Experimental Evaluation of N-tier Systems: Observation and Analysis of Multi-Bottlenecks

Simon Malkowski*, Markus Hedwig†, and Calton Pu*
*Center for Experimental Research in Computer Systems
Georgia Institute of Technology, Atlanta, GA 30332-0765, USA
{simon.malkowski, calton}@cc.gatech.edu
†Chair of Information Systems Research
Albert-Ludwigs-University of Freiburg, 79098 Freiburg, Germany
markus.hedwig@is.uni-freiburg.de

Abstract

In many areas such as e-commerce, mission-critical N-tier applications have grown increasingly complex. They are characterized by non-stationary workloads (e.g., peak load several times the sustained load) and complex dependencies among the component servers. We have studied N-tier applications through a large number of experiments using the RUBiS and RUBBoS benchmarks. We apply statistical methods such as kernel density estimation, adaptive filtering, and change detection through multiple-model hypothesis tests to analyze more than 200GB of recorded data. Beyond the usual single-bottlenecks, we have observed more intricate bottleneck phenomena. For instance, in several configurations all system components show average resource utilization significantly below saturation, but overall throughput is limited despite addition of more resources. More concretely, our analysis shows experimental evidence of multi-bottleneck cases with low average resource utilization where several resources saturate alternatively, indicating a clear lack of independence in their utilization. Our data corroborates the increasing awareness of the need for more sophisticated analytical performance models to describe N-tier applications that do not rely on independent resource utilization assumptions. We also present a preliminary taxonomy of multi-bottlenecks found in our experimentally observed data.

1. Introduction

In modern data centers, enterprise-class N-tier systems with web servers, application servers, and database servers are growing in economic importance, infrastructure footprint, and application complexity. Traditional performance analysis methods are challenged by this growth due to bottleneck phenomena that so far have been considered rare and unusual. In this paper, we show experimentally such "unusual" phenomena and how to detect and analyze them. Our data from system configurations reveal bottleneck cases with different kinds of partially saturated resources, which can be grouped according to their resource usage dependence and saturation frequency into *oscillatory* bottlenecks, *concurrent* bottlenecks, and *simultaneous* bottlenecks. In order to distinguish these cases from single-bottlenecks, which have traditionally been recognized as predominant saturation phenomenon, we use the umbrella-term of *multi-bottlenecks*.

Multi-bottlenecks have been previously studied in system theory [1]–[3] and system analysis [4], [5]. However, classical computer performance analysis [6], [7], which started with stable workloads on mainframes, is traditionally restricted to detecting single-bottlenecks. Typical assumptions of analytical models (e.g., mean value analysis in queuing theory) are the independence of arriving tasks and service times. Partially motivated by the growth of N-tier systems, some recent performance models have moved beyond these assumptions. Consequently, the importance of bursty workload conditions [8], [9] and non-stationary workload in general [10] have been recognized. Nevertheless, average utilization values remain the method of choice in top-down N-tier system analysis [11]–[14]. When mentioned [9], [11], phenomena such as multi-bottlenecks in N-tier applications have been described as "challenging case".

In this paper we analyze experimental results with multi-bottlenecks. We show that these bottlenecks may happen while system components show average resource utilization significantly below saturation. Figure 1 exemplifies this case by contrasting the throughput and response time of a saturated RUBBoS experiment with the corresponding candidate-bottleneck resources. While there is an obvious correlation between system throughput and database CPU utilization, the question why this correlated resource does not seem to saturate remains unanswered. Because none of the resources show full resource utilization levels, standard approaches to bottleneck detection in N-tier systems (e.g., [15]) are not able to diagnose them. However, instead of attributing the obvious performance limitation to some hidden (i.e., unmonitored) resource, we show—using our

methodology—that the observed throughput limitation is due to an oscillatory bottleneck in the database where CPU and disk saturate alternatively (see Section 6).

The main contribution of this paper is threefold. First, our experimental evaluation of N-tier application benchmarks (i.e., RUBiS and RUBBoS) documents the presence of presumed unusual multi-bottlenecks. Second, we introduce a simple classification of multi-bottlenecks that enables the detection of the observed phenomena. Third, using statistical methods such as kernel density estimation, adaptive filtering, and change detection through multiple-model hypothesis tests, we show how to detect multi-bottlenecks with an efficient top-down approach, even if average resource utilization is significantly below saturation.

The remainder of this paper is structured as follows. In Section 2 we introduce the simple classification of multi-bottlenecks that is used in our analysis. In Section 3 we outline the necessary methods for statistical interpretation of the measurement data. Section 4 presents the experimental setup and infrastructure. Section 5 shows a scenario with seven concurrent bottlenecks. In Section 6 we present two different oscillatory bottleneck cases. Related work is summarized in Section 7, and Section 8 concludes the paper.

2. Simple Classification of Multi-bottlenecks

The common understanding of a *system bottleneck* (or bottleneck for short) can intuitively be derived from its literal meaning as the key limiting factor for achieving higher system throughput. Hence, an improvement to the throughput of the bottleneck resource results in the highest possible system throughput improvement [6], [16]. Classical queuing theory defines the set of bottlenecks in a queuing system as follows. B is the set of all resources i that reach full utilization U_i when N, the number of jobs in the system, tends to infinity under stable class-mix conditions [1].

$$B = \left\{ i \,\middle|\, \lim_{N \to \infty} U_i(N) = 1 \right\} \tag{1}$$

Due to their simplicity and intuitiveness, definitions similar to Equation (1) have usually provided the foundation for reasoning about bottleneck behavior in computer system performance analysis. But despite their popularity, such formulations are based on assumptions that do not necessarily hold in practice. In other words, we show experimentally that Equation (1) does not allow correct identification of bottlenecks in the case of empirical N-tier application monitoring data. Our data suggest that resources in N-tier applications cannot generally be assumed to exhibit independent utilization. In fact, bottlenecks may be comprised of more than one physical resource. Similarly, the dimension of time has to be taken into account in N-tier systems, which may be subject to very strong workload fluctuations. Therefore, the assumption of stable class-mix conditions has to be relaxed as well. Finally, an infinite number of users is not technically feasible, hence, the assumption of full resource utilization has to be adapted to actual monitoring conditions.

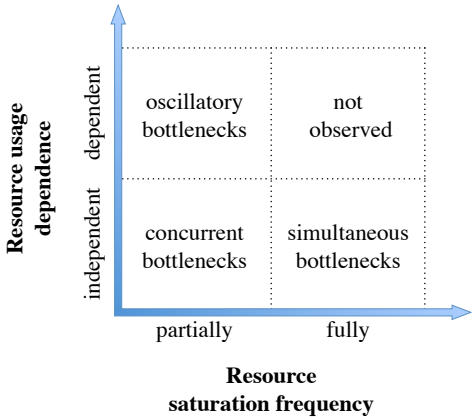
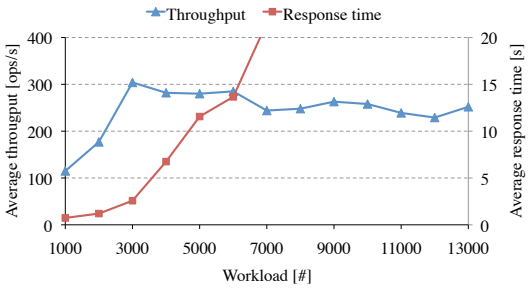


Figure 2. Simple multi-bottleneck classification.

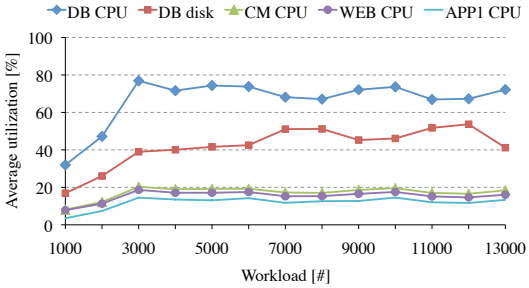
Since our data show that the aforementioned assumptions may be too rigid, the need arises to classify the observed phenomena in an alternate way. But before doing so through a formal set of definitions in the later part of this section, we first establish the key differences between our taxonomy and common approaches to N-tier bottleneck detection. Because of queuing theory, the term bottleneck is often used synonymously for single-bottleneck. In a single-bottleneck cases, the saturated resource typically exhibits a near-linearly load-dependent average resource utilization that saturates at a value of one past a certain workload. Consequently, such bottlenecks are straight-forward to detect. However, the characteristics of bottlenecks may change significantly if more than one bottleneck resource is present in the system, which is the case for many real N-tier applications with heterogeneous workloads. Therefore, we distinguish the phenomena discussed in this paper from single-bottlenecks by using the umbrella-term *multi-bottlenecks*.

Because system resources may be causally dependent in their individual usage patterns, multi-bottlenecks introduce the classification dimension of *resource usage dependence*. Additionally, greater care has to be taken in classifying bottleneck resources according to their *resource saturation frequency*. We distinguish between resources that saturate for the entire observation period (i.e., fully saturated) and resources that saturate for only certain parts of the period (i.e., partially saturated). Note that previous efforts in this area have typically omitted the notions of dependence and saturation frequency in their analysis (e.g., [15]).

Figure 2 summarizes the classification that forms the basis of our multi-bottleneck detection. The x-axis and y-axis represent resource saturation frequency and resource usage dependence, respectively. During our extensive empirical evaluation, we have only observed three of the four possible



(a) Average throughput and response time.



(b) Resource utilization for candidate-bottleneck resources.

Figure 1. 1/2/1/1L RUBBoS experiment with read/write workload.

combinatoric cases, which suggests that if resources exhibit full saturation, their resource utilization is not dependent on any other saturated resource. Under this assumption, multiple fully saturated resources, imply resource usage independence, which is classified as *simultaneous* bottleneck case. A well-known example of systems with resources that are resource usage independent are embarrassingly parallel applications. *Concurrent* bottlenecks happen if multiple resources saturate independently of each other, but only for a fraction of the observation period. As the load increases, a concurrent saturation pattern may transform into simultaneous saturation. The most notable characteristic of usage independent resources is that each resource's saturation may be evaluated separately. On the contrary, the detection of *oscillatory* bottlenecks is more challenging because multiple resources form a combined bottleneck, which can only be analyzed and resolved in union. Oscillatory bottlenecks consist of partially saturated resources and are caused by resource usage dependencies (e.g., load-balancing policies) that are an inherent characteristic of complex N-tier systems. Such dependencies result in a usage alternation between the saturated resources (i.e., there is no overlapped saturation), which is observable as interleaved saturation patterns. The latter may cause particularly low average saturation for every singel resource, and observable saturation frequency can become difficult to interpret. Therefore, it is inevitable to take resource usage dependence into account in multi-bottleneck detection. We find that once resources are grouped according to their usage dependence, it is possible to compute their combined frequency and derive a representative measure for the bottleneck magnitude.

Given these findings, the following definitions are sufficient for a simple multi-bottleneck classification. The resulting generic schema may be parameterized in a concrete setting, such as presented in this paper, and utilized to detect bottlenecks in N-tier application monitoring data. In the following, we augment the definitions with our concrete parameter choices, leaving sensitivity analysis and parameterization comparisons as an interesting topic for future research.

Critical saturation. *System resource $i \in I$, where I is a set of system resources, is critically saturated for an observation interval t of length $\lambda > 0$ iff its utilization exceeded a saturation threshold $\alpha \in [0, 1]$ for this interval, whereby α is referred to as the critical saturation threshold.*

We found that parameterizing tuple (λ, α) with 1 second and 95%, respectively, leverages workload fluctuation, technical feasibility, and standard statistical significance well in our data. Set I consists of all monitored resources, which are all CPUs, disks, and network links in our testbed.

Resource saturation frequency. *The resource saturation frequency f_R^i (or frequency for short) of resource $i \in I$ is the number of intervals with critical saturation divided by the total number of intervals in observation period Λ.*

We adopted the system observation period Λ equal to the experiment runtime of 8 minutes. Therefore, the resource saturation frequency f_R^i indicates how often resource demand exceeded the capacity of resource i during runtime.

Fully saturated resource. *A fully saturated resource $i \in I$ has maximal saturation frequency; i.e., $f_R^i \approx 1$.*

We found that $f_R^i \geq 0.95$ constitutes a convenient detection rule for full saturation of resource i.

Partially saturated resource. *A partially saturated resource $i \in I$ is not fully saturated, but has a non-negligible saturation frequency; i.e., $f_R^i \in (0, 1)$.*

We found that $0.01 < f_R^i < 0.95$ constitutes a convenient detection rule for partial saturation of resource i.

Bottleneck resource. *A bottleneck resource $b \in I$ is either a fully saturated resource or a partially saturated resource.*

Resource usage dependence. *A bottleneck resource $b_j \in I$ is resource usage dependent (or dependent for short), if there exists another bottleneck resource $b_k \in I$ such that their binary critical saturation states $B_j^t \in \{0, 1\}$ and*

$B_k^t \in \{0,1\}$ *have a mutually exclusive relationship for any given interval* t*; i.e.,* $Prob(B_j^t = 1 \wedge B_k^t = 1) \leq \varepsilon$ *with* $0 < \varepsilon \ll 1$ *for all* t *in* Λ.

We found that this definition implies an intuitive rule in a top-down approach to bottleneck detection. Because of the mutual exclusion property, dependent bottleneck resources do not exhibit overlap of critical saturation intervals among each other. Even at very high workloads, such overlap has a statistically insignificant probability; e.g., $Prob(\text{overlap}) \leq 5\%$.

Resource usage independence. *A bottleneck resource* $b_j \in I$ *is resource usage independent (or independent for short), if there exists no other bottleneck resource* $b_k \in I$ *such that their binary critical saturation states* $B_j^t \in \{0,1\}$ *and* $B_k^t \in \{0,1\}$ *have a mutually exclusive relationship for any given interval* t*; i.e.,* $Prob(B_j^t = 1 \wedge B_k^t = 1) > \varepsilon$ *with* $0 < \varepsilon \ll 1$ *for all* t *in* Λ.

In contrast to dependent bottleneck resources, independent bottleneck resources show a clearly increasing overlap of critical saturation intervals with growing workload; e.g., $Prob(\text{overlap}) > 5\%$.

Single-bottleneck. *A single-bottleneck* $\beta \in I$ *is either a fully saturated bottleneck resource or partially saturated bottleneck resource iff there exists only one bottleneck resource in the system.*

Oscillatory bottleneck. *An oscillatory bottleneck* $\beta \subseteq I$ *is a set of partially saturated, resource usage dependent bottleneck resources iff there exist more than one bottleneck resource in the system.*

In this paper, we focus on the maximal set of resources that form an oscillatory bottleneck. The study of subsets of resources in such oscillations is a subject of future research.

Concurrent bottleneck. *A concurrent bottleneck* $\beta \in I$ *is a partially saturated, resource usage independent bottleneck resource iff there exist more than one bottleneck resource in the system.*

Simultaneous bottleneck. *A simultaneous bottleneck* $\beta \in I$ *is a fully saturated, resource usage independent bottleneck resource iff there exist more than one bottleneck resource in the system.*

Multi-bottleneck. *A multi-bottleneck is either an oscillatory bottleneck, a concurrent bottleneck, or a simultaneous bottleneck.*

Bottleneck. *A bottleneck is either a single-bottleneck or a multi-bottleneck.*

Bottleneck saturation frequency. *The bottleneck saturation frequency* f_B^β *(or frequency for short) of bottleneck* β *is the number of intervals where at least one of the bottleneck resources in* β *is critically saturated divided by the total number of intervals in observation period* Λ.

The bottleneck saturation frequency indicates how often resource demand exceeded the capacity of any of the resources that are analyzed in union due to their strong usage dependence. Hence, this measure allows the assessment of how often a system under examination suffered of a particular performance limiting phenomenon.

Primary bottleneck. *The primary system bottleneck is the bottleneck with the highest bottleneck saturation frequency.*

Note that there can be more than one primary bottleneck and that any single-bottleneck or simultaneous bottleneck is also a primary bottleneck by defintion.

3. Statistical Data Interpretation

In the following we provide a brief overview of the most important statistical concepts that formed the basis of our data analysis. Since real systems are typically subject to variable request characteristics, it is necessary to analyze the distributions of resource utilization values to infer actual saturation characteristics. However, histograms are often poor estimates of unknown density functions [17], therefore, we chose estimation through kernel densities [18], instead. Given a finite sample $X_1, \ldots, X_2$ from a univariate distribution, the unknown density function g can be estimated from the observed data using kernel regression. Although literature offers various kernel functions, the symmetric Gaussian density is a popular choice in density estimation. Given a value x, the kernel function c, and a smoothness parameter h (i.e., "bandwidth") the density estimator $\bar{g}_c$ takes the following form.

$$\bar{g}_c = \frac{1}{n} \sum_{t=1}^{n} \frac{k}{h} \left(\frac{x - X_t}{h} \right) \tag{2}$$

There are different approaches to calculating bandwidth h. We choose a common strategy (i.e., AMISE estimation [18]), which simplifies to estimation with the sample standard deviation estimate $\bar{\sigma}$.

$$\bar{h}_0 = \frac{\bar{\sigma} \sqrt[3]{4}}{\sqrt[3]{n}} \tag{3}$$

An example of a resulting three-dimensional density graph is shown in Figure 4(a).

In the analysis of multi-bottlenecks, it is further necessary to asses the dependence relationship of resources in excess of their relative probabilities. In order to reduce the complexity of this analysis in an efficient top-down approach, it is possible to segment the monitoring data into piecewise constant functions. This method provides intuitive aggregation and reduces the data size significantly compared to bottom-up analysis. Such a segmenting problem is also known

(a) Software setup

Classification	Software
Web server	Apache 2.0.54
Application server	Ap. Tomcat 5.5.17 JOnAS 4.6.6
Cluster middleware	C-JDBC 2.0.2
Database server	MySQL 5.0.51a
Operating system	Redhat FC4 Kernel 2.6.12
System monitor	Systat 7.0.2

(b) Hardware node setup

Type	Components		
Normal	Processor	Xeon 3GHz	64-bit
	Memory	2GB	
	Network	6 x 1Gbps	
	Disk	2 x 146GB	10,000rpm
Low-cost	Processor	PIII 600Mhz	32-bit
	Memory	256MB	
	Network	5 x 100Mbps	
	Disk	13GB	7,200rpm

(c) Sample topology (1/2/1/2L)

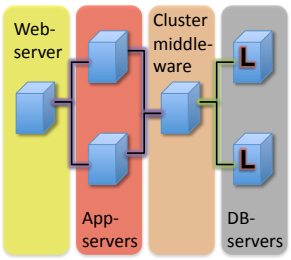


Table 1. Details of the experimental setup on the Emulab cluster.

as adaptive filtering problem that detects abrupt changes in streams of noisy data and has been previously studied in signal processing. Therefore, we perform an automated multiple-model hypothesis test on a set of signal estimation models, which are based on different assumptions each, to divide the measured time series into segments according to piecewise constant mean and variance models [19]. Given the time index t, the noise measure e, the parameter vector θ, and the noise variance γ, each signal y can be expressed as follows.

$$y(t) = \theta(t) + e(t) \tag{4}$$

$$Ee(t)^2 = \gamma(t) \tag{5}$$

In our case $y(t)$ in (4) is identical to X_t in (2). The set of final change times is determined algorithmically [19]. This approach is directly interpretable as characterization of each stable interval of the piecewise constant resource utilization functions. We characterize each utilization interval with respect to being statistically distinguishable from critical saturation (see Section 2). Such a classification is performed for each bottleneck resource metric, and the results can be summarized in a simple graph for the entire system (e.g., Figure 4(d)).

4. Experimental Setup

Among N-tier application benchmarks, RUBBoS and RUBiS have been used in numerous research efforts due to their real production system significance. In our experiments, the run consist of an 8-minute ramp-up, a 12-minute run period, and a 30-second ramp-down. Performance measurements (e.g., CPU or network utilization) are taken during the run using Linux account logging utilities (i.e., Sysstat) with one-second intervals.

RUBBoS [20] is an N-tier e-commerce system modeled on bulletin board news sites similar to Slashdot. The benchmark can be implemented as 3-tier (web server, application server, and database server) or 4-tier (with the addition of cluster middleware such as C-JDBC) systems. The benchmark places high load on the database tier. The workload consists of 24 different interactions (involving all tiers) such as register user, view story, and post comments. The benchmark includes two kinds of workloads: browse-only and read/write interaction mixes.

RUBiS [21] is an N-tier system benchmark modeled on online auction sites such as eBay. The benchmark can be implemented as a 3-tier, 4-tier, or 5-tier system. We have chosen a configuration consisting of web server, web container, EJB container, cluster middleware, and database server. Web and EJB containers are deployed together on the same physical nodes. The benchmark usually places a high load on application servers. The workload consists of 26 interactions such as register user, sell item, and place bid. RUBiS includes two kinds of workloads: browse-only and read/write interaction mixes.

The experiments used in this paper were run in the Emulab testbed [22] with various types of servers. Table 1(b) contains a summary of the hardware used in our experiments. Normal and low-cost nodes were connected over 1,000 Mbps and 100 Mbps links, respectively. The experiments were carried out by allocating a dedicated physical node to each server. In the initial setting all components were normal nodes. As an alternative, database servers were also hosted on low-cost machines. We use a four-digit notation #W/#A/#C/#D to denote the number of web servers, application servers, cluster middleware nodes, and database servers. The server node type is either *normal* or *low-cost* ("L"). If a specification is omitted, it can be assumed that the default node type (i.e., normal) has been used. A sample topology of an experiment with one web server, two application servers, one cluster middleware node, and two low-cost database servers (i.e., 1/2/1/2L) is shown in Table 1(c).

The presented dataset is part of an ongoing effort, for which we have run a very high number of experiments over a wide range of configurations and workloads. A typical RUBiS or RUBBoS experimentation cycle requires thousands of lines of code that need to be managed for each experiment. The experimental data output are system metric data points (i.e., network, disk, and CPU utilization)

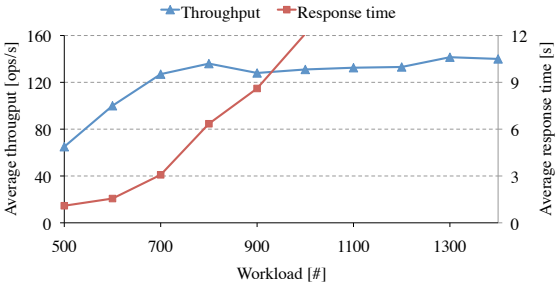


Figure 3. Average throughput and response time of a 1/6/1/1L RUBiS experiment with browse-only workload an throttled database network bandwidth (2 Mbps).

in addition to higher-level monitoring data (e.g., response times and throughput). Although the scripts contain a high degree of similarity, the differences among them are subtle and important due to the dependencies among the varying parameters. Maintaining these scripts by hand is a notoriously expensive and error-prone process.

To enable experimentation at this scale, we employed an experimental infrastructure created for the Elba project [23] to automate system configuration management, particularly in the context of N-tier system staging. The Elba approach [12] divides each automated staging iteration into steps such as converting policies into resource assignments [24], automated code generation [25], benchmark execution, and analysis of results.

5. Concurrent Bottlenecks

In this section we show a RUBiS benchmark experiment that exhibits concurrent saturation of application server CPUs and the database network link. More concretely, these data are obtained with a 1/6/1/1L RUBiS configuration using the browse-only interaction mix. Traffic shaping (implemented by the packet scheduler modules in the Linux NET3 kernel) has been used to throttle the network bandwidth between the application servers and the database server. Packets are added to a scheduler queue tree and shaped corresponding to a 2Mbps bandwidth limit using the drop-tail queuing discipline. The system workload ranges between 500 and 1,400 users in steps of 100.

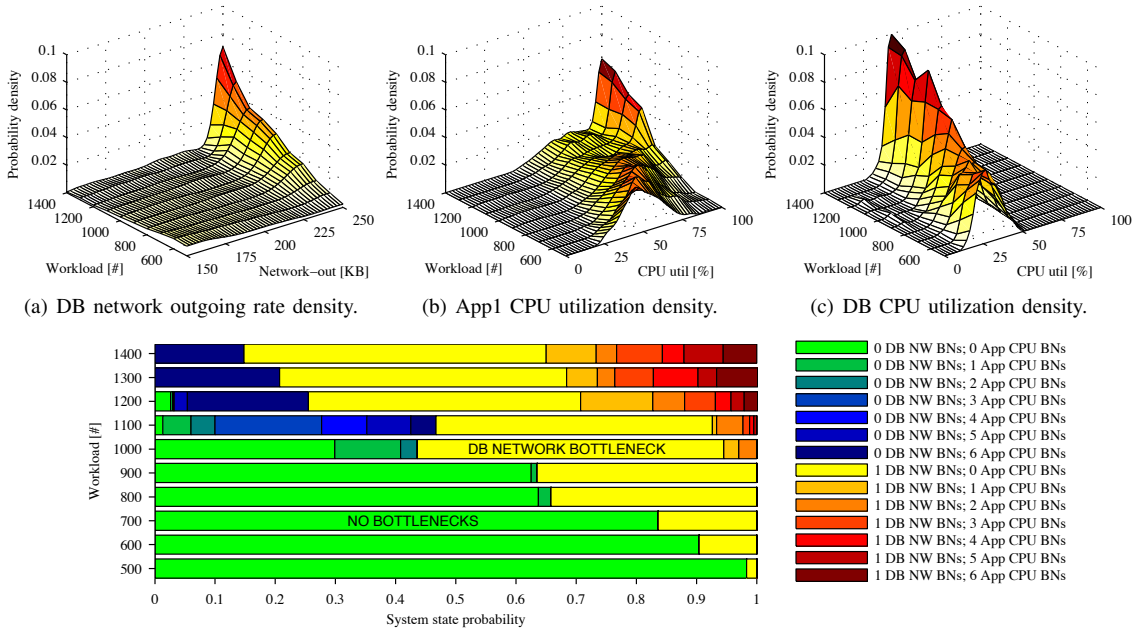
Figure 3 summarizes the overall system performance, which is observable in the clients. The throughput increases near-linearly up to the performance knee at around 800 concurrent users where the rate saturates at around 140 interactions per second. Similarly, the average response time exceeds six seconds past a workload of 800 users. Evidently, there is at least one bottleneck in the system that needs to be identified through a detailed analysis.

Figure 4 shows the summary of the bottleneck analysis for this scenario. The bottlenecks in the system are best illustrated by studying their resource utilization as a function of workload. The three density graphs in Figures 4(a), 4(b), and 4(c) show the increasing demand peak (z-axis) on the resource utilization (x-axis) as the workload grew from 500 to 1,400 users (y-axis). More specifically, the three density graphs contrast the changes in the resource consumption profiles of bottleneck resources (i.e., application server CPUs and database network) and non-saturated resources (i.e., database CPU). Note that the application server CPU graph is representative of all application servers. While the first graph (Figure 4(a)) shows the continuous saturation of the network bandwidth at 250 KBps (consistent with the link shaping parameter of 2 Mbps), the application server CPU utilization (Figure 4(b)) transitions from a normal to a multimodal saturated curve shape with a dominating mode at the upper capacity limit. This multimodality, which causes average utilization values to be misleadingly low, is the result of an unstable resource consumption profile due to non-stationary request sequences. For higher workloads the application server CPUs vary between being critically utilized and being underutilized. Overall, the seven saturated system resources limit the number of tasks going into the database server. Therefore, its CPU remains unsaturated, and its utilization density retains its normal shape despite growing intensity throughout for the entire workload span (Figure 4(c)).

In order to precisely characterize whether the uncovered bottleneck resources are concurrent or oscillatory bottlenecks, we estimated the saturation frequencies of each resource and visualized them in Figure 4(d). Although there are fourteen different combinations, the concurrent character of the seven bottlenecks is apparent. There is a clear linear decrease in probability of non-overlapping bottleneck states, and a corresponding linear increase in overlapping bottleneck states as the workload grew to 1,400 users. For workloads higher than 1,000, the probability of states without critical saturation of at least one resource is statistically insignificant. Both, the frequency of the network bottleneck and the CPU bottlenecks are solely load-dependent, which implies saturation-independence among the resources. Because non of the resources is fully saturated, we can conclude that the system suffers of seven concurrent bottlenecks in the six application sever CPUs and the database network link.

6. Oscillatory Bottlenecks

A common assumption in computer system performance analysis is that low average resource utilization implies absence of saturation. However, in the following we show that a standard RUBBoS read/write mix workload may causes request sequences in the database that result in alternating partial resource saturation with particularly low average resource utilization. While Section 6.1 details a



(a) DB network outgoing rate density. (b) App1 CPU utilization density. (c) DB CPU utilization density.

(d) Probability distribution (i.e., frequency of resource saturation states) among the fourteen different bottleneck resource saturation states.

Figure 4. Detailed bottleneck analysis of a 1/6/1/1L RUBiS experiment with browse-only workload an throttled database network bandwidth (2 Mbps).

scenario with an oscillatory bottleneck within a single node, Section 6.2 shows an example of an oscillatory bottleneck with eight resources distributed in the database tier.

6.1. Within-node Dependences

As previously shown in Figure 1(a), the 1/2/1/1L RUBBoS scenario under read/write workload exhibits a clear overload pattern past workloads of 3,000 user. Nevertheless, the average utilization values for candidate bottleneck resources (Figure 1(b)) remain far from saturation. Although the comparison of these two figures reveals a strong correlation between system throughput and database CPU utilization, the CPU utilization seems to saturate at an average of less than 80 percent.

In order to diagnose the system bottlenecks, we turn to Figure 5. The analysis of Figures 5(a) and 5(b) yields the explanation of the previously observed low utilization values. In fact, the two examined densities (database CPU and disk) are both bimodal with two distinct concentration sectors for the entire workload span. Both utilization metrics seem to vary between low and high values during the experiment time. A sample examination of the two time series for a short interval of runtime for 12,000 users (Figure 5(c)) seems to support this impression. However, it is important to note that such an examination is not scalable and requires human intuition to derive insights in the actual system behavior. Therefore, we automatically summarize the metric

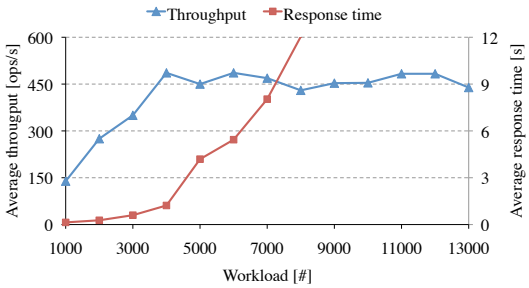
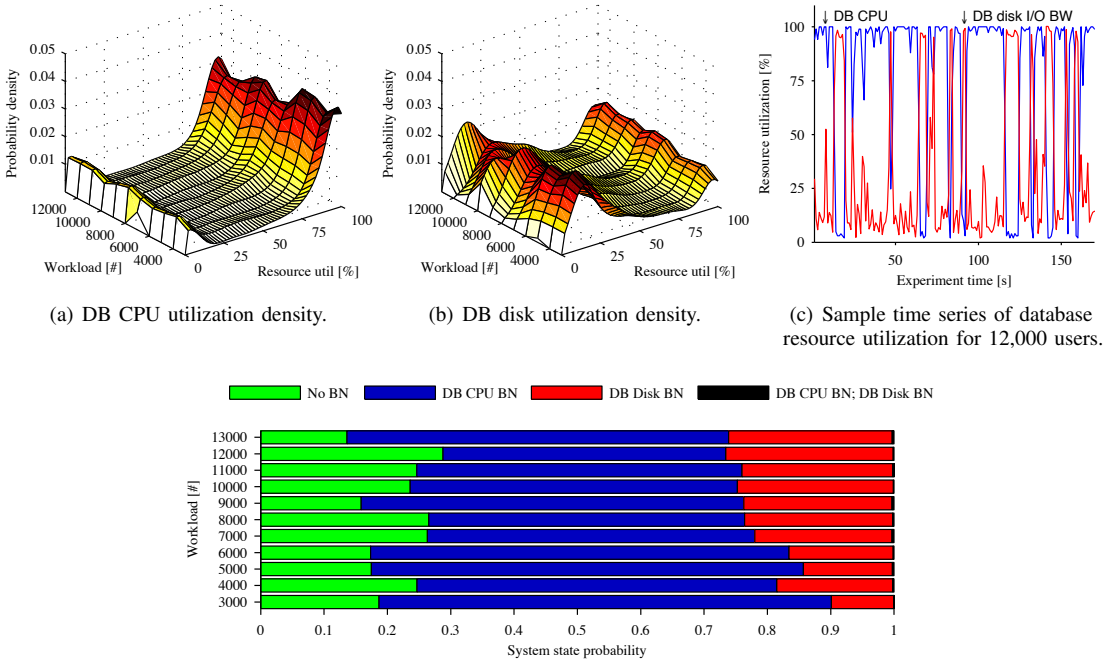


Figure 6. Average throughput and response time of a 1/1/1/8L RUBBoS experiment with read/write workload.

data in Figure 5(d), which reveals two bottleneck resources with four saturation states for the entire workload span. Clearly, overlapping saturation is extremely rare. The ratio of time fractions for the two non-overlapped bottleneck state seems to be relatively stable, which further strengthens the assumption of high dependence between the two bottleneck resources. Consequently, the reason for the system saturation lays in an oscillatory bottleneck with interleaved saturation of database CPU and disk. A thorough log analysis reveals that if many long queries are bottlenecked at the database disk, the CPU remains idle since it is waiting for I/O, and no new request are admitted to the node. This causes the observed exclusive saturation pattern.



(a) DB CPU utilization density.

(b) DB disk utilization density.

(c) Sample time series of database resource utilization for 12,000 users.

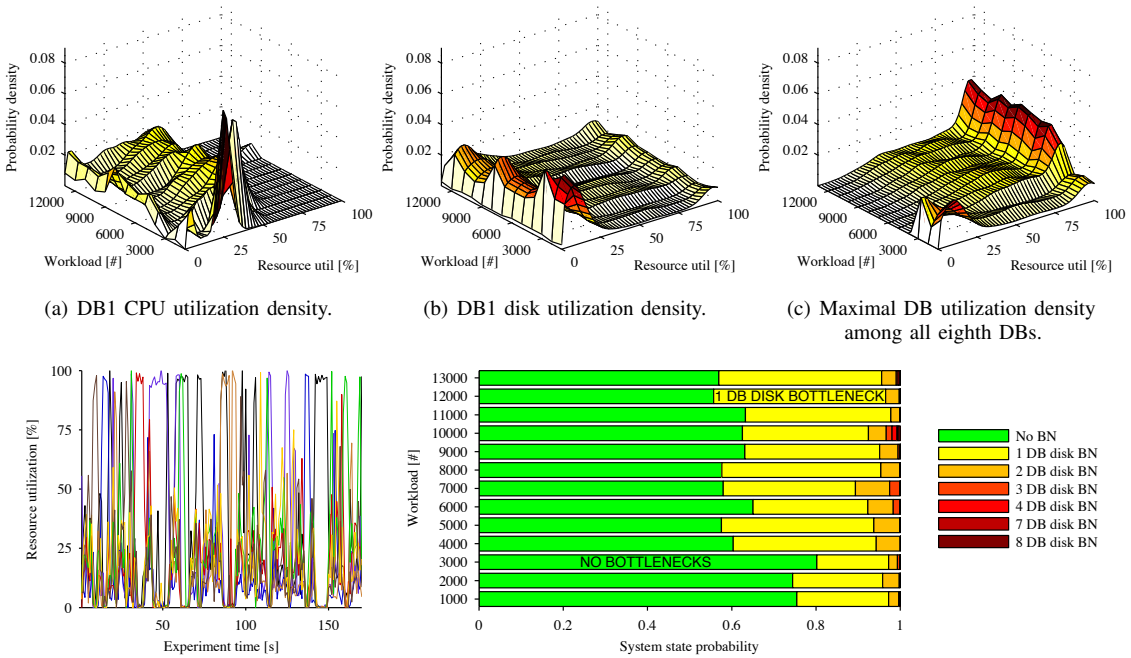
(d) Probability distribution (i.e., frequency of resource saturation states) among the four different bottleneck resource saturation states.

Figure 5. Detailed bottleneck analysis of a 1/2/1/1L RUBBoS experiment with read/write workload.

6.2. Between-Node Dependences

The performance metrics of a 1/1/1/8L RUBBoS deployment under standard read/write workload for 1,000 to 13,000 concurrent user sessions shown in Figure 6 reveal system saturation past a workload of 4,000 users. The analysis, necessary to infer the underlying bottlenecks, is summarized in Figure 7. Figure 7(a) shows that the CPU utilization in the database does not reach critical levels during the experiment time. Consequently, this resource can be disregarded as bottleneck candidate. The inspection of the density graph for the disk utilization in the first database (representative of all eight databases) reveals slightly elevated density values at the high tail of the right-skewed density (see Figure 7(b)). Unlike the previous examples, these values do not seem to explain the overall performance deterioration at first sight. The elevation levels are constant, and their overall probability remains significant but small throughout the entire experiment. In other words, the resource shows a saturation behavior, which is very infrequent during the runtime. This could suggest a strongly oscillating bottleneck with a large number of bottleneck resources. Such an interpretation is further supported by Figure 7(c), which shows the density for the maximal disk utilization among all eight databases. Past a workload of 3,000 users, the maximal value is always higher than 50 percent, and a dominant peak has appeared at the high percentiles. This means that the system exhibits a high utilization in at least one of the database disks at all times for higher workloads.

Nonetheless, the question remains whether the bottleneck resources saturate dependently. While a sample manual evaluation of the resource utilization time series (see Figure 7(d)) does not yield clear results due to the magnitude of the variability, the automated partitioning used to generate Figure 7(e) reveals the oscillatory character of this bottleneck. The probability plots are strongly dominated by two system states (i.e., single database disk bottleneck or no bottleneck at all). There is virtually no overlap between the saturation of the eight resources. The rarely observed overlap can be attributed to the noise that is introduced into the data by each resource and by the stochastic aggregation method. Therefore, we can conclude the the performance limitation in this scenario is caused by an oscillatory bottleneck with eight saturation-dependent, partially saturated database disks. A detailed log analysis shows that because of the workload distributing capabilities of C-JDBC and the relative infrequency of overly long queries, the bottleneck is strongly distributed among the eight resources. The interleaved saturation is caused by the default C-JDBC replication policy, which waits for the completion of all concurrent write-requests before committing. Writes are always immediately sent to all databases (i.e., multi-master replication), thus the entire system is bottlenecked if one database node saturates.



(a) DB1 CPU utilization density.

(b) DB1 disk utilization density.

(c) Maximal DB utilization density among all eighth DBs.

(d) Sample time series of disk utilization in all DBs with 12,000 users.

(e) Probability distribution (i.e., frequency of resource saturation states) among the seven different bottleneck resource saturation states.

Figure 7. Detailed bottleneck analysis of a 1/1/1/8L RUBBoS experiment with read/write workload.

7. Related Work

Traditional performance analysis in computer systems presumes models based on expert knowledge, employs standard statistical methods, and parameterizes them based on a certain experimentation design [6], [7]. Queuing models have been common practice in many research efforts dealing with performance prediction [26], [27]. Although these approaches have been applied very successfully, they suffer from their rigid assumptions when handling all evolution of large applications. The availability of extensive instrumentation data profiles [26] or constant mean inter-arrival times of request [27] do not hold in general since actual parameters vary widely in real applications. The characteristic load non-stationarity in N-tier systems has been exploited for performance prediction by Stewart et al. [10], who also explain how their anomaly detection can be used to invoke a bottleneck detection process such as the one presented in this paper.

Statistically induced models have been recently used to remove human intervention from the loop [28]–[30], and extensive experiments have been conducted to compare different bottleneck detection methodologies [31]. Nevertheless, all these algorithmic approaches explicitly correlate high-level application performance with low-level system behavior. In this paper, we solely use performance degradation patterns as a trigger for our analysis, similarly to real system administrators [32]. In practice, many computer manuals possess a performance tuning section, which typically relies on specialized "rules of thumb". Additionally, commercial tools (e.g., HP Open View or IBM Tivoli) offer the possibility of inspecting an abundant variety of metric data graphically without clear aggregation and analysis frameworks. Some discussion on a two-dimensional bottleneck characterization in RUBiS and RUBBoS has been previously provided by Amza et al. [13]. However, this approach solely relies on manual human diagnosis and only targets stable bottleneck characterization with small-scale experimentation.

In contrast to our work, previous bottleneck detection research in computer systems has built upon an extremely detailed understanding of the systems (e.g., invasively instrumented central system [33]) or an analysis confined to a small resource subset (e.g., network traffic or software configurations [5]). Methods for bottleneck detection and analysis have also been also discussed in literature on simulation in areas such as industrial production [4]. The latter work emphasizes a technique for dealing with shifting bottleneck behavior based on resource utilization. The notion of ordering bottlenecks by severity with the help of resource demand distributions has been introduced by Luthi [3]. He provides proof that approximating distributions with histograms yields a higher precision than conventional methods. Unlike our observation-based work, both these

approaches remain very generic without inference of domain specific phenomena.

8. Conclusion

For mission-critical applications such as e-commerce, N-tier systems have grown in complexity with non-stationary workloads and inter-task dependencies created by requests that are passed between the various servers. These complicating factors create *multi-bottlenecks* such as oscillatory bottlenecks (where inter-task dependencies cause the bottleneck to migrate among several resources) and concurrent bottlenecks (where multiple bottlenecks arise among several resources). Multi-bottlenecks are non-trivial to analyze, since they may escape typical assumptions made in classic performance analysis such as stable workloads and independence among tasks.

In this paper, we describe an experimental study of multi-bottlenecks using a large dataset of N-tier application benchmark data. We used techniques and tools developed for automated system management (e.g., code generation for experimental scripts) to collect more than 200GB of measurement data on the N-tier application benchmarks RUBiS and RUBBoS. Using statistical techniques such as kernel density estimation, we show that multi-bottleneck phenomena arise naturally in sufficiently complex N-tier systems. For instance, in 72.2% of our RUBBoS experiments with low-cost database nodes (e.g., Section 6), we were able to identify oscillatory bottlenecks, and in 83.7% of our RUBiS experiments, we found either concurrent or simultaneous bottlenecks (e.g., Section 5). Furthermore, it is non-trivial to reveal multi-bottlenecks since they often happen when no single-bottleneck is visible (i.e., average resource utilization well below saturation for all resources).

Acknowledgment

This research has been partially funded by National Science Foundation grants ENG/EEC-0335622, CISE/CNS-0646430, CISE/CNS-0716484, AFOSR grant FA9550-06-1-0201, NIH grant U54 RR 024380-01, IBM, Hewlett-Packard, Wipro Technologies, and Georgia Tech Foundation through the John P. Imlay, Jr. Chair endowment. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funding agencies and companies mentioned above.

References

[1] G. Balbo and G. Serazzi, "Asymptotic analysis of multiclass closed queueing networks: multiple bottlenecks," in *Perform. Eval. '97*.

[2] G. Casale and G. Serazzi, "Bottlenecks identification in multiclass queueing networks using convex polytopes," in *MASCOTS '04*.

[3] J. Luthi, "Interval matrices for the bottleneck analysis of queueing network models with histogrambased parameters," in *IPDS '98*.

[4] C. Roser, M. Nakano, et al., "Shifting bottleneck detection," in *WSC '02*.

[5] F. Ricciato, F. Vacirca, et al., "Diagnosis of capacity bottlenecks via passive monitoring in 3g networks: An empirical analysis," *in Comput. Netw. '07*.

[6] R. Jain, *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling.* New York, NY, USA: John Wiley & Sons, Inc., 1991.

[7] D. J. Lilja, *Measuring Computer Performance - A Practitioner's Guide*. New York, NY, USA: Cambridge University Press, 2000.

[8] N. Mi, "Performance impacts of autocorrelated flows in multi-tiered systems," in *Perform. Eval. Rev. '07*.

[9] G. Casale, N. Mi, et al., "How to parameterize models with bursty workloads," in *HotMetrics '08*.

[10] C. Stewart, T. Kelly, et al., "Exploiting nonstationarity for performance prediction," in *SIGOPS Oper. Syst. Rev. '07*.

[11] Q. Zhang, L. Cherkasova, et al., "A regression-based analytic model for dynamic resource provisioning of multi-tier applications," in *ICAC '07*.

[12] C. Pu, A. Sahai, et al., "An observation-based approach to performance characterization of distributed n-tier applications," in *IISWC '07*.

[13] C. Amza, E. Cecchet, et al., "Bottleneck characterization of dynamic web site benchmarks," in *IBM CAS '02*.

[14] E. Cecchet, A. Chanda, et al., "Performance comparison of middleware architectures for generating dynamic web content," in *Middleware '03*.

[15] M. Litoiu, "A performance analysis method for autonomic computing systems," in *ACM Trans. Auton. Adapt. Syst. '07*.

[16] Y. Wang, Q. Zhao, et al., "Bottlenecks in production networks: An overview," in *JSSSE '05*.

[17] C. Alexopoulos, "Statistical analysis of simulation output: state of the art," in *WSC '07*.

[18] B. E. Hanson, "Bandwidth selection for nonparametric distribution estimation," www.ssc.wisc.edu/~bhansen/papers/wp.htm, May 2004.

[19] F. Gustafsson, *Adaptive Filtering and Change Detection*. John Wiley & Sons, Inc., 2001.

[20] "RUBBoS," jmob.objectweb.org/rubbos.html.

[21] "RUBiS," rubis.objectweb.org.

[22] "Emulab - Network Emulation Testbed," www.emulab.net.

[23] "The Elba project," www.cc.gatech.edu/systems/projects/Elba.

[24] A. Sahai, S. Singhal, et al., "Automated generation of resource configurations through policies," in *Policy '04*.

[25] G. Jung, C. Pu, et al., "Mulini: an automated staging framework for qos of distributed multi-tier applications," in *WRASQ '07*.

[26] C. Stewart and K. Shen, "Performance modeling and system management for multi-component online services," in *NSDI'05*.

[27] B. Urgaonkar, G. Pacifici, et al., "An analytical model for multi-tier internet services and its applications," *Perform. Eval. Rev. '05*.

[28] M. K. Aguilera, J. C. Mogul, et al., "Performance debugging for distributed systems of black boxes," in *SOSP '03*.

[29] I. Cohen, M. Goldszmidt, et al., "Correlating instrumentation data to system states: a building block for automated diagnosis and control," in *OSDI'04*.

[30] C. Huang, I. Cohen, et al., "Achieving scable autoamted diagnosis of distributed systems performance problems," HP Labs, Tech. Rep., 2007.

[31] S. Malkowski, M. Hedwig, et al., "Bottleneck detection using statistical intervention analysis," in *DSOM '07*.

[32] P. Bodík, O. Fox, et al., "Advanced tools for operators at amazon.com," in *In HotAC '06*.

[33] R. Blake and J. S. Breese, "Automatic bottleneck detection," ftp://ftp.research.microsoft.com/pub/tr/tr-95-10.ps, 1995.