An autonomic admission control policy for distributed web systems

Novella Bartolini, Giancarlo Bongiovanni, Simone Silvestri
Department of Computer Science
University of Rome "La Sapienza", Italy
E-mail: {bartolini,bongiovanni,simone.silvestri}@di.uniroma1.it

***Abstract*—This paper tackles the problem of autonomic admission control for web clusters. The main contribution of this work is the proposal of a new session admission algorithm that self-configures a dynamic constraint on the rate of incoming new sessions to guarantee the respect of Service Level Agreements (SLA). Unlike other approaches, our policy does not need any prior information on the incoming traffic, nor any assumption on the probability distribution of request inter-arrival or service time. Furthermore, it does not require any manual configuration or parameter tuning.**

We performed extensive simulations under a range of operating conditions and compared our algorithm to other previously proposed approaches. The simulations show that our policy rapidly adapts to the given traffic profile and improves service throughput while respecting the response time constraints imposed by the SLAs. It also improves service quality by reducing the oscillations of response time and number of active clients common to other policies.

I. Introduction

Nowadays a huge gamma of applications is accessed through the Web. While it is impossible to formulate a unique Quality of Service (QoS) requirement for all web based applications, some key features are highly desirable and even mandatory in many cases. Among these, overload control is commonly recognized to be of primary concern. Most of the previously proposed solutions rely on laborious parameter tuning and manual configuration. Our study instead focuses on adaptivity and autonomy of parameter configuration.

We address the problem of admission control policies for server clusters offering typical session based services. Such policies are implemented on Edge Points (EP), which are access routers or application level dispatchers. EPs intercept requests and make decisions to block or accept incoming new sessions to increase the user perceived quality of admitted sessions while at the same time providing an acceptable admission probability.

Performance requirements are often detailed into Service Level Agreements (SLAs). They can be mapped onto component-level thresholds, whose suitable values are seldom known a priori even in the most stable traffic scenarios. Furthermore, the relationship between component-level and SLA parameters varies under time-varying workload scenarios and cannot be computed offline. To this purpose, we propose the use of a learning activity to be performed at runtime by the EPs, in order to monitor the application workload and the system capacity. We designed an admission control algorithm that works on the basis of such gathered information. According to our proposal, the EPs dynamically calculate an upper limit on the admission rate of new sessions. By continuously adapting this limit to the most recently gathered data, the system is also responsive to workload changes.

We designed a synthetic traffic generator, based on industrial standard benchmarks and tested the proposed policy through extensive simulations. The experiments showed that the system is capable of self configuring the component-level parameters to achieve the highest throughput compatible with the agreements on quality specified in the SLA. We compared our method to other admission control policies proposed in literature, showing that it respects the agreements on quality in all traffic scenarios, without any previous parameter tuning, except for the SLA limit values, while the performance of other admission control policies is very dependent on the initial parameter setting. Our method also outperforms the others by avoiding undesired oscillations of response time, acceptance probability and of the number of ongoing sessions.

II. Related Work

An exhaustive survey of related work in the area of admission control for Web servers is out of the scope of this paper and due to space limitations we just mention the ones that are related to the area of autonomic computing. There is an impressively growing interest in self-managing systems, starting from several industrial initiatives from IBM [1], Hewlett Packard [2] and Microsoft [3]. Nevertheless, little effort has been spent on the problem of autonomous tuning of QoS policies for web systems.

The problem of designing adaptive component-level thresholds is analyzed in [4] for a general context of autonomic computing. A critical problem of this proposal is the fact that the most common threshold policies cause on/off behaviors that often result in unacceptable performance. Our proposal is instead based on a probabilistic approach and on a learning technique, that dynamically creates a knowledge basis for the online evaluation of the best decision to make even for traffic situations that never occurred in the past history.

The authors of [5] study the problem of timing performance control in the design of QoS policies. They propose to adapt the time interval between successive decisions to the size of workload dependent system parameters, such as the processor

queue length. We show that our algorithm shows a very little dependence on the time interval between decisions.

The problem of autonomously configuring a computing cluster to satisfy SLA requirements is addressed in [6]. Unlike our work, this paper proposes a policy whose decisions concern the reconfiguration of resource allocation to services.

The authors of [7] propose a technique for learning dynamic patterns of web user behavior. A finite state machine representing the typical user behavior is constructed on the basis of past history and used for prediction and prefetching techniques.

In [8] the problem of delay prediction is analyzed on the basis of a learning activity exploiting passive measurements of query executions. The proposed method is used to enhance traditional query optimizers.

Proposals [7] and [8] contribute to improve the QoS of web systems, but differently from our work, none of them directly formulate a complete autonomic solution that at the same time gives directions on how to take measures and make corresponding admission control decisions for web cluster architectures.

III. Service Level Agreements

The usual SLA formulation for web services imposes an upper limit to the $X_{\mathtt{SLA}}\% - ile$ of the response time of undifferentiated requests. A lower bound $\lambda_{\mathtt{SLA}}$ on the admission rate of new sessions $\lambda_{\mathtt{adm}}(t)$ is also considered, imposing that at any observation instant t, $\lambda_{\mathtt{adm}}(t)$ be at least equal to $\min\{\lambda_{\mathtt{in}}(t), \lambda_{\mathtt{SLA}}\}$, where $\lambda_{\mathtt{in}}(t)$ is the incoming session rate.

Although our work may be applied to different formulations of SLA, including the most typical one cited above, we argue that a more detailed service level agreement should be in place when web clusters are based on a tiered architecture [9], [10]. In such architectures, requests involving a processing activity at different tiers may require processing times that differ by orders of magnitude.

We consider the following SLA:

- $RT^i_{\mathtt{SLA}}$: maximum acceptable value of the 95%-ile of the response time for requests of type $i \in \{1, 2, \ldots, K\}$, where K is the number of cluster tiers.
- $\lambda_{\mathtt{SLA}}$: minimum guaranteed admission rate. If $\lambda_{\mathtt{in}}(t)$ is the rate of incoming sessions, and $\lambda_{\mathtt{adm}}(t)$ is the rate of accepted sessions, this agreement imposes that $\lambda_{\mathtt{adm}}(t) \geq \min\{\lambda_{\mathtt{in}}(t), \lambda_{\mathtt{SLA}}\}$
- $T_{\mathtt{SLA}}$: observation period during which measurement samples are gathered to check the satisfaction of the SLA constraints.

IV. Autonomic Admission Control Algorithm

We propose a probabilistic admission control algorithm that accepts as many new sessions as possible within the performance constraints imposed by the SLA described in section III. The admission probability is dynamically adjusted according to a prediction of the future workload and system availability, and to an estimate of the corresponding $95\%-$ile of the response time.

Our algorithm is shown in figure 1: the described steps are iteratively executed over fixed time intervals of length $t^{\mathtt{AACA}}_{\mathtt{obs}}$. Phase *B* represents the continuous time behavior of the algorithm during the time interval $[t_n, t_{n+1})$, while phases *C*, *D* and *E* are executed at the end of each iteration.

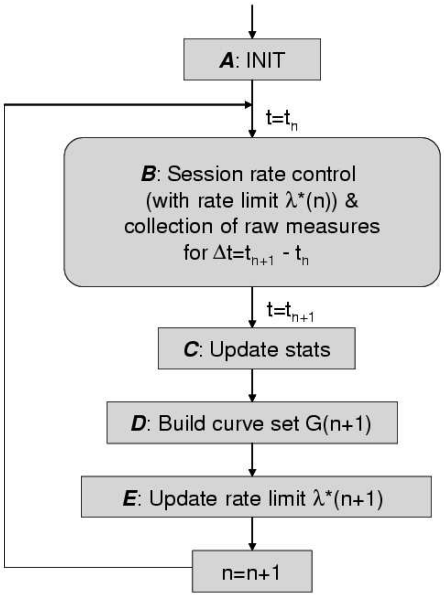

Fig. 1. Autonomic Admission Control (AACA) algorithm

For sake of simplicity, we leave the description of the parameter initialization (phase *A*) at the end of the algorithm description.

1) Session admission control and raw measurement - phase B: According to the algorithm phase *B*, the admission rate of new sessions is limited to $\lambda^*(n)$. This rate limitation is obtained by admitting new sessions with probability $p(n) = \min\{1, \lambda^*(n)/\hat{\lambda}_{\mathtt{in}}(n)\}$, where $\hat{\lambda}_{\mathtt{in}}(n)$ is a prediction of the session arrival rate for the next iteration interval. This prediction is based on the incoming session rate $\lambda_{\mathtt{in}}(n-1)$ observed during the previous time interval, that is, $\hat{\lambda}_{\mathtt{in}}(n) = \lambda_{\mathtt{in}}(n-1)$.

During each time interval, raw measures of the response time of requests belonging to ongoing sessions are collected to be processed at the execution of phase *C*. We define the set $\mathcal{T}^n_i$ as the set of observed response times (measured by the EP) for requests of type $i, i \in \{1, 2, \ldots, K\}$ during the time interval $[t_n, t_{n+1})$.

2) Statistics update - phase C: Phase *C* is executed at the end of each time interval. It consists in the calculation of some statistic parameters based on the raw measurements collected during phase *B*:

- $RT^i(n)$, that is the 95%-ile of the set $\mathcal{T}^n_i$, for $i \in \{1, 2, \ldots, K\}$,
- $\lambda_{\mathtt{in}}(n)$, that is the average incoming rate of new sessions observed during the time interval $[t_n, t_{n+1})$,
- $\lambda_{\mathtt{adm}}(n)$, that is the average rate of actually admitted sessions during the time interval $[t_n, t_{n+1})$.

3) Build curve set - phase D: Purpose of this phase is the use of the statistics collected during the previous phase, to build K time varying curves that represent the relation between the session admission rate and the observed response times of requests involving the K tiers of the cluster architecture.

A statistical measure calculated from raw samples taken during a single iteration may be not fully representative for two reasons: first, the workload is subject to variations that may cause transient effects; second, the number of samples may not be sufficient to ensure an acceptable confidence level. For this reason, we introduce a method to collect measures and associate them with a metric that gives a quantitative evaluation of their representativeness.

Let us consider the set of raw measures $\mathcal{R}_i^N \triangleq \{(\lambda_{\texttt{adm}}(n), RT^i(n)), n \in \{N_{\texttt{min}}, \ldots, N\}\}$, where $i \in \{1, 2, \ldots, K\}$, $N_{\texttt{min}} \triangleq \max\{(N - N_{\texttt{delete}}), 0\}$ and where the value $N_{\texttt{delete}}$ is introduced to limit the usage of obsolete measures, according to an aging technique based on a time window.

Let us partition the Cartesian plane into rectangular intervals of length l_λ along the $\lambda_{\texttt{adm}}$ axis as shown in figure 2. For every interval $[(k-1)l_\lambda; kl_\lambda)$, with $k = 1, 2, \ldots$ we define the set $\mathcal{P}_k^i \triangleq \{P \in \mathcal{R}_i^N, P = (\lambda_{\texttt{adm}}, RT^i),$ s.t. $\lambda_{\texttt{adm}} \in [(k-1)l_\lambda; kl_\lambda)\}$. We calculate the *barycenter* B_k^i of the k-th interval as the point with average coordinates over the points of the same interval.

An interval has no barycenter if there is not any recent measure available in $\mathcal{P}_k^i$ in any n-th interval, with $n \geq N_{\texttt{min}}$. Figure 2 shows the collected statistics taken at run-time at the database tier of an example scenario. It also evidences the calculated barycenters.

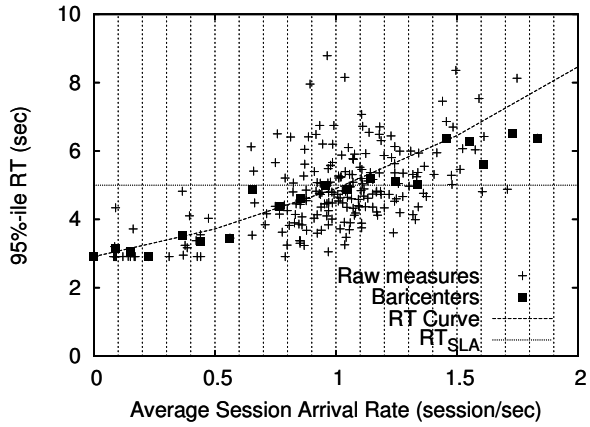

Fig. 2. Curve set construction

We define the *curve set* $G^i(n)$ as the set of barycenters calculated at the n-th iteration. We introduce the metric $v(B)$, for $B \in G^i(n)$ to give a quantitative evaluation of the reliability of the measure represented by the barycenter B. In particular, be $B_k^i = (\lambda_k^B, t_k^{Bi})$ the barycenter of the k-th interval. Be l_t an arbitrarily sized interval along the *response time* axis. Let us define the set $Square(B_k^i) \triangleq \{P \in \mathcal{P}_k^i, P = (\lambda_P, t_P),\ s.t.\ t_P \in [t_k^{Bi} - l_t, t_k^{Bi} + l_t]\}$. We define $v(B_k^i)$ as the cardinality of the set $Square(B_k^i)$.

Thanks to the aging technique and to the frequent updates, the curve set $G^i(n)$ is a highly dynamic structure, that continuously adapts itself to changing workload situations making it possible to forecast the response time corresponding to any possible workload rate, by means of interpolation techniques.

As initial setting of our curve construction phase we insert in the set $G^i(0), i \in \{1, 2, \ldots, K\}$ a point that represents the lower bound on the 95%-ile of the response times of the K types of requests. This point is the 95%-ile of response time measured when the system is in a completely idle state, that is when $\lambda_{\texttt{adm}} \widetilde{=} 0$. In order to calculate the average response time of this situation we use an offline benchmark, obtaining the point $P_{\texttt{bench}}^i = (0, RT_{\texttt{bench}}^i)$. Notice that this parameter is a characteristic of the hardware architecture in use and is not subject to manual tuning since it does not vary with time.

4) Update policy - phase E: This phase starts with a test to verify the validity of the rate limit $\lambda^*(n)$ adopted in phase *B*. To this extent, we define two types of error in the evaluation of $\lambda^*(n)$:

1) the rate limit was respected but the agreements were violated for at least one type of requests: the rate limit adopted in the present interval could have been too high and should be decreased to meet the SLA;
2) the rate limit was exceeded but none of the agreements was violated: the rate limit adopted in the present interval could have been too low and can be increased to improve the service throughput.

If none of these errors occurred, the upper limit on the admission rate of new sessions was properly set. Assuming that the traffic workload that will be observed in the next iteration will be similar to the one observed in the current iteration, there is no need to change the value of the rate limit. Therefore, in absence of errors, $\lambda^*(n+1) = \lambda^*(n)$.

If otherwise, one of these errors has occurred the value of $\lambda^*(n+1)$ needs to be updated. To this purpose phase *E* defines the construction of a function $t^i = f^i(\lambda_{\texttt{adm}})$ that represents the curve set in its most reliable points. In particular we refer to the set $\widetilde{G^i}(n) \subseteq G^i(n) = \{F \in G^i(n)\ s.t.\ v(F) \geq L \times \max_{B \in G^i(n)} v(B)\}$, where L is a given percentage (in the following experimental section VII we set $L = 80\%$). This way we consider the only barycenters that have been calculated on the basis of a sufficient number of occurrences of raw measures. The function $t^i = f^i(\lambda_{\texttt{adm}})$ is calculated by interpolating the trend between subsequent points of the set $\widetilde{G^i}(n)$. The updated rate limit can thus be computed as $\lambda^*(n) = \min_{i=1,\ldots,K} f^{i\,-1}(RT_{\texttt{SLA}}^i)$. Due to space limitations, we do not give details on the construction of the function $f^i(n)$ and on the interpolation technique. We refer the reader to the technical report [11] for further details on this topic.

5) Initial setting - phase A: As initial setting of our algorithm we use $n = 0$, $\lambda^*(0) = \lambda_{\texttt{SLA}}$ and $p(n) = 1$. The setting of the point $P_{\texttt{bench}}^i$ with value $P_{\texttt{bench}}^i = (0, RT_{\texttt{bench}}^i)$ has been detailed in section IV-3. $P_{\texttt{bench}}^i$ constitutes the first

point in the curve construction; it can be substituted with the origin $O = (0, 0)$, with no impact but a little difference in the time to converge to a stable choice of $\lambda^*(n)$.

V. Other admission control strategies

In this section we describe other previously proposed QoS policies to make performance comparisons. These policies can be formulated in many variants depending on the considered performance objectives. We limit our analysis to the optimization of response time which is tightly bounded to the user perceived quality of web applications.

A. Threshold Based Admission Control

Fixed threshold policies have been proposed in many fields of computer science, and in particular for web applications with several variants [12], [13], [14].

According to the Threshold Based Admission Control (TBAC) policy, the EP makes periodic evaluations of the 95%-ile of response time, every $t_{\mathtt{obs}}^{\mathtt{TBAC}}$ seconds. If the calculated value exceeds a threshold $T_{\mathtt{AC}}^{\mathtt{TBAC}}$, the EP rejects new sessions and only accepts requests that belong to ongoing sessions. On the contrary, if the value of the 95%-ile of the response time is lower than $T_{\mathtt{AC}}^{\mathtt{TBAC}}$, new sessions are always accepted.

B. Probabilistic Admission Control

The Probabilistic Admission Control (PAC) is a well known technique of control theory, commonly used when oscillations are to be avoided. This policy was proposed for Internet services in [15], while a similar version was also proposed in [16]. According to this policy, a new session is admitted with a certain probability, whose value depends on the measured response time. The system accepts all new sessions as long as the server load is sufficiently low, i.e. the measured response time is under the threshold $T_{\mathtt{low}}^{\mathtt{PAC}}$. The acceptance probability is gradually reduced as the load grows, while the system does not accept any new session when the workload exceeds the threshold $T_{\mathtt{high}}^{\mathtt{PAC}}$. In the general case the acceptance probability is a piece-wise linear function of the measured 95%-ile of the response time r, and has the following formulation:

$$p(r) \triangleq \begin{cases} 1 & \text{if r } \leq T_{\mathtt{low}} \\ \frac{r - T_{\mathtt{high}}}{T_{\mathtt{high}} - T_{\mathtt{low}}} & \text{if } T_{\mathtt{low}} < r \leq T_{\mathtt{high}} \\ 0 & \text{if } r > T_{\mathtt{high}} \end{cases} \quad (1)$$

The value of r and of the admission probability $p(r)$ are updated every $t_{\mathtt{obs}}^{\mathtt{PAC}}$ seconds.

VI. Simulation environment

In order to make performance comparisons among the different policies, we developed a simulator on the basis of the OPNET modeler software [17]. In this section we describe how we model the main entities of the simulation environment: clients, EP and application servers.

A. Client model and traffic generation

According to our model, each client issues a user session consisting of a sequence of logically related requests.

We assume that the interarrival time of new sessions follows a negative exponential distribution with average $1/\lambda$. The interarrival time of requests belonging to the same session is more complex: after the first request, each client injects subsequent requests waiting for the corresponding response (response time), and spending some time analyzing the content of the received response (think time).

The typical user interaction with a web application can be represented as a particular sequence of phase transitions. In order to have a realistic traffic generator, we used the phase model of an industrial standard benchmark: SPECWEB2005 [18]. This benchmark takes into account the most relevant types of dynamic content (php and jsp included). It models an e-commerce site for the sale of assembled personal computers. Each phase of this model represents a different step of the interaction of the client with the web application, from the login procedure to the customization of a product and the possible purchase. Transitions among phases may happen with given probabilities. We refer to [18] for a detailed description of the state model and of the functionalities of each phase.

Upon reception of a response, the next request is sent after the think time $T_{\mathtt{think}}$ spent by the user analyzing the received web page. Our model of $T_{\mathtt{think}}$ is based on TPC-W [19], [20] and on other works in the area of web traffic analysis such as [21]. As in the TPC-W model, we assume an exponential distribution of think times. We also assume a lower bound of 1 sec to the think time. Therefore $T_{\mathtt{think}} = \max\{-\log(r)\mu, 1\}$, where r is uniformly distributed in the interval (0,1] and $\mu = 10$ sec.

To model a realistic user behavior, we also introduce a timeout to represent the maximum response time tolerable by clients. Therefore, the interaction of a client with the web application may end for three possible reasons: *a*) session block, the EP denies service to the client's first request (no navigation session is started); *b*) session drop, no response is received within the client timeout and the ongoing session is interrupted; *c*) successful session termination, all responses are received in time and the user voluntary terminates the session.

B. Edge point model

The role of the EP is of primary importance for our proposal, since it hosts the implementation of the admission controller, as extensively detailed in section V.

C. Server model

In our experiments we refer to a typical three tier cluster organization: *pure http requests* involve the http server tier, *servlet requests* involve the application tier while *database requests* reach the third tier requiring the execution of a query to the database. Requests are served using a time sharing, round robin scheduling. Different types of requests are characterized by different processing times. We use an approximate estimate of the average processing times of the different categories on

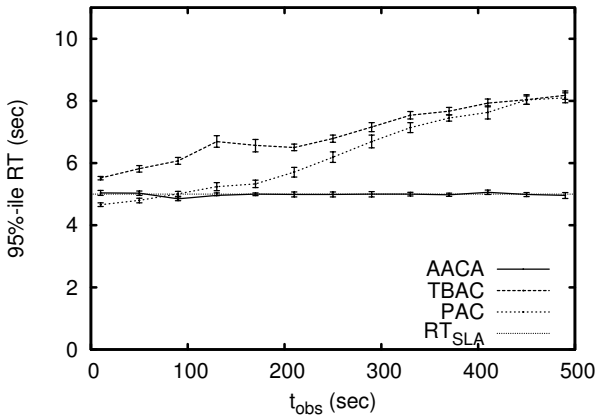

Fig. 3. 95%-ile of database RT

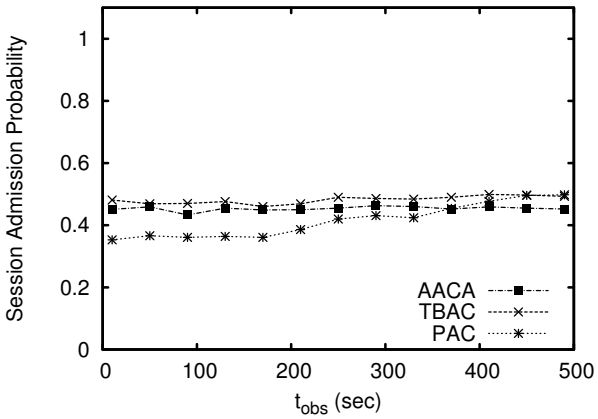

Fig. 4. Session admission probability

the basis of the experiments detailed in [13]. The service time of a single session request is exponentially distributed with the following parameters: average execution time of pure http requests is 0.001 sec, while for servlet request is 0.01 sec and for database requests is 1 sec.

In the same way we defined the client timeout, we also define a timeout at the server level, so that, in our model, servers do not waste computing resources processing requests coming from clients who already abandoned the site. Each server drops the requests that have not been completed before the expiration of the corresponding client timeout.

VII. Simulation Results

In this section we present a comparative study of the AACA, TBAC and PAC policy. For sake of brevity, we conduct our analysis on the database tier only. The database tier is in fact the one that most frequently becomes the bottleneck of typical web architectures.

All the experiments of this section are conducted with 10 application servers and a client timeout of 8 sec. The fixed threshold $T_{\mathrm{AC}}^{\mathrm{TBAC}}$ of the TBAC policy is always set in agreement with the SLA constraints on the 95%-ile of response time: $T_{\mathrm{AC}}^{\mathrm{TBAC}} = RT_{\mathrm{SLA}}$. The thresholds of the PAC policy are defined as follows: $T_{\mathrm{low}}^{\mathrm{PAC}} = 3$ sec and $T_{\mathrm{high}}^{\mathrm{PAC}} = RT_{\mathrm{SLA}}$, in agreement with the SLA constraints.

The first set of experiments (figures 3 and 4) is introduced to study the sensitivity of these policies to the parameter setting. A key parameter in the definition of the three policies is the time between successive decisions t_{obs}.

Figures 3 and 4 show that the performance of the AACA policy does not vary significantly when varying the value of t_{obs}. The same cannot be said for the other two policies: TBAC and PAC. To understand the reasons of the different behavior of the three policies let us consider first the case of AACA. For short values of t_{obs}, statistic parameter values are calculated frequently on the basis of poor sets of raw measures. The low confidence level of such metrics is compensated by the high number of points that are used to construct the curve set (phase *D* of the algorithm, as detailed in section IV-3). Such points contribute to the quantitative evaluation of the reliability function $v(\cdot)$ we define on the set of barycenters. The choice of reliable barycenters allows our algorithm to correctly reconstruct the relationship between the session admission rate and the 95%-ile of the response time and then to find the maximum admission rate that can be adopted to respect the agreements on quality. As t_{obs} grows, less points are available for the construction of the curve set. In this case, although the curve set is constructed on the basis of a smaller number of points, it still permits a good estimate of the maximum adoptable admission rate because these points have a higher confidence level. Although the period between subsequent decisions is a parameter that requires manual sizing, this set of experiments shows that for the AACA policy there is no need to perform fine and laborious tuning. On the contrary, the TBAC and PAC policy benefit from shorter periods between succeeding decisions. Short decision periods make the system capable of rapidly correcting wrong decisions and of reacting to workload changes. Long decision periods instead induce more intense oscillations in case of high load and cause scarce utilization in low load situations. The plot of error bars in figure 3 as well as in the following figure 5 shows the effective differences in performance among the three policies and the reliability of the simulation results.

Aim of the second set of results (figures 5 and 6) is to show the behavior of the three policies under varying workloads.

In the following experiments we set the SLA threshold, RT_{SLA}, to 5 sec. Since the TBAC and PAC policy take advantage of short periods, while the AACA policy is almost independent of this setting, in order to have fair comparisons among the three policies we use the following parameter setting for the subsequent experiments: $t_{\mathrm{obs}}^{\mathrm{TBAC}} = t_{\mathrm{obs}}^{\mathrm{PAC}} = 10$ sec (a sufficiently short period that allows the TBAC and PAC policy to work at their best), while we set $t_{\mathrm{obs}}^{\mathrm{AACA}} = 60$ sec (in order to relieve the system of the too frequent unnecessary decisions that would be made with shorter inter-observation periods).

As figure 5 shows, when the traffic load is high, the AACA policy finds the suitable session arrival rate and admits as many

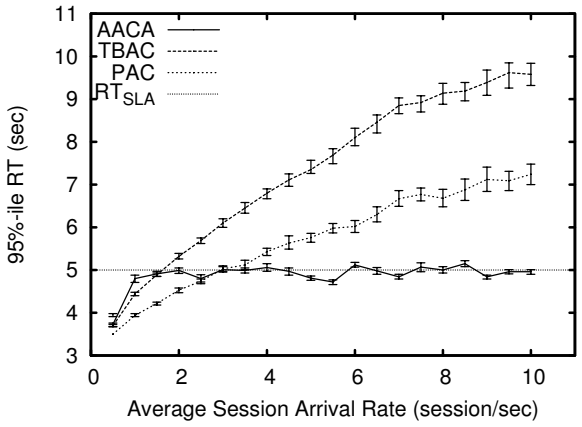

Fig. 5. 95%-ile of database RT

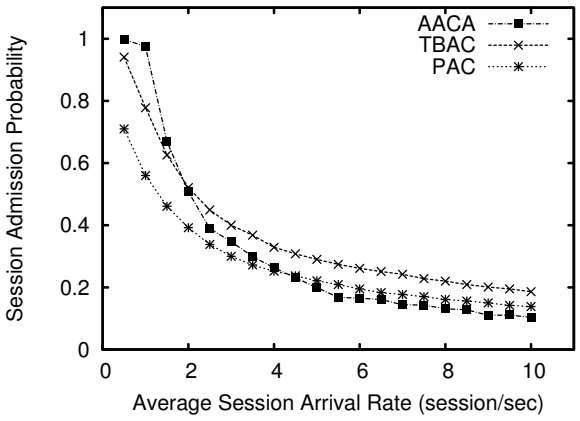

Fig. 6. Session admission probability

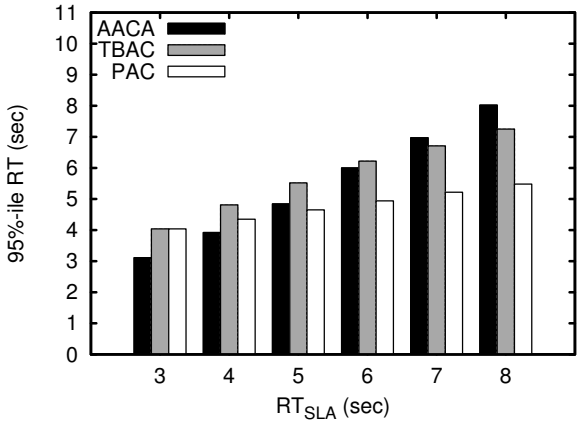

Fig. 7. 95%-ile of database RT

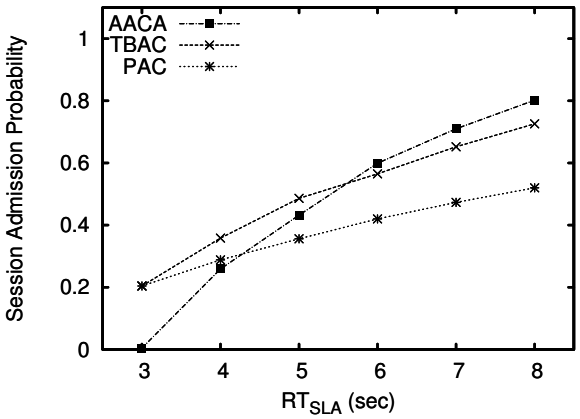

Fig. 8. Session admission probability

sessions as possible to remain under the SLA limits. When the traffic is low, it accepts almost all incoming sessions, as figure 6 points out.

On the other hand, the other two policies work properly only for a short range of operating conditions. As usually happens with non adaptive policies, TBAC and PAC under-utilize the system resources in low workload conditions, and violate the QoS agreements when the workload is too high.

Figures 5 and 6 reveal the possible difficulties in the choice of the proper parameter setting for the TBAC and PAC policies, since a value that may work for a particular traffic scenario (e.g. thresholds $T_{\mathrm{AC}}^{\mathrm{TBAC}} = 5$ sec when $\lambda = 1.5$ sessions/sec or $T_{\mathrm{high}}^{\mathrm{PAC}} = 5$ sec when $\lambda = 3$ sessions/sec) may cause SLA violations or system under-utilization in other cases.

Purpose of the third set of simulations (figures 7 and 8) is to test the policy behavior with different SLAs. We tested the AACA, TBAC and PAC policies under six different values of the SLA threshold RT_{SLA}, ranging from 3 to 8 seconds. We set $T_{\mathrm{AC}}^{\mathrm{TBAC}} = T_{\mathrm{high}}^{\mathrm{PAC}} = RT_{\mathrm{SLA}}$, while $T_{\mathrm{low}}^{\mathrm{PAC}} = 3$ sec.

Figures 7 and 8 show the 95%-ile of the database response time and of the new session admission probability respectively. These figures evidence the main drawbacks of the TBAC and PAC policies discussed earlier. In case of low threshold (from 3 to 6 seconds), TBAC does not prevent SLA violations, while for higher values it causes system under-utilization. This behavior is confirmed by the trend of the session admission probability: too many sessions are accepted or refused for low and high SLA threshold values respectively.

In the same scenario the PAC policy performs a more conservative admission control, admitting less sessions than the TBAC policy. Although the PAC policy admits less sessions than the TBAC, it still violates the SLA under low thresholds (from 3 to 4 seconds), and under-utilizes the system resources when high thresholds (from 5 to 8 seconds) are in use. The AACA policy, instead, always respects the SLA, guaranteeing a high probability of admitting new sessions without under utilizing the system.

The last set of results (figure 9) studies the problem of performance stability. To this extent we analyze how the number of active users varies over time.

The TBAC policy shows an evident oscillatory behavior due to its on/off nature. The PAC policy, although introduced with the explicit goal to reduce the oscillations that commonly characterize the threshold based policies, also shows a high variability of the number of active sessions.

One of the main reasons why these policies cannot guarantee a stable behavior is that the admission controller works at session granularity. In fact, an admitted session traverses the cluster tiers according to a given life-cycle phase model (as discussed in section VI-A), and for this reason does not have an immediate impact on all cluster tiers. At the same time, a decision to stop admitting new sessions will have an impact on the perceived performance only after the end of a sufficient number of sessions. This induces the TBAC policy to accept all incoming sessions for too many iterations before the measured response time exceeds the threshold. Once the threshold is exceeded, this policy reacts refusing all incoming sessions, but the response time remains over the threshold until the end of a sufficient number of sessions. The PAC policy reduces the amplitude of the oscillations, with respect to TBAC, but does not solve this problem completely. The AACA policy, instead, shows a stable behavior thanks to the learning mechanism that allows the system to properly adapt the session admission rate.

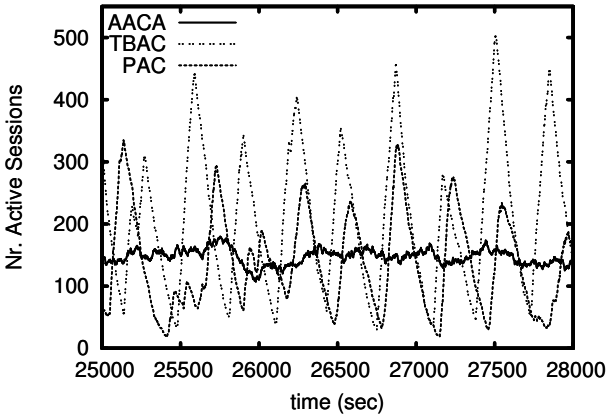

Fig. 9. Number of active sessions

VIII. Conclusions

In this paper we address the problem of admission control for web based systems. We introduce an original policy, named AACA, that is based on a self-learning measurement framework that permits the self-configuration of its parameter setting and a rapid adaptivity. Our policy does not require any prior knowledge of the incoming traffic, nor any assumption on the probability distribution of request inter-arrival and service time. Unlike other proposals in the area, our policy works under a wide range of operating conditions without the need of laborious manual parameter tuning. It is entirely implemented on edge points, be them access routers or application level dispatchers, and does not require any modification of client and server software.

We compared our policy to other previously proposed approaches. Extensive simulations show that it permits an excellent utilization of system resources while always respecting the limits on response time imposed by service level agreements. We also show that our policy improves service quality by reducing oscillations of response time and number of active sessions (common to other policies that work at session granularity).

The impact of network load on the self-learning activity at the basis of the AACA policy is under study and will be considered as an extension of this work.

References

[1] "Ibm: the vision of autonomic computing," *http://www.research.ibm.com/autonomic/manifesto*.

[2] "Hewlett packard: Adaptive enterprise design principles," *http://h71028.www7.hp.com/enterprise/cache/80425-0-0-0-121.html*.

[3] "Microsoft: The drive to self-managing dynamic systems," *http://www.microsoft.com/windowsserversystem/dsi/default.mspx*.

[4] D. Breitgand, E. Henis, and O. Shehory, "Automated and adaptive threshold setting: enabling technology for autonomy and self-management," *Proceedings of the International Conference on Autonomic Computing (ICAC)*, 2005.

[5] X. Liu, R. Zheng, J. Heo, Q. Wang, and L. Sha, "Timing performance control in web server systems utilizing server internal state information," *Proceedings of the IEEE Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services (ICAS/ICNS)*, 2005.

[6] Y. Li, K. Sun, J. Qiu, and Y. Chen, "Self-reconfiguration of service-based systems: a case study for service level agreements and resource optimization," *Proceedings of the IEEE International Conference on Web Services (ICWS)*, 2005.

[7] S-Pradeep, C. Ramachandran, and S. Srinivasa, "Towards autonomic web-sites based on learning automata," *Proceedings of the ACM World Wide Web Conference (WWW)*, 2005.

[8] J.-R. Gruser, L. Raschid, V. Zadorozhny, and T. Zhan, "Learning response time for websources using query feedback and application in query optimization," *The International Journal on Very Large Data Bases*, vol. 9, no. 1, March 2000.

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

[10] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, "An analytical model for multi-tier internet services and its applications," *IEEE Transactions on the Web*, to appear 2007.

[11] N. Bartolini, G. Bongiovanni, and S. Silvestri, "An autonomic admission control policy for distributed web systems," *University of Rome, La Sapienza, Research Report nr.: 05/2007*, 2007.

[12] L. Cherkasova and P. Phaal, "Session based admission control: a mechanism for peak load management of commercial web sites," *IEEE Transactions on Computers*, vol. 51, no. 6, 2002.

[13] S. Elnikety, E. Nahum, J. Tracey, and W. Zwaenepoel, "A method for transparent admission control and request scheduling in e-commerce web sites," *Proceedings of the ACM World Wide Web Conference(WWW)*, May 2004.

[14] N. Bartolini, G. Bongiovanni, and S. Silvestri, "Distributed server selection and admission control in replicated web systems," *The IEEE Proceedings of the 6th International Symposium on Parallel and Distributed Computing (ISPDC)*, 2007.

[15] Z. Xu and G. v. Bochmann, "A probabilistic approach for admission control to web servers," *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, July 2004.

[16] J. Aweya, M. Ouelette, D. Y. Montuno, B. Doray, and K. Felske, "An adaptive load balancing scheme for web servers," *International Journal of Network Management*, vol. 12, pp. 3–39, 2002.

[17] "Opnet technologies inc." *http://www.opnet.com*.

[18] "Specweb2005 design document," *http://www.spec.org/web2005/docs/designdocument.html*.

[19] "The transaction processing council (tpc). tpc-w," *http://www.tpc.org/tpcw*.

[20] D. Menasce, "Tpc-w: A benchmark for e-commerce," *IEEE Internet Computing*, May/June 2002.

[21] H. Weinreich, H. Obendorf, E. Herder, and M. Mayer, "Off the beaten tracks: Exploring three aspects of web navigation," *Proceedings of the ACM World Wide Web Conference (WWW)*, 2006.