An Adaptive Admission Control Policy for Geographically Distributed Web Systems

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

ABSTRACT

This paper deals with the problem of admission control for geographically distributed web servers in presence of several access routers.

The main contribution of this paper is the proposal of a scalable admission control scheme with the purpose to accept as many new sessions as possible within the constraints on response time imposed by service level agreements. The proposed policy autonomously configures and periodically adapts its component level parameters to the time-varying traffic situations.

Extensive simulations of our policy show that our algorithm always guarantees the adherence to SLAs under different traffic scenarios. The proposed method shows a stable behavior during overload by smoothing flash crowd effects. It also improves the successful session termination probability and the utilization of system resources when compared to other traditional admission control schemes.

Categories and Subject Descriptors

C.2.4 [**Computer Communication Networks**]: Distributed Systems; C.4 [**Performance of Systems**]: Performance Attributes, Measurement techniques

General Terms

Experimentation, Performance, Measurements

Keywords

Adaptive admission control, self-configuration, autonomic computing, quality of service

1. INTRODUCTION

The past decade was characterized by the growing phenomenon of web based applications and to the contextual diffusion of geographically distributed service architectures, such as grid computing facilities, peer-to-peer networks, content distribution architectures, shared resource pools, just to name a few.

This paper studies the problem of admission control for geographically replicated servers in presence of many access routers (ARs), for typical web services demanding for guaranteed quality. In such a scenario several ARs concurrently adopt Quality of Service (QoS) policies based on local information. This scenario is typical of active networks, based on application layer anycast, but the proposed methodology can be applied also to many of the architectures listed above. The ARs are not informed about the QoS policies adopted by other ARs and every decision they make will likely interfere with other AR decisions, possibly generating unpredictable and unwanted effects.

The main contribution of this paper is the proposal of a probabilistic admission control policy that self-configures its initial parameter setting and adapts it to time-varying workload and external traffic. The proposed policy does not require any prior knowledge of the incoming traffic and is not based on any assumption on request interarrival and service time distribution.

The application of this policy does not require any modification of client and server software, as it can be implemented in ARs, be them access routers or dispatchers. Throughout this paper we assume ARs operate at level 7 of the ISO/OSI stack and have some application level knowledge, such as the capability to associate incoming requests to ongoing service sessions. Although this assumption is at the basis of the performance evaluation presented in this paper, the same policy can be adopted for services that are not based on the concept of session, or to scenarios where the ARs do not have any session information, for example when request admission and dispatching procedures are operated at proxy level.

According to our proposal, each AR needs to perform a measurement activity to gather information regarding the behavior of other ARs and to keep into account the time-varying status of non-dedicated network links. Measures are taken with passive techniques, by simply observing the response time of intercepted requests.

Purpose of the proposed policy is to accept as many new sessions as possible within the performance limits on response time imposed by the service level agreements (SLA).

In order to evaluate the proposed technique we designed a synthetic traffic generator, whose sizing is compliant with the assumptions of industrial standard benchmarks. The think time and session interarrival time are the same of the

Infoscale 2007 Suzhou, China

Rice TPC-W workload generator [12, 22], while the session model is the same of SPECWEB2005 [21]. We tested our policy and two preexisting ones on a simulator based on the OPNET modeler software [20].

The experiments reveal that thanks to an adaptive probabilistic admission control our policy always meets the SLA requirements under different traffic scenarios, while the other policies either violate the agreements or under-utilize the system resources.

Unlike other policies, our proposal is independent of parameters such as the inter-decision period, that would require manual tuning. Our algorithm is based on an autonomous initial self-configuration that quickly converges to the component level parameter setting that enables the respect of the SLA constraints with the maximum rate of admitted sessions. The measurement based adaptation of the acceptance probability also permits a rapid reconfiguration of the policy to follow workload variations at runtime.

The paper is organized as follows. In Section 2 we introduce the state of the art of admission control in geographically distributed web systems. In Section 3 we give some details on the reference scenario of a geographically distributed web system with multiple access routers, with service level agreements specifying strict requirements on user perceived quality. In Section 4 we introduce an original admission control policy as well as other well known policies to be implemented on ARs. In Section 5 we describe the experimental setup, while in Section 6 we present a comparative performance study of the proposed policies in different traffic scenarios. Section 7 concludes the paper.

2. RELATED WORK

The problem of admission control for Web servers has been extensively studied in the recent literature but the majority of the works only consider locally distributed architectures with single access routers such as those described in [4]. Little effort has been spent on the problem of autonomous tuning of QoS policies for web systems. Most policies often rely on laborious parameter tuning, that are seldom known a priori.

Many works consider the admission control problem in single server architectures. Both [5] and [7] show how considering the admission of whole sessions rather than individual requests can improve service quality in some circumstances. Both these papers locate the admission control policy at the server level, for this reason the proposed methodology cannot scale to a geographically distributed systems, with replicated content.

The authors of [19] analyze the problem of parameter oscillation when performing on/off admission control to web servers. Differently from our proposal, this scheme only considers single web servers, and single access routers, which means that the suggested admission control policy may not scale to the scenario considered in this paper.

The authors of [17] introduce the problem of QoS policies in a distributed architecture with only one dispatching point. A prediction of the incoming workload is considered while making admission control decisions with the purpose of maximizing the provider's income. Requests are scheduled on the basis of an estimate of the service time.

The authors of [8] study a multi-tier cluster architecture and propose the use of a proxy server to transparently intercept database requests. Such a proxy maintains several measurement-based estimates on the total capacity of the database, the current database load and the workload generated by each query type. On the basis of these measures the proxy performs adaptive admission control and request redirection, achieving overload control and improving response time.

The authors of [9] propose and evaluate a delay prediction and control scheme based on feedback and feed-forward strategies. In [11] the authors present a control-theoretic approach to provide guaranteed absolute or relative delays between different service classes of a single web server. Unlike our work, the cited schemes [8,9,11,17] cannot be applied to a geographically distributed environment with many replicated servers and several access routers.

In [6] a single server architecture is considered, in presence of the so called bystander traffic, that is, non application specific traffic on the non-dedicated links between the clients and the server. An application level mechanism is introduced to mitigate flash crowds. Unlike our work, the authors of [6] do not take into account session information when performing admission control. They only address informational services, typically composed by static contents.

In [10] the problem of anycast flow admission control is discussed. The authors propose three algorithms for admission control and propose the adoption of a probabilistic redirection mechanism. This work does not analyze the problem of session based traffic and does not study the scalability problems imposed by the presence of several access routers and for this reason it is not directly comparable to ours.

The problem of designing adaptive component-level threshold is analyzed in [3] for a general context of autonomic computing. The mechanism proposed in the paper consists in monitoring the policy threshold values in use by keeping track of false positive and false negative alarms with respect to possible violations of service level agreements. Unlike this paper, our algorithm employs a very simple linear rule to calculate the new session admission probability as a function of the measured response time. Of this paper we criticize the usual threshold approach and underline how the most common threshold policies cause on/off behaviors that often result in unacceptable performance especially in large scale architectures. Our proposal is instead based on a probabilistic approach that smoothes oscillations and performs better overload control.

3. REFERENCE SCENARIO

The reference architecture of this paper consists of a set of geographically distributed servers, accessed through several ARs. This scenario models several typical replication strategies [16] for web applications such as ACDN [14], shared server pools [15] and other multi-broker architectures. SLAs for internet based applications usually specify constraints on user perceived performance levels. The most common specification is in terms of an upper bound on the $X\%-$ile of request response time (with X usually set to 90 or 95) and a lower bound on the number of concurrent client sessions (if the new session request rate is sufficiently high) [13]. Although our work may be applied to different formulations of SLA, including the most typical one cited above, we argue that a more detailed SLA should be in place when clusters based on multiple heterogeneous tiers are considered. A common cluster architecture for web services, requiring

the use of a database, is based on the differentiation of requests in three categories involving different tiers: *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. Request processing at different tiers happens with significantly different service time distribution, and the average processing times of the three request types differ of one to three orders of magnitude. Web services are normally based on the concept of *session*. A session is a sequence of temporally and logically related requests issued by the same client. Service sessions can be modelled as a sequence of service phases alternated to think phases. The sequence of phases traversed by the session during its lifetime strictly depends on the particular application, while all service sessions normally start with the same request (typically an http request to the index page of the site hosting the application). The typical session life-cycle can be modeled with a state diagram representing all possible phase transitions involving different tiers such the one utilized in the specification of the benchmark SPECWEB2005 [21]. The probability that a session reaches a state after many phase transitions is strictly dependent on the workload being served. In fact as the system utilization increases, there is a higher chance that longer sessions are interrupted before reaching their natural end. Therefore the probability that a session reaches high order states (i.e. states that can only be reached after several interactions of the end user with the system) decreases with increasing system utilization. For this reason, under varying traffic load, with or without admission control, the average duration time of sessions also varies, leading to a variable request mix being served by different tiers. The percentage of requests involving the bottleneck tier, in this case the database tier, decreases with increasing session rate. Since the response time of these requests is significantly higher than the response time of the other types of requests, the monotonicity of the response time as a function of the incoming session rate cannot be stated a priori, because long processing requests become less frequent due to session dropping in case of high load. If this is the case, a single SLA threshold on the response time of the overall request mix cannot be used. In fact, the same value of the statistic parameter representing the response time can be related to two very different situations, i.e. normal load with acceptable performance and very high load where the majority of sessions get dropped at the bottleneck tier, thus traversing the bottleneck no more than once in the lifetime of each session and causing the session abrupt interruption. For this reason we consider an SLA agreement on the 95%−ile of the database response time. This assumption is without loss of generality since it causes a more conservative evaluation of the system performance, as the 95%−ile of request response time calculated on the whole set of requests is always lower than considering database requests only.

4. ADMISSION CONTROL POLICIES

ARs operate admission control and server selection on the basis of locally available information on the quality of service that can be perceived by clients located in proximity.

In this section we describe our policy in deeper details. We give it the name of Adaptive Probabilistic Admission Control (APAC). We also introduce other preexisting policies as benchmarks for comparisons. In particular, we introduce the Threshold Based Admission Control policy (TBAC, an adaptation of the policy proposed in [7] to the scenario described in section 4) and the Probabilistic Admission Control (PAC, an adaptation of the policy proposed in [19]). We assume the ARs intercept requests and use a probabilistic redirection scheme to select the best suited server from the shared pool. Any time a new requests arrive to an AR, a server is selected with a higher probability among the servers that showed lower response times as described in [2]. In order to have fair comparisons, the same redirection scheme is used for all the admission control policies.

4.1 Adaptive Probabilistic Admission Control (APAC) policy

The aim of this policy is to accept as many new incoming sessions as possible within the constraints imposed by the SLA. As detailed in section 3, we restrict the evaluation of SLA fulfillment only at the bottleneck tier, that is the database. During the passive measurement phase, the access routers periodically evaluate the 95%-ile of database response time, which from now on will be indicated as $RTT_{\mathtt{DB}}$. This represents the 95%-ile of the user perceived latency of database requests with sufficient approximation if we assume that the user to AR latency is negligible if compared to the latency observed in the links between AR and servers. The algorithm is iterated every $t_{\mathtt{obs}}^{\mathtt{APAC}}$ seconds on each AR. ARs are not synchronized with each other.

During the n-th time slot the i-th AR accepts new sessions with probability $p^i(n)$, (with $p^i(1) = 1$). To estimate the value of $p^i(n+1)$ the policy assumes that the new session arrival rate does not change between two consecutive intervals. Possible errors of this estimate are corrected in subsequent iterations. At the beginning of the n-th time slot the i-th AR acts as follows:

1. Evaluates the 95%-ile response time of the database requests received during the last interval : $RTT^i_{\mathtt{DB}}(n-1)$
2. If $\{(RTT^i_{\mathtt{DB}}(n-1) > RTT_{\mathtt{SLA}}) \wedge (RTT^i_{\mathtt{DB}}(n-1) < RTT_{\mathtt{SLA}}(1-\alpha))\}$ the session admission probability must be updated (go to step 3), otherwise $p^i(n) = p^i(n-1)$ and go to the next iteration (go to step 1). The factor α introduces an hysteresis cycle to avoid too frequent policy reconfigurations and possible oscillations. We use $\alpha = 10\%$. The value $RTT_{\mathtt{SLA}}$ is the upper bound on the 95%−ile of response time, as specified in the SLA.
3. Policy update:
   - Let $\gamma^i(n-1) = \frac{RTT_{\mathtt{SLA}}}{RTT^i_{\mathtt{DB}}(n-1)}$ be the ratio between the upper limit on the 95%−ile of response time specified in the SLA and 95%−ile of database request response time observed during the previous time interval.
   - The session admission probability is updated as follow:

$$p^i(n) \triangleq \begin{cases} 1 & \text{if } n = 1 \\ \min\{p^i(n-1)\ \gamma^i(n-1), 1\} & \text{otherwise} \end{cases}$$

This method assumes that the relationship between the new session admission probability and the user perceived round trip time (RTT) can be linearly approximated. Since

the actual relationship is obviously more complex, this method results in a coarse approximation during the first steps of the algorithm, that will be improved by subsequent iterations.

Because of the slow impact of session based decisions on the effective user perceived RTT, the decision period $t_{\mathtt{obs}}^{\mathtt{APAC}}$, must be large enough to guarantee proper consideration of the effects of the last decisions in terms of database request response time. As we will show in section 6 our policy has a stable behavior and meets the SLA in all the considered traffic situations provided $t_{\mathtt{obs}}$ is sufficiently large.

4.2 Threshold Based Admission Control (TBAC)

This policy uses a threshold value $T_{\mathtt{AC}}^{\mathtt{TBAC}}$ to make decisions whether new sessions should be accepted for service. It consists in the evaluation of the current 95%−ile of the database tier response time and acceptance of new sessions only if this value does not exceed the threshold $T_{\mathtt{AC}}^{\mathtt{TBAC}}$. Because of the on/off nature of this policy, it has an inherent oscillatory behavior, that can be smoothed by choosing short inter-decision periods $t_{\mathtt{obs}}^{\mathtt{TBAC}}$.

Since the number of response time samples gathered in short inter-decision periods may not be sufficient to give an accurate estimate of the 95%−ile, we assume the ARs gather samples for longer sliding time windows of width $t_{\mathtt{W}}^{\mathtt{TBAC}}$. Therefore, every $t_{\mathtt{obs}}^{\mathtt{TBAC}}$ seconds each AR calculates the value of $RTT_{\mathtt{DB}}$, that is the 95%−ile of database response time, given a set of samples gathered during the previous $t_{\mathtt{W}}^{\mathtt{TBAC}}$ seconds. If $RTT_{\mathtt{DB}}$ exceeds the threshold $T_{\mathtt{AC}}^{\mathtt{TBAC}}$ all new session requests are rejected during the subsequent interval, otherwise new session requests are served as well as requests belonging to already ongoing sessions.

As we show in section 6, simulation results reveal that this policy contributes to lowering the 95%-ile of the request round trip time. We also show that this policy cannot guarantee the fulfillment of SLA agreements even if $T_{\mathtt{AC}}^{\mathtt{TBAC}}$ is set to $RTT_{\mathtt{SLA}}$. The behavior of this policy is in fact strictly dependent on a proper parameter tuning, specifically in terms of admission threshold $T_{\mathtt{AC}}^{\mathtt{TBAC}}$ and inter-decision period $t_{\mathtt{obs}}^{\mathtt{TBAC}}$, which directly affect the rate of requests actually reaching the servers. Proper parameter tuning of this policy cannot be done manually and offline for an architecture based on multiple ARs, since it is dependent on the amount of workload being sent to the application servers as we will show in section 6.

4.3 Probabilistic Admission Control (PAC)

The probabilistic control is at the basis of control theory when oscillations should be avoided. The policy we introduce in this section was proposed for internet services in [19], while a similar version was introduced in [1]. According to this policy, a new session is admitted with a certain probability, which value depends on the measured server response time. The system accepts all new session requests as long as the server load is very low (i.e. the measured response time is under the threshold $T_{\mathtt{low}}^{\mathtt{PAC}}$) and it gradually reduces the acceptance probability as load grows, zeroing it when response time exceeds a threshold $T_{\mathtt{high}}^{\mathtt{PAC}}$. In the general case the acceptance probability is a piece-wise linear function of the server response time measured in the previous time interval $RTT_{\mathtt{DB}}(n)$ (based on a time window of length $t_{\mathtt{w}}^{\mathtt{PAC}}$) and has the following form:

$$p(n) \triangleq \begin{cases} 1 & \text{if } RTT_{\mathtt{DB}}(n-1) \leq T_{\mathtt{low}}^{\mathtt{PAC}} \text{ or } n = 0 \\ \frac{RTT_{\mathtt{DB}}(n-1) - T_{\mathtt{high}}^{\mathtt{PAC}}}{T_{\mathtt{high}}^{\mathtt{PAC}} - T_{\mathtt{low}}^{\mathtt{PAC}}} & \text{if } T_{\mathtt{low}}^{\mathtt{PAC}} < RTT_{\mathtt{DB}}(n-1) \leq T_{\mathtt{high}}^{\mathtt{PAC}} \\ 0 & \text{if } RTT_{\mathtt{DB}}(n-1) > T_{\mathtt{high}}^{\mathtt{PAC}} \end{cases} \quad (1)$$

As the performance comparisons of section 6 will show, although this policy reduces the problem of oscillations of round trip time and number of ongoing sessions, it cannot be used in presence of service level agreements requiring guaranteed quality. In fact, the two threshold values $T_{\mathtt{high}}^{\mathtt{PAC}}$ and $T_{\mathtt{low}}^{\mathtt{PAC}}$ that characterize this policy are arbitrarily set offline independently of the observed incoming session rate and of the inter-observation period $t_{\mathtt{w}}^{\mathtt{PAC}}$(the time between two subsequent policy decisions). This way the adherence of this policy to the SLA limits cannot be guaranteed, since it is highly dependent on the parameters cited above, as will be clear from the experiments detailed in section 6.

5. SIMULATION ENVIRONMENT

We developed a simulator on the basis of the OPNET modeler software [20] to comparatively study the performance of the QoS policies introduced in section 4. The main entities of the simulation environment are clients, ARs and application servers. We assume that clients are partitioned into geographic areas, and for each area there is a single AR. In this way all clients of the same area send their requests to the same AR, and latency on the links between clients and the AR can be ignored. In the experimental setup we do not introduce bystander traffic because we want to focus on the effects of the policies in presence of several ARs adopting concurrent admission control decisions.

5.1 Client model and traffic generation

According to our model, each client issues a user session, in the shape of a series of correlated requests.

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

In order to have a realistic traffic generator, we use the phase model of an industrial standard benchmark: SPECWEB2005 [21]. It takes into account the most relevant types of dynamic content (php and jsp included) and models an e-commerce web site selling personal computers, with typical requests such as browse, login, search, customization of selection, add to cart, buy and logout functionalities. We refer to [21] for a detailed description of the state model and of the functionalities of each phase.

Client sessions start from the home page and follow the state model according to its transition probabilities, possibly ending in any state if the user voluntary abandons the site before completing the whole session life-cycle, or reaching the last state of the model where an item can be purchased or not. Upon reception of a response, the next request is sent after a certain amount of time, called *User Think Time (UTT)*, that represents the time the user spends analyzing the received web page. Our model of UTT is based on TPC-W [22] and on other works in the area of web traffic analysis

that justify the assumption of heavy tailed distributions of the user think time [18]. As in the TPC-W model, we assume a logarithmic distribution of think times and a lower bound of 1 sec. Therefore $UTT = max\{-log(r)\mu, 1\}$, where r is uniformly distributed in the interval [0,1] and $\mu = 20sec$.

To model a realistic user behavior, we also introduce a timeout on the maximum time the user can wait for a response before abandoning the site. The interaction of a client with the web application may end for three possible reasons: *a*) session block: the AR 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.

5.2 Access router model

ARs are the key entities of our model since all QoS policies are enforced by them. They adopt a two-way dispatching policy, so ARs receive client requests and dispatch them to the servers. They also receive server responses and forward them to the clients. ARs enable the adoption of QoS policies for admission control and request redirection. In section 6 we analyze the performance of the AR policies.

5.3 Server model

Servers communicate with the clients via the ARs. Each server creates a new thread upon reception of a client request and adds it to the waiting queue. Servers use a time sharing, round robin scheduling. The thread processing time depends on the type of request being served. We assume each phase of the session state model can be mapped onto a specific tier of the typical three tier organization of internet applications. Thus we consider three categories of requests depending on which tiers are involved in the request processing: *pure http*, *servlet* and *database requests*. We gave an approximate estimate of the average processing times of the different categories on the basis of the experiments detailed in [8]. In our simulations we assume each session phase requires an exponentially distributed execution time. Execution times are set as follows (waiting time due to preemptive processor scheduling are not considered in these measures): average execution time for pure http requests is 0.001sec, while for servlet request is 0.01sec and for database requests is 1sec.

6. SIMULATION RESULTS

This section is dedicated to a comparative analysis of the performance of the APAC, TBAC and PAC policies.

To this extent we introduce three sets of simulations. In the first set we analyze how the performance is affected by the session arrival rate, while in the second set we study the behavior of the cited policies under different SLAs. A third set of experiments studies the dependence of these policies on manually tuned parameters.

In the first two sets of experiments we use the following simulation parameter setting: 5 access routers, 15 application servers, client timeout 8 sec. The only manually tuned parameter for the APAC policy is the length of the inter-decision period $t_{\mathtt{obs}}^{\mathtt{APAC}}$ that has been set to 250sec. The following parameters define the behavior of the TBAC policy: inter-decision period $t_{\mathtt{obs}}^{\mathtt{TBAC}} = 10$sec, time window $t_{\mathtt{TW}}^{\mathtt{TBAC}} = 60$sec and threshold $T_{\mathtt{AC}}^{\mathtt{TBAC}} = RTT_{\mathtt{SLA}}$ on the 95%−ile of the response time. The parameters of the PAC policy are defined as follows: inter-decision period $t_{\mathtt{obs}}^{\mathtt{PAC}} = 10sec$, time window $t_{\mathtt{TW}}^{\mathtt{PAC}} = 20$sec and thresholds on the 95%−ile of the response time $T_{\mathtt{low}}^{\mathtt{PAC}} = 3$sec and $T_{\mathtt{high}}^{\mathtt{PAC}} = RTT_{\mathtt{SLA}}$. Notice that the choice of the same value of $t_{\mathtt{obs}}$ for all the policies would lead to unfair comparisons since long inter-observation periods constitute an advantage for the APAC policy while TBAC and PAC are privileged by short periods. Notice also that manual tuning of the TBAC and PAC policy is a complex task as the proper parameter setting is dependent on the particular traffic situations as will be shown in figure 1. In fact, whatever threshold values are chosen, they can only work for a specific session arrival rate. In order to have fair comparisons we let TBAC and PAC have the same upper threshold on RTT: $T_{\mathtt{AC}}^{\mathtt{TBAC}} = T_{\mathtt{high}}^{\mathtt{PAC}} = RTT_{\mathtt{SLA}}$.

Purpose of the first result set is to show how the behavior of our APAC policy, differently from TBAC and PAC, is not dependent on the system load. It adapts itself to different workload situations, and admits as many new sessions as possible within the SLA limits.

In the following experiments we set the SLA threshold to 5sec.

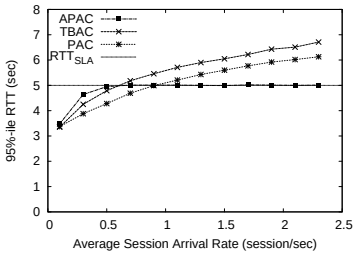

Figure 1: 95%-ile of database response time

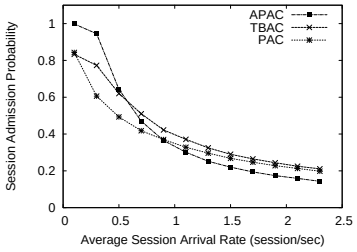

Figure 2: New session admission probability

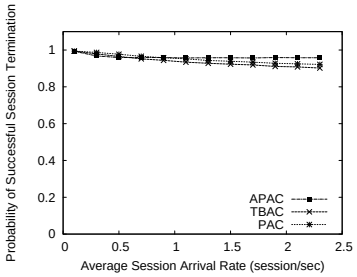

Figure 3: Successful termination probability of admitted sessions

Figure 1 shows the 95%−ile of database response time with varying session arrival rate. This figure points out how the APAC policy meets the SLA requirements independently of the AR session arrival rate. The TBAC policy shows an

increasing response time with growing session arrival rate, causing either a system under-utilization or the violation of the agreements on quality. Similarly, the PAC policy shows an even more evident system under-utilization for low arrival rates while it also exceeds the SLA limits for more intense workloads. This figure reveals 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 $T_{\mathtt{AC}}^{\mathtt{TBAC}} = 5$sec when $\lambda = 0.6$ sessions/sec or $T_{\mathtt{high}}^{\mathtt{PAC}} = 5$sec when $\lambda = 0.9$ sessions/sec) may cause SLA violations or system under-utilization in other cases.

As shown in figure 1, the TBAC policy has a higher value of the 95%-ile of request RTT, with respect to the PAC policy, in any load scenario. This is due to the particular threshold setting of the two policies.

The 95%-ile of all the policies is under the SLA threshold, of 5 seconds, for low values of arrival rate ($\lambda \leq 0.3$ sessions/sec), but while the APAC policy accepts almost all incoming new sessions, the TBAC policy shows a lower admission probability and a consequent under-utilization of the system resources, as shown in figure 2. The PAC policy blocks even more sessions than TBAC due to the particular parameter settings. Notice that, as previously pointed out, another tuning of the threshold parameters could have been beneficial for the TBAC and PAC in low traffic scenarios, but would have had negative effects on their behavior during medium and high traffic situations.

These results highlight an important problem of the TBAC and PAC policies: the dependency of the threshold tuning on the system load. On one hand if the system load is so low that the response time only occasionally reaches the imposed threshold, these policies cause under-utilization of the system, blocking more sessions then strictly necessary. On the other hand, if the system load is high these policies cannot guarantee the respect of the SLA as they let the system be overloaded when the decision to accept new sessions is made.

This behavior of the TBAC policy is due to its on/off nature. Even in a low workload scenario, momentary peaks of requests can cause the AR to measure high values of the 95%-ile of the response time, possibly detecting a SLA violation for a short time interval. The TBAC policy reacts to this situation refusing all new sessions for one or more intervals, likely causing a system under-utilization. In high load scenarios, the TBAC policy shows performance problems as well. When the measures reveal that new sessions can be admitted under the SLA constraints, the TBAC policy admits all incoming new sessions for the subsequent inter-observation period, possibly causing an uncontrolled growth of the system load and oscillations of RTT.

The PAC policy has similar problems due to the way it calculates the new session admission probability for the next interval. Equation 1 assumes a fixed linear dependence of the measured RTT on the session admission probability. Given a measured RTT, the PAC policy selects a single value for the new session admission probability, independently of the incoming session rate. Such value may not be suitable to different workload scenarios. Furthermore, this policy shows a high dependence on the tuning of the inter-observation period. This parameter is of primary importance because session based admission decisions do not have an immediate effect on RTT. A wrong tuning of this parameter leads to an oscillatory policy behavior although with minor intensity than with the TBAC policy.

Summarizing, the on/off behavior of the TBAC policy causes system under-utilization in low load situations and performance oscillations during periods of heavy load. The PAC policy reduces the oscillations but has the same drawbacks. They only behave properly under stable scenarios when the thresholds are correctly tuned.

Figure 3 shows the probability of admitted session successful termination with increasing rate of incoming new session requests. While the TBAC and PAC policy show a decreasing successful termination probability, due to the higher rate of incoming new sessions overloading the system, the APAC policy shows a more stable behavior. Our policy has the advantage to guarantee a stable, non oscillatory system behavior in every load scenario, even under variable traffic conditions.

The aim of the second set of simulations is to test the policy behavior with different SLAs. We consider three different workload situations: low ($\lambda = 0.4$ sessions/sec), medium ($\lambda = 0.7$ sessions/sec) and high ($\lambda = 1.9$ sessions/sec). We tested the APAC, TBAC and PAC policies under five different values of the SLA threshold $RTT_{\mathtt{SLA}}$, varying from 4 to 8 seconds. We set $T_{\mathtt{AC}}^{\mathtt{TBAC}} = T_{\mathtt{high}}^{\mathtt{PAC}} = RTT_{\mathtt{SLA}}$.

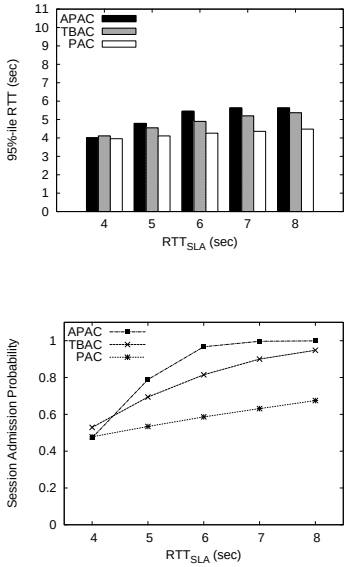

Figure 4: 95%-ile of database request RTT and Session Admission Probability - low load scenario

Figure 4 shows the 95%-ile of database response time in the low load scenario. This figure points out the main drawbacks of the TBAC and PAC policies discussed earlier. In the case with low threshold (4 seconds) the TBAC does not prevent SLA violations, while for higher values it causes system under-utilization. This is confirmed by 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 shows a more conservative behavior, admitting less sessions than the TBAC policy. It results in a more evident under-utilization of the system resources except for the lowest value of the SLA threshold we considered (4 seconds), where it meets the SLA thanks to the specific combination of parameters. Our policy, instead, always respects the SLA, guaranteeing a high

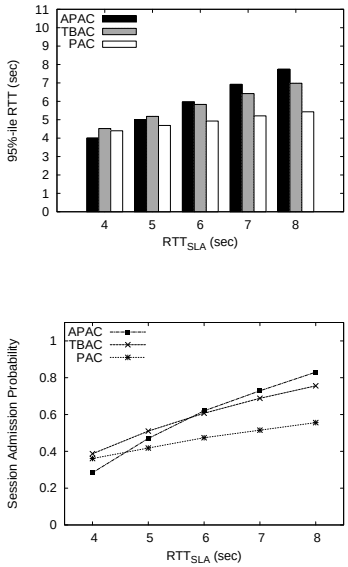

Figure 5: 95%-ile of database request RTT and Session Admission Probability - medium load scenario

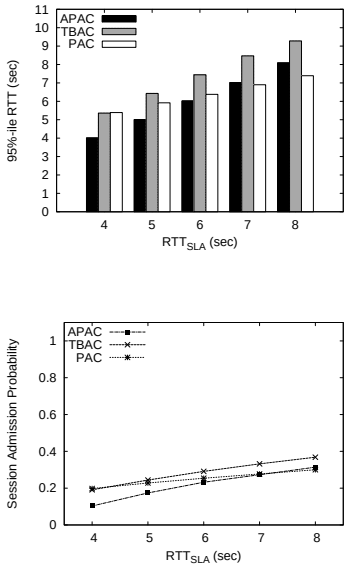

Figure 6: 95%-ile of database request RTT and Session Admission Probability - high load scenario

probability of admitting new sessions without under utilizing the system in low load scenarios.

Similar results are obtained in the medium load scenario showed in figure 5. APAC always respects the SLAs while the TBAC and PAC policies either violate the SLA when low thresholds are used or under utilize the system when higher thresholds are considered.

This figure points out the main problem of the PAC policy: it does not consider the actual session arrival rate to estimate the admission probability. While in the previous experiment (with low load) it could meet the SLA with a threshold of 4 seconds, a small increment of the session arrival rate results in a SLA violation, as reported in figure 5.

Figure 6 shows the results obtained for the high load scenario. In this case the TBAC policy does not respect the SLA in any of the studied cases. As already stated, the on/off behavior causes the admission of too many sessions after an admitting decision, worsening the user perceived RTT. Similarly, the PAC policy often violates the SLA and still under-utilizes the system for the highest threshold value we considered (8 seconds). Thanks to the probabilistic admission, the PAC policy reduces the entity of SLA violations with respect to the TBAC, but does not guarantee the fulfillment of the agreement except in those rare situations in which parameters are correctly configured for the actual traffic volume. In this scenario our policy guarantees the fulfillment of the SLAs in all the studied cases.

In addition to the good behavior in terms of performance, our policy has the advantage of self-tuning the admission control probability without the need of prior knowledge about traffic workload intensity and user behavior. Our policy starts accepting all new incoming sessions and rapidly adapts its acceptance probability to the time-varying workload situations. A complete analysis of the time to converge to the final parameter settings is in progress. The only policy parameter that requires manual tuning for APAC is the length of the inter-observation period.

The last simulation results of this section show that the behavior of the APAC policy is only marginally influenced by the setting of this parameter. The only constraint on the length of the inter-observation period comes from the necessity that the last computed probability value be given enough time to manifest itself (in terms of impact on the RTT) before the calculation of the next probability value.

We now focus on the impact of the length of the inter-observation period on the performance of the APAC, TBAC and PAC policy. In this experiment we use the following simulation parameter settings: 5 access routers, 15 application servers, client timeout of 8 sec, $RTT_{\mathtt{SLA}} = 5$sec, $\lambda = 0.7$ sessions/sec. In these experiments we use the same value $t_{\mathtt{obs}}$ of the inter-observation period for all the policies, therefore $t_{\mathtt{obs}} = t_{\mathtt{obs}}^{\mathtt{APAC}} = t_{\mathtt{obs}}^{\mathtt{TBAC}} = t_{\mathtt{obs}}^{\mathtt{PAC}}$

Figure 7 shows the 95%-ile of database request RTT under varying values of $t_{\mathtt{obs}}$. This figure reveals that the APAC policy needs relatively long periods to be able to meet the SLA, but is not adversely influenced by even longer periods.

On the contrary the use of long values for $t_{\mathtt{obs}}^{\mathtt{TBAC}}$ and $t_{\mathtt{obs}}^{\mathtt{PAC}}$ worsen the performance of the TBAC and PAC policy because they induce more intense oscillations in case of high load and cause scarce utilization in low load situations.

Figure 7 shows how the PAC policy behaves similarly to the TBAC policy with growing inter-observation period. With long values of $t_{\mathtt{obs}}$, in fact, the PAC policy oscillates between periods with high admission probability, where the SLA is likely to be violated, followed by periods with low admission probability, where the system is under-utilized.

Once again both the TBAC and PAC policy show a high dependence on their parameter tuning. On the contrary, the APAC policy has no need of precise and laborious parameter tuning. The APAC parameter $t_{\mathtt{obs}}^{\mathtt{APAC}}$ can be set so that ARs do not have to check the system status too often, avoiding the risk to become a system bottleneck during overload.

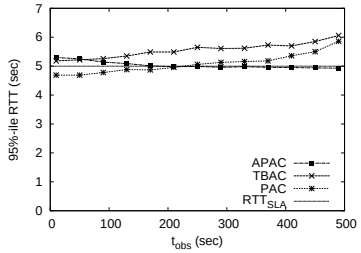

Figure 7: 95%-ile of database request RTT

7. CONCLUSIONS

This paper addresses the problem of distributed admission control in a geographically replicated server environment with several access routers. The reference architecture models several service paradigms such as active content delivery networks, edge computing, anycast based services and other multi-broker services as well. In such an environment every policy decision made by each AR will likely interfere with other AR decisions. According to our proposal, each AR can keep into account the behavior of other ARs by means of passive measures of the user perceived performance. We propose a novel admission control scheme that autonomously configures and periodically adapts its component level parameters to the time-varying traffic situations. As the simulation results show, the proposed policy outperforms other previously proposed approaches, in terms of both acceptance probability of new session requests and fulfillment of service level agreements on response times. Our policy also shows a stable behavior during overloads and better resource utilization without prior knowledge of traffic characteristics, without the need of manual parameter tuning. The proposed policy, in fact, self-configures the initial parameter settings of its component level thresholds and shows good adaptivity to time varying traffic situations.

8. REFERENCES

[1] 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*, 12:3–39, 2002.

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

[3] 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.

[4] V. Cardellini, E. Casalicchio, and M. Colajanni. The state of the art in locally distributed web server systems. *ACM Computing Surveys*, 34(2):263–311, 2002.

[5] J. Carlstrom and R. Rom. Application aware admission control and scheduling in web servers. *Proc. of the IEEE Conf. on Computer Communications (INFOCOM)*, 2002.

[6] X. Chen and J. Heidemann. Flash crowd mitigation via adaptive admission control based on application-level observation. *ACM Trans. on Internet Technology*, 5(3):532–569, 2005.

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

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

[9] D. Henriksson, Y. Lu, and T. Abdelzaher. Improved prediction for web server delay control. *Proc. of the IEEE Euromicro Conf. on Real-time systems (ECRTS)*, 2004.

[10] W. Jia, D. Xuan, W. Tu, L. Lin, and W. Zhao. Distributed admission control for anycast flows. *IEEE Trans. on Parallel and Distributed Systems*, 15(8), August 2004.

[11] C. Lu, Y. Lu, T. F. Abdelzaher, J. A. Stankovic, and S. H. Son. Feedback control architecture and design methodology for service delay guarantees in web servers. *IEEE Trans. on Parallel and Distributed Systems*, 17(9):1014–1027, 2006.

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

[13] J. Philippe, N. D. Palma, S. Bouchenak, F. Boyer, and D. Hagimont. A black-box approach for web application sla. *Proc. of ACM Symposium on Applied Computing (SAC)*, 2006.

[14] M. Rabinovich, Z. Xiao, and A. Aggarwal. Computing on the edge: A platform for replicating internet applications. *Proc. of the Int. Workshop on Web Content Caching and Distribution (IWCW)*, 2004.

[15] J. Rolia, L. Cherkasova, M. Arlitt, and V. Machiraju. Supporting application quality of service in shared resource pools. *Communications of the ACM*, 49(3), March 2006.

[16] S. Sivasubramanian, G. Pierre, M. van Steen, and G. Alonso. Analysis of caching and replication strategies for web applications. *Internet computing*, to appear 2007.

[17] A. Verma and S. Ghosal. On admission control for profit maximization of networked service providers. *Proceedings of ACM World Wide Web (WWW)*, 2003.

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

[19] Z. Xu and G. v. Bochmann. A probabilistic approach for admission control to web servers. *Proc. of the Int. Symp. on Performance Evaluation of Computer and Telecommuinication Systems (SPECTS)*, July 2004.

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

[21] Specweb2005. *http://www.spec.org/.*

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