Jean Arnaud, Sara Bouchenak

Performance, Availability and Cost of Self-Adaptive Internet Services

Chapter of *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*

IGI Global, 2011

Performance, Availability and Cost of Self-Adaptive Internet Services

Jean Arnaud
INRIA, Grenoble, France
Jean.Arnaud@inria.fr

Sara Bouchenak
University of Grenoble – INRIA, Grenoble, France
Sara.Bouchenak@inria.fr

ABSTRACT
Although distributed services provide a means for supporting scalable Internet applications, their ad-hoc provisioning and configuration pose a difficult tradeoff between service performance and availability. This is made harder as Internet service workloads tend to be heterogeneous, and vary over time in amount of concurrent clients and in mixture of client interactions. This chapter presents an approach for building self-adaptive Internet services through utility-aware capacity planning and provisioning. First, an analytic model is presented to predict Internet service performance, availability and cost. Second, a utility function is defined and a utility-aware capacity planning method is proposed to calculate the optimal service configuration which guarantees SLA performance *and* availability objectives while minimizing functioning costs. Third, an adaptive control method is proposed to automatically apply the optimal configuration to the Internet service. Finally, the proposed model, capacity planning and control methods are implemented and applied to an online bookstore. The experiments show that the service successfully self-adapts to both workload mix *and* workload amount variations, and present significant benefits in terms of performance and availability, with a saving of resources underlying the Internet service.

INTRODUCTION
A challenging issue in management of distributed Internet services tems from the conflicting goals of, on the one hand, high performance and availability, and on the other hand, low cost and resource consumption. In the limit, high performance and availability can be achieved by assigning all available machines in a data center to an Internet service. Symmetrically, it is possible to build a very-low cost Internet service by allocating very few machines, which induces bad performance and data center downtime. Between these two extremes, there exists a configuration such that distributed Internet services achieve a desirable level of service performance and availability while cost is minimized. This chapter precisely addresses the problem of determining this optimal configuration, and automatically applying it to build a self-adaptive Internet service.

The chapter describes a capacity planning method for distributed Internet services that takes into account performance and availability constraints of services. We believe that both criteria must be taken into account collectively. Otherwise, if capacity planning is solely performance-oriented, for instance, this may lead to situations where 99% of service clients are rejected and only 1% of clients are serviced with a guaranteed performance. To our knowledge, this is the first proposal for capacity planning and control of distributed Internet services that combines performance and availability objectives. To do so:

- A utility function is defined to quantify the performance, availability and cost of distributed Internet services.
- A utility-aware capacity planning method is proposed; given SLA performance *and* availability constraints, it calculates a configuration of the Internet service that guarantees SLA constraints while minimizing the cost of the service (i.e. the number of host machines).
- The capacity planning method is based on a queuing theory model of distributed Internet services. The model accurately predicts service performance, availability and cost.
- An adaptive control of online Internet services is proposed to automatically detect both workload mix *and* workload amount variation, and to reconfigure the service with its optimal configuration.

Finally, the proposed utility-aware methods for modeling, capacity planning and control were implemented to build self-adaptive distributed Internet services. The chapter presents experiments conducted on an industry standard Internet service, the TPC-W online bookstore. The results of the experiments show that the Internet service successfully self-adapts to workload variations, and present significant benefits in terms of service performance and availability, with a saving of resources of up to 67% on the underlying Internet service.

The remainder of the chapter first presents the background on Internet services. Then, it defines the motivations and objectives of this work. It then presents the utility function of Internet services, the proposed analytic model, the proposed capacity planning method, and the adaptive control of Internet services. An evaluation is then presented; and a related work is discussed. Finally, conclusions of this work are drawn.

BACKGROUND

Underlying System

Internet services usually follow the classical client-server architecture where servers provide clients with some online service (e.g. online bookstore, e-banking service, etc.). A client remotely connects to the server, sends it a request, the server processes the request and builds a response that is returned to the client before the connection is closed. We consider synchronous communication systems, that is, when the client sends its request to the server, it blocks until it receives a response. Furthermore, for scalability purposes Internet services are built as multi-tier systems. A multi-tier system consists of a series of M server tiers T_1, T_2,...,T_M. Client requests flow from the front-end tier T_1 to the middle-tier and so on until reaching the back-end tier T_M.

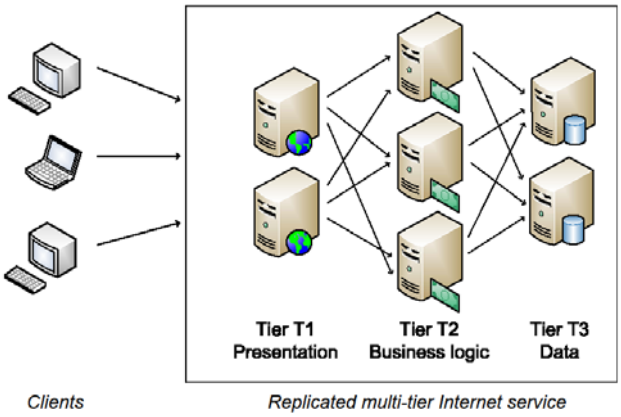

Figure 1: Replicated multi-tier services

Each tier is tasked with a specific role. For instance, the front-end web tier is responsible of serving web documents, and the back-end database tier is responsible of storing non-ephemeral data. Moreover, to face high loads and provide higher service scalability, a commonly used approach is the replication of servers in a set of machines. Here, a tier consists of a set of replicated services, and client requests are dynamically balanced between replicated services. Figure 1 presents an example of a replicated multi-tier Internet service with two replicas on the front-end presentation tier T_1, three replicas on the business tier T_2, and two replicas on the back-end database tier T_3.

Service Performance, Availability and Cost

SLA (Service Level Agreement) is a contract negotiated between clients and their service provider. It specifies service level objectives (SLOs) that the application must guarantee in the form of constraints on quality-of-service metrics, such as performance and availability. Client request latency and client request abandon rate are key metrics of interest for respectively quantifying the performance and availability of Internet services.

The *latency* of a client request is the necessary time for an Internet service to process that request. The average client request latency (or latency, for short) of an Internet service is denoted as ℓ. A low latency is a desirable behavior which reflects a reactive service.

The *abandon rate* of client requests is the ratio of requests that are rejected by an Internet service compared to the total number of requests issued by clients to that service. It is denoted as α. A low client request abandon rate (or abandon rate, for short) is a desirable behavior which reflects the level of availability of an Internet service.

Besides performance and availability, the *cost* of an Internet service refers to the economical and energetic costs of the service. Here, the cost ω is defined as the total number of servers that host an Internet service.

Service Configuration

The configuration κ of an Internet service is characterized in the following by a triplet $\kappa(M, AC, LC)$, where M is the fixed number of tiers of the multi-tier service, AC and LC are respectively the architectural configuration and local configuration of the Internet service that are detailed in the following.

The *architectural configuration* describes the distributed setting of a multi-tier Internet service in terms of the number of replicas at each tier. It is conceptualized as an array $AC < AC_1, AC_2, ..., AC_M >$, where AC_i is the number of replica servers at tier T_i of the multi-tier service.

The *local configuration* describes the local setting applied to servers of the multi-tier service. It is conceptualized as an array $LC < LC_1, LC_2, ..., LC_M >$. Here, LC_i represents servers MPL (Multi-Programming Level) at tier T_i of the multi-tier service. The MPL is a configuration parameter of a server that fixes a limit for the maximum number of clients allowed to concurrently access the server (Ferguson, 1998). Above this limit, incoming client requests are rejected. Thus, a client request arriving at a server either terminates successfully with a response to the client, or is rejected because of the server's MPL limit.

Service Workload

Service workload is characterized, on the one hand, by *workload amount*, and on the other hand, by *workload mix*. Workload amount is the number of clients that try to concurrently access a server; it is denoted as N. Workload mix, denoted as X, is the nature of requests made by clients and the way they

interleave, e.g. read-only requests mix vs. read-write requests mix. There is no well established way to characterize the workload mix X of an Internet service. In the following, a workload mix is characterized with the n-uplet X(Z, V, S, D) where:

- Z is the average client think time, i.e. the time between the reception of a response and the sending of the next request by a client.
- $V<V_1,..,V_M>$ are the visit ratios at respectively tiers $T_1..T_M$. More precisely, V_i corresponds to the ratio between the number of requests entering the multi-tier service (i.e. at the front-end tier T_1) and the number of subsequent requests processed by tier T_i. In other words, V_i represents the average number of subsequent requests on tier T_i when issuing a client request to the multi-tier Internet service. Thus, V_i reflects the impact of client requests on tier T_i. Note the particular case of $V_1 = 1$.
- $S<S_1,..,S_M>$ are the service times at tiers $T_1..T_M$. Thus, S_i corresponds to the average incompressible time for processing a request on tier T_i when the multi-tier Internet service is not loaded.
- $D<D_1,..,D_M>$ are the inter-tier communication delays. D_i is the average communication delay between tier T_{i-1}, if any, and tier T_i, with $i > 1$. Note the particular case of $D_1 = 0$.

Furthermore, service workload may vary over time, which corresponds to different client behaviors at different times. For instance, an e-mail service usually faces a higher workload amount in the morning than in the rest of the day. Workload variations have a direct impact on the quality-of-service as discussed later.

PROBLEM ILLUSTRATION

Both the configuration of an Internet service and the workload of the service have a direct impact on the quality-of-service. This section illustrates this impact through examples.

Impact of configuration. To illustrate the impact of service configuration, the TPC-W multi-tier online bookstore service is considered in the following. It consists of a front-end web tier and a back-end database tier (see the Evaluation Section for more details about TPC-W). Two distinct static ad-hoc configurations of the Internet service are considered here: κ_1(M = 2, AC < 1, 1 >, LC < 500, 500 >) and κ_2(M = 2, AC < 3, 6 >, LC < 200, 100 >). With κ_1, the capacity of the system is increased through an increased setting of the local configuration of servers. While with κ_2, the local configuration is set to the default one (i.e. default MPLs in Tomcat front-end server and MySQL back-end server), and the capacity of the system is extended by adding servers to increase the architectural configuration of the system. With both κ_1 and κ_2 configurations, the TPC-W online service is run with 1000 concurrent clients accessing the service using a read-only workload mix.

Figure 2 presents the client request latency and Figure 3 gives the abandon rate obtained with each configuration of the Internet service. κ_1 provides bad service performance and availability with a latency of 10s and an abandon rate of 71%. Obviously, this is due to too few resources that are assigned to the service. κ_2 provides better performance but still induces a bad service availability (40% of abandon rate) because of the inadequate default local configuration of the service.

Thus, both local and architectural configurations have an impact of the performance, availability and cost of Internet services. These configurations should therefore be carefully chosen in order to meet quality-of-service objectives and minimize the cost.

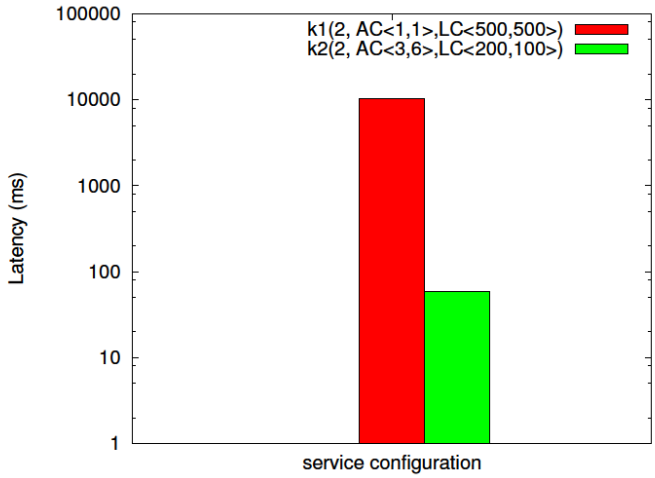

Figure 2: Impact of service configuration on performance

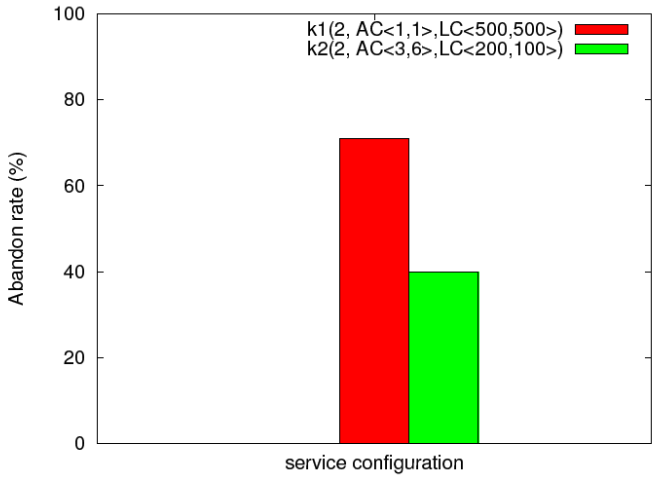

Figure 3: Impact of service configuration on availability

Impact of workload. Figure 4 and Figure 5 respectively present the impact of client workload variation on the performance, availability and cost of the TPC-W two-tier Internet service. Here, several workload variation scenarios are considered. The workload successively varies from a first stage with workload mix X_1 to a second stage with workload mix X_2 (see the Evaluation Section for more details about TPC-W and workload mixes). During each stage, the workload amount N (i.e. #clients) varies between 250 and 1250 clients. An ad-hoc medium configuration of the multi-tier Internet service is considered as follows: $\kappa(M = 2, AC < 2, 3 >, LC < 500, 500 >)$. The service latency and abandon rate of this configuration are presented in Figure 4 and Figure 5. Obviously, different workloads have different behaviors in terms of service performance and availability. For instance, with 1250 clients, latency is 4.5 s with workload mix X_1, and 10.7 s with workload mix X_2. This induces an abandon rate of 29% with mix X_1 vs. 39% with mix X_2. And within the same workload mix, service latency and abandon rate vary when the number of clients varies.

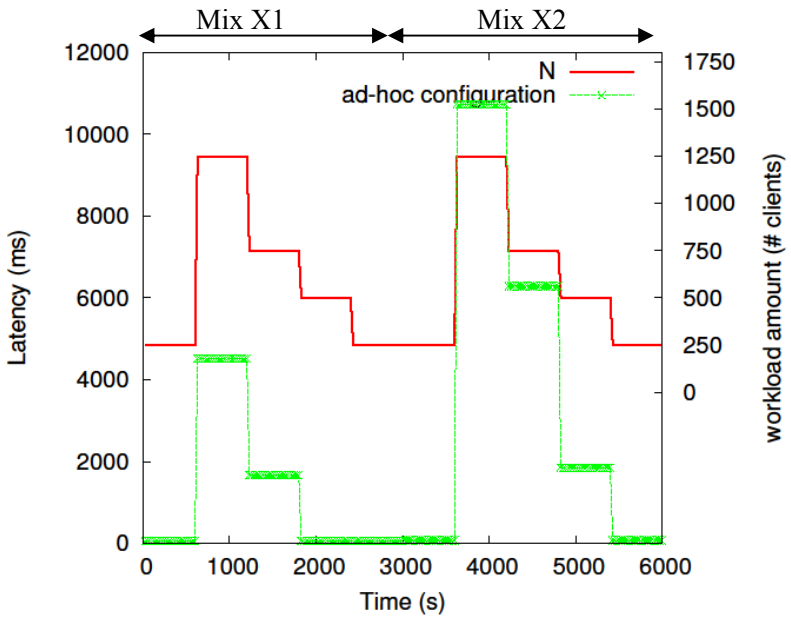

Figure 4: Impact of workload on performance

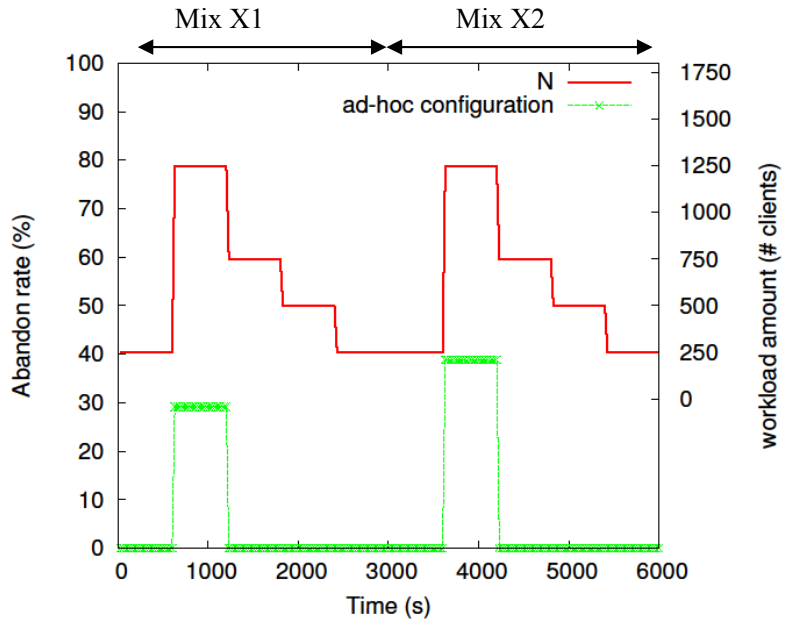

Figure 5: Impact of workload on availability

Nonlinear behavior. Figure 6 illustrates the behavior of the two-tier TPCW Internet service when the service workload amount varies for workload mix X1 (see the Evaluation Section for more details about TPC-W and workload mixes). Here, the workload amount (i.e. number of concurrent clients) increases linearly over time. However, the service latency does not vary linearly. This clearly shows that linearity assumptions that are made on multi-tier Internet services and linear control do not hold for these systems.

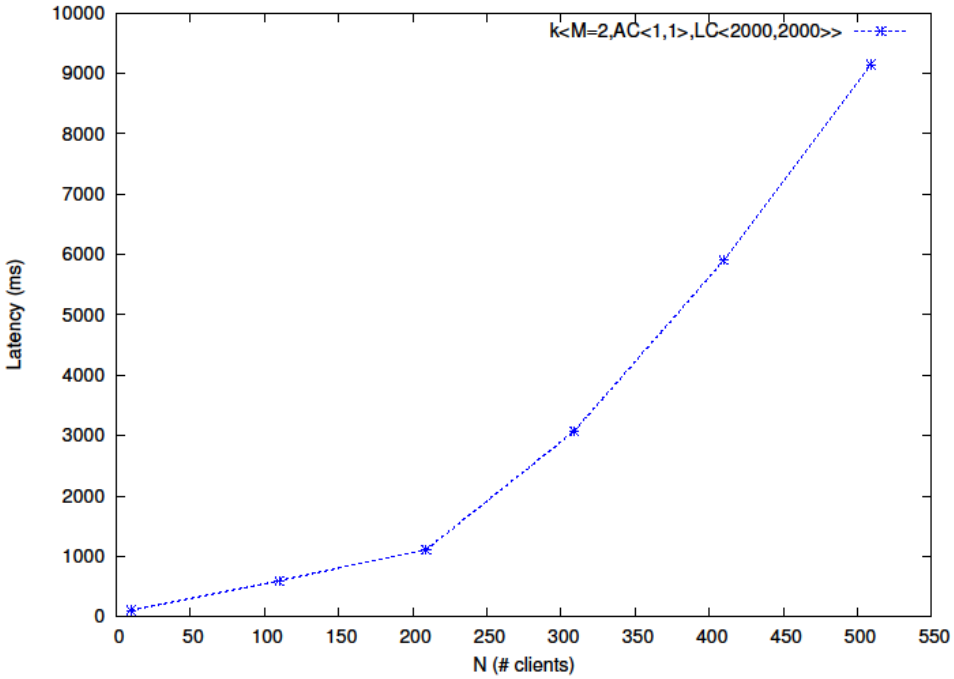

Figure 6: Nonlinear behavior of Internet services

ADAPTIVE CONTROL OF INTERNET SERVICES

Both service workload and service configuration have an impact on the performance, availability and cost of services. The workload of Internet services is an exogenous input, which variation can not be controlled. Thus, to handle workload variations and provide guaranties on performance and availability, Internet services must be able to dynamically adapt their underlying configuration. Several objectives are targeted here:

- Guarantee SLA constraints in terms of service performance and availability, while minimizing the cost of the Internet service.
- Handle *nonlinear* behavior of Internet services taking into account both workload amount *and* in workload mix variations over time.
- Provide self-adaptive control of Internet services that provides online automatic reconfigurations of Internet services.

We propose MoKa, a nonlinear utility-aware control for self-adaptive Internet services. First, MoKa is based on a utility function that characterizes the optimality of the configuration of an Internet service in terms of SLA requirements for performance and availability, in conjunction with service cost. Second, a nonlinear model of Internet services is described to predict the performance, availability and cost of a service. Third, a capacity planning method is proposed to calculate the optimal configuration of the Internet service. Finally, an adaptive nonlinear control of Internet services is provided to automatically apply optimal configuration to online Internet services. MoKa is built as a feedback control of multi-tier Internet services as described in Figure 7, with three main elements: (i) online monitoring of the Internet service, (ii) adaptive control of the Internet service, and (iii) online reconfiguration of the Internet service.

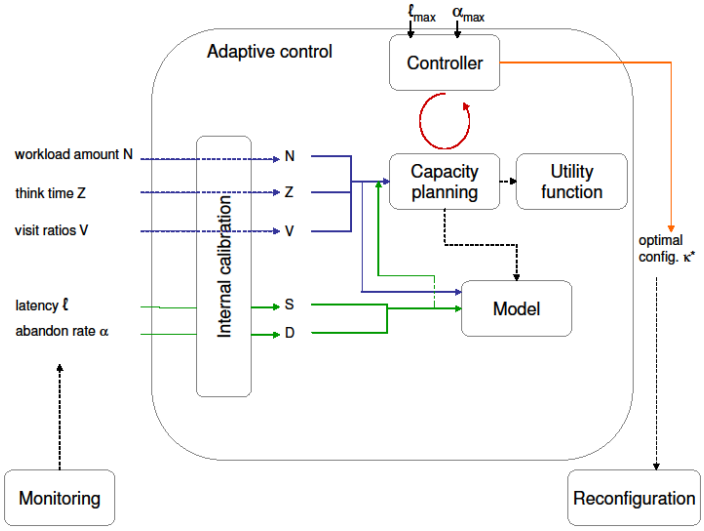

Figure 7: Adaptive control of Internet services

Online monitoring aims at observing the Internet service and producing the necessary data in order to, on the one hand, automatically calibrate MoKa's model, and on the other hand, trigger MoKa's capacity planning and control. MoKa's model calibration is performed online and automatically. This allows rendering the dynamics of service workload mix and workload amount, without requiring human intervention and manual tuning of model parameters, which makes the model easier to use. Automatic calibration is described in the Section titled "*Automatic and online MoKa calibration*". Therefore, the controller calls the utility-aware capacity planning method to calculate the optimal configuration κ^* for the current workload amount and workload mix. That optimal configuration guarantees the SLA performance and availability objectives while minimizing the cost of the Internet service. Finally, the new calculated configuration κ^* is applied to the Internet service. In the following, we successively present MoKa utility function, modeling, capacity planning and automatic calibration.

UTILITY FUNCTION OF INTERNET SERVICES

We consider an SLA of an Internet service that specifies service performance and availability constraints in the form of maximum latency ℓ_{max} and maximum abandon rate α_{max} not to exceed. *Performability Preference* (i.e. performance and availability preference) of an Internet service is defined as follows:

$$PP(\ell, \alpha) = (\ell \leq \ell_{max}) \cdot (\alpha \leq \alpha_{max}) \quad (1)$$

where ℓ and α are respectively the actual latency and abandon rate of the Internet service. Note that $\forall \ell, \forall \alpha, PP(\ell,\alpha) \in \{0, 1\}$, depending on whether Eq. 1 holds or not.

Based on the performability preference and cost of an Internet service, the utility function of the service combines both criteria as follows:

$$\theta(\ell, \alpha, \omega) = \frac{M \cdot PP(\ell,\alpha)}{\omega} \quad (2)$$

where ω is the actual cost (i.e. #servers) of the service, and M is the number of tiers of the multi-tier Internet service. M is used in Eq. 2 for normalization purposes. Here, $\forall \ell, \forall \alpha, \forall \omega, \theta(\ell, \alpha, \omega) \in [0, 1]$, since $\omega \geq M$ (at least one server per tier) and $PP(\ell,\alpha) \in \{0, 1\}$.

A high value of the utility function reflects the fact that, on the one hand, the Internet service guarantees service level objectives for performance and availability and, on the other hand, the cost underlying the service is low. In other words, an optimal configuration of an Internet service is the one that maximizes its utility function.

MODELING OF INTERNET SERVICES

The proposed analytic model predicts the latency, abandon rate and cost of an Internet service, for a given configuration κ of the Internet service, a given workload amount N and a given workload mix X. The input and output variables of the model are depicted in Figure 8.

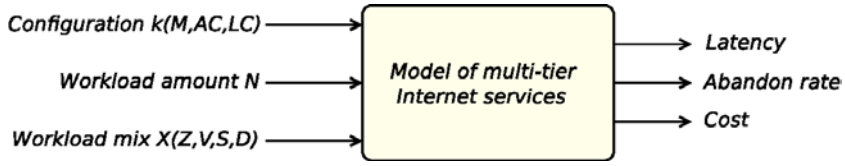

Figure 8: Model input and output variables

The model follows a queueing network approach, where a multi-tier system is modeled as an M/M/c/K queue. Moreover, Internet services are modeled as closed loops to reflect the synchronous communication model that underlies these services, that is a client waits for a request response before issuing another request. Figure 9 gives an example of a three-tier system with a configuration κ(M=3, AC<1,1,2>, LC<20,15,3>), a workload amount of 30 clients and a workload mix characterized, among others, by tier visit ratios V<1, 0.5, 2>.

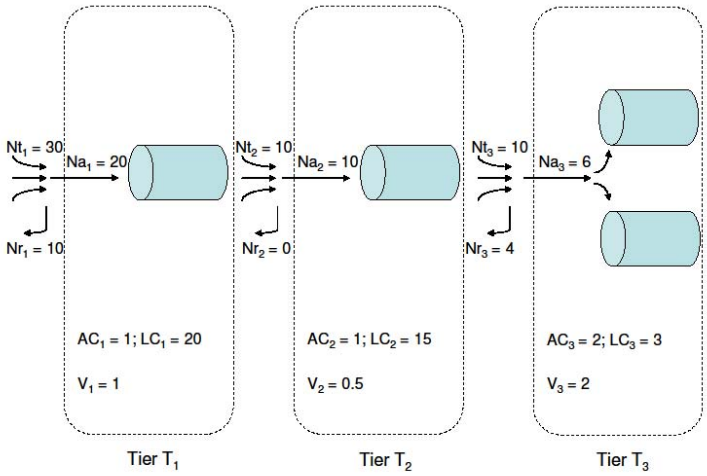

Figure 9: Replicated multi-tier Internet services as a queueing network

The example illustrates how the requests of the 30 incoming clients fly through the different tiers and server queues. For instance, among the $Nt_1 = 30$ clients that try to access tier T_1, $Nr_1 = 10$ are rejected because of local configuration (i.e. MPL limit) at that tier and $Na_1 = 20$ clients are actually admitted in the server at T_1. Then, the 20 client requests processed by T_1 generate $Nt_2 = 10$ subsequent requests that try to access tier T_2 (with a visit ratio $V_2 = 0.5$). All the 10 requests are admitted in the server of T_2 because they

are below T_2's MPL local configuration (i.e. $Na_2 = 10$). Finally, the 10 requests on T_2 would induce in total 40 requests to T_3 (with a visit ratio $V_3 = 2$). However, due to synchronous communication between the tiers of a multi-tier service, a request on T_2 induces at a given time at most one request to T_3, and in average 4 successive requests to T_3. Thus, $Nt_3 = 10$ subsequent requests tentatively access T_3. Among these 10 requests, $Nr_3 = 4$ requests are rejected because of T_3's MPL local configuration and $Na_3 = 6$ requests are admitted and balanced among the two server replicas of that tier. In summary, among the 30 client requests attempting to access the multi-tier service, a total of 14 are rejected and 16 are serviced, resulting in an abandon rate of 47%.

More generally, Algorithm 1 describes how the model predicts the latency, abandon rate and cost of a replicated multi-tier Internet service. This algorithm builds upon the MVA (Mean Value Analysis) algorithm (Reiser, 1980). It extends it to take into account the following main features: server replication (i.e. architectural configuration), server's MPL (i.e. local configuration), different workload mixes, and to include predictions of abandon rate and service cost. The algorithm consists of the following four parts.

The first part of the algorithm (lines 1–13) calculates the admitted requests and rejected requests at each tier of the multi-tier service following the method introduced in the example of Figure 9. It considers, in particular, the following model inputs: the service configuration $\kappa(M, AC, LC)$, the workload amount N and the workload mix X with its service visit ratios. Lines 1–6 of the algorithm calculate Nt_i, the number of requests that try to access tier T_i considering the visit ratios of the tiers. Line 6 guarantees that if Na_{i-1} requests are admitted to tier T_{i-1}, there would not be more than Na_{i-1} requests that try to access T_i, because of synchronous communication between the tiers. Lines 7–9 apply T_i's MPL local configuration to calculate Na_i, the number of requests admitted to T_i, and Nr_i, the number of requests rejected by T_i.

Furthermore, because a request admitted to T_i may be rejected by one of the following tiers $T_{i+1}..T_M$, lines 10–11 calculate Na_i', the number of requests admitted at T_i and not rejected by the following tiers. Finally, lines 12–13 produce the total number of rejected requests, Nr, and the total number of admitted requests, Na.

The second part of the algorithm (lines 14–32) is dedicated to the prediction of service request latency. First, lines 15–17 initialize queues' length and service demand at each tier (i.e. the amount of work necessary to process a request on a tier T_i, excluding the inter-tier communication delays D_i). Lines 18–31 consider the different tiers from the back-end to the front-end in order to estimate the cumulative request latency at each tier T_i: ℓa_i is the latency of a request admitted at tier T_i and admitted at the following tiers $T_{i+1}..T_M$, and ℓr_i is the latency of a request admitted at T_i and rejected at one of the following tiers. The latter case of requests will not be part of the final admitted requests but it is considered in the algorithm because it has an impact on queue length and, thus, on response times and latency calculation of admitted requests. Lines 19–22 introduce requests one by one at tier T_i, calculate service demand for each server replica at T_i, and estimate request response time for that tier. In line 22, the Max function is used to ensure that service demand is not lower than the incompressible necessary time W_i, as induced by service times characterizing the service workload mix. Then, lines 23–28 cumulate response times to calculate ℓa_i, the latency of requests admitted at tiers $T_i..T_M$, and ℓr_i, the latency of requests admitted at T_i but then rejected at one of the following tiers. These values are then used in lines 29–31 to calculate the queue length using Little's law. Finally, the overall latency ℓ of a client request is provided in line 32 as the latency of a request admitted to T_1 and never rejected in the following tiers.

Input:
$\kappa(M, AC, LC)$: service configuration
N: workload amount
$X(Z, V, S, D)$: workload mix
Output:
ℓ: service latency
α: service abandon rate
ω: service cost

/* **Admitted and rejected requests** */
for $i := 1\ ; i \leq M\ ; i++$ **do**
  if $i = 1$ **then**
    $Nt_i := N$
  else
    $Nt_i := Min(Na_{i-1}, Na_{i-1} * \frac{V_i}{V_{i-1}})$
  $MPL_i := LC_i \cdot AC_i$ /* *total MPL at* T_i */
  $Na_i := Min(MPL_i, Nt_i)$ /* *requests admitted at* T_i */
  $Nr_i := Nt_i - Na_i$ /* *requests rejected at* T_i */
for $i := 1\ ; i \leq M\ ; i++$ **do**
  $Na'_i := Na_i - \sum_{j=i+1}^{M} Nr_j$ /* *requests admitted at* $T_i..T_M$ */
$Nr := \sum_{i=1}^{M} Nr_i$ /* *total rejected requests* */
$Na := N - Nr$ /* *total admitted requests* */
/* **Service latency** */
for $i := 1\ ; i \leq M\ ; i++$ **do**
  $Ql_i := 0$ /* *total queue length at* T_i */
  $W_i := (S_i - D_i) \cdot V_i$ /* *service demand at* T_i*/
for $i := M\ ; i \geq 1\ ; i--$ **do**
  /* *introduce requests one by one at* T_i */
  for $j := 1\ ; j \leq Na_i\ ; j++$ **do**
    $W'_i := \frac{(1 + Ql_i) \cdot W_i}{AC_i}$ /* *service demand per server replica at* T_i */
    $R_i = Max(W'_i, W_i) + D_i \cdot V_i$ /* *response time of request admitted at* T_i */
    if $i < M$ **then**
      $\ell a_i := R_i + \ell a_{i+1}$ /* *latency of request admitted at* $T_i..T_M$ */
      $\ell r_i := R_i + \ell r_{i+1}$ /* *latency of request admitted at* T_i *and rejected at* $T_{i+1}..T_M$ */
    else
      $\ell a_i := R_i$ /* *latency of request admitted at* $T_i..T_M$ */
      $\ell r_i := R_i$ /* *latency of request admitted at* T_i *and rejected at* $T_{i+1}..T_M$ */
    $\tau a_i = \frac{j \cdot Na'_i / Na_i}{\ell a_i + Z}$ /* *throughput of requets admitted at* $T_i..T_M$ */
    $\tau r_i = \frac{j \cdot (Na_i - Na'_i)/Na_i}{\ell r_i + Z}$ /* *throughput of requets admitted at* T_i *and rejected at* $T_{i+1}..T_M$ */
    $Ql_i = (\tau a_i + \tau r_i) \cdot R_i$ /* T_i*'s total queue length with Little's law* */
$\ell := \ell a_1$ /* *final latency of requests admitted at* $T_1..T_M$ */
/* **Service abandon rate** */
$\tau a := \frac{Na}{\ell a_1 + Z}$ /* *throughput of requets admitted at* $T_1..T_M$ */
$\tau r := \frac{Nr - Nr_1}{\ell r_1 + Z}$ /* *throughput of requets admitted at* T_1 *and rejected at* $T_2..T_M$ */
$\tau r' := \frac{Nr_1}{Z}$ /* *throughput of requests rejected at* T_1 */
$\alpha := \frac{\tau r + \tau r'}{\tau r + \tau r' + \tau a}$ /* *total abandon rate* */
/* **Service cost** */
$\omega := \sum_{i=1}^{M} AC_i$; /* *total servers* */

Algorithm 1: Modeling replicated multi-tier Internet services

The third part of the algorithm (lines 33–37) is dedicated to the estimation of service abandon rate. It first calculates τa, the throughput of requests admitted at (and never rejected from) tiers $T_1..T_M$, τr, the throughput of requests admitted at T_1 but then rejected by one of the following tiers, and $\tau r'$, the throughput of requests rejected at T_1 due to MPL limit. These different values are then used to produce the total service request abandon rate α. Finally, the fourth and last part of the algorithm (lines 38–39) calculates the total cost ω of the replicated multi-tier service in terms of servers that underlie the system.

Thus, the algorithmic complexity of the proposed model is $O(M \cdot N)$, where M is the number of tiers of the multi-tier Internet service and N is the workload amount of the service.

CAPACITY PLANNING FOR INTERNET SERVICES

The objective of the capacity planning is to calculate an optimal configuration of a multi-tier Internet service, for a given workload amount and workload mix, to fulfill the SLA in terms of latency and abandon rate constraints, and to minimize the cost of the service. Thus, an optimal configuration κ^* is a configuration that has the highest value of the utility function θ^*. Figure 10 illustrates capacity planning.

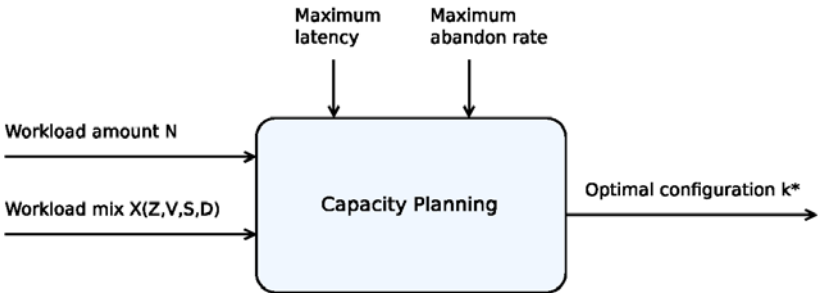

Figure 10: Capacity planning of multi-tier Internet services

The main algorithm of capacity planning is given in Algorithm 2. The algorithm takes as inputs a workload amount and a workload mix of an Internet service. It additionally has as input parameters the number of tiers of the Internet service, the SLA latency and abandon rate constraints to meet, and the underlying service analytic mode (see previous section). The algorithm produces an optimal configuration of the Internet service that guarantees the SLA and minimizes the cost of the service. The algorithm consists of two main parts: a first part that calculates a preliminary configuration that guarantees the SLA constraints, and a second part that minimizes service cost.

The first part of the algorithm (lines 1–12) first increases the number of servers assigned to all tiers of the Internet service (cf. line 6). It then adjusts the local configuration of servers to distribute client requests among server replicas at each tier (cf. line 8). However, if a request on a tier induces in total more than one request on the following tier (i.e. $V_{i-1} \leq V_i$), and due to synchronous communication between the tiers, the number of concurrent requests on the following tier does not exceed the number of concurrent requests on the current tier (cf. line 10). Afterward, the resulting service configuration along with the workload amount and workload mix are used to predict the latency, abandon rate and cost of the service. This process is repeated until a configuration that meets the SLA is found.

Input:
N: workload amount
$X(Z, V, S, D)$: workload mix
Output:
$\kappa(M, AC, LC)$: service configuration
Parameter:
M: #tiers of the service
ℓ_{max}: SLA latency constraint
α_{max}: SLA abandon rate constraint
$MO(\kappa, N, X) \rightarrow (\ell, \alpha, \omega)$: service prediction model

```
/* Meet SLA */
for i := 1 ; i ≤ M ; i++ do
    AC_i := 0
repeat
    for i := 1 ; i ≤ M ; i++ do
        AC_i++                                   /* Increase number of servers */
        if (i = 1) ∨ (i > 1 ∧ V_{i-1} > V_i) then
            LC_i := (N · V_i) / AC_i             /* A request on T_{i-1} induces at most one request on T_i */
        else
            LC_i := (LC_{i-1} · AC_{i-1}) / AC_i /* A request on T_{i-1} induces in total more than one request on T_i */
        (ℓ, α, ω) := MO(κ(M, AC, LC), N, X)      /* Call the prediction model */
until PP(ℓ, α) = 1                               /* κ guarantees SLA */
/* Minimize cost */
for i := 1 ; i ≤ M ; i++ do
    minAC := 1 ; maxAC := AC_i
    while minAC ≠ maxAC do
        AC_i := minAC + (maxAC − minAC)/2        /* Dichotomic search to reduce AC_i, and thus ω */
        if (i = 1) ∨ (i > 1 ∧ V_{i-1} > V_i) then
            LC_i := (N · V_i) / AC_i             /* A request on T_{i-1} induces at most one request on T_i */
        else
            LC_i := (LC_{i-1} · AC_{i-1}) / AC_i /* A request on T_{i-1} induces in total more than one request on T_i */
        (ℓ, α, ω) := MO(κ(M, AC, LC), N, X)      /* Call the prediction model */
        if PP(ℓ, α) = 1 then
            maxAC := AC_i                        /* Enough servers at T_i, search in minAC..AC_i */
        else
            if α > α_max then
                minAC := AC_i + 1                /* Too few servers, search in AC_i + 1..maxAC */
            else
                /* ℓ > ℓ_max: too high client concurrency level, reduce LC to decrease ℓ */
                for j := 1 ; j ≤ M ; j++ do
                    minLC := 1
                    maxLC := LC_j
                    while minLC ≠ maxLC do
                        LC_j := minLC + (maxLC − minLC)/2   /* Dichotomic search to increase LC_j, and thus reduce ℓ */
                        (ℓ, α, ω) := MO(κ(M, AC, LC), N, X) /* Call the prediction model */
                        if (PP(ℓ, α) = 1) ∨ (ℓ > ℓ_max) then
                            maxLC := LC_j            /* Enough or high concurrency at T_j, search in minLC..LC_j */
                        else
                            minLC := LC_j + 1        /* Low concurrency at T_j, search in LC_j + 1..maxLC */
                    /* End of dichotomic search on LC at T_1..T_M for current value of AC_i */
                    if PP(ℓ, α) = 1 then
                        maxAC := AC_i                /* Enough concurrency and servers at T_i, search in minAC..AC_i */
                    else
                        minAC := AC_i + 1            /* Maximum concurrency but too few servers at T_i, search in AC_i + 1..maxAC */
/* End of dichotomic search on AC at T_1..T_M */
```

Algorithm 2: Capacity planning of replicated multi-tier Internet services

The second part of the algorithm (lines 13–45) aims to reduce the number of servers assigned to the service in order to minimize its overall cost, that is to calculate the minimum values of AC_i. Basically, the minimum number of servers on tier T_i that is necessary to guarantee the SLA is between 1 and the value of AC_i calculated in the first part of the algorithm. To efficiently estimate the minimum value of AC_i, a dichotomic search on that interval is performed (cf. lines 15–17). The local configuration LC_i is adjusted accordingly to distribute requests among server replicas at tier T_i (lines 18–21). Then, the latency, abandon rate and cost of the resulting service configuration are predicted using the analytic model. If the configuration meets SLA constraints, the number of servers at that tier is reduced by pursuing the search of a lower value of AC_i in the lowest dichotomy [minAC..AC_i] (lines 23–24). Otherwise, the new configuration does not meet the abandon rate SLO or the latency SLO. The former case means that too few servers are assigned to the service; and the search of a higher value of AC_i is conducted in the highest dichotomy]AC_i..maxAC] (lines 26–27). If the abandon rate SLO is met but not the latency SLO, there may be two causes. Either the client request concurrency level is too high which increases the latency. In this case, lower values of LC_1..LC_M are efficiently calculated using a dichotomic search (cf. lines 30–39). If the new value of local configuration allows to successfully meet the SLA, the algorithm pursues its search of a lower architectural configuration (lines 41–42). Otherwise, that means that too few servers are assigned to the service, and the algorithm pursues the search of a higher architectural configuration (lines 43–44).

Thus, the proposed capacity planning method has an algorithmic complexity given by Eq. 3, where M is the number of tiers of the multi-tier Internet service, N is the workload amount of the service, AC_{max} and LC_{max} are respectively the maximum values of the architectural configuration and local configuration respectively used in the loops at lines 16 and 33 of Algorithm 2. Furthermore, the logarithmic cost on AC_{max} and LC_{max} are due to the dichotomic search on architectural and local configurations. As a comparison, the algorithmic complexity of an exhaustive search on the optimal architectural and local configurations of a multi-tier Internet service is $O(M^2 \cdot N \cdot AC_{max} \cdot LC_{max})$. The proposed capacity planning method outperforms the exhaustive search by orders of magnitude depending on the size of the service.

$$O(M^2 \cdot N \cdot \log_2(AC_{max}) + M^3 \cdot N \cdot \log_2(LC_{max})) \quad (1)$$

Moreover, the main capacity planning algorithm is complemented with an optional part presented in Algorithm 3.This is motivated by the fact that Algorithm 2 may result with a service configuration that is optimal for a workload amount N but where the local configuration is too restrictive for a workload amount higher than N. Indeed, the configuration produced by Algorithm 2 may allow to meet the SLA for a given value of workload amount N but not for a higher workload amount, although the architectural configuration would be sufficient to handle that higher workload. This would result in additional future service reconfigurations to handle higher workloads. Thus, Algorithm 3 aims to reduce future service reconfigurations and system oscillations by calculating, based on the result of Algorithm 2, the highest value of local configuration that guarantees the SLA for N and still guarantees the SLA for workload amounts higher than N.

Input:
N: workload amount
$X(Z,V,S,D)$: workload mix
Output:
$\kappa(M,AC,LC)$: service configuration
Parameter:
M: #tiers of the service
ℓ_{max}: SLA latency constraint
α_{max}: SLA abandon rate constraint
$MO(\kappa,N,X) \rightarrow (\ell,\alpha,\omega)$: service prediction model

1 **/* Reduce future service reconfiguration and oscillations */**
2 $N' := N$
3 $guaranteedSLA := true$
4 **repeat**
5 $N' := N' + 1$ /* Introduce a new client request in $T_1..T_M$ */
6 **for** $i := 1; i \leq M; i++$ **do**
7 **if** $(i = 1) \vee (i > 1 \wedge V_{i-1} > V_i)$ **then**
8 $LC_i := \frac{N' \cdot V_i}{AC_i}$ /* A request on T_{i-1} induces at most one request on T_i */
9 **else**
10 $LC_i := \frac{LC_{i-1} \cdot AC_{i-1}}{AC_i}$ /* A request on T_{i-1} induces in total more than one request on T_i */
11 /* After introducing a new client request in $T_1..T_M$ */
12 $(\ell,\alpha,\omega) := MO(\kappa(M,AC,LC),N',X)$ /* Call the prediction model with increased client requests */
13 **if** $PP(\ell,\alpha) = 0$ **then**
14 $guaranteedSLA := false$
15 **until** $guaranteedSLA = false$
16 /* $guaranteedSLA = false$: reduce admitted client requests in $T_1..T_M$ */
17 $N' := N' - 1$
18 **for** $i := 1; i \leq M; i++$ **do**
19 **if** $(i = 1) \vee (i > 1 \wedge V_{i-1} > V_i)$ **then**
20 $LC_i := \frac{N' \cdot V_i}{AC_i}$ /* A request on T_{i-1} induces at most one request on T_i */
21 **else**
22 $LC_i := \frac{LC_{i-1} \cdot AC_{i-1}}{AC_i}$ /* A request on T_{i-1} induces in total more than one request on T_i */

Algorithm 3: Additional part to capacity planning algorithm

Proofs

This section first describes properties that underlie multi-tier Internet services, before presenting the proofs of optimality and termination of the proposed capacity planning method.

Properties

P1. The service level objectives specified in the SLA are expressed in a reasonable way; that is the latency and abandon rate constraints of the SLA are eventually achieved with (enough) servers assigned to the Internet service.

P2. Adding servers to a multi-tier Internet service does not degrade the performance and availability of the service. Furthermore, if there is a latency or abandon rate bottleneck at a tier, adding (enough) servers to that tier will eventually improve the latency/abandon rate of the service, and eventually remove the bottleneck from that tier.

P3. Augmenting the server concurrency level (i.e. the MPL) will eventually increase the latency of the service and reduce the abandon rate of the service. Decreasing the server concurrency level will eventually reduce the latency of the service and increase the abandon rate.

Proof of Optimality

An optimal configuration of an Internet service for a given workload is a configuration that guarantees the SLA and that induces a minimal cost for the service. Let κ*(M, AC*, LC*) be the optimal configuration of a multi-tier Internet service consisting of M tiers. Thus

$$\mathrm{PP}(\kappa^*) = 1$$

This is possible thanks to property P1 that states that the SLA is achievable. Furthermore, let κ(M, AC, LC) be any configuration of the service.

$$\forall \kappa, PP(\kappa) = 1 \Rightarrow \sum \mathrm{AC_i} \geq \sum \mathrm{AC_i}^*$$

That is

$$\forall \kappa, \theta(\kappa) \leq \theta(\kappa^*) \wedge \theta(\kappa^*) > 0$$

In the following, we will first show that the configuration produced by the proposed capacity planning algorithm meets the SLA, and then we will demonstrate that this configuration has a minimal cost.

Let κ(M, AC, LC) be the configuration produced as a result of the capacity planning of Algorithm 2. Suppose that κ does not guarantee the SLA. First, lines 1–12 of the algorithm iterate and increase the servers assigned to the Internet service until the SLA is met. Indeed, based on properties P1 and P2, this loop will eventually terminate with a configuration that guarantees the SLA at line 12. Then, suppose that the remainder of the algorithm (lines 13–45) results in a configuration that does not meet the SLA. This corresponds to one of the three following cases: line 26, line 38 or line 43 of the algorithm. In both cases of lines 26 and 43, the number of servers assigned to the service is increased, which will allow to eventually meet the SLA (cf. properties P1 and P2). Line 38 corresponds to the case where the abandon rate constraint is not met and where the server concurrency level is augmented. This will either allow to meet the SLA constraints based on property P3 (cf. line 41), or will be followed by an increase of the servers assigned to the service which eventually guarantees the SLA based on properties P1 and P2 (cf. line 43). Thus, this contradicts the supposition that the configuration produced by the capacity planning algorithm does not meet the SLA.

Suppose now that the configuration κ(M, AC, LC) produced by the capacity planning algorithm, which guarantees the SLA, does not have a minimal cost. That is

$$(PP(\kappa) = 1) \wedge (\sum \mathrm{AC_i} > \sum \mathrm{AC_i}^*)$$

By definition, removing any server from the optimal service configuration would result in SLA violation (i.e. performability preference violation) and the occurrence of a bottleneck at the tier where the server was removed.

$$\forall \kappa, \exists i \in [1..M], \mathrm{AC_i} < \mathrm{AC_i}^* \Rightarrow PP(\kappa) = 0$$

Thus, if the configuration κ resulting from the capacity planning algorithm does not have a minimal cost

$$\exists i \in \left[1..M\right](PP(\kappa) = 1) \wedge (\mathrm{AC_i} > \mathrm{AC_i}^*) \quad (2)$$

That means that, in Algorithm 2, the dichotomic search on $\mathrm{AC_i}$ iterated on the high dichotomy $]\mathrm{AC_i}..\mathrm{maxAC}]$ instead of iterating on the low dichotomy $[\mathrm{minAC}..\ \mathrm{AC_i}]$. This corresponds to one of the two cases at line 27 or line 44 of the algorithm. However, in both cases, SLA constraints are not met,

which contradicts Eq. 4 and thus, contradicts the supposition that the configuration produced by the capacity planning algorithm does not have a minimal cost.

Proof of Termination

Obviously, the model algorithm presented in Algorithm 1 terminates in $O(M \cdot N)$ calculation steps. Furthermore, Algorithm 2 that describe the capacity planning method consists of three successive parts. The first part (lines 1–3) evidently terminates in M steps. The second part (lines 4–12) iterates until a service configuration that guarantees SLA is found. Based on properties P1 and P2, this second part of the algorithm eventually terminates after $O(M^2 \cdot N \cdot \log_2(AC_{max}))$ calculation steps. Finally, the third part of Algorithm 2 (13–45) terminates after $O(M^2 \cdot N \cdot \log_2(AC_{max}) + M^3 \cdot N \cdot \log_2(LC_{max}))$ steps. Thus, the capacity planning algorithm is guaranteed to terminate.

AUTOMATIC AND ONLINE MOKA CALIBRATION

Online monitoring and internal calibration allow to automatically calibrate the proposed model and capacity planning methods with their input values without requiring manual calibration or profiling from a human administrator. This enables MoKa to self-adapt to changes in the workload mix and to precisely exhibit the new behavior of the Internet service.

Online monitoring of Internet services is, first, performed using sensors that periodically measure the state of the service and collect data such as the workload amount N, the client think time Z, and the visit ratios $V<V_1,..,V_M>$ of the multi-tier service. Then, average values of the collected data are calculated using the EWMA (Exponentially Weighted Moving Average) filter (Box, 2009). This filter produces average values of past observations on a time window; and the weighting, which decreases exponentially for older observations, gives more importance to recent observations. Thus, once collected by the sensors, the data is filtered and the average values are given as inputs for the modeling and capacity planning methods that underlie adaptive control.

Other input variables are needed by the modeling and capacity planning methods such as the service times $S<S_1,\ldots,S_M>$ and inter-tier delays $D<D_1,\ldots,D_M>$. Because these variables are too sensitive to monitoring, they are automatically calculated by an internal calibration process of the proposed adaptive control system as depicted by Figure 7. This is done using the descending gradient method, a first-order optimization method that allows to efficiently calculate the parameter values that provide the best accuracy for the model predictions (Avriel, 2003). To do so, the latency and abandon rate of the multi-tier service are monitored online as described earlier. Roughly speaking, this monitored data is compared with the predictions of the model when using different values of S and D (in addition to N, Z and V that were obtained as described earlier), and the values of S and D that finally maximize the accuracy of the model are chosen.

EVALUATION

Experimental Environment

We implemented the MoKa adaptive control of Internet services as a Java software prototype. The MoKa prototype consists of three mains parts: one for the modeling of services, one for the capacity planning of services, and one for the service controller. Furthermore, MoKa is designed as an open framework that can be easily extended to include new model and capacity planning algorithms, and to compare them regarding their accuracy, optimality and efficiency. Moreover, MoKa follows a proxy-based approach in order to integrate the proposed adaptive control to an Internet service in a non-intrusive way. This allows MoKa, for instance, to integrate monitoring sensors and reconfiguration actuators of an Internet service.

The evaluation of the proposed MoKa modeling and adaptive control was conducted using the TPC-W benchmark (TPC-W, 2010). TPC-W is an industry-standard benchmark from the Transaction Processing Performance Council that models a realistic web bookstore. TPC-W comes with a client emulator which generates a set of concurrent clients that remotely access the bookstore application. They emulate the behavior of real web clients by issuing requests for browsing the content of the bookstore, requesting the best-sellers, buying the content of the shopping cart, etc. The client emulator generates different profiles by varying the workload amount and workload mix (the ratio of browse to buy). In our experiments, the on-line bookstore was deployed as a two-tier system, consisting of a set of replicated web/business front-end servers, and a set of replicated back-end database servers. The client emulator was running on a dedicated machine to remotely send requests to the online bookstore. Two workload mixes were used for our experiments: mix X1 representing a version of TPC-W's browsing mix with read-only interactions, and mix X2 that extends X1 for a heavier workload on the back-end tier. Whereas the original TPC-W client emulator allows to specify given static workload amount and workload mix, we modified the client emulator in order to introduce more dynamics to the generated workload. Thus, during a given experiment, the workload amount and the workload mix vary over time.

The following experiments with MoKa were conducted on the Grid'5000 experimental platform (Grid'5000, 2010). The machines consist of Intel Xeon processors running at 2.33 GHz, they embed a memory of 4 Go, and are interconnected via a Gigabit Ethernet network. The machines run the Linux 2.6.26 kernel, Apache Tomcat 5.5 for web/application servers, and MySQL 5.0 for database servers. Round-robin was used to dynamically balance the load among server replicas.

Model Evaluation

This section evaluates the accuracy of the proposed analytic model of Internet services. It considers a workload that varies over time in amount and in mix, as described in Figure 11. Here, the workload mix varies from mix X1 to mix X2, and for each mix, the workload amount varies between 250 and 1250 clients. In this context, the behavior of the real multi-tier Internet service is compared with the predictions of the proposed model. Furthermore, two (static) configurations of the multi-tier Internet service are considered: $\kappa 1(2,\langle 1,1\rangle,\langle 500,500\rangle)$ and $\kappa 2(2,\langle 3,6\rangle,\langle 200,100\rangle)$. The former configuration represents a minimal architectural configuration where the capacity of the system is increased by increasing the local configuration. While the latter configuration uses default local configurations of Tomcat front-end server and MySQL back-end server, and increases the capacity of the system by increasing the architectural configuration. The use of these two configurations intends to mimic human administrators who apply ad-hoc configuration to increase the capacity of Internet services.

Figure 12 compares the latency measured on the online Internet service vs. the latency predicted by the model, and Figure 13 compares the real abandon rate of the online Internet service with the abandon rate calculated by the model. Both figures show that the model is able to accurately predict the latency and abandon rate of the multi-tier Internet service. For instance, the average difference between the real latency and the predicted latency is 8% for $\kappa 1$ and an absolute difference of 20ms for $\kappa 2$. The abandon rate is predicted with an average error not exceeding 2%. Furthermore, since the prediction of the model regarding the cost of an Internet service is straightforward, it is thus accurate and not presented here.

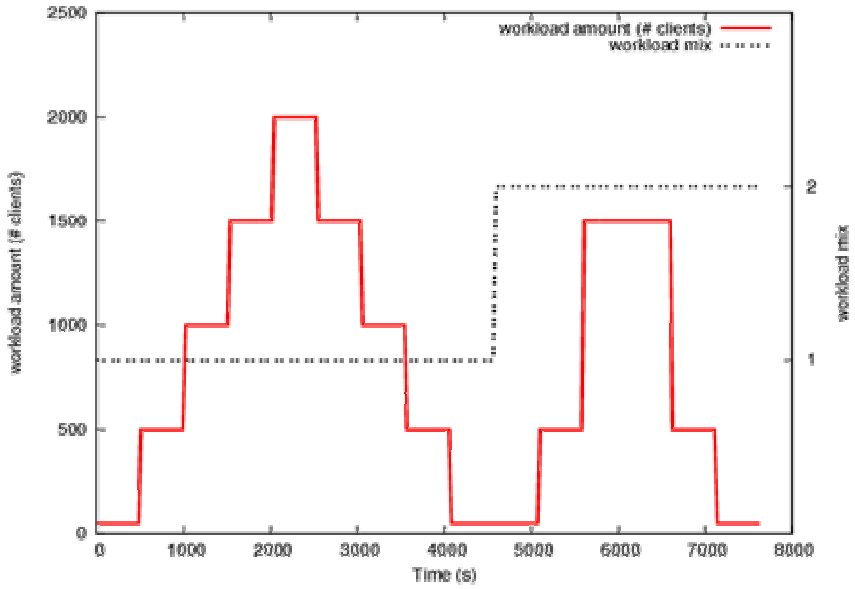

Figure 11: Workload variation

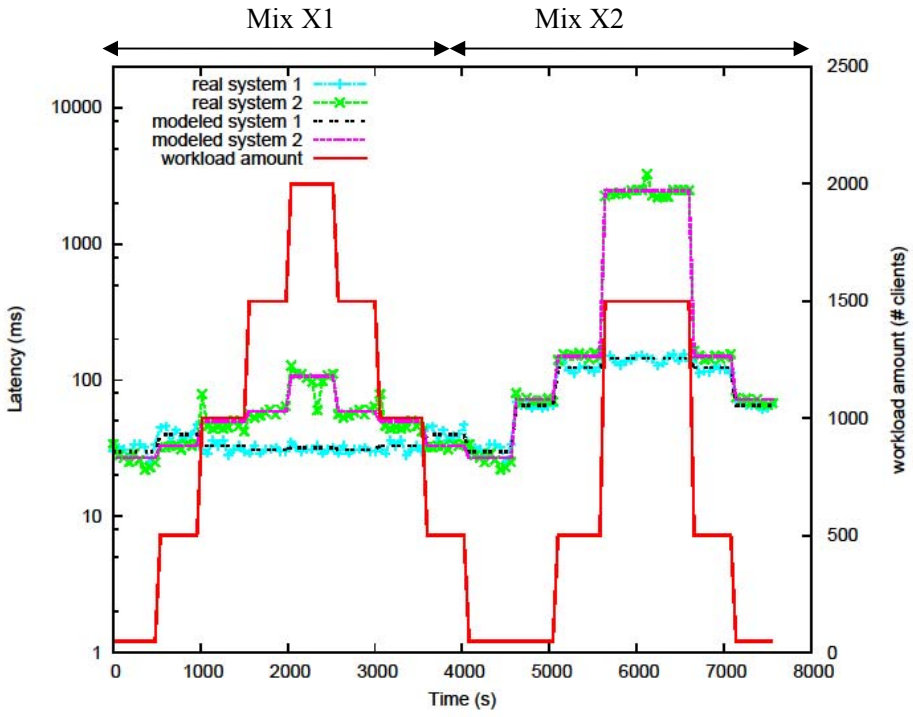

Figure 12: Accuracy of performance – real latency vs. predicted latency

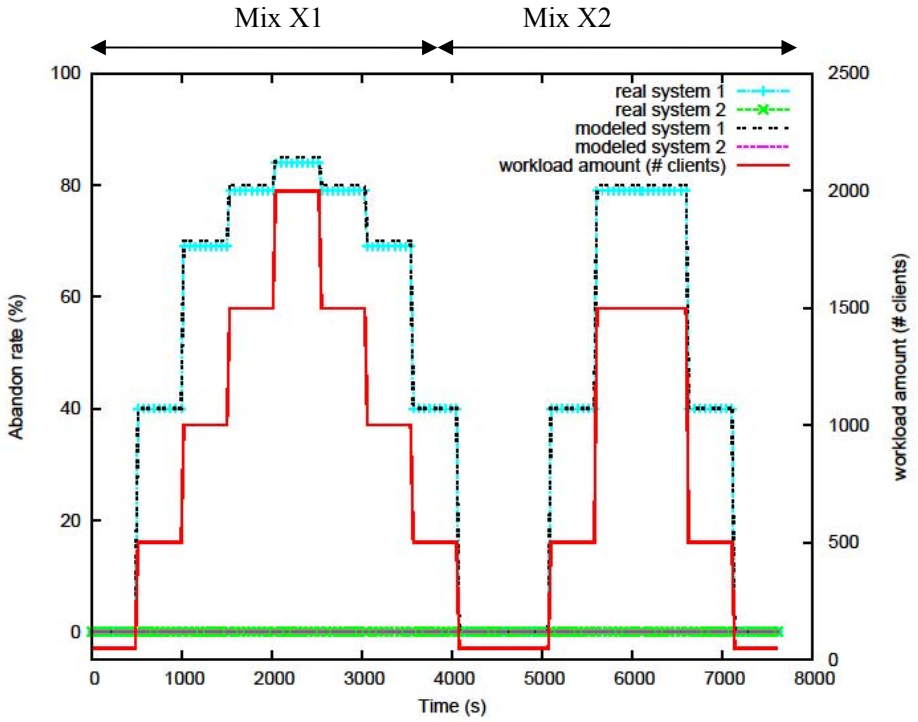

Figure 13: Accuracy of availability – real abandon rate vs. predicted abandon rate

Capacity Planning Evaluation

This section evaluates the proposed capacity planning method with regard to the optimality of the configuration produced by the method. To do so, the following SLA is considered with a maximum service latency ℓ_{max} of 1s and a maximum service abandon rate α_{max} of 10%. That means that capacity planning must produce a service configuration with a minimal cost while guaranteeing that at least 90% of client requests are processed within 1s.

Here, different workload mixes and different workload amount values are considered for the two-tier TPC-W Internet service. For each workload value, the proposed capacity planning method calculates an architectural configuration and a local configuration of the Internet service that are respectively presented in Figure 14 and Figure 15. Furthermore, we compare the result of the proposed capacity planning method with the result of another method based on an exhaustive search. This latter method performs a search on the set of possible architectural and local configurations of the Internet service. It compares all possible configurations and produces the optimal configuration that guarantees the SLA and minimizes the cost of the service.

Figure 14 and Figure 15 compare the two methods with regard to their calculated architectural and local configurations, and show that the proposed capacity planning method produces the optimal configuration of the Internet service.

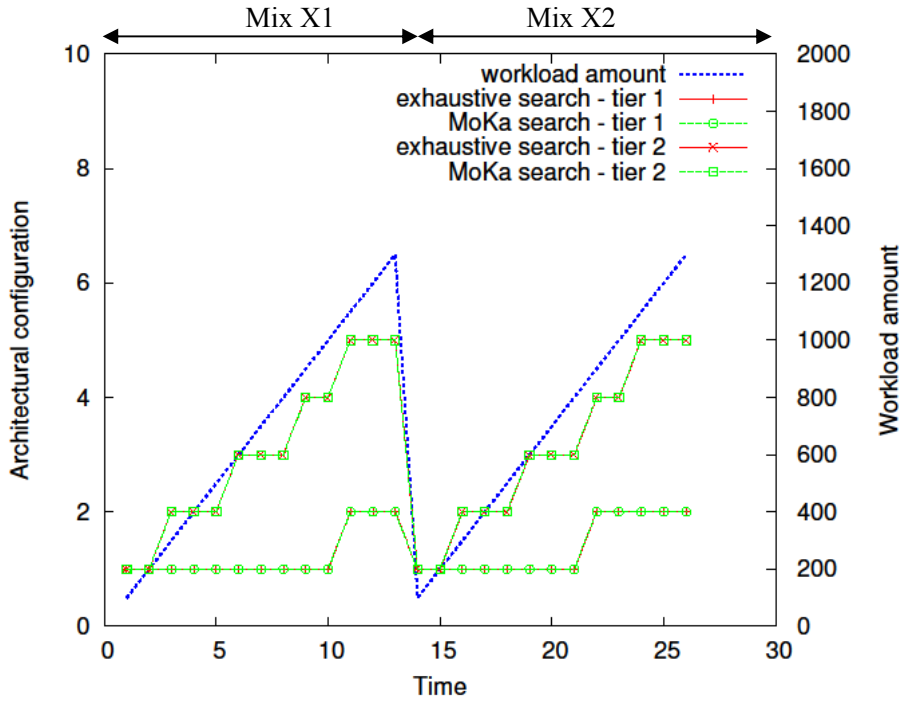

Figure 14: Optimality of architectural configuration

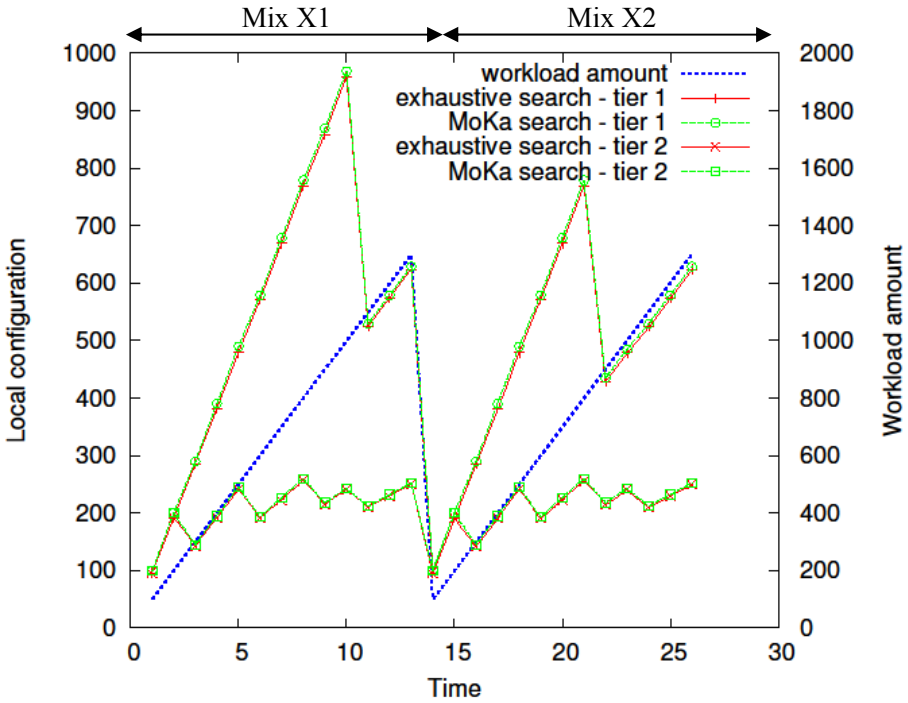

Figure 15: Optimality of local configuration

Adaptive Control Evaluation

This section presents the evaluation of the MoKa-based control applied to the online two-tier TPC-W Internet service. Here, the SLA specifies a maximum service latency ℓ_{max} of 1s and a maximum service abandon rate α_{max} of 5% (these values are chosen for illustration purposes, although ℓ_{max} and α_{max} can have other values, e.g. α_{max} = 0% to represent a service that is always available). Figure 16 describes the variation over time of the Internet service workload, i.e. the variation of the workload mix from mix X1 to mix X2 and, for each mix, the variation of the workload amount between 250 and 1000 concurrent clients. In this context, the behavior of the MoKa-based controlled system is first compared with two baseline systems: one with a small (static) configuration κ1(2,<1,1>,<500,500>) and another with a larger (static) configuration κ2(2,<3,6>,<200,100>). Figure 17 and Figure 18 respectively present the service latency and service abandon rate of the multi-tier Internet service, comparing the MoKa-based controlled system with the two non-controlled baseline systems κ1 and κ2. These figures show that κ1 is not able to meet the SLA performance constraint, and that κ2 is not able to meet the SLA availability constraint when the workload is too heavy. In comparison, MoKa is able to control the configuration of the Internet service in order to meet SLA constraints. The different points where the MoKa-based controlled system is above the SLA limits correspond to the occurrence of workload changes and the necessary time for the system to reconfigure and stabilize. Here, there is an order of magnitude between the average latency of the non-adaptive κ1 system and the average latency of the MoKa-based system; and there is a factor of 3 between the abandon rate of κ1 vs. the abandon rate of the MoKa-based system.

The adapted architectural and local configurations of the MoKa-based controlled system are shown in Figure 19 and Figure 20, and the cost is given in Figure 21. This shows that MoKa is able to assign to the Internet service the strictly necessary servers to guarantee the SLA, with a saving of up to 67% of servers. Finally, MoKa is able to automatically detect service workload changes and self-calibrate with the current workload amount (N between 250 and 1000 in the present experiment), and the current workload mix (e.g. mix X1 ($Z = 7$ s; $V <V_1 = 1.0, V_2 = 3.0>$; $S <S_1 = 9$ ms, $S_2 = 14$ ms>; $D <D_1 = 0, D_2 = 2$ ms>), and mix X2 ($Z = 7$ s; $V <V_1 = 1.0, V_2 = 1.5>$; $S <S_1 = 11.5$ ms, $S_2 = 27$ ms>; $D <D_1 = 0, D_2 = 4$ ms>)) Based on this automatic MoKa calibration, optimal configuration is dynamically applied to the online service.

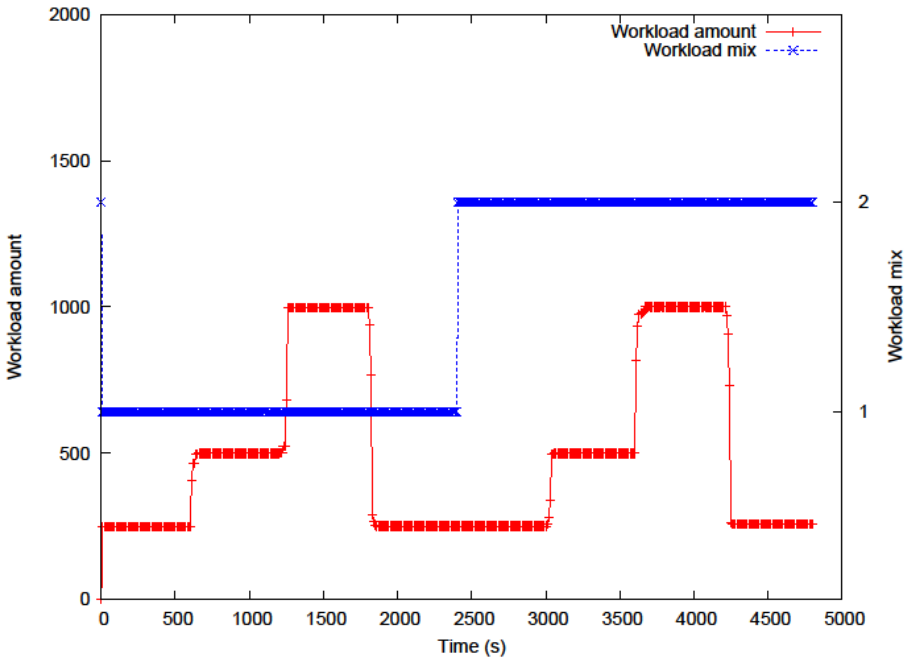

Figure 16: Workload variation

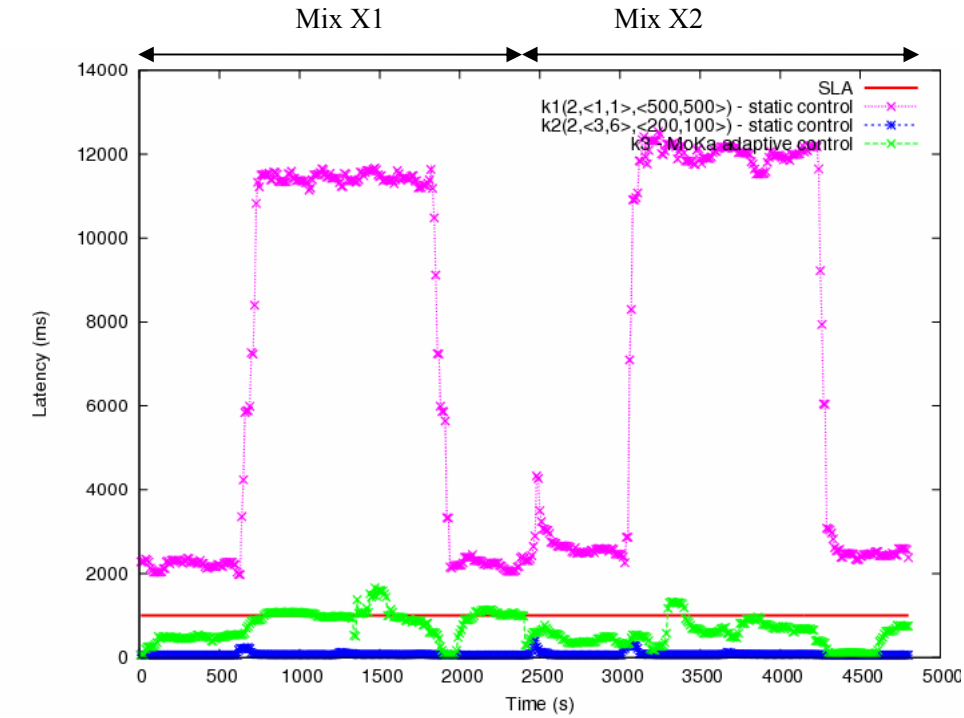

Figure 17: Service latency with and without MoKa control

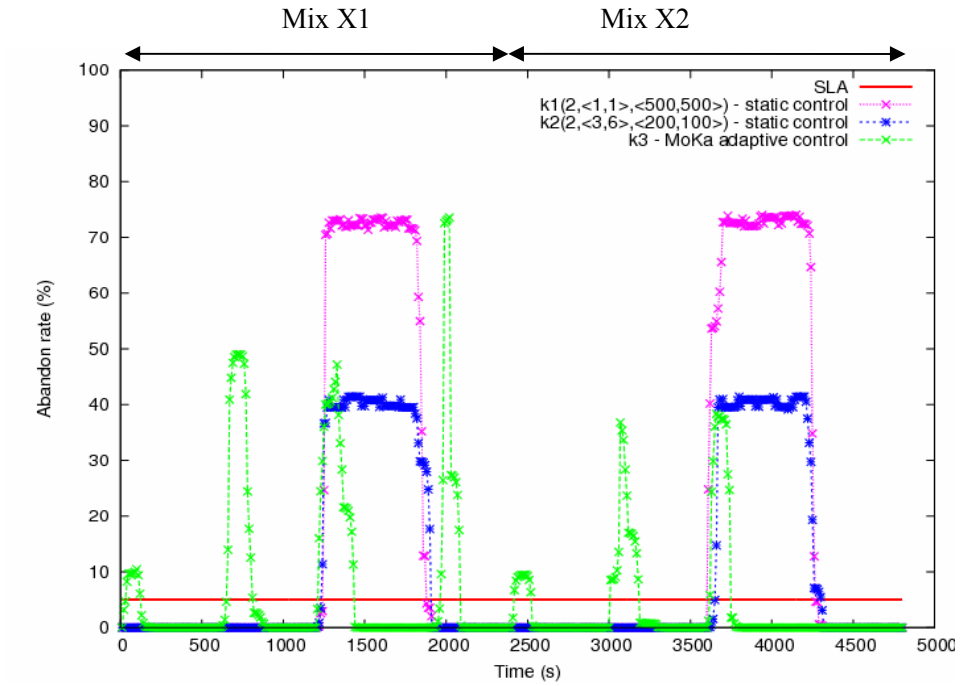

Figure 18: Service abandon rate with and without MoKa control

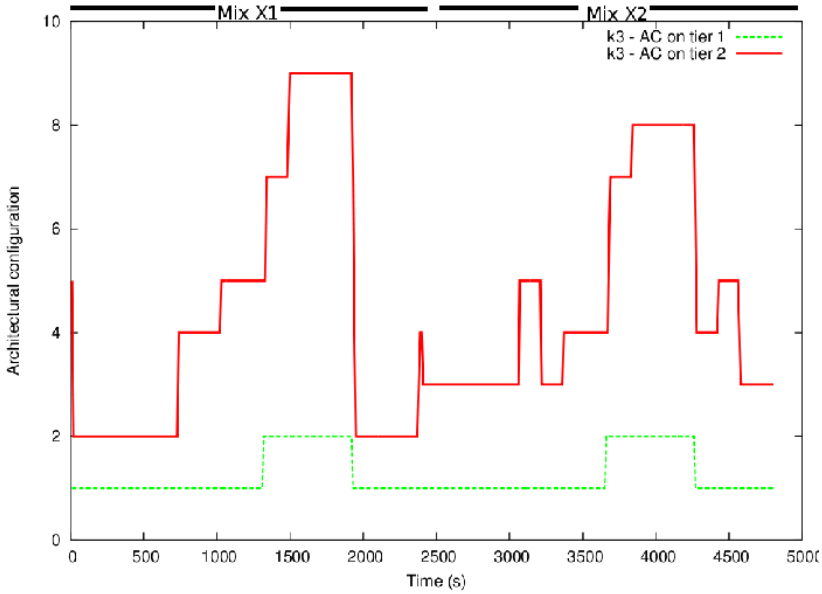

Figure 19: Service architectural configuration with and without MoKa control

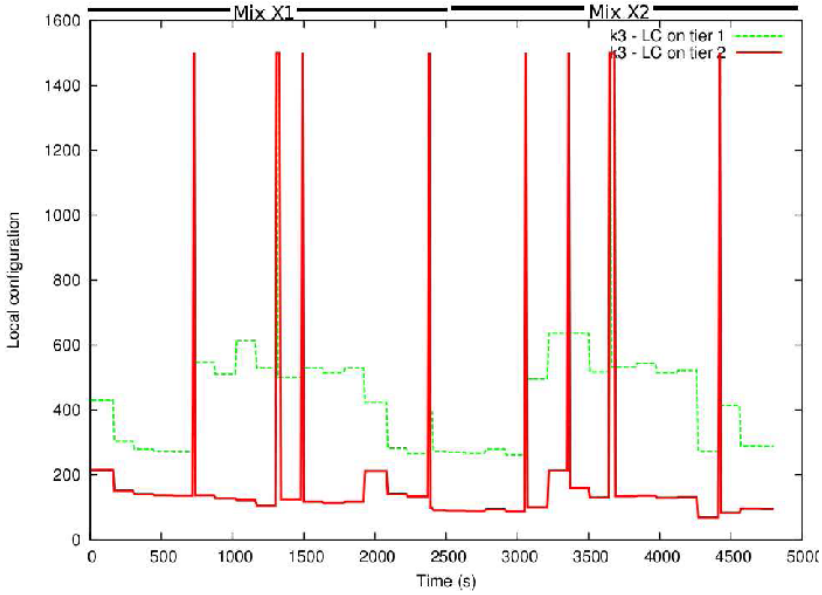

Figure 20: Service local configuration with and without MoKa control

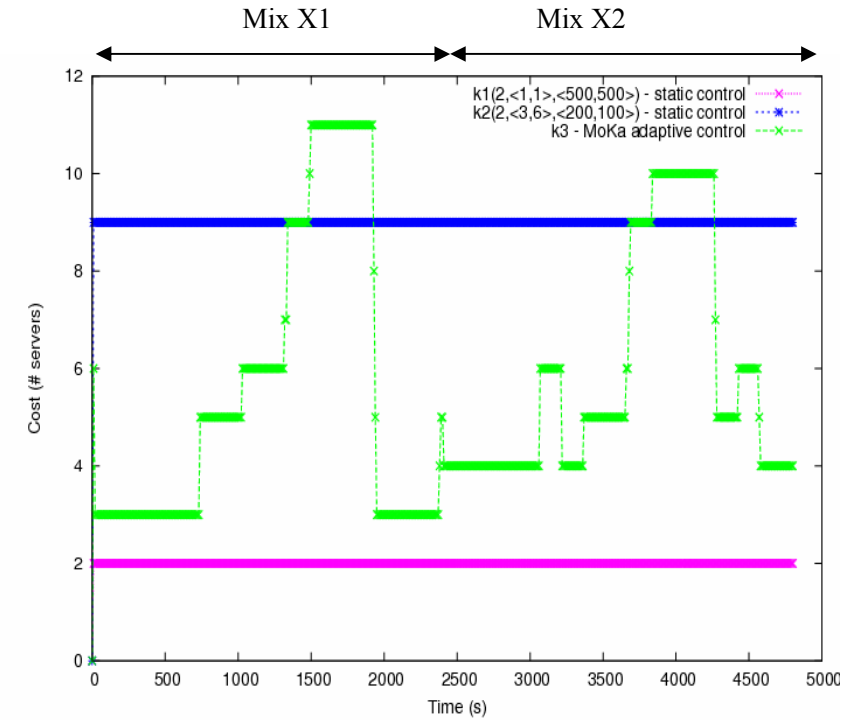

Figure 21: Service cost with and without MoKa control

In addition to the previous comparison of MoKa with ad-hoc static configurations, we also compare MoKa with a linear control approach, a technique classically used for Internet service provisioning. Here again, the SLA specifies a maximum service latency ℓ_{max} of 1s and a maximum service abandon rate α_{max} of 5%. And the Internet service workload varies over time, with a workload mix variation from mix X1 to mix X2 and a workload amount variation between 200 and 4000 concurrent clients.

Roughly speaking, the linear control is first calibrated with two internal parameters: a reference service configuration κ_0 and a reference workload amount N_0. The reference workload amount is the maximum number of concurrent clients that the reference service configuration can handle while guaranteeing the SLA. These internal parameters of the linear controller are obtained through preliminary profiling of the Internet service. Afterwards, the controller applies a linear capacity planning method that simply calculates an architectural configuration that is proportional to the current workload amount N and the reference workload amount N_0 and reference service configuration κ_0. In the following experiments, the reference service configuration and workload amount for the calibration of the linear control are respectively κ_0(<2,<1,3>,<600,200>) and N_0 = 630 clients.

Figure 22 and Figure 23 respectively present Internet service latency and abandon rate, and compare the MoKa-based controlled service with the linearly controlled service. The figures show that both control approaches are able to meet SLA requirements. However, MoKa keeps the latency and abandon rate near the SLA limits, which allows it to improve resource usage, and thus to reduce service cost compared to the linear control. This is shown in Figure 24 where, compared to MoKa optimal control, the linear control induces a cost overhead of 23%.

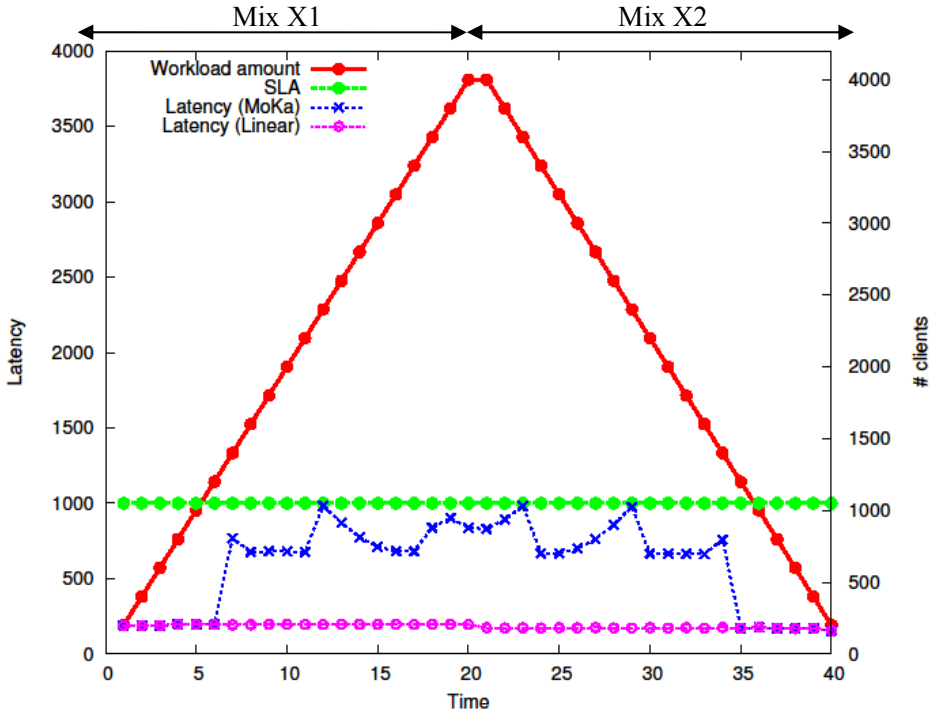

Figure 22: Service latency with different control techniques

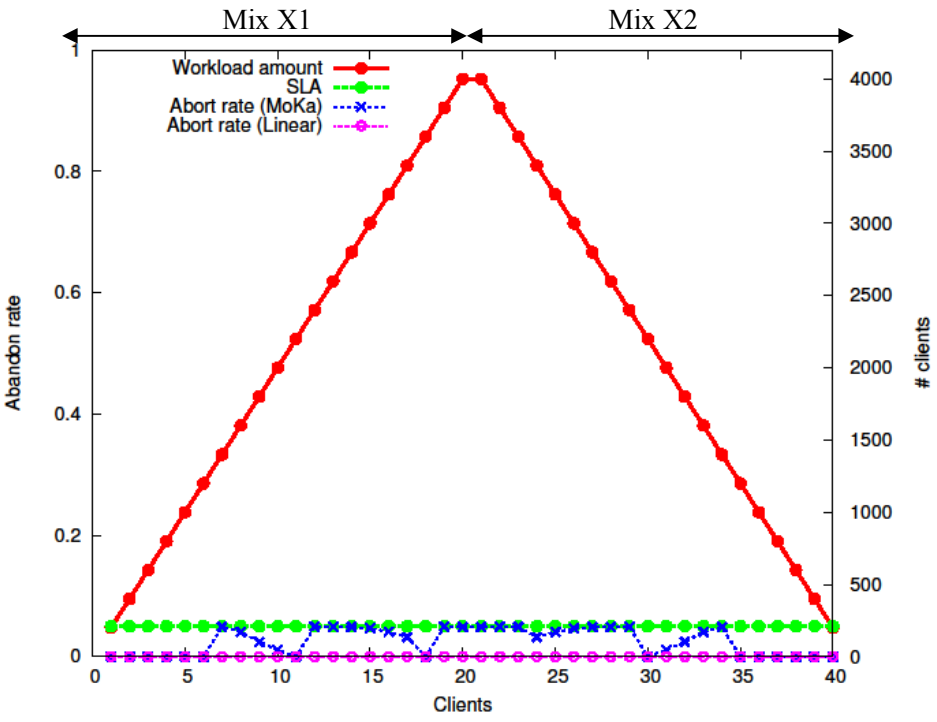

Figure 23: Service abandon rate with different control techniques

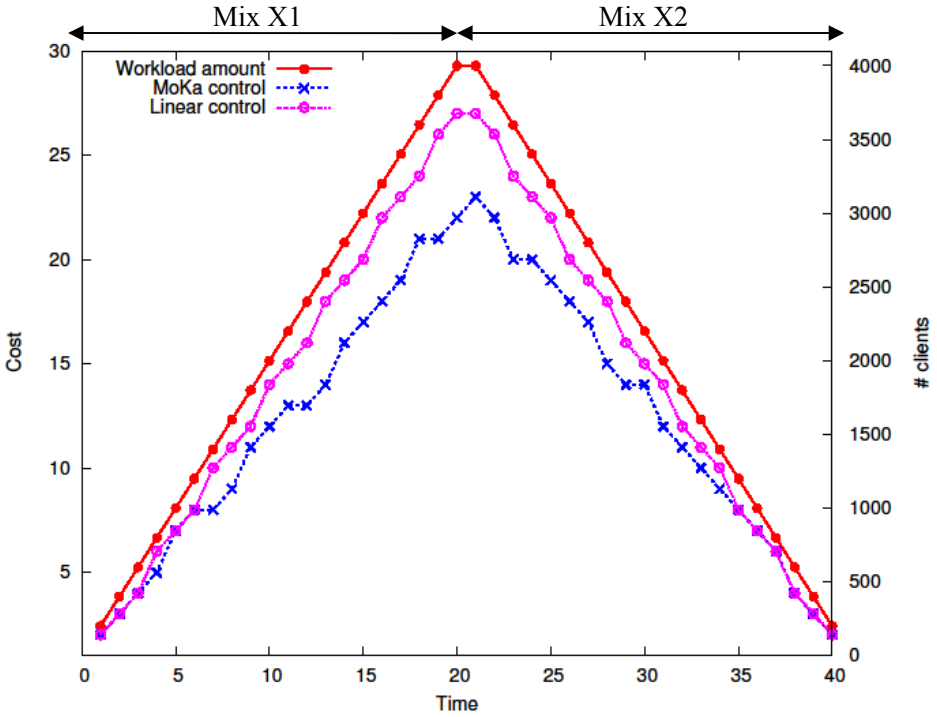

Figure 24: Service cost with different control techniques

RELATED WORK

The control of services to guarantee the SLA is a critical requirement for successful performance and availability management of Internet services (Loosley, 1997; Marcus, 2003; Menascé, 2001). The management of service performance and availability is usually achieved by system administrators using ad-hoc tuning (Brown, 2010; Microsoft, 2010). However, new approaches tend to appear to ease the management of such systems. These approaches differ with regard to several criteria: tackling performance and/or availability objectives, handling Internet service workload variations in terms of workload amount and/or workload mix, the used control techniques, and the applied control mechanism (i.e. actuators).

Different control mechanisms may be considered to manage service performance and availability, such as server provisioning, admission control, service differentiation, service degradation, and request scheduling (Guitart, 2010). In the following, we will focus on approaches using the two first techniques, namely admission control for a local configuration of the concurrency level of a server, and server provisioning for an architectural configuration of the size of a replicated distributed Internet service.

Admission control fixes the MPL concurrency level of a multi-programming system (e.g. multi-threaded servers). It has been extensively studied in server systems, and it was applied to a web server (Elnikety, 2004), a database server (Milan-Franco, 2004), or a multi-tier system (Menascé, EC 2001). Some admission control solutions are proposed in the form of heuristics (Heiss, 1991; Chen, 2003; Milan-Franco, 2004). Hill-climbing is a well-known heuristic applied in several solutions of admission control. These solutions have the advantage to be simple to implement; however, they provide a best-effort behavior without guarantees on the quality-of-service and SLA of the Internet services.

Other approaches tend to provide strict guarantees on the quality-of-service, and are usually based on analytic models to characterize the system and control it. For instance, there are linear models and nonlinear models (Diao, 2002; Parekh, 2002; Tipper, 1990; Wang, 1996), queuing theory-based models or control theory-based models (Parekh, 2002; Diao, 2002; Malrait, 2009), models for central systems or for distributed services (Bouchenak, 2006; Sivasubramanian, 2006; Urgaonkar, 2007), used for providing guaranties on a unique QoS criterion or for combining multiple criteria (Chase, 2001; Menascé, EC 2001), applying a unique or multiple control mechanisms, i.e. actuators, (Diao, 2002; Milan-Franco, 2004).

Other approaches control Internet services by provisioning/unprovisioning servers to the service. Autonomic provisioning of database servers is presented in (Chen, 2006), and server provisioning in multi-tier systems is described in (Bouchenak, 2006). While these systems are based on heuristics, other approaches tend to better characterize multi-tier applications through analytic modeling for provisioning multi-tier systems (Villela, 2007; Urgaonkar, 2007). However, these approaches are restricted to performance management and do not take into account service availability objectives. Furthermore, they require extensive model calibration with appropriate parameter values; and this calibration is tied to a given workload mix and must be changed each time the workload mix changes, which is not easily detectable.

In summary, MoKa differs from the other approaches in many aspects: (i) it takes into account and combines service performance and service availability SLA objectives, (ii) it combines admission control with server provisioning for a better usage of resources and service cost minimization, and (iii) it automatically handles both workload amount and workload mix variations without requiring manual recalibration of MoKa for an adaptive control of Internet services.

CONCLUSION

This chapter presented MoKa, a system for adaptive control of Internet services to guarantee performance and availability objectives and to minimize cost. The contribution of MoKa is multifold. First, a utility function is defined to quantify the performance, availability and cost of distributed Internet services. Second, a utility-aware capacity planning method is developed; given SLA performance *and* availability constraints, it calculates a configuration of the Internet service that guarantees the constraints while minimizing the cost of the service. Third, a queuing theory-based analytic model of multi-tier Internet services is proposed; the model accurately predicts service performance, availability and cost, and is used as a basis of the capacity planning. Finally, an adaptive control of online Internet services is proposed in the form of a feedback control loop that automatically detects workload mix and workload amount variation, and reconfigures the service with its optimal configuration.

The proposed model, capacity planning and control methods are implemented and applied to an online bookstore. The experiments show that the Internet service successfully self-adapts to both workload mix *and* workload amount variations, and present significant benefits in terms of service performance and availability, with a saving of resources underlying the Internet service. We hope that such a method will lead to a more principled, less ad-hoc implementations of resource management in Internet services and cloud computing environments. The reader is also suggested to consult Chapter 4 related to utility-aware performance management of composite services.

ACKNOWLEDGEMENTS

Experiments presented here were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (Grid'5000, 2010).

REFERENCES

M. Avriel. *Nonlinear Programming: Analysis and Methods*. Dover Publishing. 2003.

G. E. P. Box, A. Luceno, et M. Del Carmen Paniagua-Quinones. *Statistical Control by Monitoring and Adjustment*. Broché, May 2009.

M. Brown. *Optimizing Apache Server Performance*. Retrieved May 20, 2010, from http://www.serverwatch.com/tutorials/article.php/3436911

J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing Energy and Server Resources in Hosting Centers. *The 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, New York, NY, USA, 2001.

X. Chen, H. Chen, and P. Mohapatra. Aces : An Efficient Admission Control Scheme for QoS-aware Web Servers. *Computer Communications*, 26(14), Mar. 2003.

J. Chen, G. Soundararajan, and C. Amza. Autonomic Provisioning of Backend Databases in Dynamic Content Web Servers. *The 3rd IEEE International Conference on Autonomic Computing (ICAC 2006)*, Dublin, Ireland, Jun. 2006.

Y. Diao, N. Gandhi, J. Hellerstein, S. Parekh, and D. Tilbury. Using MIMO Feedback Control to Enforce Policies for Interrelated Metrics with Application to the Apache Web Server. *Network Operations and Management Symposium (NOMS)*, April 2002.

Y. Diao, J. L. Hellerstein, S. Parekh, H. Shaikh, and M. Surendra. Controlling Quality of Service in Multi-Tier Web Applications. *The 26th International Conference on Distributed Computing Systems (ICDCS 2006)*, Lisbon, Portugal, Jul. 2006.

S. Elnikety, J. Tracey, E. Nahum, and W. Zwaenepoel. A method for transparent admission control and request scheduling in e-commerce web sites. *The 13th International conference on World Wide Web (WWW 2004)*, New York, NY, USA, 2004.

P. Ferguson, G. Huston. *Quality of Service: Delivering QoS on the Internet and in Corporate Networks*. John Wiley & Sons, 1998.

Grid'5000. *Grid'5000*. Retrieved May 20, 2010, from http://www.grid5000.fr/

J. Guitart, J. Torres, and E. Ayguadé. A Survey on Performance Management for Internet Applications. *Concurrency Computation – Practice & Experience*, 22(1), 2010.

H.-U. Heiss and R. Wagner. Adaptive Load Control in Transaction Processing Systems. *The 17th International Conference on Very Large Data Bases (VLDB 1991)*, Barcelona, Spain, Sep. 1991.

C. Loosley, F. Douglas, and A. Mimo. *High-Performance Client/Server*. John Wiley & Sons, November 1997.

L. Malrait, S. Bouchenak, and N. Marchand. Fluid Modeling and Control for Server System Performance and Availability. *The 39th Annual IEEE/IFIP Conference on Dependable Systems and Networks (DSN 2009)*, Jun. 2009.

L. Malrait, S. Bouchenak, and N. Marchand. Experience with ConSer: A System for Server Control Through Fluid Modeling. *IEEE Transactions on Computers*, 2010.

E. Marcus and H. Stern. *Blueprints for High Availability*. Wiley, Sep. 2003.

D. A. Menascé and V. A. F. Almeida. *Capacity Planning for Web Services: Metrics, Models, and Methods*. Prentice Hall, 2001.

D. A. Menascé, D. Barbara, and R. Dodge. Preserving QoS of E-Commerce Sites Through Self-Tuning: A Performance Model Approach. *The ACM Conference on Electronic Commerce (EC'01)*, Tampa, FL, Oct. 2001.

J. Milan-Franco, R. Jimenez-Peris, M. Patino-Martinez, and B. Kemme. Adaptive Middleware for Data Replication. *The 5th ACM/IFIP/USENIX international conference on Middleware (Middleware 2004)*, New York, NY, USA, 2004.

Microsoft. *Optimizing Database Performance*. Retrieved May 20, 2010, from http://msdn.microsoft.com/enus/library/aa273605(SQL.80).aspx

S. S. Parekh, N. Gandhi, J. L. Hellerstein, D. M. Tilbury, T. S. Jayram, and J. P. Bigus. Using Control Theory to Achieve Service Level Objectives in Performance Management. *Real-Time Systems*, 23(1-2), 2002.

M. Reiser and S. S. Lavenberg. Mean-Value Analysis of Closed Multi-Chain Queuing Networks. *Journal of the ACM*, 27(2), pp. 313-322, 1980.

S. Sivasubramanian, G. Pierre, M. van Steen, and S. Bhulai. SLA-Driven Resource Provisioning of Multi-Tier Internet Applications. Technical Report, Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, 2006.

D. Tipper and M. Sundareshan. Numerical Methods for Modeling Computer Networks Under Nonstationary Conditions. *IEEE Journal on Selected Areas in Communications*, 8(9), Dec. 1990.

TPC-W. *TPC-W: a transactional web e-Commerce benchmark*. Retrieved May 20, 2010, from http://www.tpc.org/tpcw/

B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. Analytic Modeling of Multi-Tier Internet Applications. *ACM Transactions on theWeb (ACM TWEB)*, 1(1), 2007.

D. Villela, P. Pradhan, and D. Rubenstein. Provisioning Servers in the Application Tier for E-Commerce Systems. *ACM Transactions Interet Technolies*, 7(1), 2007.

W.-P. Wang, D. Tipper, and S. Banerjee. A Simple Approximation for Modeling Nonstationary Queues. The 15th Annual Joint Conference of the IEEE Computer and Communications Societies, Networking the Next Generation (IEEE INFOCOM' 96), San Francisco, CA, USA, Mar. 1996.

Q. Zhang, L. Cherkasova, and N. Mi. A Regression-Based Analytic Model for Capacity Planning of Multi-Tier Applications. *Journal of Cluster Computing*, 11(3), 2008.