Performance measurements and modeling of database servers

Kihl, Maria; Cedersjö, Gustav; Robertsson, Anders; Aspernäs, Bertil

2011

Performance measurements and modeling of database servers

Maria Kihl, Gustav Cedersjö
Dept. of Electrical and Information Technology
Lund University, Sweden
+46 46 222 9010
maria.kihl@eit.lth.se

Anders Robertsson
Dept. of Automatic Control
Lund University, Sweden
andersro@control.lth.se

Bertil Aspernäs
Ericsson AB
Karlskrona, Sweden
bertil.aspernas@ericsson.com

ABSTRACT

In this paper we present some experiments on the MySQL database server. The objective of the experiments was to investigate the high load dynamics for varying relation sizes and requests. We show that the dynamics for SELECT (read) requests can be modeled as a modified M/M/1 system, whereas, the dynamics for UPDATE (write) are completely different. Our results can be used for designing control and optimization algorithms for database servers.

1. INTRODUCTION

Resource management of computing systems has gained much attention in the last years, since poorly managed resources can degrade the performance of a computer system severely. Enterprise e-services are often networked, implemented as large server clusters that communicate across IP-networks. The systems are subjected to external load disturbances, as traffic surges and changes in user behavior. The experience is that enterprise servers are often the bottlenecks, whereas the network backbone is often underutilized. Therefore, the server systems must provide performance guarantees in the face of external load. The guarantees must satisfy the service-level agreements (on delay, QoS etc) that the system operator has set up with its clients. Also, the system must provide graceful degradation during overload.

Therefore, the challenge is how to control server performance while providing guarantees on convergence and disturbance rejection. The solution is based on *dynamic control schemes*, which monitors the systems, and provides actions when needed. Several types of resource–management mechanisms have been proposed and evaluated in the literature. In larger computer systems, *load balancing* is performed in order to distribute the need for resources uniformly over a number of resource units (Computers, CPUs, memory, etc.), thus avoiding that some units are overloaded while others are idle [5][6]. During overload periods, when more resources are requested than are available, *admission control* mechanisms reduce the amount of work by blocking some of the requests [2][7][8][14]. For Internet applications, virtualized server systems can be used to divide physical resources into a number of separated platforms where different web applications are allowed to operate without affecting one another. *Dynamic resource allocation* between the virtualized platforms serves as a new and easier way to perform resource optimization on web server systems [3][11][12][13]. In the last years, the field of *power and energy management* has become important. Large software systems have high energy consumption, and therefore, dynamic resource optimization of these systems may considerably lower the Operating Expenditures (OPEX) for the network operators [1][4][9][10].

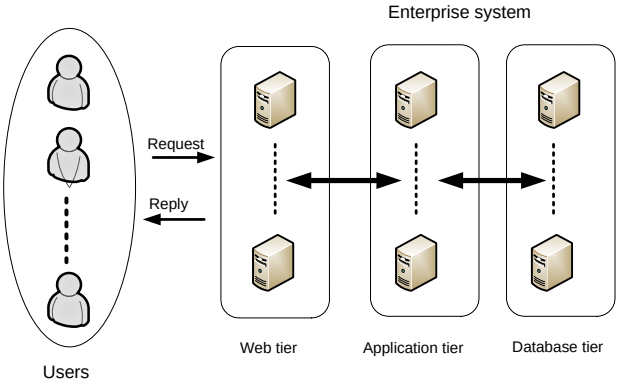

Figure 1 Multi-tier Enterprise system

However, all these optimization techniques require accurate performance models of the involved computing systems. The operation region is mainly high traffic load scenarios, which means that the computing systems have non-linear dynamics that needs to be characterized accurately. In a previous work [13], we have shown that web servers with dynamic content can be modeled as single server systems with processor sharing. However, some recent experiments on databases have shown that the high load dynamics of database servers are completely different than for web servers. Since database servers are important components in future Internet systems, as cloud computing and data centers, we have performed a number of experiments in order to capture and model the high load dynamics of databases. To our knowledge, this is the first paper that proposes performance models for database servers.

2. SYSTEM EXAMPLES

In this section we present two examples of computing systems commonly used in Internet and telecommunication networks, which use database servers.

2.1 Multi-tier Enterprise systems

Enterprise system, also called e-Business systems, as internet banks and web shops, are often implemented as distributed server clusters in several tiers, see Figure 1. The *Web tier* provides the web interface to the users, accepting service requests. The *Application tier* processes the requests, which may include requests to the *Database tier*, providing customer and service information. The performance challenge for the system operator is to optimize the system resources to avoid situations where parts of the system are overloaded when other parts are underutilized.

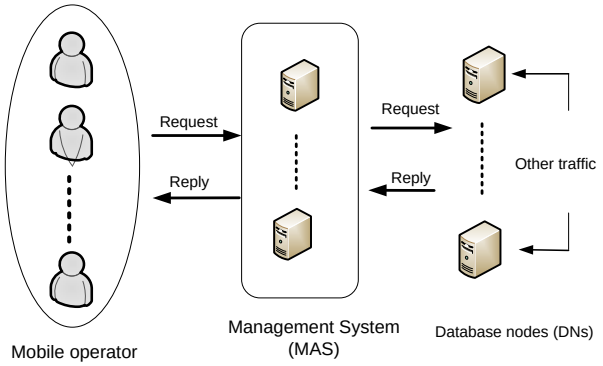

Figure 2 A Telecom service management system

2.2 Telecom Service management systems

In telecommunication service oriented architectures, as mobile networks, data transfer is separated from control signaling. Therefore, all services, either user services as telephony, or administrative services as location updates or billing, are handled by a service management system with its own networks and protocols. The service management systems have a complex architecture, usually implemented as large distributed server systems, with application servers processing service requests from the telecom networks, and databases storing subscriber and service data. The nodes can be owned by different network operators, limiting the available information of traffic loads and service progress. All signaling is performed across IP networks.

3. Performance models

A software system is basically a network of queues, as examples, the CPU ready queue, Semaphore queues, socket queues, and I/O device queues, which stores requests in waiting of service in the processors. Therefore, queuing models can be used when describing the dynamic behavior of server systems [18][19][20][21]. Also, control theory offers a range of structures, tools and analysis methods for adjusting systems to the given environment, which might change over time. Therefore, control theory is very useful when designing performance control schemes for computing systems [22].

However, when combining queuing models with control system design, it is important that the models capture the important dynamics from a control theory point of view. Most service performance metrics such as response time, service rate and processing delay depend on queue state dynamics. For the objective of performance control, simple models, based on the assumption of a single-server queue, are often preferred. The model should only capture the dominating load dynamics of the system, since a well-designed control system can handle many model errors [23]. However, neglecting some hidden buffer dynamics may cause control instability [24].

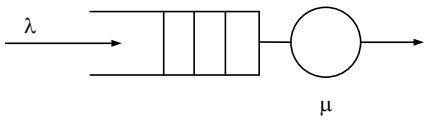

Figure 3 M/M/1 model

3.1 M/M/1 model approximation

The classical M/M/1 model, where a single-server queue processes requests that arrive according to a Poisson process with exponential distributed service times, see Figure 3, has been shown to accurately capture the response time dynamics of a web server system [13].

In a more general form, a system can be assumed to have M/M/1 dynamics, when the average response time, T, for an arrival rate, λ, and a service rate, μ, can be derived as

$$T(\lambda) = T_0 + \frac{1}{\mu - \lambda} \qquad (1)$$

where T_0 is the response time when there is only one request in the system. T_0 may include other delays than the processing delay, for example, protocol handling delays. This model captures the dominating dynamics for the response times. However, it assumes that the capacity is CPU bounded, which means that it only works for CPU intensive applications. Further, the model is valid also for processor sharing systems, as the Apache web server, due to dynamics of time-shared systems [25].

3.2 M/M/1 model with load dependency

In [26] it is proposed that computing systems show load dependencies in the service time, since each request waiting for service will add load in, for example, the operation system and memory. Therefore, the service time for a request will be dependent on the number of concurrent requests in the system. Each new incoming request causes an additional multiplicative increase, p, in the remaining service time duration on all requests in progress. p $(0 < p \leq 1)$ is denoted as the load dependency parameter of the server. At service completion, the request leaving the system will unstress the system, resulting in a corresponding percentage decrease in the remaining service time duration on all remaining requests in progress. Assuming that there are n concurrent requests in the system, the average service time for each request is given by

$$\bar{x}_n = \bar{x} \cdot (1+p)^{n-1} \qquad (2)$$

where $\bar{x}$ is the average service time if there is only one request in the system.

4. Testbed

We have performed a set of experiments in order to develop performance models for database servers. The experiments were performed in our testbed, consisting of three computers acting as two traffic generators and one MySQL database server. An illustration of the testbed is shown in Figure 4. The computers used in the experiments are ordinary desktop computers with some open source server software installed. The computer hardware is a DELL OptiPlex GX270. It is equipped with a 2.0 GHz Intel Celeron processor, 256MB main memory, a single hard-drive and a Gigabit Ethernet network interface. The computers are connected with a Fast Ethernet switch. The Linux distribution Ubuntu 8.04 LTS Server Edition is running on the computers. One computer has a MySQL Server 5.1 installed.

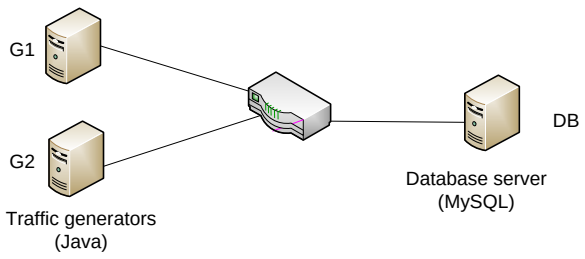

Figure 4 Experimental testbed

4.1 Traffic generators

Since the traffic generator is the primary measurement instrument in the lab, it is important that it generates the desired traffic and measures the response times accurately. To build a high load traffic generator with quality, there are some real challenges: The measurements of the response times must be accurate, the traffic generator cannot become overloaded itself, and the traffic must be generated with the desirable arrival distribution.

4.1.1 Database experiments

The traffic generator in the database experiments was implemented on the Java platform in the Scala Programming Language and relies heavily on the concurrency primitives of JSR-166 (in the package `java.util.concurrent`) and Java Database Connectivity (JDBC). The generator has three main types of concurrent activities. The first generates the requests, the second processes the requests, and the third logs the response times. These activities are represented by Java threads. A `QueryGenerator` creates requests according to a Poisson process, and a fixed number of `Worker`s execute the requests on the database server. Before the generation starts, all threads and connections are set up. The query generator sends the requests over a `BlockingQueue` to the workers. Each worker has established a connection to the database and is waiting for a request to arrive to the queue. As soon as a request arrives, a worker takes it, starts a timer and sends the request over the connection to the database server.

When the response is received, the timer is stopped and both start and stop times are sent to `TimeLogger` where they are logged to a file.

4.2 MySQL database server

In the experiments, we have used the default configuration of MySQL that comes with the Ubuntu package, with the exception that it is opened up for external connections and the maximum number of concurrent connections increased to 500 connections. MySQL has support for different database engines, but all databases in the lab use the same engine, namely MyISAM, which is also the default engine in MySQL.

The MySQL server contains one database with several relations. All relations use the same schema, but with different number of tuples. The basic structure of the relation is inspired by the database benchmark developed by Bitton, DeWitt and Turbyfill called the "Wisconsin Benchmark" [15]. The actual structure comes from a newer version of this benchmark that is more scalable with respect to relation size [16]. The maximum number of tuples is limited to the number of unique values of a 32 bit integer. The relation has 16 attributes, and each tuple occupies 200B. The largest possible relation would occupy 800GB. Three different sizes were used in the experiments: one million, five million and ten million tuples.

Two of the attributes in the relation, `unique1` and `unique2`, are unique integers from zero to the number of tuples. The integers in `unique2` are sequential, while `unique1` are random, and all other attributes are derived from either of them.

A Haskell program was developed to generate the data for the relation from the specifications of the benchmark. Haskell is a lazy functional programming language [17]. A linear feedback shift register that generates a maximum length sequence was used to generate the sequences for `unique1` and `unique2`. The program generates not just the sequences, but complete INSERT-statements for the whole relation. This program was pipelined to the "MySQL command-line tool" that sent the actual statements to the server.

5. Experiments

Before the generation of traffic started, all connections to the server were set up and all parts of the traffic generator were initialized and ready to go. Requests were sent on the connections as a Poisson process with a specified rate, and all response times were logged in a file. Since the response times are measured by the traffic generator, these times includes latency introduced by the network and the computer with the traffic generator. This experiment run for a specified amount of time, typically minutes. All connections were then closed and the generator was stopped. The log file was saved with a filename that includes the arrival rate. The same experiment was performed with different arrival rates. The log files from the whole set of experiments were put in a folder together with the traffic generator's configuration used during the experiments.

The basic structure was the same in all experiments. A computer with a traffic generator that measures response times was connected to a computer with a database server. The traffic generator generated requests to the database server.

5.1 Experiment set E1: Relation size

The objective of the first set of experiments was to investigate the performance behavior for different database sizes. The first set of experiments was performed on the relation described above with three different sizes. The request was a `SELECT`-query that looked like this.

```
SELECT unique3 FROM wisc WHERE unique2=?;
```

The attribute `unique2` was indexed and each `id` had a unique value between zero and the number of tuples. The question mark in the query was replaced with random numbers, uniformly distributed over the range from zero to the number of tuples. The number of tuples was set to $1x10^6$, $5x10^6$, and $1x10^7$.

5.2 Experiment set E2: SELECT and UPDATE

A database engine has to be able to handle several connections concurrently trying to read and update the same tuple in a relation. Usually, it is done by locking some part of the database for exclusive access to one connection at the time. With MyISAM, the database engine we use, the whole table that is targeted is locked during `UPDATE`s. Writing requires exclusive access. However, two connections concurrently reading a tuple do not need any locking. This could result in different response times for concurrent `UPDATE` and concurrent `SELECT`.

To find out if the request type affects the overload detection, a new set of experiments was performed. The new experiments

compared response times from concurrent SELECTs with response times from concurrent UPDATEs. First the set of experiments with SELECT was performed. After that, the experiments with UPDATE were performed.

6. Results

We measured response times of SQL-requests to a MySQL database server. The requests were generated as a Poisson process by a traffic generator. The traffic generator registered timestamps for each request and response, from which the response times were derived.

6.1 M/M/1 approximation

In some of the experiments, we were able to approximate the response time dynamics with a modified M/M/1 model, according to the analysis below.

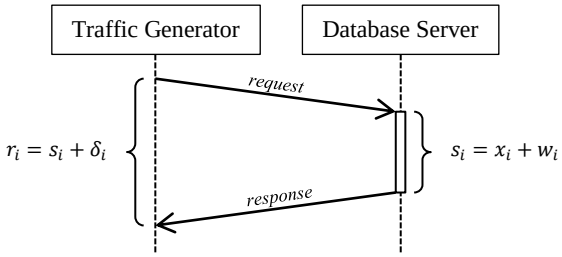

Figure 5 Sequence diagram showing response time r_i and system time s_i for the i:th request

Let s_i be the time in the system for the i:th request. The system time s_i is the sum of the service time x_i and the waiting time w_i. The response time r_i is measured by the traffic generator and is the sum of the system time s_i and some latency δ_i. (See Figure 5) We assume that the latency δ_i is independent of the load. The average system time for an M/M/1 system is

$$\bar{s} = \frac{1/\mu}{1-\rho} = \frac{1}{\mu - \lambda}$$

where $\rho = \lambda/\mu$ is the utilization, λ is the arrival rate to the system, and $\mu = 1/\bar{x}$ is the service rate [25]. Suppose the response time measurements comes from an M/M/1 system, then the average response time can be written as

$$\bar{r} = \bar{s} + \bar{\delta} = \frac{1}{\mu - \lambda} + \bar{\delta} \qquad (3)$$

The service rate μ is estimated from the response time measurements where λ is small. By expanding $\lim_{\lambda \to 0} \bar{r}$ an estimated of the service time and the service rate is derived.

$$\bar{r} = \bar{s} + \bar{\delta} = \frac{1}{\mu - \lambda} + \bar{\delta} \to \frac{1}{\mu} + \bar{\delta} = \bar{x} + \bar{\delta}, \qquad \lambda \to 0$$
$$\bar{r} - \bar{\delta} \to \bar{x}, \qquad \lambda \to 0$$

The service rate μ is estimated with $\hat{\mu} = 1/(\bar{r}_0 - \bar{\delta})$ where $\bar{r}_0$ is the average response time from an experiment where λ is low, or near zero. The estimated service rate $\hat{\mu}$ is inserted in equation (3) to get an estimation of the average response time.

$$\hat{r} = \frac{1}{\hat{\mu} - \lambda} + \bar{\delta} = \frac{1}{\frac{1}{\bar{r}_0 - \bar{\delta}} - \lambda} + \bar{\delta}$$

The only parameter that is unknown is the average latency $\bar{\delta}$, since λ is chosen for each experiment, and $\bar{r}_0$ is calculated from the response times of an experiment with low arrival rate. The experiment is now performed for several different arrival rates.

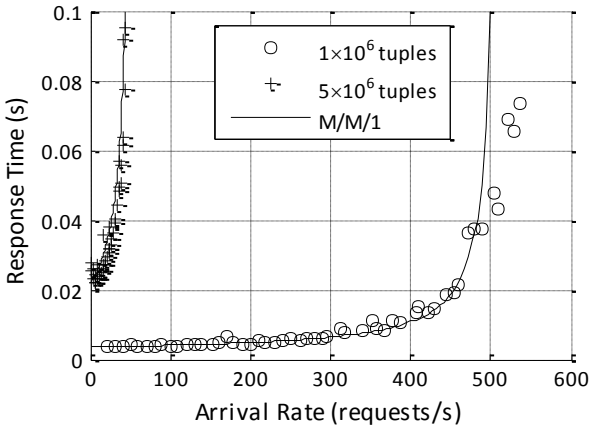

Figure 6 Response times of SELECT-queries

For each experiment the average response time $\bar{r}$ is calculated and plotted versus the arrival rate λ of that experiment. The estimation of the average response time for an M/M/1 system $\hat{r}$ is also plotted. The parameter $\bar{\delta}$ in $\hat{r}$ is estimated by fitting the curve $\hat{r}$ to the measured average response times.

6.2 Experiment E1

In experiment set E1 we measured response times with relations of different sizes. We tested with $1x10^6$, $5x10^6$, and $1x10^7$ tuples. Figure 6 shows the response time of SELECT-queries to a relation with $1x10^6$ tuples and a relation with $5x10^6$. It also shows curves for M/M/1 queuing systems that behave approximately the same under these circumstances. The curves for the M/M/1 systems were plotted with $\bar{\delta} = 0.0016$s for the $1x10^6$-tuple relation and $\bar{\delta} = 0.0035$s for the $5x10^6$-tuple relation.

Figure 7 shows response times of SELECT-queries to relations with $5x10^6$ tuples and $1x10^7$ tuples, where the response times to the relation with $1x10^7$ tuples is compared to an M/M/1 queuing system plotted with $\bar{\delta} = 0.0035$. The difference when increasing from $1x10^6$ to $5x10^6$ tuples is much larger than the difference when increasing from $5x10^6$ to $1x10^7$ tuples.

6.3 Experiment E2

In experiment set E2 we measured response times of SELECTs and UPDATEs to the same relation. Figure 8 shows response times for those kinds of queries and a curve from an M/M/1 system. The response time curve for SELECTs is quite similar to the curve from the M/M/1 system.

The UPDATEs however, does not look like any M/M/1 system. We have some preliminary results showing that UPDATE requests behave as a load dependent server system, however, more work is needed on this subject. Also, in a real system, the requests will be a mix of SELECTs and UPDATEs. Further, different database engines work differently, which means that it will be necessary to investigate different types of database servers.

7. Conclusions

Database servers are vital components of the future Internet services, as data centers, e-business systems, and cloud computing. The accurate control and optimization of these systems require good performance models capturing the dynamics during high loads. In this paper, we present some preliminary experiments on MySQL, showing that during some conditions, the system can be modeled as a modified M/M/1 system. However, more work is needed on the subject.

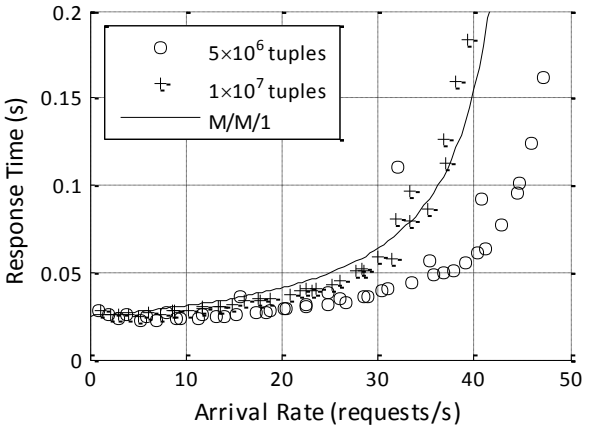

Figure 7 Response times of `SELECT`-queries

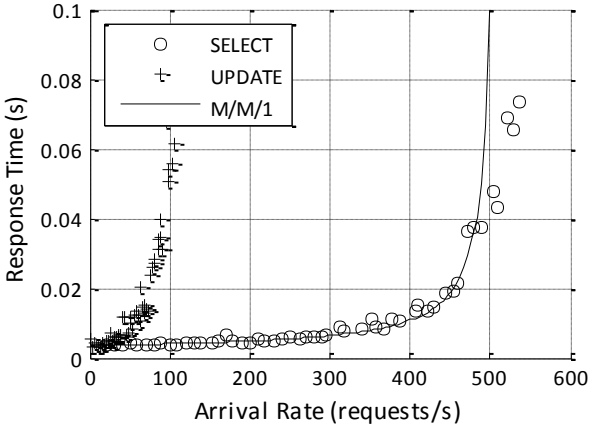

Figure 8 Response times of `SELECT`- and `UPDATE`-queries

8. ACKNOWLEDGMENTS

This work has been partly funded by the Lund Center for Control of Complex Engineering Systems (LCCC) and the Swedish Research Council grant VR 2010-5864. Maria Kihl is funded in the VINNMER program at VINNOVA.

9. REFERENCES

[1] R. Bianchini and R. Rajamony, "Power and energy management for server systems," *IEEE Computer*, vol. 37, no. 11, 2004.

[2] M. Kihl, A. Robertsson, M. Andersson, and B. Wittenmark, "Control theoretic analysis of admission control mechanisms for web server systems," *The World Wide Web Journal*, Springer, vol. 11, no. 1, 2008.

[3] M. Kjaer, M. Kihl, and A. Robertsson, "Resource Allocation and Disturbance Rejection in Web Servers using SLAs and Virtualized Servers", *IEEE Transaction on Network and Service Management*, Vol. 6, No. 4, 2009.

[4] H. Claussen, L.T.W Ho, F. Pivit, "Leveraging advances in mobile broadband technology to improve environmental sustainability", *Telecommunications Journal of Australia*, Vol. 59, No. 1, 2009.

[5] Y. Diao, C. Wu, J. Hellerstein, A. Storm, M. Surendra, S. Lightstone, S. Parekh, C. Garcia-Arellano, M. Carroll, L. Chu, and J. Colaco, "Comparative studies of load balancing with control and optimization techniques," in *Proc. American Control Conference*, 2005.

[6] Y. Fu, H. Wang, C. Lu, and R. Chandra, "Distributed utilization control for real-time clusters with load balancing," in *Proc. IEEE International Real–Time Systems Symposium*, 2006.

[7] X. Chen, H. Chen, and P. Mohapatra, "Aces: an efficient admission control scheme for QoS-aware web servers," *Computer Communication*, vol. 26, no. 14, 2003.

[8] X. Liu, J. Heo, L. Sha, and X. Zhu, "Adaptive control of multi–tiered web applications using queuing predictor," in *Proc. 10th IEEE/IFIP Network Operation Management Symposium*, 2006.

[9] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu, "Dynamic voltage scaling in multitier web servers with end-to-end delay control," *IEEE Transactions on Computers*, vol. 56, no. 4, 2007.

[10] E. Elnozahy, M. Kistler, and R. Rajamony, "Energy-efficient server clusters," in *Lecture Notes in Computer Science 2325*. Springer-Verlag Berlin Heidelberg, 2003..

[11] W. Xu, X. Zhu, S. Singhal, and Z.Wang, "Predictive control for dynamic resource allocation in enterprise data centers," in *Proc. 10th IEEE/IFIP Network Operation Mangement Symposium*, 2006.

[12] Z. Wang, X. Liu, A. Zhang, C. Stewart, X. Zhu, T. Kelly, and S. Singhal, "AutoParam: automated control of application-level performance in virtualized server environments," in *Proc. 2nd IEEE International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks, 2007.*

[13] J. Cao, M. Andersson, C. Nyberg and M. Kihl, "Web Server Performance Modeling using an M/G/1/K*PS Queue", Proc. of the International Conference on Telecommunication, 2003

[14] V. Mathur and V. Apte, "A computational complexity-aware model for performance analysis of software servers", IEEE 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOT), 2004.

[15] D. Bitton, D.J. DeWitt, and C. Turbyfill. Benchmarking database systems a systematic approach. In Proceedings of the 9th International Conference on Very Large Data Bases, 1983.

[16] D.J. DeWitt. The Wisconsin benchmark: Past, present, and future. The Benchmark Handbook for Database and Transaction Processing Systems, 1, 1991.

[17] S. Marlow. Haskell 2010 Language Report, 2009.

[18] M. Kihl, A. Robertsson, and B. Wittenmark, "Performance modelling and control of server systems using non-linear control theory", In Proc. of the 18th International Teletraffic Congress, 2003.

[19] K-J. Åström and B. Wittenmark, *Computer–Controlled Systems*. Upper Saddle River, NJ: Prentice Hall, 1997.

[20] L. Malrait, N. Marchand and S. Bouchenak, "Modeling and Control of Server Systems: Application to Database Systems", European Control Conference, 2009.

[21] L. Malrait, S. Bouchenak and N. Marchand, "Fluid Modeling and Control for Server System Performance and Availability, IEEE International Conference on Dependable Systems and Networks, 2009.

[22] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury, "Control engineering for computing systems," *IEEE Control Syst. Mag.*, vol. 25, no. 6, 2005.

[23] V. Mathur and V. Apte, “A computational complexity-aware model for performance analysis of software servers”, IEEE 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOT), 2004.

[24] X. Liu, J. Heo and L. Sha, “Modeling 3-tiered web applications”, 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOT), 2005.

[25] L. Kleinrock, *Queueing Systems, Volume I: Theory*, Wiley Interscience, New York, 1975.

[26] Curiel, M. & Puigjaner, R., “Using load dependent servers to reduce the complexity of large client-server simulation models”, *Performance Engineering, LNCS 2047, Springer-Verlag Berlin Heidelberg*, 2001.