Generic Online Optimization of Multiple Configuration Parameters with Application to a Database Server

Yixin Diao[1], Frank Eskesen[1], Steven Froehlich[1], Joseph L. Hellerstein[1], Lisa F. Spainhower[2], and Maheswaran Surendra[1]

[1] IBM T.J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598
[2] IBM Server Group, 2455 South Rd., Poughkeepsie, NY 12601
{diao,eskesen,stevefro,hellers,lisa,suren}@us.ibm.com

Abstract. Optimizing configuration parameters is time-consuming and skills-intensive. This paper proposes a generic approach to automating this task. By generic, we mean that the approach is relatively independent of the target system for which the optimization is done. Our approach uses online adjustment of configuration parameters to *discover* the system's performance characteristics. Doing so creates two challenges: (1) handling interdependencies between configuration parameters and (2) minimizing the deleterious effects on production workload while the optimization is underway. Our approach addresses (1) by including in the architecture a rule-based component that handles interdependencies between configuration parameters. For (2), we use a feedback mechanism for online optimization that searches the parameter space in a way that generally avoids poor performance at intermediate steps. Our studies of a DB2 Universal Database Server under an e-commerce workload indicate that our approach can be effective in practice.

1 Introduction

The advent of e-Commerce has created a need for responsive and cost-effective information technology (IT) services. However, the increasing complexity of computing systems has resulted in a correspondingly larger human effort for system configuration, especially the optimization of configuration parameters. This paper describes a generic approach to automating configuration optimization. The approach is generic in that it is relatively independent of the target system for which the optimization is done.

Enterprise level software, especially middleware, has tens to hundreds of configuration parameters. For example, IBM's DB2 Universal Database Server has approximately 100 to 200 configuration parameters (e.g., buffer pool sizes, time delay for writing commit records, maximum number of database applications). The challenges here are well recognized, as evidenced by efforts such as IBM's autonomic computing initiative to develop self-managing systems [1]. In particular,

M. Brunner and A. Keller (Eds.): DSOM 2003, LNCS 2867, pp. 3–15, 2003.

addressing self-configuration and self-optimization often depends on subtleties in the workload and system configuration. This motivates the need for a generic approach that discovers the performance impact of configuration parameters by interacting with the target system. However, such an approach creates two challenges: (1) handling interdependencies between configuration parameters and (2) minimizing the deleterious effects on production workload while the optimization is underway (e.g., minimize oscillations, avoid large response times).

There are several efforts related to our work. Some researchers have studied the regulation of system resources to achieve policy objectives, especially using control theory [2]. Examples here include controlling buffer length in Internet routers [3] and response times for web service differentiation [4]. The problem we address differs from these efforts in that we seek to optimize a service level metric (e.g., response time) rather than regulate it, which typically means that a search is required. Thus, closer to our current work is [5], who describe a system that performs on-line optimization of a web server by using hill climbing techniques. However, the approach taken requires a detailed knowledge of the system being optimized in order to construct the queueing models. A similar concern arises with the approach in [6] who consider how to maximize profits based on queueing-theoretic formulas. [7] proposes a fuzzy control approach to minimize response time using a combination of feedback control system and qualitative insights into the effect of tuning parameters on QoS. However, only one configuration parameter is considered. [8] presents case studies of using randomly generated configuration settings for application servers. This approach is simple and generates good results for off-line parameter optimization, but the absence of a guided search may turn out to be problematic for online optimization (as discussed in Section 4). A further concern with all of the foregoing is that none of the approaches consider the architecture necessary to support online optimization, especially handling interdependencies between configuration parameters.

This paper describes a generic, online approach to optimizing configuration parameters. The approach builds on recent work in the area of metric discovery, especially the use of the Common Information Model (CIM) to discover metrics and configuration parameters and the architecture necessary to support this [9]. The problem of interdependencies between configuration parameters is addressed by architecting rule-based components on the target system that handle such interdependencies. The challenge of online search is addressed by employing an existing optimization technique that generally avoids poor performance at intermediate steps. We apply our architecture to IBM's DB2 Universal Database Server since automated configuration of databases is a pressing issue [10]. While there have been many efforts in this area (e.g., [11, 12]), our approach differs in that it requires no prior knowledge of the target system being optimized, although we do require access to the sensors (e.g., response time measurements) and effectors (e.g., buffer pool sizes).

The remainder of the paper is organized as follows. Section 2 details our control architecture, and Section 3 describes how we optimize the setting of

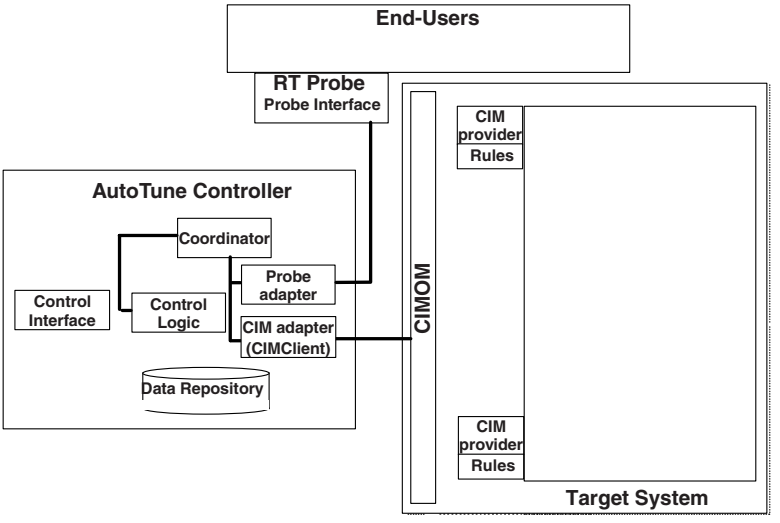

Fig. 1. Diagram of system architecture

configuration parameters. Section 4 presents the testbed setup and experimental results. Our conclusions are contained in Section 5.

2 System Architecture

This section describes the architecture used to support online optimization of configuration parameters.

Figure 1 depicts our architecture. The main elements are the target system being controlled, the AutoTune Controller that dynamically adjusts configuration parameters, and the RT Probe[1] that collects response times from the end-users (the workload). We use the Common Information Model (CIM) [13] to describe the target system (e.g., a database with tables and tablespaces) and the response time probe, especially their metrics (e.g., response times, rows read, sort times) and configuration parameters (e.g., buffer pool sizes). The architecture that we present extends our prior work with metric discovery [9]. Indeed, we discover configuration parameters in the same manner as is done with discovering resource metrics.

The operation of the system is depicted in Figure 2. Every control interval, the AutoTune Controller sends a request to the target system to modify a subset of the configuration parameters. The target system makes these modifications, and the RT probe provides data on the achieved performance. Based on this, the AutoTune Controller computes new values of the configuration parameters.

One point should be underscored in the foregoing. The only way that the AutoTune Controller knows the performance of a setting of configuration pa-

[1] In general, this could be a measurement source for any service level metric.

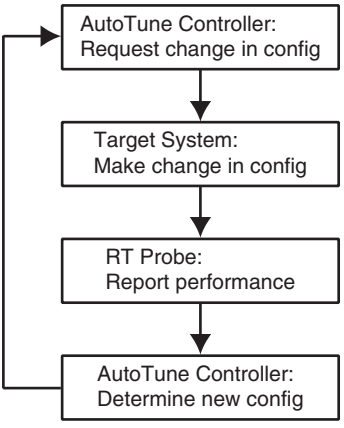

Fig. 2. High level flow of the operation of online optimization of multiple configuration parameters

rameters is to observe the system under those settings. Since this is taking place during normal system operation, caution must be exercised so that there is minimal impact on end-users both in terms of poor performance and the variability of performance.

Now consider the target system. Access to both metrics and configuration parameters is handled by the CIM Object Manager (CIMOM). The CIMOM provides a standard object-oriented interface to this information. The information itself is obtained by various CIM Providers. Note that CIM Providers have rules associated with them to handle conflicts that arise from interdependencies between settings of parameter values. For example, a CIM Provider for memory would handle requests for re-sizing buffer pools that result in demands that exceed memory constraints. That the rules are associated with CIM Providers is appealing in that the specifics of the constraints are known to the providers. A disadvantage is that there may be constraints that involve multiple CIM Providers, although we have not encountered these to date.

Now consider the element schemas used by the CIMOM. The CIM Providers extract data from various parts of the target system (e.g., table spaces, buffer pools) and provide this to the CIMOM. This design is done according to the principle that all descriptive and capability-related information of a managed element is modeled as properties of the class representing the resource itself, while its statistical data is put in an associated class, subclassed from `CIM_StatisticalData`. Specifically: (1) the `controllerName` property is used in multiple controller environments to specify which controller is to be used; (2) the `physicallyEnabled` indicator determines whether the provider is physically capable of setting a control (e.g., the DB2 provider is not capable of setting a control when DB2 is stopped), and (3) the `logicallyEnabled` indicator is intended to be set by the system administrator to override the controller.

The RT Probe provides response time information. Its element schema describes how to operate the probe (e.g., what synthetic transactions can be sent, the resource to which these transactions are sent) and the response times reported (e.g., by transaction type and resource).

The AutoTune Controller is an agent based system that uses the Agent Building and Learning Environment (ABLE) [14]. ABLE is a Java-based toolkit for developing and deploying hybrid intelligent agent applications. Built on top of ABLE is a general AutoTune agent framework that facilitates the construction of control agents by separating the control interface (e.g., probe adaptor, DB2 adaptor) with control logic (e.g., the direct search method described in Section 3). This general and extensible ABLE/AutoTune based controller architecture allows us to easily target our controller to a controlled element (for example, DB2) simply by adding an "adaptor" component that knows how to interface with the controlled element. The control logic typically requires a history of control actions. These data are accumulated by using the control interface to find relevant data for the element (by querying the element schema) and then subscribing to updates of element data (which are then placed in the controller's historical data repository).

3 Optimization Technique

From Figure 2, we see that configuration parameters are changed on an on-going basis. This places two requirements on the approach taken to optimization. First, once a "bad" setting of configuration parameters is identified, we want to move quickly back to "good" settings. Second, we want to minimize the variability in performance due to exploring parameter settings.

Many optimization methods have been studied and applied to solve real world problems (e.g., [15]). However, not all of them are suitable for online optimization. For example, gradient methods (e.g., steepest descent, conjugate gradient, Newton's [16]) are widely used. These techniques evaluate the derivative of performance with respect to configuration parameters, and change parameters in the direction in which they improve performance. However, these techniques are quite sensitive to noise, and noisy data are common in computing systems. Also, obtaining the gradients or Hessians information through approximations and using the line search algorithm to choose the step size require more evaluation samples. They are costly for online optimization. A second class of techniques that are fairly robust to noise are stochastic optimization methods (e.g., random search [8], genetic algorithms) in which randomness is used to evaluate different settings of configuration parameters. However, these approaches can have longer times to converge and more extreme variations in performance during convergence.

Our starting point is the Nelder-Mead simplex method [17, 18, 19], a robust version of a gradient method that has been used with success in many practical optimization problems. This technique is also referred to as a direct search method as it does not actually compute the gradient. Rather, it uses informa-

tion about local minima and maxima to determine the direction in which the parameter space should be searched.

Some notation is introduced for the following discussion. We denote the vector of configuration parameters by $\theta = (\theta_1, \cdots, \theta_n)$. Thus, each θ can be viewed as a point in n-space. Let $J(\theta)$ be the performance of the system if the setting of the configuration parameters is θ. Our objective is to navigate points so that the best performance is achieved in a short time without extreme values of $J(\theta)$ at intermediate configurations. In the following, we assume that *best* means lowest value, as in minimizing response times (although the approach applies to maximizing values as well). That is, we want to find θ^* such that $\min_\theta J(\theta) = J(\theta^*)$. (Technically, what is found is only a local minimum, not a global minimum. This should be fine for most resource allocation problems where the cost function is convex and a local minimum is also a global minimum. However, if the function is not convex, random search or genetic algorithms may perform better.)

Below, we briefly summarize the steps in the direct search method. (1) Initialize: Define a simplex with $n+1$ vertices $\theta(1), \theta(2), \ldots, \theta(n+1)$; (2) Evaluate: Evaluate the performance (i.e., compute $J(\theta)$) of all vertices; (3) Navigate: Several moves should be considered including reflection (mirroring the worst vertex), contraction (converging towards the best vertex), and expansion (moving further along the good direction), which identifies a θ that replaces the worst vertex and forms a new simplex with better performance; (4) Test for completion: If the new simplex is sufficiently small, then the optimization is complete; otherwise, go to Step 2.

Figure 3 illustrates the direct search method for a simulation of two DB2 buffer pools in which response times are obtained from an analytic model. The four plots in the figure represent four iterations of the algorithm. Each contains a contour plot showing how the two configuration parameters affect response time. (The contour plot can be constructed because the example is generated by simulation.) Iteration 0 corresponds to Step 1 above. We see that the simplex is below and to the left of the region of small response times that are in the middle of the range of parameter values plotted. In iteration 1, a new simplex is constructed by applying Step 3 to move towards a region in which response times are smaller. The new simplex is constructed by using the new θ to replace the one with the largest response time. Iteration 2 results in a simplex constructed in a similar manner. In iteration 3, we see that the new point has a larger response time than the one previously encountered, but it is smaller than the vertex with the largest response time in iteration 2.

The online optimization technique that we employ handles some practical considerations from some well-known techniques [18, 19, 20]: (1) Re-evaluate J at the vertices before the simplex is contracted to help convergence in the existence of stochastics, and when the simplex is converges to adapt to workload nonstationarity. (2) Limit and fix the smallest size of the simplex to better handle variations in workload and avoid unnecessary oscillations, which can be costly especially for online optimization of computer configurations. (3) Incorporate constraints on parameter interdependencies to handle boundary conditions. The

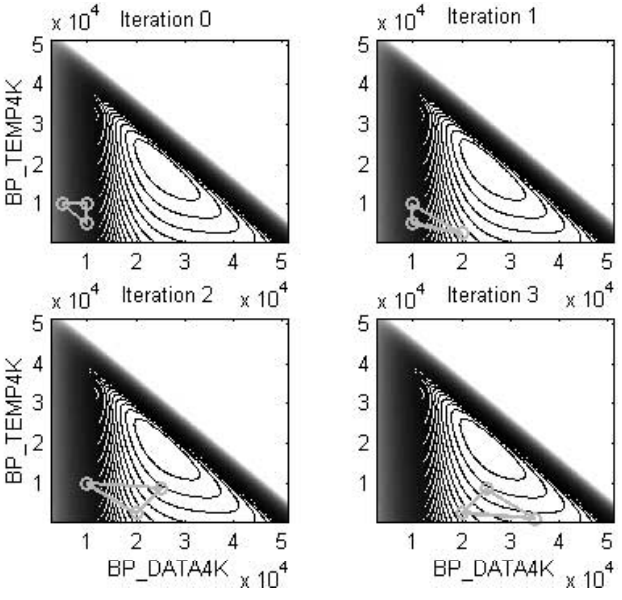

Fig. 3. Illustration of the direct search method for a simulation of two DB2 buffer pools. The contour plots illustrate the response times with axes for buffer pool sizes. The triangle in each plot depicts the simplex used in that iteration of the search

constraints are obtained from the CIM provider, and also enforced by it. For example, the buffer pool size cannot be negative, and the total buffer pool size cannot exceed a limit otherwise the system may crash. Thus, it is a constrained optimization problem. The projection method is used to enforce the parameter constraints.

4 Experimental Assessment

4.1 Testbed Setup

To assess the applicability of our approach to online optimization of configuration parameters, we study it in the context of a database server. IBM's DB2 version 8.1 provides a plethora of tuning parameters that can be changed programmatically in an online environment. Among them are some memory related parameters such as buffer pool size, package cache size, and sort heap size, which have drastic impact on database performance. In this paper, we consider buffer pool tuning.

Our evaluations are done using TPC-W, an industry standard e-commerce benchmark [21]. We used three buffer pools, `BP_INDEX4K` for indexing spaces, `BP_TEMP4K` for cached data, and `BP_DATA4K` for all remaining data. Generally,

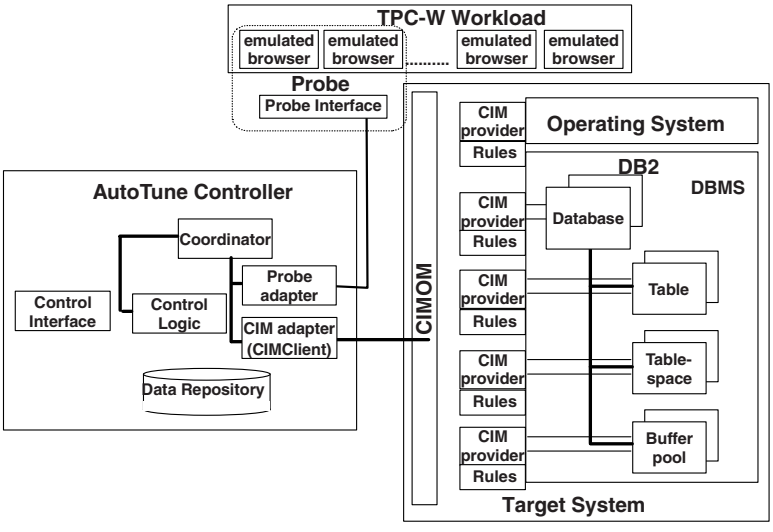

Fig. 4. Diagram of testbed system on which experiments were conducted

having larger buffer pool size results in smaller user response time. However, the total buffer pool size cannot go to infinity due the limited memory space.

Figure 4 displays how we instantiated the architecture in Figure 1. The testbed itself consists of three machines for the database server, the database client, and the AutoTune controller. The database server machine is an IBM RS/6000 model 7044-170 with 768 megabytes of RAM storage. This machine uses the AIX operating system and contained IBM DB2 V8.1 server and the CIM server code including DB2 CIM providers. The database client machine, an IBM RS/6000 model 7044-270 with 1 gigabyte of RAM storage, also uses the AIX operating system. This machine drives the client application using DB2 V8.1 client support. The client application is an emulated browser (EB), an online ordering load generator using TPC-W to emulate database access characteristics of a web e-commerce environment. The AutoTune controller runs on an IBM model T20 ThinkPad with Windows 2000.

Our experiments proceed as follows. A set of 50 emulated browsers (EBs) were running which executed transactions against the database server according to the TPC-W benchmark specifications. A subset (5) of these EBs were also instrumented to provide client side response time for the AutoTune controller. The AutoTune controller used the direct search method to determine the buffer pool sizes and the desired values were sent to the database for real time adjustment. In this environment, while the DB2 client and controller machines were lightly loaded, the DB2 server was always driven at 100% CPU utilization. The control interval was set as 20 minutes. The buffer pool sizes were changed at the start of the interval but the client side response time was measured 5 minutes later (for 15 minutes) to avoid the transient effect of buffer pool resizing.

Table 1. Results of the first six iterations of direct search on the testbed

Sample	BP_DATA4K (4K pages)	BP_INDEX4K (4K pages)	BP_TEMP4K (4K pages)	RT (sec.)
1	29199	23359	2919	15.8
2	14599	23359	2919	17.8
3	29199	11679	2919	16.7
4	29199	23359	1459	19.2
5	19465	15572	4379	15.0
6	14598	11679	5839	18.4

4.2 Experimental Results

Table 1 displays some data obtained from applying direct search to our testbed system. Intuitively, the direct search method operates by evaluating different buffer pool settings within a small region, and moving towards the direction that reduces the response time. Since we use three buffer pools, the initial simplex was composed of 4 vertices and 4 sample buffer pool combinations were selected and evaluated accordingly (as shown in the first four lines in Table 1). The first sample takes the default buffer pool size, and the reset three are defined by halving the size, one at each sample. The constraint is that the total size cannot go beyond 150% of the default total size. Among the four samples, the first sample had minimum response time (15.8 seconds) and the fourth sample had maximum response time (19.2 seconds). Next, we evaluate the reflection point as shown in the fifth sample, which is defined as the mirror point of the maximum point through the centroid. Since its response time (15.0 seconds) is even smaller than the minimum response time we get in the first sample, we evaluate the expansion point as shown in the sixth sample, which was expanded further along this direction. However, the expansion results in higher response time (18.4 seconds). Therefore, according to the direct search algorithm, we complete the first iteration and use the vertices at sample 1, 2, 3, and 5 to compose the simplex for the next iteration.

Figure 5 displays the results of applying direct search until convergence is achieved. In the upper plot, the dashed line indicates the buffer pool size for BP_DATA4K, the dotted line indicates BP_INDEX4K, the dashed-dotted line indicates BP_TEMP4K, and the solid line indicates the total buffer pool size. Note that as the optimizing continues, BP_DATA4K and BP_TEMP4K increase, and BP_INDEX4K decreases. This indicates that indexing space (BP_INDEX4K) need not be so large, but more space is required for cached data (BP_TEMP4K) and the remaining data (BP_DATA4K). During the search there is one instance of a large response time (e.g., the 16th sample) since not all the moves in navigation are heading towards the correct direction. However, the wrong moves will not be continued. Generally, the algorithm convergence time is up to the applications, e.g., the number of configuration parameters, the initial parameter values, the performance function shape, and the convergence criteria. Our experiments indicate a reasonable convergence requires roughly 10*(number of parameters) samples.

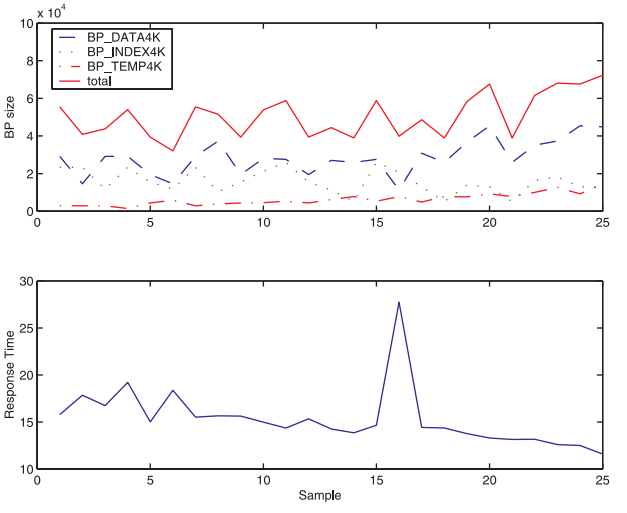

Fig. 5. Experimental results of DB2 buffer pool tuning with the direct search method

It is also interesting to compare direct search with the techniques used in [8] for off-line parameter optimization (but we used it here in an on-line scenario). We conducted experiments in which the total buffer pool size is chosen from a uniform distribution over the range 300,000 to 900,000 4K pages, and then the selected total pool size is randomly sub-divided into three buffer pools. Some of the experimental results are shown in Figure 6. Note that this random approach achieves performance comparable to direct search. However, there is much more variability in performance during the optimization, and considerably longer response times at intermediate values. This comes no surprise as the direct search method chooses the navigation directions based on the sample values that are just evaluated, but the above method goes randomly without guidance.

5 Conclusions

Optimizing configuration parameters to improve performance is time-consuming and skills-intensive. We have proposed an architecture and an algorithm for automating this process through the simultaneous optimization of multiple configuration parameters in a manner that minimizes prior knowledge of the target system (the system being optimized).

Central to our approach is minimizing knowledge of the target system. We accomplish this by *discovering* the effect of configuration parameters on the performance of the target system by making changes to configuration parameters and observing the effects of these changes. Doing so has two implications. First, we must handle interdependencies between configuration parameters (e.g., limits

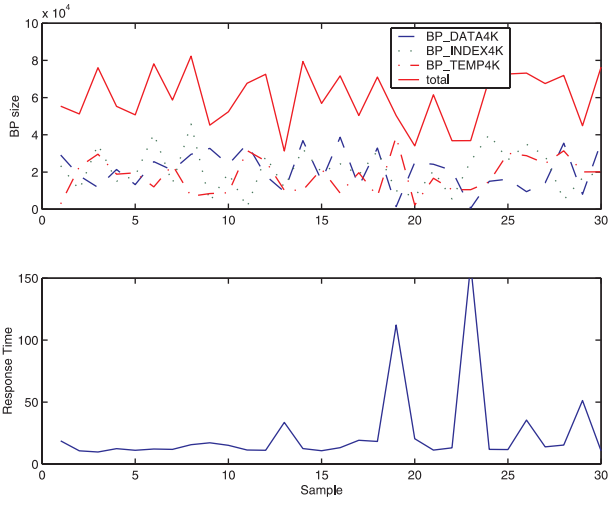

Fig. 6. Experimental results of DB2 buffer pool tuning with the stochastic optimization method

on the total size of a buffer pool). Second, the search for better configurations should be done in a manner that is unlikely to degrade performance excessively and restores performance quickly when it is degraded. Our approach to the first incorporates rules into the CIM Providers of the target system so that changes in configuration parameters are properly constrained. The second is addressed by using the direct search algorithm, a feedback mechanism that exploits the performance function to optimize performance with guidance. This generic approach is expected to be applicable for a wide class of online optimization problems such as configuring server parameters to increase system utilization, differentiating services from different customers, and balancing workload among multi-tiered systems.

In this paper we have applied our architecture and approach to IBM's DB2 Universal Database Server to optimize the setting of buffer pool sizes. This results in approximately a 25% response time reduction in an e-commerce benchmarking environment. We compare these results with an alternative - randomly selecting configuration settings. While the random approach achieves almost the same reduction in response time, its intermediate settings of configuration parameters result in highly variable performance and some very large response times.

Acknowledgements

We would like to thank Dr. Rajarshi Das for many valuable discussions on applying the direct search method for a related Apache Web server optimization problem.

References

[1] IBM, "Autonomic computing: IBM's perspective on the state of information technology," *available at http://www.research.ibm.com/autonomic/*, 2001. 3

[2] G. F. Franklin, J. D. Powell, and A. Emani-Naeini, *Feedback Control of Dynamic Systems*. Reading, Massachusetts: Addison-Wesley, third ed., 1994. 4

[3] C. V. Hollot, V. Misra, D. Towsley, and W. B. Gong, "On designing improved controllers for AQM routers supporting TCP flows," in *INFOCOM*, 2001. 4

[4] C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, S. Son, and M. Marley, "Performance specifications and metrics for adaptive real time systems," in *Proceedings 21st IEEE Real Time Systems Symposium*, pp. 13–24, Nov. 2000. 4

[5] D. Menasce, D. Barbara, and R. Dodge, "Preserving QoS of e-commerce sites through self-tuning: A performance model approach," in *Proceedings of 2001 ACM Conference on E-commerce*, 2001. 4

[6] Z. Liu, M. S. Squillante, and J. L. Wolf, "On maximizing service-level-agreement profits," in *Proceedings of the ACM Conference on Electronic Commerce*, 2001. 4

[7] Y. Diao, J. L. Hellerstein, and S. Parekh, "Optimizing quality of service using fuzzy control," in *Proceedings of Distributed Systems Operations and Management*, 2002. 4

[8] M. Raghavachari, D. Reimer, and R. Johnson, "The deployer's problem: Configuring application servers for performance and reliability," in *Proceedings of the International Conference on Software Engineering, Portland, OR*, 2003. 4, 7, 12

[9] Y. Diao, F. Eskesen, S. Froehlich, J. L. Hellerstein, A. Keller, L. Spainhower, and M. Surendra, "Generic on-line discovery of quantitative models for service level management," in *Proceedings of IEEE/IFIP Symposium on Integrated Network Management*, 2003. 4, 5

[10] G. Weikum, A. Moenkeberg, C. Hasse, and P. Zabback, "Self-tuning database technology and information services: from wishful thinking to viable engineering," in *International Conference on Very Large Data Bases*, 2002. 4

[11] G. M. Lohman and S. S. Lightstone, "Smart: Making db2 (more) autonomic," in *Proceedings of the 28th International Conference on Very Large Data Bases, Hong Kong, China*, 2002. 4

[12] J. Rao, C. Zhang, G. M. Lohman, and N. Megiddo, "Automating physical database design in a parallel database," in *SIGMOD*, 2002. 4

[13] "Common Information Model (CIM) Core Model, Version 2.4," white paper, Aug. 2000. http://www.dmtf.org/var/release/Whitepapers/DSP0111.pdf 5

[14] J. P. Bigus, D. A. Schlosnagle, J. R. Pilgrim, W. N. Mills III, and Y. Diao, "ABLE: A toolkit for building multiagent autonomic systems," *IBM Systems Journal*, vol. 41, no. 3, 2002. 7

[15] D. G. Luenberger, *Linear and nonlinear programming*. Addison-Wesley, Reading, MA, 1984. 7

[16] X. Liu, L. Sha, Y. Diao, S. Froehlich, J. L. Hellerstein, and S. Parekh, "Online response time optimization of apache web server," in *Proceedings of the 11th International Workshop on Quality of Service*, pp. 461–478, 2003. 7

[17] J. A. Nelder and R. Mead, "A simplex method for function minimizatioin," *Computer Journal*, 1965. 7

[18] F. H. Walters, J. L. R. Parker, S. L. Morgan, and S. N. Deming, *Sequential Simplex Optimization: A technique for improving quality and productivity in research, development, and manufacturing*. CRC Press, 1991. 7, 8

[19] C. H. Brooks, "An introduction to amoeba," *available at http://nexus.cs.usfca.edu/~brooks/papers/amoeba.pdf* 7, 8
[20] J. O. Kephart, R. Das, and J. K. MacKie-Mason, "Two-sided learning in an agent economy for information bundles," in *AmEC IJCAI*, 1999. 8
[21] W. D. Smith, "TPC-W: Benchmarking an ecommerce solution," in *http://www.tpc.org/tpcw* 9