in "Self-Star Properties in Complex Information Systems", O.Babaoglu et. al., eds., LNCS 3460, 2005.

On the Use of Online Analytic Performance Models in Self-Managing and Self-Organizing Computer Systems

Daniel A. Menascé, Mohamed N. Bennani, and Honglei Ruan

George Mason University, Department of Computer Science, MS 4A5, Fairfax, VA 22030, USA

Abstract. Current computing environments are becoming increasingly complex in nature and exhibit unpredictable workloads. These environments create challenges to the design of systems that can adapt to changes in the workload while maintaining desired QoS levels. This paper focuses on the use of online analytic performance models in the design of self-managing and self-organizing computer systems. A general approach for building such systems is presented along with the algorithms used by a Quality of Service (QoS) controller. The robustness of the approach with respect to the variability of the workload and service time distributions is evaluated. The use of an adaptive controller that uses workload forecasting is discussed. Finally, the paper shows how online performance models can be used to design QoS-aware service oriented architectures.

1 Introduction

The next generation of large distributed systems will consist of millions of interconnected heterogeneous devices and of a very large number of sources that generate data in widely different formats. These devices have significantly different characteristics in terms of processing power, bandwidth, reliability, battery life, and connectivity (wired or wireless). Many different types of applications with different and competing Quality of Service (QoS) requirements may share a common computing, communication, and data storage infrastructure. Applications running in these environments will i) be component-based for increased reusability, ii) service-oriented, iii) need to operate in unattended mode and possibly in hostile environments such as battlefields or natural disaster relief situations, iv) be composed of a large number of "replaceable" components discoverable at runtime, and v) have to run on a multitude of unknown and heterogeneous hardware and network platforms.

Under these circumstances, systems must be adaptable and self-configurable in order to continuously meet QoS requirements at the application and component level in the presence of changes in workload intensity. Adaptability and self-configuration is also necessary to cope with attacks and failures in order to meet availability and security requirements. Therefore, because of the dynamic aspects of complex distributed computer systems, they must be self-configurable, self-optimizing, self-healing, and self-protecting.

Some important challenges must be addressed:

- The structure of applications changes dynamically as new services are added or removed.
- The workload is hard to characterize due to its unpredictable nature, dynamically changing services, and application adaptation.
- It is difficult to build static performance models because the system is in constant evolution.
- There is a multitude of QoS metrics at various levels. Some examples include response time, jitter, throughput, availability, survivability, recovery time after attack/failure, call drop rate, access failure rate, packet delays, packet drop rates.
- There are tradeoffs between QoS metrics (e.g., response time vs. availability [15], response time vs. security [18]).
- Transient analysis of QoS compliance, and not just steady-state analysis, of system behavior is necessary in critical times such as terrorism attacks or catastrophic failures.
- Global QoS goals have to be mapped to locally enforced and monitored QoS goals [14].
- There is a need for protocols and mechanisms for efficient QoS goal negotiation, monitoring, and enforcement. Given the heterogeneous nature of these systems, QoS goals and contracts have to be specified in platform-neutral terms.
- Resource management mechanisms including resource reservation, resource allocation, and admission control in non-dedicated resources, as in Grid computing [13], are generally complex.

There has been a growing interest in self-managing systems and self-configuring systems as illustrated by the papers in a recent workshop [5] and in [1, 2, 6, 7, 8, 9, 10, 11, 17, 20, 22, 25]. In this paper, we describe our approach which consists of using analytic performance models in the design of self-configurable and self-managing computer systems. This approach is exemplified in various of our papers [4, 16, 17, 19, 20]. We provide here an all encompassing framework, describe the challenges, and summarize result obtained. Section two discusses our general approach to controlling computer systems. Section three presents an example of the results obtained by applying these ideas to control a Web server. The next section shows how analytic performance models are robust when the workload and service time distributions exhibit high variability. Section five describes the design of an adaptive controller that uses workload forecasting and presents an example of results for such a controller. Section six shows how online performance models can be used to build QoS-aware service oriented architectures that perform QoS negotiation and admission control. Finally, Section seven presents some concluding remarks.

2 General Approach

We use Fig. 1 to illustrate our general approach to designing self-organizing and self-managing computer systems. We consider that a system is subject to a workload, which may consist of any mix of online transactions and batch jobs. There is a multitude of parameters and settings that may affect the performance of such systems. Examples of these parameters include, among others, TCP, web server, application level, database server, operating system, and load balancer parameters. An analysis of the effects of various configurable parameters in E-commerce systems can be found in [23].

The set of parameters is divided into uncontrolled parameters and controlled parameters. Uncontrolled parameters (1) are those that are not changed dynamically by the controller. Typically, these parameters are the ones that have relatively little impact on performance or that require a system restart or reboot in order for their effect to take place. Controlled parameters (2) are those whose settings are changed dynamically by the *controller* (3) by executing a controller algorithm. The goal of this algorithm is to find settings of the controlled parameters that optimize a given *goal function* (4), which can be expressed in the form of a utility function [24] or any other function of the values of the primary responses. Examples will be given in the remaining sections of the paper. The result of the current goal is passed to the controller, which may request goal evaluations for different possible settings of the controller parameters.

A set of responses are generated as a result of the workload (5) and of the settings for all parameters—controlled and uncontrolled. The responses are typically divided into primary responses (6) and secondary responses (7). The former are those whose values must be kept within desired ranges as specified by Service Level Agreements (SLAs) or QoS goals (8). Examples of primary responses may include response time, throughput, and probability of rejection. Secondary responses are those for which no QoS goals are set but that may be used by the controller algorithm. An example is the utilization of the various devices of the controlled system.

2.1 The Controller Algorithm

The size of the state space of possible configurations grows in a combinatorial way with the number of controlled parameters. Therefore, an exhaustive search of that space is not feasible. The controller uses combinatorial search techniques [21] such as hill-climbing and beam-search to find a close-to-optimal configuration for which the value of the goal function is as close as possible to its desired level.

Figure 2 displays an example of a portion of a state space. Each point represents a configuration of the controlled parameters and the numerical value associated with each point represents the value of the goal function. Suppose that the current configuration is point A, which has value 10, and that through a hill-climbing search, a new configuration, point B with value 35, is found.

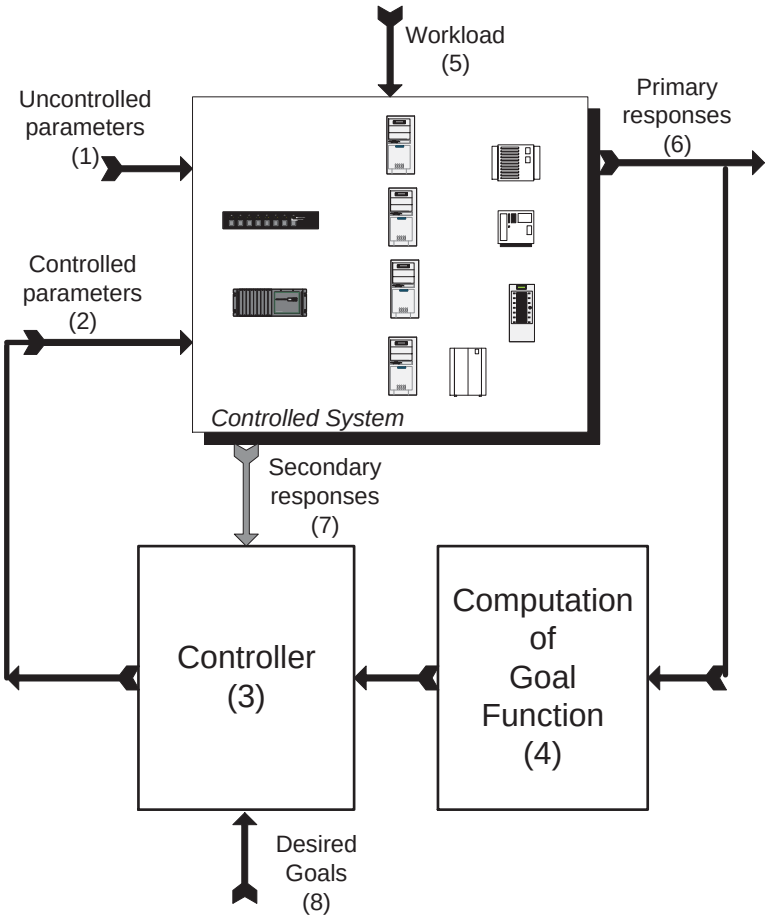

Fig. 1. General Approach to Self-managing and Self-organizing Computer Systems

An important question is how is the goal value computed for each configuration point? The goal value for the current configuration is obtained from measurements obtained from the system. However, as the search technique explores the state space, the goal values have to be computed through the use of models that can predict the value of response variables for configurations different from the current one. Our approach consists in using analytic performance models of the system to obtain the values of the primary responses.

This *online* use of predictive performance models is a departure from their common use in capacity planning [12]. In those cases, performance models are used to analyze and compare scenarios over relatively long (in the order of months) periods of time. In the case of self-configuring and self-managing sys-

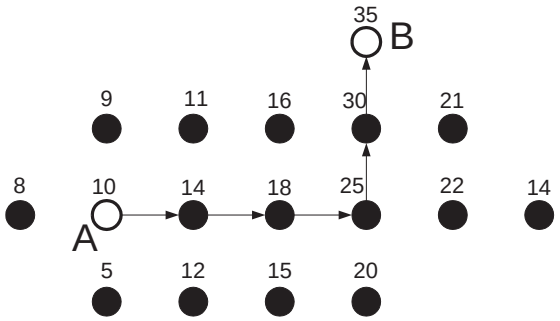

Fig. 2. Example of State Space Search

tems, configurations may have to change very frequently (at a few-minute intervals).

2.2 Controller Types

Controllers can be classified according to the control frequency and workload forecasting method used.

- Control frequency. The interval between two successive executions of the controller algorithm is called the *control interval (CI)*. Controllers can be classified according to the length of the controller interval as follows:
  - Fixed CI: the length of the control interval is constant.
  - Adaptive CI: If the CI is too small and the workload intensity is relatively stable, the controller algorithm will be executed too often with little or no effect. If the CI is too large and the workload intensity varies very rapidly, the controller will not run frequently enough to be effective. Thus, a CI that adjusts itself to the workload intensity can be more effective than a fixed CI.
- Workload forecasting. The online performance models used by the controller algorithm use two types of parameters: workload intensity (e.g., arrival rates of requests) and service demands of the requests on the various resources of the computer system [12]. The service demands can be obtained by monitoring the utilization of the various system resources (e.g., CPU, disks, and network segments). Controllers can be classified according to their use of the workload intensity as follows:

- No forecasting: the workload intensity used to run the performance models in a given CI is the same as the workload intensity seen in the previous CI.
- Workload forecasting: the workload intensity used to run the performance models in a given CI is a forecast of the workload intensity based on workload intensity values for a certain number of previous intervals. Workload forecasting techniques such as exponential smoothing, weighted moving averages, and polynomial regression [12] can be used. It was shown [4] that the use of workload forecasting can improve the QoS of a controlled system, especially when the workload intensity approaches its saturation value.

3 An Example: A Controlled Web Server

In this section we show the results of applying the techniques described above to the QoS control of an actual Web server. The HTTP server is Apache 1.3.12, which was modified to allow for a dynamic change of the number of active threads (m) and the maximum number of requests in the system (n). These parameters, m and n, are the controlled parameters. The workload used to drive the server is generated by SURGE, a workload generator for Web servers [3], using two client machines sending requests to a third machine that runs the Web server. SURGE generates references matching empirical measurements regarding file size distributions, relative file popularity, embedded file references, and temporal locality of references. This workload generator was selected because it was demonstrated [3] that, unlike other Web server benchmarks, it exercises servers in a manner that is consistent with actual empirical distributions observed in Web traffic. A fourth machine runs the QoS controller. All four machines are Intel-based and run either Windows 2000 Professional or Windows XP Professional. All machines are connected through a 100-Mbps LAN switch.

The primary responses are the response time of an HTTP request (R), the throughput of the HTTP server (X_0), and the probability that a request is rejected (P_{rej}). The goal function is a QoS value defined as $QoS = w_R \times \Delta QoS_R + w_X \times \Delta QoS_X + w_P \times \Delta QoS_P$, where ΔQoS_R, ΔQoS_X, and ΔQoS_P are relative deviations of the average response time, average throughput, and probability of rejection, with respect to their SLAs, and w_R, w_X, and w_P are the relative weights of these deviations with respect to the QoS value [4, 17]. These deviations are defined in such a way that their value is in the range $[-1, 1]$. When the response metric meets its goal the deviation is zero. The deviation is negative when the response metric does not meet its goal and positive when the goal is exceeded. Therefore, the value of QoS is also in the range $[-1, 1]$ and the larger its value the better. The weights w_R, w_X, and w_P have to be chosen in a way that reflects the relative importance of the three performance metrics—response time, throughput, and probability of rejection—to the management of the Web site. The SLAs and respective weights for the experiment described here are:

$R \leq 0.3$ seconds, $w_R = 0.5$, $X_0 \geq 50$ requests/sec, $w_X = 0.2$, $P_{\text{rej}} \leq 0.05$, and $w_P = 0.3$.

Figure 3 shows the variation of the QoS during the experiment. The x-axis is a time axis labeled in units of control intervals. The workload intensity started at 5 requests/sec and climbed to 19 requests/sec at CI = 19. Then, the workload intensity was reduced to 14 requests/sec. The experiment in question lasted 30 CIs and each CI is equal to two minutes. Results are shown for two types of combinatorial search techniques: hill climbing and beam search (see top two curves). As it can be seen, the QoS for the uncontrolled Web server (bottom curve) becomes negative when the load reaches its peak value, indicating that at least one of the three metrics is not meeting its SLA. On the other hand, the QoS for the controlled Web server always remains in positive territory for both hill-climbing and beam search. We noticed in the various experiments we carried out that beam search tends to provide slightly better results than hill-climbing. This is probably due to the fact that the latter combinatorial search technique may at times be trapped at local optima. However, the difference between the two techniques was never significant.

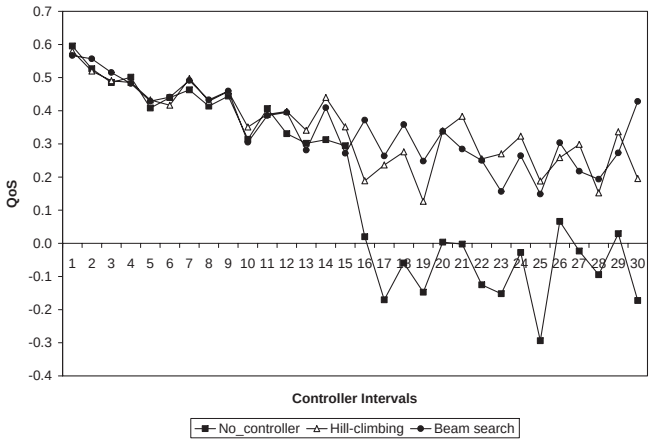

Fig. 3. A controlled Web Server

4 The Robustness of Online Models

Many real workloads exhibit some sort of high variability in their intensity and/or service demands at the different resources. Therefore, it is important to investigate the behavior of the proposed technique for self-managing computer

systems in such environments. To this end, we conducted a set of experiments to study the impact of the variability in the request inter-arrival time and service times distributions using a simulated multi-threaded server with one CPU and one disk. The server has m threads and at most n requests can be in the server, waiting for a thread or being executed by a thread. The goal function is the one used in Section 3. The SLAs and respective weights for the experiment described here are: $R \leq 1.2$ seconds, $w_R = 0.25$, $X_0 \geq 5$ requests/sec, $w_X = 0.3$, $P_{\text{rej}} \leq 0.05$, and $w_P = 0.45$.

The variability of the distributions of the inter-arrival time and service times distributions is represented by their respective coefficients of variation (COV) (i.e., the standard deviation divided by the mean): C_a and C_s. Figure 4 shows the value of the QoS during an experiment in which $C_a = C_s = 2.0$. The x-axis is labeled in CIs and each CI is equal to two minutes. The workload intensity varies in the same manner as in Section 3. The top two curves correspond to hill climbing and beam search and the bottom curve corresponds to the situation in which the controller is disabled. It can be clearly seen that even at the peak intensity value (CI = 20), the QoS decreases only slightly (from 0.8 to 0.55) when the controller is used. On the other hand, the QoS decreases from 0.8 to less than 0.1 when the controller is disabled. Other results for different values of C_a and C_s are presented in our previous work [4]. These results show the robustness of analytic models when used for QoS control. Even though these models assume exponential service and interarrival times (i.e., $C_s = 1.0$ and $C_a = 1.0$), they do a good job at predicting the trends of QoS metrics when these assumptions are violated. The reason is that it is more important to correctly compare, QoS-wise, two points in the search space than knowing their absolute QoS values.

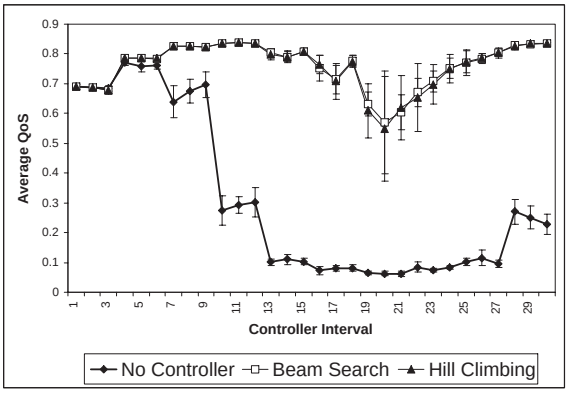

Fig. 4. QoS Controller Performance for $C_a = 2$ and $C_s = 2$.

5 Adaptive Controller Intervals

Figure 5 shows an algorithm that can be used to dynamically vary the length of the control interval. This algorithm sets the length of the CI as a multiple, K, of the smallest possible control interval $\mathrm{CI}_{\mathrm{min}}$. When the currently measured value of the QoS, $\mathrm{QoS}_{\mathrm{curr}}$, is less than or equal to a minimum value $\mathrm{QoS}_{\mathrm{min}}$ for the QoS, the controller interval is set to its minimum value $\mathrm{CI}_{\mathrm{min}}$. Otherwise, the controller interval is set to a multiple of $\mathrm{CI}_{\mathrm{min}}$ according to the relative error ϵ between the QoS value, $\mathrm{QoS}_{\mathrm{prev}}$, measured last time the controller was activated and the currently measured value of the QoS, $\mathrm{QoS}_{\mathrm{curr}}$.

If $\mathrm{QoS}_{\mathrm{curr}} < \mathrm{QoS}_{\mathrm{min}}$
then $CI \leftarrow \mathrm{CI}_{\mathrm{min}}$
else begin
$\epsilon = \left| \frac{\mathrm{QoS}_{\mathrm{curr}} - \mathrm{QoS}_{\mathrm{prev}}}{\mathrm{QoS}_{\mathrm{prev}}} \right|$
If $0 \leq \epsilon \leq 0.05$ then $K = 12$
If $0.05 < \epsilon \leq 0.1$ then $K = 6$
If $0.1 < \epsilon \leq 0.2$ then $K = 5$
If $0.2 < \epsilon \leq 0.3$ then $K = 4$
If $0.3 < \epsilon \leq 0.4$ then $K = 3$
If $0.4 < \epsilon \leq 0.5$ then $K = 2$
If $\epsilon > 0.5$ then $K = 1$
$CI \leftarrow K \times \mathrm{CI}_{\mathrm{min}}$
end

Fig. 5. Algorithm for adjusting control interval length.

Figure 6 shows the variation of the QoS when the control interval varies according to the algorithm of Fig. 5 when $C_a = C_s = 1.0$. In these curves, workload forecasting is always used. The workload used in that experiment has two peaks: one at time 6 and another at time 20. It can be clearly seen from the figure that the use of a dynamically adjusted controller interval yields better QoS values. For example, at peak loads (see monitoring intervals 6 and 20), the QoS for the dynamically adjusted system is always positive. These curves also show that the QoS values obtained when adaptive control intervals are used are generally higher than those achieved when the controller runs at fixed intervals.

One could ask the question whether these improvements come mainly from the dynamic adjustment of the control interval and not because of workload forecasting. To this end, we conducted another set of experiments in which we compare the average QoS values obtained for the cases when dynamic controller intervals were used alone against the cases when they were used jointly with workload forecasting. The results are reported in Fig. 7 for $C_a = C_s = 1.0$. The

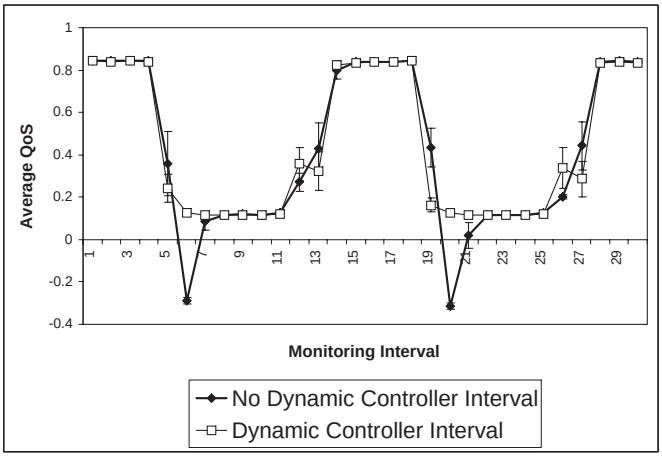

Fig. 6. Dynamic controller interval impact on QoS (forecasting always used)

curves in this figure clearly show that there is a statistically significant performance gain when forecasting is enabled in conjunction with dynamic controller interval. There is an accompanying increase in the QoS gain as a result of using dynamic controller intervals combined with workload forecasting.

6 QoS-aware Service Oriented Architectures

In [16] we presented a framework for the design of QoS-aware software components. This section presents another application of online performance models in the design of QoS-aware Service Oriented Architectures (SOAs). Figure 8 presents an architecture that consists of QoS-aware service providers (SPs), clients that make requests to the SPs, and a QoS Broker (QB). SPs register with the QB. During the registration process, the QB engages in a protocol aimed at characterizing the services provided by an SP in terms of their service demands. A client that needs a service contacts the QB to discover the service, authenticate itself, and negotiate QoS requirements in terms of response time and required throughput for its session with the SP. The QB keeps track of all accepted QoS requests and uses an online performance model to negotiate new requests. Based on the results of the performance evaluation, a request may be accepted, rejected, or a counteroffer may be sent to the client. Once a session is accepted, the SP is informed and the client makes requests directly to the SP, which is responsible for admission control.

We implemented the approach in Java and performed several experiments to validate the approach. A comparison between the QoS-based case and the non-QoS case (i.e., the case in which no QoS negotiation takes place) is shown

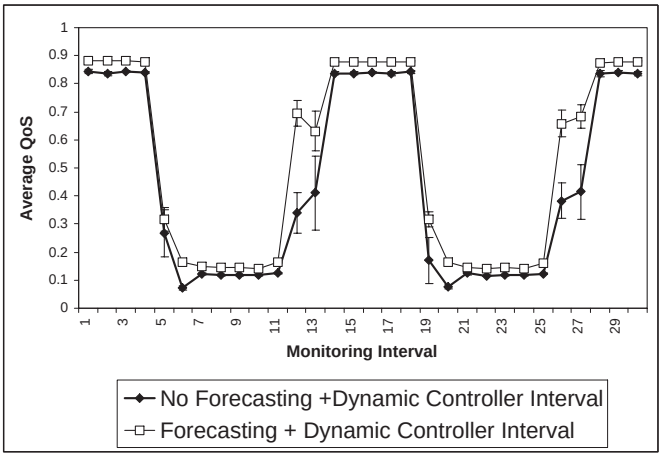

Fig. 7. Impact of workload forecasting on QoS (dynamic controller intervals always used)

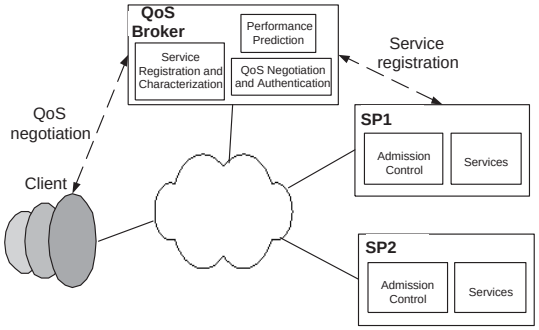

Fig. 8. A Service Oriented Architecture with a QoS broker.

in Fig. 9. The experiments generate a random workload which is submitted to the two SPs without using the QoS broker. The workload is replayed against the same SPs but this time using the QoS negotiation protocol. During this second phase, a reduction on the response time observed in the non-QoS case is requested. Let f be the reduction factor in the response time.

Figure 9 compares response times and throughputs between the QoS-brokered and the non-QoS brokered cases for all sessions executed by SP 1 for a reduction factor of $f = 35\%$. The top graph of that figure shows the average request response time for the two cases and the bottom one shows the throughput in requests/sec. The x-axis is labeled by session ID, each of which is unique in the

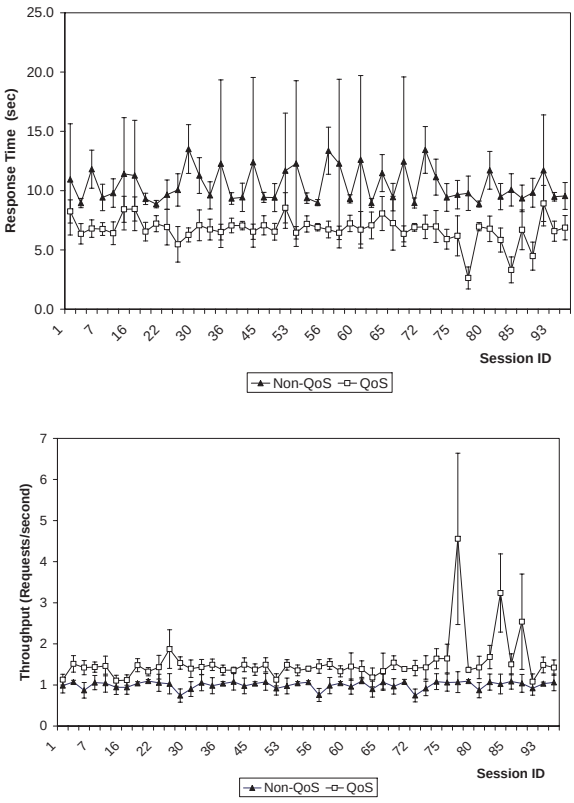

Fig. 9. Response time (top) and throughput (bottom) for the non-QoS and QoS negotiated cases for $f = 35\%$.

experiments. The value at each point is an average over 500 values collected during the experiments. The curves also display the 95% confidence intervals for the average values. As it can be seen from the figure, there is a significant performance gain, i.e., decreased response time and increased throughput for the service provider. As shown in Table 1, with an $f = 35\%$ response time reduction requirement at QoS negotiation time, the actual response time reduction is 35.2%, which perfectly matches the QoS requirement.

We also conducted similar experiments for $f = 0\%$ and $f = 20\%$. The summary results for the two SPs are shown in Table 1. In this table, %RT Reduction stands for the percentage reduction of the average response time and %XPUT Increase stands for the percent increase in throughput relative to the non-QoS case. Table 1 also shows, for each SP, the percent of sessions that are rejected, the percent of session requests that received a counter offer, and the percent

of sessions that were accepted by the QoS broker. As it can be seen, with the QoS broker, the SPs always achieve better performance than without it. Moreover, the actual response time reductions achieved matched pretty closely the QoS goals for different values of f. These results demonstrate the applicability and effectiveness of online analytic performance models for building QoS-aware Service Oriented Architectures.

Table 1. Summary of Results for $f = 0.0$, $f = 0.20$, and $f = 0.35$

		$f = 0\%$	$f = 20\%$	$f = 35\%$
SP 1	% RT Reduction	4.1	16.7	35.2
	% XPUT Increase	9.0	24.0	54.7
	% Reject	17.0	22.7	36.9
	% Counter Offer	6.9	9.3	9.9
	% Acceptance	76.1	68.0	53.2
SP 2	% RT Reduction	4.9	21.2	35.0
	% XPUT Increase	12.1	24.0	54.7
	% Reject	15.2	29.3	39.8
	% Counter Offer	10.8	9.7	10.4
	% Acceptance	74.0	61.0	49.8

7 Concluding Remarks

This paper presented a general approach that can be used to build self-managing/organizing computer systems. We showed how online analytic performance models can be used in an efficient and effective manner for that purpose. We discussed and presented results for different design alternatives for the controller component of these autonomic systems. These design alternatives include the selection of a combinatorial search technique, the frequency at which the controller algorithm is invoked, and the importance of a workload forecasting feature. We provided promising results obtained from a real web server subject to a workload generated by the SURGE benchmark. The robustness of these techniques with respect to the variability of interarrival times and service times and the effectiveness of using an adaptive controller were evaluated on a simulated multithreaded server. We also showed how online performance models can be used to design QoS-aware service oriented architectures. We are currently expanding our work along several lines. First, we are looking into the use of online analytic performance models that are subject to heterogeneous classes of requests. We are also investigating how the techniques presented here can be used to determine optimal resource allocation for autonomic data centers. In this case, control has to be carried out in a distributed manner. Local controllers have to coordinate with global controllers to maximize a global utility function.

Acknowledgements

This work was partially supported by grant NMA501-03-1-2022 from the National Geospatial-Intelligence Agency (NGA) and by grant ACI 0203872 from the National Science Foundation.

References

1. Anderson, E., Hobbs, M., Keeton, K., Spence, S., Uysal, M., Veitch, A.: Hippodrome: running circles around system administration. Conference on File and Storage Technologies (FAST'02), Monterey, CA, Jan. (2002)
2. Babaoglu, O., Jelasity, M., Montresor, A.: Grassroots Approach to Self-Management in Large-Scale Distributed Systems. In Proceedings of the EU-NSF Strategic Research Workshop on Unconventional Programming Paradigms, Mont Saint-Michel, France, 15-17 September (2004)
3. Barford, P., Crovella, M.: Generating Representative Web Workloads for Network and Server Performance Evaluation. Proc. 1998 ACM Sigmetrics, Madison, Wisconsin, June 22-26, (1998)
4. Bennani, M., Menascé, D.A.: Assessing the Robustness of Self-managing Computer Systems under Variable Workloads. *Proc. IEEE International Conf. Autonomic Computing (ICAC'04)*, New York, NY, May 17–18, (2004)
5. Chase, J., Goldszmidt, M., Kephart, J.: eds., Proc. First ACM Workshop on Algorithms and Architectures for Self-Managing Systems. San Diego, CA, June 11, (2003)
6. Chase, J., Anderson, D., Thakar, P., Vahdat, A., Doyle, R.: Managing Energy and Server Resources in Hosting Centers. 18th Symp. Operating Systems Principles, Oct. (2001)
7. Diao, Y., Gandhi, N., Hellerstein, J. L., Parekh, S., Tilbury, D. M.: Using MIMO Feedback Control to Enforce Policies for Interrelated Metrics With Application to the Apache Web Server. Proc. IEEE/IFIP Network Operations and Management Symp., Florence, Italy, April 15-19, (2002)
8. Doyle, R., Chase, J., Asad, O., Jin, W., Vahdat, A.: Model-Based Resource Provisioning in a Web Service Utility. Fourth USENIX Symposium on Internet Technologies and Systems, March (2003)
9. Garlan, D., Cheng, S., Schmerl, B.: Increasing System Dependability through Architecture-based Self-repair. Architecting Dependable Systems, R. de Lemos, C. Gacek, A. Romanovsky (eds.), Springer-Verlag, (2003)
10. Jelasity, M., Montresor, A., Babaoglu, O.: A Modular Paradigm for Building Self-Organizing Peer-to-Peer Applications. In Post-Proceedings of ESOP03: International Workshop on Engineering Self-Organising Applications, Lecture Notes in Computer Science, vol. 2977, Springer-Verlag, Berlin (2004).
11. Kermarrec, A.: Self-clustering in Peer-to-Peer overlays. In International Workshop on Self-* Properties in Complex Information Systems, Bertinoro, Italy, February (2004) 89-92
12. Menascé, D. A., Almeida, V.A.F., Dowdy, L.W.: *Performance by Design: Computer Capacity Planning by Example*, Prentice Hall, Upper Saddle River, NJ, (2004)
13. Menascé, D.A., Casalicchio, E.: A Framework for Resource Allocation in Grid Computing, Proc. 12th Annual Meeting of the IEEE/ACM International Symposium on

Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Volendam, The Netherlands, October 5-7, (2004)
14. Menascé, D.A.: Mapping Service Level Agreements in Distributed Applications. IEEE Internet Computing (2004) September/October , Vol. 8, No. 5, 100-102
15. Menascé, D.A.: Performance and Availability of Internet Data Centers. IEEE Internet Computing (2004) May/June, Vol. 8, No. 3, 94-96
16. Menascé, D.A., Ruan, H., Gomaa, H.: A Framework for QoS-Aware Software Components. *Proc. ACM 2004 Workshop on Software and Performance*, San Francisco, CA, January 14–16, (2004)
17. Menascé, D.A., Bennani, M.: On the Use of Performance Models to Design Self-Managing Computer Systems. Proc. 2003 Computer Measurement Group Conf., Dallas, TX, Dec. 7-12, (2003)
18. Menascé, D.A.: Security Performance. IEEE Internet Computing (2003) May/June, Vol. 7, No. 3, 84-87
19. Menascé, D.A.: Automatic QoS Control. IEEE Internet Computing (2003) January/February, Vol. 7, No. 1, 92-95
20. Menascé, D. A., Dodge, R., Barbará, D.: Preserving QoS of E-commerce Sites through Self-Tuning: A Performance Model Approach. Proc. 2001 ACM Conf. E-commerce, Tampa, FL, Oct. 14-17, (2001)
21. Rayward-Smith, V. J., Osman, I. H., Reeves, C.R., eds, Modern Heuristic Search Methods, John Wiley & Sons, Dec. (1996)
22. Schintke, F., Schutt, T., Reinefeld, A.: A Framework for Self-Optimizing Grids Using P2P Components. Intl. Workshop on Autonomic Computing Systems, Sep. (2003)
23. Sopitkamol, M.: Ranking configuration parameters in multi-tiered e-commerce sites. ACM Sigmetrics Performance Evaluation Review, Dec. (2004), Special Issue in E-commerce and Services.
24. Walsh, W.E., Tesauro, G., Kephart, J.O., Das, R.: Utility Functions in Autonomic Computing. *Proc. IEEE International Conf. Autonomic Computing (ICAC'04)*, New York, NY, May 17–18, (2004)
25. Wickremisinghe, R., Vitter, J., Chase, J.: Distributed Computing with Load-Managed Active Storage. IEEE Int. Symp. High Performance Distr. Computing, July (2002)