of the autocorrelation function indicates the impossibility of an accurate prediction. A visual inspection of these figures leads us to conclude that the resource measures have low or null correlation for any workload, and



(a) ACF of resource measures (light service demand)

(b) ACF of resource measures (heavy service demand)

Fig. 17. Autocorrelation functions of the resource measures.



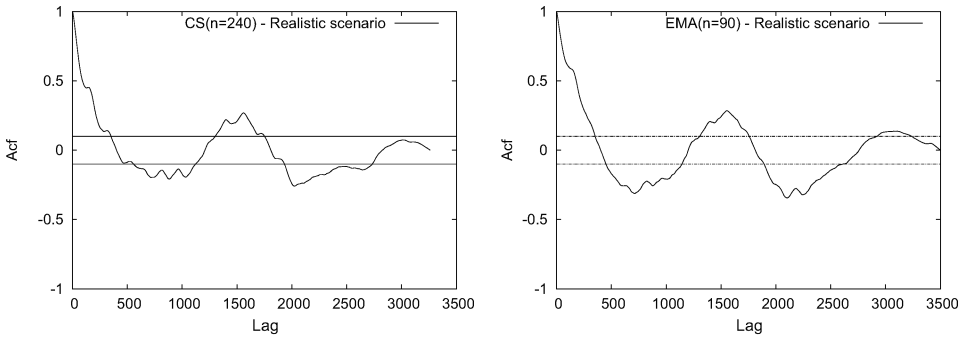
(a) ACF of CS_{60} values

(b) ACF of EMA_{30} values

Fig. 18. Autocorrelation functions for two load trackers with adequate precision (light service demand).

in particular for the realistic-user scenario and heavy service demand shown in Figure 17(b).

We now move on to evaluate the autocorrelation functions when a load tracker model is used as the basis for prediction. For each workload, we evaluate the ACF of some of load trackers that in Section 4.3 have shown adequate precision: EMA_{30} and CS_{60} for light service demand (Figure 18); EMA_{90} and CS_{240} for heavy service demand (Figure 19). In these cases, the autocorrelation seems higher than that shown by resource measures. However, for a more precise analysis, in Table VI, we report some significant results as a function of the prediction window k. From this table, we can see that the autocorrelation decreases for higher values of k. However, the degree of the decrease differs for different workloads and considered values. Resource measures are poorly correlated for any value of k when we consider the workload characterized by a realistic-user scenario and heavy service demand and even for the other workload the ACF tends to decrease below 0.5 soon after $k = 10$. On the other hand, we can appreciate that the ACFs of the two considered load trackers decreases much less rapidly. For any workload, their ACF



(a) ACF of CS_{240} values (b) ACF of EMA_{90} values

Fig. 19. Autocorrelation functions for two load trackers with adequate precision (heavy service demand).

Table VI. Autocorrelation Values

Staircase user scenario and light service demand				
	k = 10	**k = 30**	**k = 60**	**k = 100**
Measures	0.59	0.45	0.32	0.22
EMA_{30}	0.94	0.76	0.51	0.18
CS_{60}	0.90	0.68	0.47	0.21
Realistic user scenario and heavy service demand				
	k = 10	**k = 30**	**k = 60**	**k = 100**
Measures	0.30	0.25	0.18	0.18
EMA_{90}	0.96	0.85	0.71	0.61
CS_{240}	0.96	0.83	0.63	0.48

is around or well above 0.7 until $k = 30$. This result is important because, when consecutive values show a high degree of correlation, it is more likely to achieve an accurate load prediction. We limit the prediction window of interest for our studies to an interval of 30 seconds because, for an extremely dynamic system, larger prediction windows could lead to a wrong view of the load conditions.

6.2 Load Prediction Function

Thanks to the two-phase strategy, we can expect that even simple linear predictors may be sufficient to forecast the future load of a resource. Indeed, previous studies [Lingyun et al. 2003; Sang and Li 2000; Baryshnikov et al. 2005] demonstrate that simple linear models, such as the autoregressive model or the linear interpolation, are adequate for prediction when the correlation of consecutive resource measures is high. For example, in Dinda and O'Hallaron [2000], it is shown that the UNIX load average can be accurately predicted with low computational cost through an autoregressive model that takes into account the last 16 measures ($AR(16)$). In this article, we consider a set of load predictors $LP_k(\overrightarrow{L_q}(t_i))$ that are based on the linear regression of two load tracker values. Each predictor in this class is characterized by two values:

—the predicted window k that represents the size of the prediction interval;
—the past time window q, where q is the size of the load tracker vector $\overrightarrow{L_q}(t_i)$, that is the distance between the first and the last considered load tracker value.

This linear load predictor is actually a class of load predictors that are based on the linear regression of two load tracker values. Each predictor in this class is characterized by the values of the past window q and of the prediction window k. Let us take two load tracker values l_{i-q} and l_i. The load predictor $LP_k(\overrightarrow{L_q}(t_i))$ of the load tracker is the line that intersects the two points (t_{i-q}, l_{i-q}) and (t_i, l_i) and returns $\hat{l}_{i+k}$ that is the predicted value of the load tracker l_{i+k} at time t_{i+k}:

$$LP_k(\overrightarrow{L_q}(t_i)) = m * (t_{i+k}) + l_{i-q} - m * (t_{i-q}), \tag{14}$$

where $m = \frac{l_i - l_{i-q}}{q}$. We should point out that this class of functions is just an example of application of the two-phase strategy. Indeed, any other linear and nonlinear predictor could be integrated into the proposed framework.

6.3 Evaluation of the Load Predictors

In the context of the two-phase framework, the strength of a predictor depends on its accuracy to evaluate the future values of the load tracker. The common measure of the accuracy of a predictor is based on the evaluation of the relative error between a load tracker value and the corresponding predicted value $\hat{l}_{i+k}$. A load predictor characterized by a low prediction error is able to evaluate future load tracker values accurately. Let us consider a load tracker $LT(\overrightarrow{S_n}(t_i))$ and a load predictor $LP_k(\overrightarrow{L_q}(t_i))$ that at time t_i forecasts $LT(\overrightarrow{S_n}(t_{i+k}))$, where $k > 0$. We define the *prediction error* ϵ_{i+k} at time t_{i+k} as the relative error between the actual load tracker value l_{i+k} and the predicted value $\hat{l}_{i+k}$:

$$\epsilon_{i+k} = \frac{|l_{i+k} - \hat{l}_{i+k}|}{l_{i+k}}. \tag{15}$$

Small values of ϵ_{i+k} indicate a good accordance between l_{i+k} and $\hat{l}_{i+k}$. We evaluate the accuracy of the load predictors defined in Equation (14) as a function of k (*prediction window*) and q (*past window*) when they are applied to some of the load trackers proposed in Section 3.

In Tables VII and VIII we report the sum of the relative prediction errors normalized by the number of predictions carried out during the experiment for the staircase user scenario and light service demand, and for the realistic user scenario and heavy service demand, respectively.

The first important result coming from all of our experiments is that the load predictors based on a linear load tracker such as EMA always performs better than the load predictors based on a nonlinear load tracker such as CS. This result is characterized by a total relative error always higher than 0.3 when $k = 30$. This depends on two factors: a linear load tracker is characterized by a reduced number of oscillations; we are using a linear function as a load predictor. Analyses for different load prediction functions are out of the scope of this article.

Table VII. Prediction Errors as a Function of the Past Window Value (Staircase-User Scenario and Light Service Demand)

	Prediction window $k = 10$		
	q = 5	**q = 10**	**q = 20**
EMA$_{30}$	0.12	0.14	0.18
CS$_{60}$	0.14	0.15	0.21
	Prediction window $k = 30$		
	q = 10	**q = 15**	**q = 20**
EMA$_{30}$	0.25	0.15	0.18
CS$_{60}$	0.46	0.36	0.38

Table VIII. Prediction Errors as a Function of the Past Window Value (Realistic-User Scenario and Heavy Service Demand)

	Predicition window $k = 10$		
	q = 5	**q = 10**	**q = 20**
EMA$_{90}$	0.11	0.14	0.18
CS$_{240}$	0.19	0.23	0.26
	Prediction window $k = 30$		
	q = 10	**q = 15**	**q = 20**
EMA$_{90}$	0.22	0.19	0.25
CS$_{240}$	0.35	0.32	0.40

Another important result is that for short prediction windows, such as $k = 10$ seconds, the results are rather stable for any set of q past values. On the other hand, for further predictions (e.g., $k = 30$), the choice of the right values for q is more important. An empirical observation coming from Table VII and from all other results about different scenarios is that, for adequate predictions, it is convenient to use a set of past values in the interval $k/2 \leq q < k$. The reason for this is that with too few q values, the prediction line takes into account only the very recent trend of the load tracker. Hence, if the load tracker is not smooth enough, the prediction error tends to augment. On the other hand, too many q values tend to give excessive importance to the past trends, and this causes another type of prediction error.

In the graphs in Figure 20, we give a visual interpretation of the load prediction behavior achieved by EMA_{90} and CS_{240}. The parameters of these figures are $q = 5$ for the prediction window of $k = 10$, and $q = 15$ for a prediction window of $k = 30$ seconds. We show the load tracker values and the predicted values for the realistic-user scenario and heavy service demand. All predicted curves follow the load trackers fairly well even for $k = 30$ seconds. Moreover, these figures confirm the better results of a predictor based on an EMA load tracker compared to on based on a CS load tracker.

7. CASE STUDIES

In this section, we validate the proposed two-phase strategy by applying it to support runtime management decisions in two distributed environments. The considered systems share the common characteristics that their resource

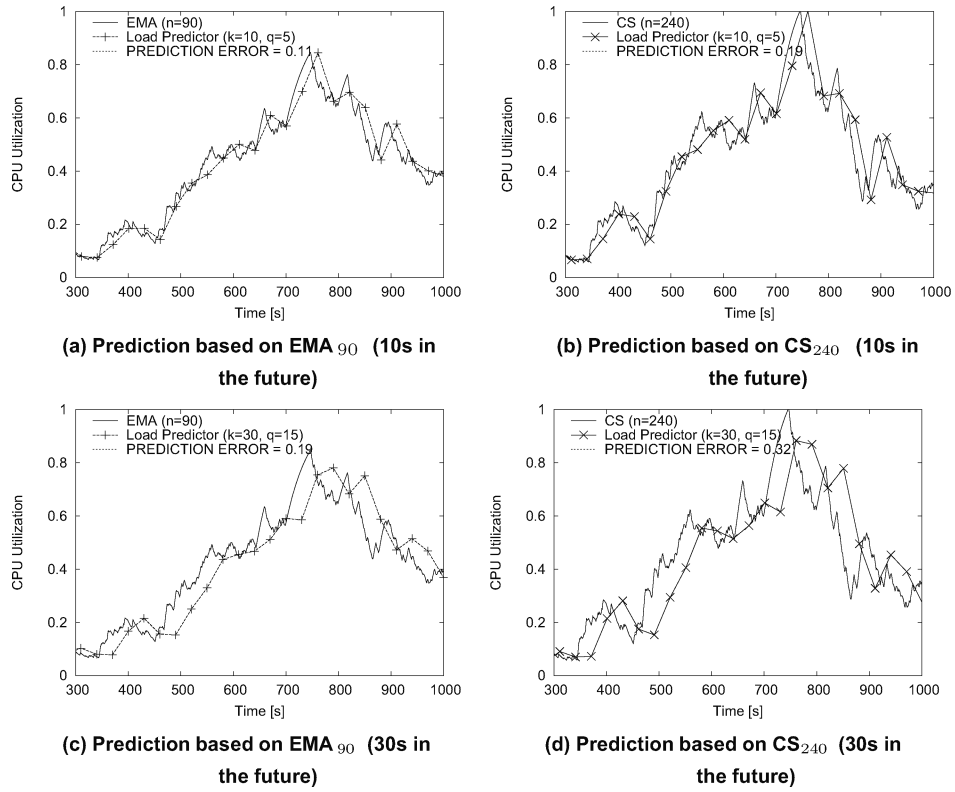


Fig. 20. Load predictors for a workload characterized by realistic-user scenario and heavy service demand.

measures obtained by monitors present large oscillations. In Section 7.1, we support a threshold-based admission controller and a request dispatcher applied to a Web cluster system. In Section 7.2, we consider a completely different system to demonstrate the flexibility of the proposed framework, and we support a dynamic load balancer applied to a locally distributed Network Intrusion Detection System (NIDS).

7.1 Admission Control and Dispatching for a Web Cluster System

Two main problems affect the performance of an e-commerce infrastructure [Elnikety et al. 2004]: overload risks when the volume of requests temporarily exceed the capacity of the system and slow response time leading to lower usage of a site and consequent reduced revenues. To mitigate these two problems, the software infrastructure can be enriched with an admission controller that accepts new client requests only if the system is able to process them with some guaranteed performance level [Elnikety et al. 2004; Cherkasova and Phaal 1999; Chen and Heidemann 2005]. Many decisions about accepting or refosing a client request are based on punctual load information of some critical component of the infrastructure. If the observed resource measure lies below a predefined threshold, the system accepts the request; otherwise, the request

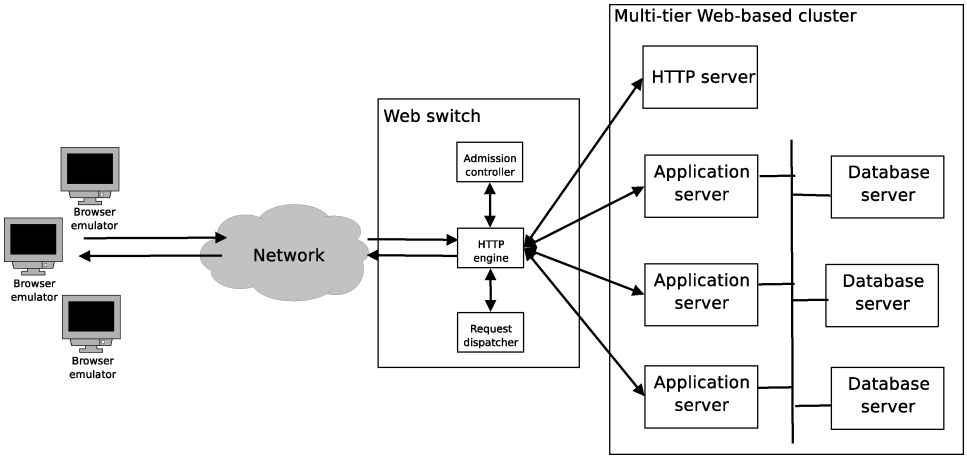
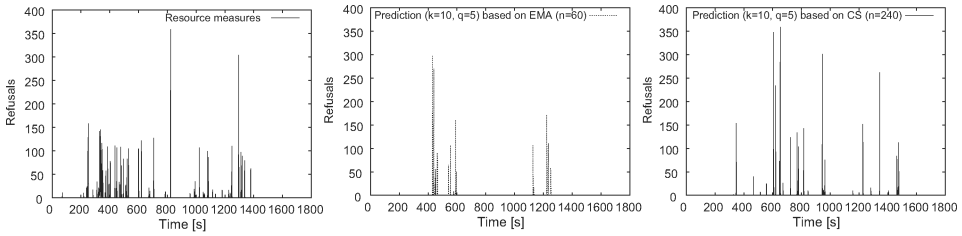


Fig. 21. Architecture of the multitier Web cluster.

is dropped. This approach may lead to frequent and unnecessary activations of the admission control mechanism. Even worse, highly variable and burst Web patterns may make it very difficult to activate the admission control mechanism on time. In this section, we show how the use of the proposed two-phase strategy with a load tracker and a load prediction can mitigate these problems and improve the overall performance of the system.

We refer to a locally distributed multitier system whose architecture is described in Figure 21. The system is based on the implementation presented in Cain et al. [2001]. The application servers are deployed through the Tomcat [2005] servlet container and are connected to MySQL database servers [MySQL 2005]. In our experiments, we exercise the system through real traces; each experiment has a duration of 30 minutes. The Web switch node, running a modified version of the Apache Web server [Apache 1999], is enriched with a threshold-based admission control mechanism and a weighted round-robin dispatcher where weights are based on resource measures or load predicted values. Upon the arrival of an HTTP request, the admission controller decides whether to serve or to refuse it by using direct or filtered load monitoring information coming from each server of the cluster. The admission threshold is set to 95% of the maximum processing capacity of the backend nodes, which are the most critical components of the system. If a request is admitted into the system, the dispatcher forwards it to the Apache-based HTTP server if it is for a static object, otherwise, it chooses through the weighted round-robin algorithm, a suitable application server if the request is for a dynamically generated resource.

We consider three instances of the admission controller and of the dispatcher. One is based on resource measures, the others are based on the two-phase framework where the load tracker uses the EMA_{90} or the CS_{240} models, and the load prediction is based on the model of Section 6 for $q = 5$ and $k = 10$. The activities of these three instances of the admission control mechanism in terms of refused requests are shown in Figure 22. From this figure, we can observe that



(a) Admission control based on resource measures

(b) Admission control based on the two-phase framework (EMA_{90})

(c) Admission control based on the two-phase framework (CS_{240})

Fig. 22. Number of refused requests during the entire experiment.

Table IX. Evaluation of the Two Admission Control Mechanisms

	90-percentile of the Web page response time	Percentage of refused requests	Activations of the admission refusal
Resource measures	3.08s	31%	140
Two-phase framework (EMA_{90} and prediction)	3.71s	10%	35
Two-phase framework (CS_{240} and prediction)	3.26s	17.34%	72

the use of the two-phase strategy tends to reduce the number of unnecessary activations of the admission control mechanism, which are due to transient load spikes, and consequently allows the system to reject a minor number of requests. However, there is visual evidence that the EMA and CS load trackers have different effects that we motivate in the following. Table IX summarizes the quantitative results of this case study. The first important result is that the two-phase framework does not penalize the overall performance of the system. Even if it accepts a much larger quantity of requests, the impact on the 90-percentile of the response time is not perceived by a user. Moreover, the use of the two-phase strategy reduces some (unnecessary) activations of the refusal mechanism and limits the number of refusals. These positive effects are due to the combined benefits of the dispatching algorithm and the admission control mechanism based on predicted values.

From Figure 22 and Table IX, we can also conclude that the prediction based on an EMA load tracker supports admission control algorithms more efficiently than the CS-based alternative. This is in complete accordance with the results shown in Section 6 where the prediction errors affecting the CS_{240} predictor were significantly higher than those characterizing the EMA_{90} predictor.

7.2 Locally Distributed Network Intrusion Detection System

In an Internet scenario characterized by a continuous growth of network bandwidth and traffic, the network appliances that have to monitor and analyze all flowing packets are reaching their limits. These issues are critical especially for a Network Intrusion Detection System (NIDS) that looks for evidences of

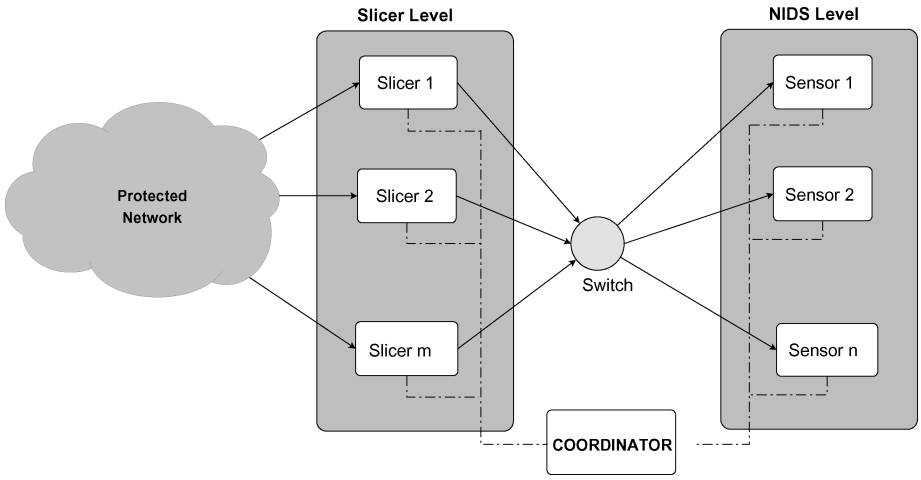


Fig. 23. Architecture of the distributed NIDS.

illicit activities by tracing all connections and examining every packet flowing through the monitored links.

Here, we consider a locally distributed NIDS (Figure 23) with multiple sensors that receive traffic slices by a centralized dispatcher as in Colajanni and Marchetti [2006]. The overall NIDS performance is improved if the number of packets reaching each traffic analyzer does not overcome its capacity and the load among the traffic analyzers is well balanced. To this end, the considered locally distributed NIDS is enriched by a load balancer that dynamically redistributes traffic slices among the traffic analyzers. This balancer is activated when the load of a traffic analyzer reaches a given threshold. In such a case, the load balancer redistributes traffic slices to other less-loaded traffic analyzers in a round-robin way until the load on the alarmed analyzer falls below the threshold. The distributed NIDS are exercised through the IDEVAL traffic dumps that are considered standard workloads for attacks [Lippmann et al. 2000].

The considered system shares an important characteristic of Internet-based servers, that is, a marked oscillatory behavior of the samples measured in each component that complicates load balancing decisions. As examples, we report the load on a distributed NIDS consisting of three traffic analyzers in Figure 24. The load is measured as a network throughput (in Mbps) that is shown to be the best load indicator. The horizontal line at 12Mbps denotes the threshold for the activation of the dynamic load balancer. The small vertical lines on top of each figure indicate the activation of a load redistribution process on that traffic analyzer. The consequences of making balancing decisions on the basis of periodic samples of the traffic throughput are clear: the mechanism for load redistribution is activated too frequently (63 times during the experiment lasting for 1200 seconds), but the load on the traffic analyzers is not balanced at all.

We apply the two-phase framework to the same NIDS system. In particular, we integrate the load balancer with a load-change detection model based on the

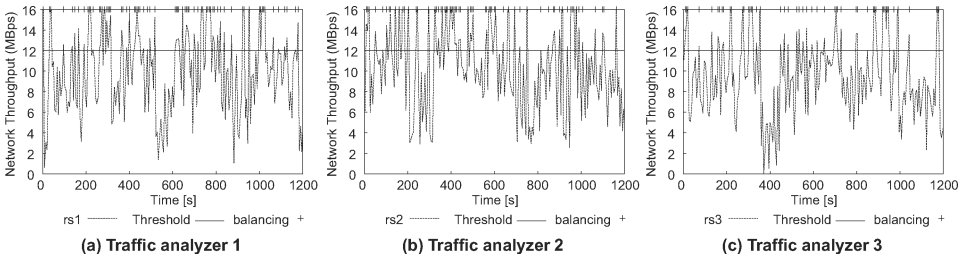


Fig. 24. Load on NIDS traffic analyzers when load balancing is based on resource measures.

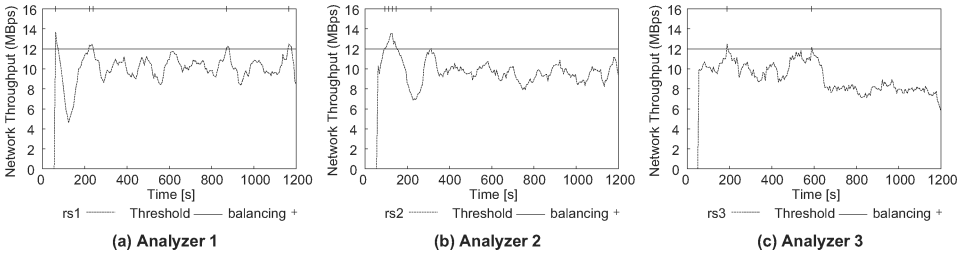


Fig. 25. Load on NIDS traffic analyzers when load balancing is based on a two-phase framework (SMA_{10} load tracker).

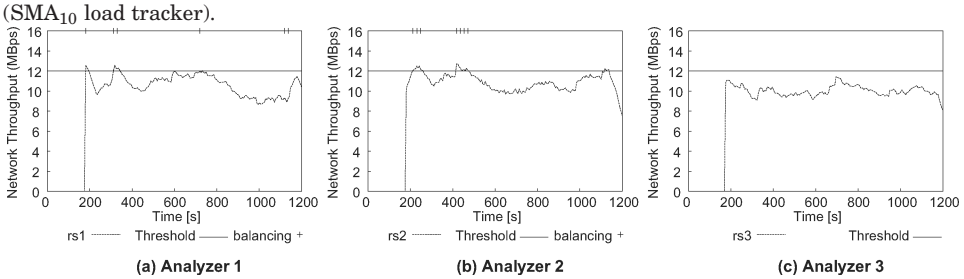


Fig. 26. Load on NIDS traffic analyzers when load balancing is based on a two-phase framework (EMA_{30} load tracker).

SMA and EMA of the last measures of the network throughput. Figure 25 and Figure 26 show the load balancing activities on the three traffic analyzers when the load change detector is based on SMA_{10} and EMA_{30}, respectively. A cross comparison among the Figures 24, 25 and 26 gives a first immediate result. Thanks to the two-phase framework, the mechanism for load redistribution is activated only a few times especially in the first part of the experiment. After an initial transient phase where the load balancer has to redistribute traffic among the traffic analyzers, the load remains more evenly distributed below the threshold and the number of load balancer activations decreases significantly.

The reduction of unnecessary activations of the load redistributor is an important result, but we are also interested to know which mechanism improves load balancing of the three traffic analyzers. To this purpose, we evaluate the coefficient of variation of the load on each traffic analyzer for the load change detector models based on EMA_{30}, SMA_{10}, and also resource samples for further comparison.

Table X. Evaluation of Load Balancing Mechanisms

	90-percentile of the Coefficient of Variation	Total number of load re-distributions
Activation based on samples	0.58	63
Activation based on SMA_{10}	0.20	12
Activation based on EMA_{30}	0.10	13

Table X summarizes the results of this case study. The load balancing systems that use the two-phase framework both reduce redistribution activities and improve the quality of load balancing: the 90-percentile of the coefficient of variation of the load change detector based on EMA_{30} is almost six time smaller than that based on resource measures. These results give a further confirmation that most of the redistributions carried out during the experiment based on resource measures were not only useless but also had a negative impact on load balancing.

8. RELATED WORK

Detecting significant and permanent load changes of a system resource and predicting its future load behavior are the basis for most runtime decisions for the management of Web distributed systems. Some examples of applications include load balancers [Pai et al. 1998; Castro et al. 1999; Bryhni 2000; Andreolini et al. 2003; Mitzenmacher 2000; Ferrari and Zhou 1987; Gautama and van Gemund 2006; Bahi et al. 2006], overload and admission controllers [Pai et al. 1998; Pandey and Barnes 1998; Kamra et al. 2004; Abdelzaher et al. 2002; Chen and Mohapatra 2003], request routing mechanisms and replica placement algorithms [Rabinovich et al. 2003; Karbhari et al. 2002; Pierre and Van Steen 2001; Sivasubramanian et al. 2004], distributed resource monitors [Rabinovich et al. 2006; Wolski et al. 1999]. The common method to represent resource load values for runtime management systems is based on the periodic collection of samples from server monitors and on the direct use of these values. Some low-pass filtering of network throughput samples has been proposed in Sang and Li [2000], but the large majority of proposals detect load changes and predict future values on the basis of some functions that work directly on resource measures. Even the studies that are based on a control theoretical approach to prevent overload or to provide guaranteed levels of performance in Web systems [Kamra et al. 2004; Abdelzaher et al. 2002] refer to direct resource measures (e.g., CPU utilization, average Web object response time) as feedback signals.

The problem with these approaches is that most modern Web-based systems are characterized by complex hardware/software architectures and by highly variable workloads that cause instability of system resource measures. Hence, real-time management decisions based on the direct use of these measures may lead to risky, it not completely wrong, actions. Our preliminary experimental results motivate the proposal for a two-phase strategy that first aims to represent the load trend of a resource (namely, load tracker), and then uses this load representation as the input for load change detectors and load predictors that are at the basis of many runtime decision systems. An initial idea of the

two-phase approach applied to the load prediction problem has been proposed by the authors in Andreolini and Casolari [2006]. However, this is the first paper that proposes a thorough study and a general two-phase methodology to support runtime decisions in the context of complex architectures and heavy-tailed workloads characterizing modern Web-based services. Moreover, in this article, we implement and integrate the overall methodology into a framework that has been demonstrated to work well for quite different distributed contexts. The architecture of many sophisticated load monitoring strategies and management tasks and the characteristics of heavy-tailed workloads are often too complex for an analytical representation [Luo and Marin 2005; Fishman and Adan 2006]. Unlike our article, which is based on a view of real systems, many previous studies have been oriented to simulation models of simplified Web-based architectures [Abdelzaher et al. 2002; Pai et al. 1998; Cherkasova and Phaal 2002; Stankovic 1984; Cardellini et al. 2000]. Although the simulation of a Web-based system is a challenging task in itself [Floyd and Paxson 2001] and has characterized many research efforts of the same authors, we have to admit that real systems open novel interesting and challenging issues.

There are many studies on the characterization of resource loads, albeit related to systems that are subject to quite different workload models than those considered in this article. Hence, many of the previous results cannot be applied directly to the Web-based systems considered here. For example, in Mitzenmacher [2000], the author evaluates the effects of different load representations on job load balancing through a simulation model that assumes a Poisson job interarrival process. A similar analysis concerning UNIX system is carried out in Ferrari and Zhou [1987]. Dinda and O'Hallaron [2000] investigate the predictability of the CPU load average in a UNIX machine subject to CPU bound jobs. The adaptive disk I/O prefetcher proposed in Tran and Reed [2004] is validated through realistic disk I/O interarrival patterns referring to scientific applications. On the other hand, the workload features considered in all these pioneer papers differ substantially from the load models characterizing Web-based servers that show high variability, bursty patterns, and heavy tails even at different time scales.

Some more recent studies refer to Web-based workloads but in the context of specific applications or tasks that are mainly oriented to admission control mechanisms. For example, Cherkasova and Phaal [2002] validate their session-based admission controller for Web servers through the SPECWeb96 workload [SpecWEB96 1996], that is now considered fairly obsolete [Iyer 2001; SpecWEB05 2005] with respect to the TPC-W workload [TPC-W 2004] that is becoming the de-facto standard benchmark for the analysis of Web-based systems for dynamic services. An interesting example of application is in Kamra et al. [2004] where the authors propose a self-tuning admission controller for multitier Web sites. However, no previous study is oriented to propose a general methodology for load tracking, load change detection, and load prediction.

The focus on runtime operations and consequent constraints is another key difference of this article with respect to previous literature. The most common

method for investigating the efficacy of load representation for runtime management tasks is offline analysis of samples collected from access or resource usage logs [Sang and Li 2000; Dinda and O'Hallaron 2000; Baryshnikov et al. 2005; Lingyun et al. 2003; Choi et al. 2003; Kelly 2005]. In this article, the need for runtime decision supports in a highly variable Web context has led to evaluate the feasibility of simple yet effective load models and predictors and the possibility of integrating them in an online framework. All the considered models must be characterized by low computational complexity. In our article, we consider linear and nonlinear models, which may be used as a trend indicator in other contexts (see, e.g., the cubic spline function in Eubank and Eubank [1999], Wolber and Alfy [1999], Poirier [1973]). The distributed resource monitor called Network Weather Service (NWS) [Wolski et al. 1999] collects resource measures periodically, and forecasts future sample values by means of linear averages, median estimates, or autoregressions. However, the NWS predictions are just one-step-ahead and are related to measured values; on the other hand, the proposed framework is able to generate k-step-ahead predictions of the load trend values.

Other linear models are widely adopted for load representation and prediction. For example, in Baryshnikov et al. [2005], the authors demonstrate how a simple linear extrapolation can predict a hot spot with good approximation. The simulation results presented in Cherkasova and Phaal [2002] show that the exponential moving average of the CPU utilization can be used as a valid indicator for the Web server load. We can confirm through real experiments applied to different distributed contexts that this hypothesis is in accordance with some of the results of this article. On the other hand, we can conclude that linear-time series models, which are often adopted to predict future load values [Tran and Reed 2004; Lingyun et al. 2003; Sang and Li 2000], are not really suitable to support runtime decisions for Web-based systems. The problem is that, in highly variable contexts, an autoregressive model such as ARIMA requires continuous updating of the parameters and is unsuitable to support most runtime management decisions.

9. CONCLUSIONS

In this article, we address two important issues that are the basis for several runtime management decisions in Web-based systems, detecting nontransient changes of the load conditions of a system resource, and predicting future load values of a resource.

Existing runtime management systems evaluate load conditions of system resources and, on this basis, decide whether and which action(s) it is important to carry out. We have shown that in the context of Web-based systems characterized by highly variable workload and complex hardware/software architectures, it is inappropriate to make decisions solely on the basis of system resource measures. The values obtained from load monitors of Web-based servers offer an instantaneous view of the load conditions of a resource and they are of little help in understanding the real load trends and for anticipating future load conditions.

For this reason, we propose a two-phase strategy that first aims to get a representative view of the load trend from resource measures through linear and nonlinear models that are computationally compatible to runtime constraints. Then, it uses the estimated load trends for solving decision problems such as the load change detection and the load prediction that are considered in this article.

We have integrated the two-phase methodology into a framework that is suitable for supporting different decision systems in real contexts. In this article, we have experimented with the proposed framework in a multitier Web system, in a Web cluster, and in a distributed NIDS for job dispatching, load balancing, and admission control purposes, as well as for a large set of representative workload models. In all contexts, the achieved results are quite encouraging. For this reason, we think that the proposed two-phase strategy can be extended to other problems, such as long-term prediction and trend analysis, and to many other applications that require precise and runtime decisions such as load sharing, load balancing, and request redirection even at a geographical scale. Web systems based on autonomic properties and GRID infrastructures are other interesting areas where the proposed framework and models could find immediate application.

ACKNOWLEDGMENTS

We would like to express our gratitude to the many people who offered input into this work. Special and sincere thanks go to the anonymous reviewers for their helpful comments. We thank Novella Bartolini and Francesco Lo Presti for their valuable hints. We acknowledge that the two-sided quartile-weighted median was suggested by Francesco, and the scatter plot by the first reviewer. We thank Mirco Marchetti who integrated the two-phase framework into his distributed NIDS architecture and helped us with the experiments.

REFERENCES

ABDELZAHER, T., SHIN, K. G., AND BHATTI, N. 2002. Performance guarantees for Web server end-systems: A control-theoretical approach. *IEEE Trans. Paral. Distrib. Syst. 13*, 1, 80–96.

ANDREOLINI, M. AND CASOLARI, S. 2006. Load prediction models in Web-based systems. In *Proceedings of the 1th International Performance Evaluation Methodologies and Tools Conference (VALUETOOLS'06)*. Pisa, Italy.

ANDREOLINI, M., COLAJANNI, M., AND NUCCIO, M. 2003. Scalability of content-aware server switches for cluster-based Web information systems. In *Proceedings of 12th International World Wide Web Conf. (WWW'03)*. Budapest, Hungary.

APACHE. 1999. Apache HTTP server project. `http://www.apache.org`.

ARLITT, M., KRISHNAMURTHY, D., AND ROLIA, J. 2001. Characterizing the scalability of a large Web-based shopping system. *IEEE Trans. Intern. Techn. 1*, 1, 44–69.

BAHI, J., CONTASSOT-VIVIER, S., AND COUTURIER, R. 2006. Dynamic load balancing and efficient load estimators for asynchronous iterative algorithms. *IEEE Trans. Paral. Distrib. Syst. 16*, 4, 289–299.

BARFORD, P. AND CROVELLA, M. E. 1998. Generating representative Web workloads for network and server performance evaluation. In *Proceedings of the 1st the Joint International Conference on Measurement and Modeling of Computer Systems (ACM SIGMETRICS'98/Performance'98)*. Madison, WI.

BARYSHNIKOV, Y., COFFMAN, E., PIERRE, G., RUBENSTEIN, D., SQUILLANTE, M., AND YIMWADSANA, T. 2005. Predictability of Web server traffic congestion. In *Proceedings of 10th International Workshop of Web Content Caching and Distribution (WCW'05)*. Sophia Antipolis, France.

BONETT, D. 2006. Approximate confidence interval for standard deviation of nonnormal distributions. *Comput. Statis. Data Anal. 50*, 3, 775–882.

BOX, G., JENKINS, G., AND REINSEL, G. 1994. *Time Series Analysis Forecasting and Control*. Prentice Hall.

BROCKWELL, B. L. AND DAVIS, R. A. 1987. *Time Series: Theory and Methods*. Springer-Verlag.

BRYHNI, H. 2000. A comparison of load balancing techniques for scalable web servers. *IEEE Netw. 14*, 4, 58–64.

CAIN, H. W., RAJWAR, R., MARDEN, M., AND LIPASTI, M. H. 2001. An architectural evaluation of Java TPC-W. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture (HPCA'01)*. Nuovo Leone, Mexico.

CANALI, C., XIAO, Z., AND RABINOVICH, M. 2004. Utility computing for Internet applications. In *Web Content Delivery*, X. Tang, J. Xu, and S. Chanson, Eds. Vol. 2. Springer Verlag, 131–151.

CARDELLINI, V., CASALICCHIO, E., COLAJANNI, M., AND YU, P. 2002. The state of the art in locally distributed Web-server system. *ACM Comput. Surv.* 2, 263–311.

CARDELLINI, V., COLAJANNI, M., AND YU, P. 2003. Request redirection algorithms for distributed Web systems. *IEEE Trans. Paral. Distrib. Syst. 14*, 5, 355–368.

CARDELLINI, V., COLAJANNI, M., AND YU, P. S. 2000. Geographic load balancing for scalable distributed Web systems. In *Proceedings of the 8th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'00)*. San Francisco, CA.

CASTRO, M., DWYER, M., AND RUMSEWICZ, M. 1999. Load balancing and control for distributed World Wide Web servers. In *Proceedings of the International Conference on Control Applications (CCA'99)*. Kohala Coast, HI.

CECCHET, E., CHANDA, A., ELNIKETY, S., MARGUERITE, J., AND ZWAENEPOEL, W. 2003. Performance comparison of middleware architectures for generating dynamic Web content. In *Proceedings of the 4th Middleware Conference*. Rio de Janeiro, Brazil.

CHALLENGER, J., DANTZIG, P., IYENGAR, A., SQUILLANTE, M., AND ZHANG, L. 2004. Efficiently serving dynamic data at highly accessed Web sites. *IEEE/ACM Trans. on Networ. 12*, 2, 233–246.

CHEN, H. AND MOHAPATRA, P. 2002. Session-based overload control in QoS-aware Web server. In *Proceedings of the 21th IEEE International Conference on Computer Communications (INFOCOM'02)*.

CHEN, H. AND MOHAPATRA, P. 2003. Overload control in QoS-aware Web servers. *Comput. Netw. 42*, 1, 119–133.

CHEN, X. AND HEIDEMANN, J. 2005. Flash crowd mitigation via an adaptive admission control based on application-level observations. *IEEE Trans. Inter. Tech. 5*, 3, 532–569.

CHERKASOVA, L. AND PHAAL, P. 1999. Session based admission control: A mechanism for improving performance of commercial Web sites. In *Proceedings of the 7th International Workshop on Quality of Service (IWQoS'99)*. London, UK, 226–235.

CHERKASOVA, L. AND PHAAL, P. 2002. Session-based admission control: A mechanism for peak load management of commercial Web sites. *IEEE Trans. Comput. 51*, 6, 669–685.

CHOI, B., PARK, J., AND ZHANG, Z. 2003. Adaptive random sampling for traffic load measurement. In *Proceedings of the 16th IEEE International Conference on Communications (ICC'03)*. Anchorage, AL.

COLAJANNI, M. AND MARCHETTI, M. 2006. A parallel architecture for stateful intrusion detection in high traffic networks. In *Proceedings of the IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation (MonAM'06)*. Tuebingen, Germany.

CROVELLA, M. E., TAQQU, M. S., AND BESTAVROS, A. 1998. Heavy-tailed probability distributions in the World Wide Web. In *A Practical Guide To Heavy Tails*. Chapman and Hall, 3–26.

DAHLIN, M. 2000. Interpreting stale load information. *IEEE Trans. Paral. Distrib. Syst. 11*, 10, 1033–1047.

DINDA, P. AND O'HALLARON, D. 2000. Host load prediction using linear models. *Cluster Comput. 3*, 4, 265–280.

DODGE, R. C., MENASCÉ, D. A., AND BARBARÁ, D. 2001. Testing e-commerce site scalability with TPC-W. In *Proceedings of the 27th International Computer Measurement Group Conference*. Orlando, FL.

DUFFIELD, N. G. AND LO PRESTI, F. 2000. Multicast inference of packet delay variance at interior network links. In *Proceedings of the 19th IEEE International Conference on Computer Communications (INFOCOM'00)*. Tel Aviv, Israel.

ELNIKETY, S., NAHUM, E., TRACEY, J., AND ZWAENEPOEL, W. 2004. A method for transparent admission control and request scheduling in e-commerce Web sites. In *Proceedings of the 13th International World Wide Web Conference*. New York, NY.

EUBANK, R. L. AND EUBANK, E. 1999. *Non parametric regression and spline smoothing*. CRC Press.

FERRARI, D. AND ZHOU, S. 1987. An empirical investigation of load indices for load balancing applications. In *Proceedings of the 12th IFIP WG 7.3 International Symposium on Computer Performance Modeling, Measurement and Evaluation (PERFORMANCE'87)*. Brussels, Belgium.

FISHMAN, G. AND ADAN, I. 2006. How heavy-tailed distributions affect simulation-generated time averages. *ACM Trans. Model. Comput. Simul. 16*, 2, 152–173.

FLOYD, S. AND PAXSON, V. 2001. Difficulties in simulating the Internet. *IEEE/ACM Trans. Networ. 9*, 3, 392–403.

FORSYTHE, G. E., MALCOLM, M. A., AND MOLER, C. B. 1977. *Computer Methods for Mathematical Computations*. Prentice-Hall.

GANEK, A. G. AND CORBI, T. 2003. The dawning of the autonomic computing era. *IBM Syst. J. 42*, 1, 5–18.

GAUTAMA, H. AND VAN GEMUND, A. 2006. Low-cost static performance prediction of parallel stochastic task compositions. *IEEE Trans. Paral. Distrib. Syst. 17*, 1, 78–91.

IYER, R. 2001. Exploring the cache design space for Web servers. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium (PDPS'01)*. San Francisco, CA.

JUNG, J., KRISHNAMURTHY, B., AND RABINOVICH, M. 2002. Flash crowds and denial of service attacks: Characterization and implications for CDNs and Web sites. In *Proceedings of the 11th International World Wide Web Conference (WWW'02)*. Honolulu, HW.

KAMRA, A., MISRA, V., AND NAHUM, E. M. 2004. Yaksha: A self-tuning controller for managing the performance of 3-tiered sites. In *Proceedings of the 12th International Workshop on Quality of Service (IWQOS'04)*. Montreal, Canada.

KARBHARI, P., RABINOVICH, M., XIAO, Z., AND DOUGLIS, F. 2002. ACDN: A content delivery network for applications. In *Proceedings of the 21st ACM International Conference on Management of Data (SIGMOD'02)*. Madison, WI.

KELLY, T. 2005. Detecting performance anomalies in global applications. In *Proceedings of the USENIX Workshop on Real, Large Distributed Systems (WORLDS'05)*. San Francisco, CA.

KENDALL, M. AND ORD, J. 1990. *Time Series*. Oxford University Press.

KEPHART, J. O. AND CHESS, D. M. 2003. The vision of Autonomic Computing. *IEEE Comput. 36*, 1, 41–50.

LILJA, D. J. 2000. *Measuring Computer Performance. A Practitioner's Guide*. Cambridge University Press.

LINGYUN, Y., FOSTER, I., AND SCHOPF, J. M. 2003. Homeostatic and tendency-based CPU load predictions. In *Proceedings of the 8th International Parallel and Distributed Processing Symposium (IPDPS'03)*. Nice, France.

LIPPMANN, R., HAINES, J. W., FRIED, D., AND KORBA, J. DAS, K. 2000. Analysis and results of the 1999 DARPA off-line intrusion detection evaluation. In *Proceedings of the 3rd International Workshop on Recent Advances in Intrusion Detection (RAID'00)*. London, UK.

LUO, S. AND MARIN, G. 2005. Realistic Internet traffic simulation through mixture modeling and a case study. In *Proceedings of the 37th IEEE Winter Simulation Conference (WSC'05)*. Orlando, FL.

MENASCÉ, D. AND KEPHART, J. 2007. Autonomic computing. *IEEE Intern. Comput. 11*, 1, 18–21.

MITZENMACHER, M. 2000. How useful is old information. *IEEE Trans. Paral. Distrib. Syst. 11*, 1, 6–20.

MySQL 2005. MySQL Database server. http://www.mysql.com/.

PAI, V. S., ARON, M., BANGA, G., SVENDSEN, M., DRUSCHEL, P., ZWAENEPOEL, W., AND NAHUM, E. M. 1998. Locality-aware request distribution in cluster-based network servers. In *Proceedings of the 8th*

ACM Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'98). San Jose, CA.

PANDEY, R. AND BARNES, J. F. OLSSON, R. 1998. Supporting quality of service in HTTP servers. In *Proceedings of the 17th ACM Symposium on Principles of Distributed Computing (PODC'98)*. Puerto Vallarta, Mexico.

PIERRE, G. AND VAN STEEN, M. 2001. Globule: A platform for self replicating Web documents. In *Proceedings of the 6th Conference on Protocols for Multimedia systems (PROMS'01)*. Enschede, The Netherlands.

POIRIER, D. J. 1973. Piecewise regression using cubic spline. *J. Amer. Statist. Ass. 68*, 343, 515–524.

PRADHAN, P., TEWARI, R., SAHU, S., CHANDRA, A., AND SHENOY, P. 2002. An observation-based approach towards self-managing Web servers. In *Proceedings of the 10th International Workshop on Quality of Service (IWQOS'02)*. Monterey, CA.

RABINOVICH, M., TRIUKOSE, S., WEN, Z., AND WANG, L. 2006. DipZoom: The Internet measurement marketplace. In *Proceedings of the 9th IEEE Global Internet Symposium*. Barcelona, Spain.

RABINOVICH, M., ZHEN, X., AND AGGARWAL, A. 2003. Computing on the edge: A platform for replicating Internet applications. In *Proceedings of the 8th International Workshop of Web Content Caching and Distribution (WCW'03)*. Hawthorne, NY.

RAMANATHAN, P. 1999. Overload management in real-time control applications using (m,k)-firm guarantee. *Perform. Eval. Rev. 10*, 6, 549–559.

SANG, A. AND LI, S. 2000. A predictability analysis of network traffic. In *Proceedings of the 19th IEEE International Conference on Computer Communications (INFOCOM'00)*. Tel Aviv, Israel.

SATYANARAYANAN, M., NARAYANAN, D., TILTON, J., FLINN, J., AND WALKER, K. 1997. Agile application-aware adaptation for mobility. In *Proceedings of the 16th ACM International Symposium on Operating Systems Principles (SOSP'97)*. Saint-Malo, France.

SIVASUBRAMANIAN, S., PIERRE, G., AND VAN STEEN, M. 2004. Replication for Web hosting systems. *ACM Comput. Surv. 36*, 3, 291–334.

SpecWEB05 2005. The SPECWeb05 benchmark. http://www.spec.org/osg/web2005/.

SpecWEB96 1996. The SPECWeb96 benchmark. http://www.spec.org/osg/web96/.

STANKOVIC, J. A. 1984. Simulations of three adaptive, decentralized controlled, job scheduling algorithms. *Comput. Netw. 8*, 3, 199–217.

Tomcat 2005. The Tomcat Servlet Engine. http://jakarta.apache.org/tomcat/.

TPC-W 2004. TPC-W transactional Web e-commerce benchmark. http://www.tpc.org/tpcw/.

TRAN, N. AND REED, D. 2004. Automatic ARIMA time series modeling for adaptive I/O prefetching. *IEEE Trans. Paral. Distrib. Syst. 15*, 4, 362–377.

UTTS, J. M. 2004. *Seeing Through Statistics*. Thomson Brooks/Cole.

WILDSTROM, J., STONE, P., WITCHEL, E., MOONEY, R., AND DAHLIN, M. 2005. Towards self-configuring hardware for distributed computer systems. In *Proceedings of the 2nd International Conference on Autonomic Computing (ICAC'05)*. Seattle, WA.

WOLBER, G. AND ALFY, I. 1999. Monotonic cubic spline interpolation. In *Proceedings of the International Conference on Computer Graphics*. Canmore, CA, 188.

WOLSKI, R., SPRING, N. T., AND HAYES, J. 1999. The Network Weather Service: A distributed resource performance forecasting service for metacomputing. *Future Generation Comput. Syst. 15*, 5, 757–768.

Received Janyary 2007; revised July 2007, October 2007; accepted February 2008