Using MIMO Linear Control for Load Balancing in Computing Systems

Yixin Diao, Joseph L. Hellerstein, Adam J. Storm, Maheswaran Surendra,
Sam Lightstone, Sujay Parekh, and Christian Garcia-Arellano

***Abstract*— Load balancing is widely used in computing systems as a way to optimize performance by equalizing loads to reduce delays, such as adjusting the size of memory pools to balance resource demands in a database management system. Load balancing is generally approached as a nonlinear constrained optimization in which dynamics are ignored. We approach load balancing differently - as a feedback controller design problem using a multiple input multiple output linear quadratic regulator (LQR) that achieves the constrained optimization objective. Such an approach allows us to exploit well established techniques for handling disturbances (e.g., due to changes in workloads) and to incorporate the cost of control (e.g., throughput reductions due to resizing buffer pools) by properly selecting the LQR Q and R matrices. From studies of DB2 Universal Database Server using industry standard benchmarks, we show that the controller obtains a factor of three increase in throughput for an OLTP workload and a 59% reduction in response times for a DSS workload.**

I. Introduction

A central concern of businesses with mission critical computing is quality of service, the ability to manage utilization of computing resources so as to maintain response times and throughputs at acceptable levels. One widely used technique for achieving better resource utilization is load balancing whereby incoming requests and/or system resources are balanced so as to minimize the utilization of the bottleneck resource, a technique that generally minimizes response times and maximizes throughputs. Existing approaches to load balancing do not consider dynamics such as the effect of disturbances (e.g., changes in workload) and the cost of control actions (e.g., overheads for changing the sizes of buffer pools). With the growth of the Internet, dynamics have become increasingly important, especially the challenges associated with "flash events" that can cause very rapid changes in workloads [1]. This paper describes the use of a multiple input multiple output (MIMO) linear quadratic regulation (LQR) controller for load balancing that addresses dynamics.

We begin with an example of load balancing. Figure 1 shows the architecture of a database server that provides load balancing across multiple memory pools which can be of different types and characteristics. The system operates as follows. Database clients interact with the database server through the database agents which are computing elements that access copies of disk data in the memory pools. The memory pools are used to cache disk pages to reduce the number of disk input/output operations for the agents. Central to the performance of a database management system is the management of its memory pools. Increasing the size of a memory pool can dramatically reduce the time to access disk data since there is a higher probability that a copy of the data is cached in memory. We refer to this reduction in disk response time obtained from an increase in memory allocation as the **response time benefit** (or just benefit) in the measure of saved disk response time per unit memory increase. Since the total size of the memory pools is fixed, increasing the size of one pool necessarily means decreasing the size of another pool. The memory tuner adjusts pool allocations with the intent of reducing overall disk response time for data access. An intuitive approach is to allocate memory to pools so that each has the same benefit, which is a kind of load-based balancing. We also note in passing that no matter what scheme is used to dynamically allocate memory, care must be taken to not change memory pools too frequently since excessive adjustments introduce substantial resizing overheads that can decrease throughput and increase response time.

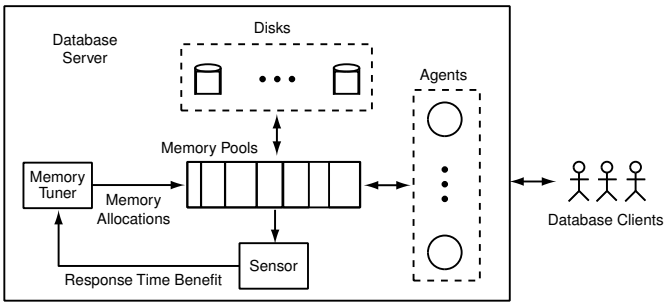


Fig. 1. *Architecture of load balancing for a database server. The server processes query requests by database clients for information on disk. Memory pools are used to cache the disk information to reduce the number of disk input/output operations. The memory tuner automatically adjusts the size of memory pools to optimize performance.*

Many researchers have addressed online optimization of computing systems. [2] describes a system that performs on-line optimization of a web server using hill climbing techniques. [3] considers how to maximize server farm profits based on queueing-theoretic formulas. [4] proposes a fuzzy control approach to minimize web server response time. More generally, constrained optimization has been

Y. Diao, J. L. Hellerstein, M. Surendra, and S. Parekh are with IBM Thomas J. Watson Research Center, Hawthorne, New York, USA.

A. J. Storm, S. Lightstone, and C. Garcia-Arellano are with IBM Toronto Lab, Markham, Ontario, Canada.

0-7803-8335-4/04/$17.00 ©2004 AACC

addressed using techniques such as constraint programming, genetic algorithms, and heuristic search [5], [6], [7]. Unfortunately, none of these approaches consider the dynamics of computing systems, and the overheads of adjusting configuration parameters. On the other hand, the control literature contains extensive studies of dynamic systems with constraints in which optimization is done. However, the control objective is typically regulation or tracking (but not optimizing resource allocations in computing systems), and the optimization technique is applied to control performance indexes (e.g., minimize the tracking error) [8], [9], [10], [11]. Recently, in computer science literature several researchers have employed control theory to analyze system dynamics including flow and congestion control [12], [13], differentiated caching and web service [14], multimedia streaming [15], web server performance [16], and email server control [17], [18]. However, none of these approaches address load balancing and optimization of computing systems resources. There is a vast literature on load balancing, including its use in multiple source routing [19], memory allocation [20], implementations for L4 switches [21], techniques for balancing loads in data warehouses [22], agent based algorithms [23], and redirection algorithms for web-server systems [24]. While the focus of these efforts is more on load balancing in a specific context, there have also been studies that analyze general strategies, especially static load balancing (which makes use of long-term trends) versus dynamic load balancing (which exploits current changes in state) [25]. However, none of these studies consider the impact of transient load imbalances and the overheads of changing resource allocations.

This paper proposes the use of a MIMO LQR controller for load balancing in computing systems. Our approach is structured so that load balancing is equivalent to constrained performance optimization, and is equivalent to driving the control error to zero. The controller and its design provide a way to incorporate system dynamics, to consider the cost of control actions, and to analyze the effects of disturbances (e.g., variations in workload). We show that our approach works well in practice through experimental studies of load balancing memory allocations in IBM's DB2 Universal Database Enterprise Server Edition for Linux, Unix and Windows Version 8.1.

The remainder of the paper is organized as follows. Section II describes the load balancing problem and justifies the use of this heuristic for a database management system. Section III details the control architecture and algorithms. Section IV assesses our controller for a DB2 Universal Database Server. The conclusions are contained in Section V.

II. Problem Formulation

This section formulates the load balancing problem and shows that load balancing provides a way to optimize memory allocations in a database management system.

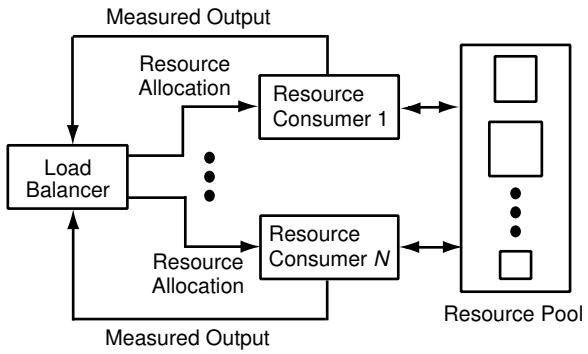
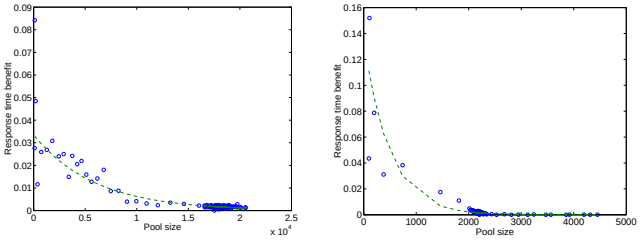


Fig. 2. *General architecture for load balancing and resource allocation. The load balancer controls the pool of resources, assigning instances to consumers in a way so as to equalize the value of a measured output (e.g., response time, utilization).*

Figure 2 depicts a general architecture for load balancing and resource allocation. The resource pool contains multiple instances of a resource type (e.g., system resource such as memory, CPU, disk, or load resource such as customer requests). The instances need not be identical (e.g., CPUs with different speeds or memory pages with different size) and the number of instances can be infinite. There are finite N consumers of the resource type, each of which receives allocations of the resource from the load balancer and provides to the load balancer one or more measured outputs that quantify the consumers performance (e.g., response time, utilization). The load balancer allocates instances of the resource type in a way that equalizes the measured outputs of the resource consumer. For the example of server farm, the resource type is web requests, the consumers are web servers, and the measured output is server utilization. In the database server example, one possible resource type is memory (to provide faster access to disk pages) and the consumers are memory pools such as buffer pools for caching data and index pages, and sort memory for performing in-memory sorting of disk pages. The measured output can be response time, throughput, or utilization. However, we choose the response time benefit (obtained as a result of the change memory allocation to reduce the costly disk I/O time) to be the measured output, so that load balancing leads to optimal memory performance.

We start from modeling the relationship between resource allocations and measured outputs. In some cases, this can be a linear relationship. For example, linearity holds if the resource pool consists of requests for server and the measured output is web server throughput and the request arrival rate is smaller than the service rate. (Of course, if the request arrival rate is always larger than the service rate, then the server utilization is 1 and the slope of this linear relationship becomes 0.) However, the linearity assumption does not always hold in general. Consider the relationship between the memory pool size in a database server and the response time benefit. Figure 3 depicts this relationship for experiments done on an IBM DB2 Universal



(a) A data buffer pool *(b) An index buffer pool*

Fig. 3. *Example empirical models of the response time benefit of DB2 buffer pools of caching data pages and index pages for an online transaction processing workload.*

Database Server with an industry standard benchmark for two memory pools. (Details of the experimental setup are described in Section IV). The horizontal axis is the size of the memory pool in the unit of 4K page, the vertical axis is the response time benefit of adding an increment of memory in the unit of second per 4K page, and the circles indicate results obtained from testbed experiments. We assume the relationship between the pool size and the saved disk response time is approximately $x_i = a_i(1-e^{-b_i u_i})$ so that the relationship between the pool size and the response time benefit is $y_i = \frac{dx_i}{du_i} = a_i b_i e^{-b_i u_i}$ where u_i denotes the memory allocated to pool i, x_i denotes the saved disk response time for pool i, y_i denotes the response time benefit for pool i, and a_i, b_i are model parameters that can be obtained empirically. The modeling results are shown in Figure 3 by the dashed lines.

We now define what is meant by a **load balancing problem (LBP)**. Let $u_1, \cdots, u_N$ be resource allocations for N resource consumers and let $y_1, \cdots, y_N$ be their measured outputs. The LBP is to choose the u_i such that $y_1 = \cdots = y_N$ subject to the constraint that $\sum u_i = U$ where U defines the total resource. Note that the sum of changes in resource allocations is always 0.

The motivation for load balancing is that the performance of computing systems is largely affected by the most loaded resource. Thus, by equalizing the (appropriately chosen) measured outputs, we hope to optimize the performance. Such an effect is clear in Figure 3. For example, suppose that the size of the data buffer pool is 15,000 pages, and the size of the index buffer pool is 500 pages. The corresponding response time benefits are 0.002 and 0.05 seconds per page, respectively. If we deallocate 100 pages of memory from the data buffer pool and allocate this memory to the index buffer pool, the benefits from the two pools will be more close to each other. From the view of total response time, doing so slightly increases the response time for accessing the data buffer pool (a increase of 0.2 seconds as the integral of benefit from 14,900 to 15,000), but this is more than offset by the large decrease in response for accesses to the index buffer pool (a decrease of 5 seconds as the integral of benefit from 500 to 600). More generally, such effects are common in queueing systems for actions that affect resource utilizations (e.g., allocation CPU shares) and measured outputs such as response time and number in system.

While load balancing is a widely used heuristic in computing systems, in some cases it results in the optimal solution, specifically, when the objective function is composed of a sum of resource consumer metrics and the measured outputs for the load balancer are the derivatives of such metrics. This turns out to be the case for the database memory management problem with models defined above. The optimization problem is to maximize the total saved response time f, or $\max_{u_1,\cdots,u_N} f$ where $f = \sum_{i=1}^{N} x_i = \sum_{i=1}^{N} a_i(1 - e^{-b_i u_i})$ subject to the constraint of the total available memory $\sum_{i=1}^{N} u_i = U$. To see this, note that $f = \sum_{i=1}^{N-1} x_i + x_N = \sum_{i=1}^{N-1} a_i(1 - e^{-b_i u_i}) + a_N(1 - e^{-b_N(U-\sum_{i=1}^{N-1} u_i)})$ and $\max f$ can be found where the gradient is zero, that is, $\frac{\partial f}{\partial u_i} = a_i b_i e^{-b_i u_i} + \frac{\partial a_N(1-e^{-b_N(U-\sum_{i=1}^{N-1} u_i)})}{\partial u_i} = a_i b_i e^{-b_i u_i} - a_N b_N e^{-b_N u_N} = y_i - y_N = 0$ for $i = 1, 2, \ldots, N-1$. This gives the optimal memory setting at $y_i = y_j$ for $i, j = 1, 2, \ldots, N$. Since f is a convex function of $\mathbf{u}$, the optimal solution is unique, i.e., the local optimum is also the global optimum.

III. MIMO Controller for Load Balancing

Current approaches to load balancing (e.g., gradient methods [5] and simplex methods [26]) do not consider the dynamics of resource allocations, an omission that can impair performance (e.g., oscillation due to overreaction and negligence of the control delay). Here, we treat load balancing as a MIMO control problem, an approach that does address dynamics. By dynamics of load balancing, we mean the trajectory of resource allocations. There are two costs associated with this trajectory. The first is the cost of a suboptimal resource allocations. A memory allocation with unbalanced response time benefit would result in longer system response time and less server throughput. The second cost is that of the control actions, such as changes in memory allocations, which consume resource and reduce throughput as well.

To design a MIMO controller for load balancing, we model the plant as the set of resource consumers with control inputs $u_1(k), \cdots, u_N(k)$ and measured outputs $y_1(k), \cdots, y_N(k)$, where k indicates the unit of discrete time. System identification is used to approximate a local linear model in a desired operating region. Two types of disturbances are considered: the perturbation of the resource allocations $d_i^I(k)$ that occurs on input to the resource consumer, and the distortion of the measured output $d_i^O(k)$. The measured output of the i-th resource consumer is $w_i(k) = y_i(k) + d_i^O(k)$. We define the average measured output $\bar{w}(k) = \frac{1}{N}\sum_i w_{i=1}^{N}(k)$ as the control reference. The i-th control error at time k is $e_i(k) = \bar{w}(k) - w_i(k)$. The control objective is to make $e_i(k) = 0$, i.e., balancing the measured outputs so as to maximize the total saved response time. Note that different to using a static value or external

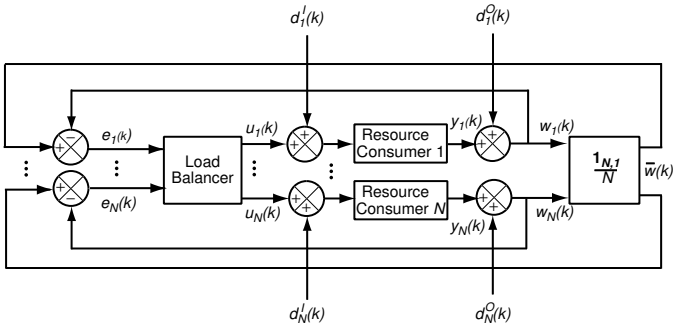


Fig. 4. *Block diagram of using MIMO control for load balancing with disturbances.*

signal as the reference, we specify the reference as a linear transformation of measured outputs. Figure 4 displays a block diagram of this system in which a MIMO controller acts as a load balancer to control resource allocations for N resource consumers. Clearly, $\sum_{i=1}^{N} e_i(k) = 0$. Also note that $\bar{w}(k) = \begin{bmatrix} w_1(k) & \cdots & w_N(k) \end{bmatrix} \frac{1}{N} \mathbf{1}_{N,1}$, where $\mathbf{1}_{N,1}$ is a vector of N 1's.

We formulate the control problem using a state space model and design a state feedback controller to achieve the load balancing objective. Consider the following state space model for the database memory system

$$\begin{aligned} \mathbf{y}(k+1) &= \mathbf{A}\mathbf{y}(k) + \mathbf{B}(\mathbf{u}(k) + \mathbf{d}^I(k)) \quad (1) \\ \mathbf{e}(k) &= (\frac{1}{N}\mathbf{1}_{N,N} - \mathbf{I})(\mathbf{y}(k) + \mathbf{d}^O(k)) \quad (2) \end{aligned}$$

where $\mathbf{y}(k)$ and $\mathbf{u}(k)$ are $N \times 1$ vectors representing the measured output (e.g., response time benefit) and resource allocation (e.g., memory pool size) in Figure 2. (Note that although database memory allocation is known to be a nonlinear system, a linear approximation around the operating region appears to be accurate enough regarding to the overall control performance as shown in later experimental studies.) The vectors $\mathbf{e}(k)$, $\mathbf{d}^I(k)$, and $\mathbf{d}^O(k)$ are $N \times 1$ vectors representing the control error, control disturbance, and measurement disturbance. The $N \times N$ matrices $\mathbf{A}$ and $\mathbf{B}$ contain model parameters from system identification. $\mathbf{I} = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 1 \end{bmatrix}$, $\mathbf{1}_{N,N} = \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$ are $N \times N$ matrices used to compute the benefit difference.

Load balancing is achieved by eliminating the difference between the measured outputs of all resource consumers. This occurs if $e_1(k) = \cdots = e_N(k) = 0$. We design a dynamic state feedback controller to do this. The system state is augmented by the integral of error $\mathbf{e_I}(k+1) = \mathbf{e}_I(k) + \mathbf{e}(k)$ and the state feedback controller is defined as $\mathbf{u}(k) = \mathbf{K}\mathbf{e}_I(k)$, so that the closed loop system is

$$\begin{aligned} \mathbf{y}(k+1) &= \mathbf{A}\mathbf{y}(k) + \mathbf{B}\mathbf{K}\mathbf{e}_I(k) + \mathbf{B}\mathbf{d}^I(k) \\ \mathbf{e}_I(k+1) &= \mathbf{e}_I(k) + (\frac{1}{N}\mathbf{1}_{N,N} - \mathbf{I})(\mathbf{y}(k) + \mathbf{d}^O(k)) \\ &= (\frac{1}{N}\mathbf{1}_{N,N} - \mathbf{I})\mathbf{y}(k) + \mathbf{e}_I(k) + (\frac{1}{N}\mathbf{1}_{N,N} - \mathbf{I})\mathbf{d}^O(k) \end{aligned}$$

Load balancing can be achieved when the system reaches the steady state (i.e., $\mathbf{e}_I(k+1) = \mathbf{e}_I(k)$ and so $\mathbf{e}(k) = 0$). There is, however, a significant shortcoming with the foregoing controller. There may exist disturbances for which steady state is not achieved. To see this, note that in the steady state we have

$$\begin{bmatrix} \mathbf{I} - \mathbf{A} & -\mathbf{B}\mathbf{K} \\ -\frac{1}{N}\mathbf{1}_{N,N} + \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{y_{ss}} \\ \mathbf{e_{I,ss}} \end{bmatrix} = \begin{bmatrix} \mathbf{B}\mathbf{d}^I(k) \\ (\frac{1}{N}\mathbf{1}_{N,N} - \mathbf{I})\mathbf{d}^O(k) \end{bmatrix}$$

where $\begin{bmatrix} \mathbf{I} - \mathbf{A} & -\mathbf{B}\mathbf{K} \\ -\frac{1}{N}\mathbf{1}_{N,N} + \mathbf{I} & \mathbf{0} \end{bmatrix}$ is singular since $(\frac{1}{N}\mathbf{1}_{N,N} - \mathbf{I})$ is singular. Hence, $\mathbf{y_{ss}}$ and $\mathbf{e_{I,ss}}$ may not exist for some $\mathbf{d}^I(k)$ and $\mathbf{d}^O(k)$.

To make the plant controllable we must consider the constraint $\sum_{i=1}^{N} u_i = U$. Thus, we redefine the state space model to only include $N-1$ states and inputs and the N^{th} resource consumer is treated as external whose input can be obtained from the constraint.

$$\begin{aligned} \mathbf{y}(k+1) &= \mathbf{A}\mathbf{y}(k) + \mathbf{B}(\mathbf{u}(k) + \mathbf{d}^I(k)) \\ \mathbf{e}(k) &= (\frac{1}{N}\mathbf{1}_{N-1,N-1} - \mathbf{I})(\mathbf{y}(k) + \mathbf{d}^O(k)) \\ &\quad + \frac{1}{N}\mathbf{1}_{N-1,1}(y_N(k) + d_N^O(k)) \\ y_N(k+1) &= a_N y_N(k) + b_N(U - \mathbf{1}_{1,N-1}\mathbf{u}(k) + d_N^I(k)) \end{aligned}$$

where $\mathbf{y}(k)$ and $\mathbf{u}(k)$ are $(N-1) \times 1$ vectors, and $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{I}$ are $(N-1) \times (N-1)$ matrices. We design the feedback controller for the $N-1$ resource allocations $\mathbf{u}(k)$ with $\mathbf{u}(k) = \mathbf{K}\mathbf{e}_I(k)$ where the integral of control error is defined on the $N-1$ resource consumers with $\mathbf{e}_I(k+1) = \mathbf{e}_I(k) + \mathbf{e}(k)$ as before. For the N^{th} resource allocation, $u_N(k) = U - \sum_{i=1}^{N-1} u_i(k)$. The closed loop system is

$$\begin{aligned} \mathbf{y}(k+1) &= \mathbf{A}\mathbf{y}(k) + \mathbf{B}\mathbf{K}\mathbf{e}_I(k) + \mathbf{B}\mathbf{d}^I(k) \\ \mathbf{e}_I(k+1) &= (\frac{1}{N}\mathbf{1}_{N-1,N-1} - \mathbf{I})(\mathbf{y}(k) + \mathbf{d}^O(k)) + \mathbf{e}_I(k) \\ &\quad + \frac{1}{N}\mathbf{1}_{N-1,1}(y_N(k) + d_N^O(k)) \end{aligned}$$

which gives

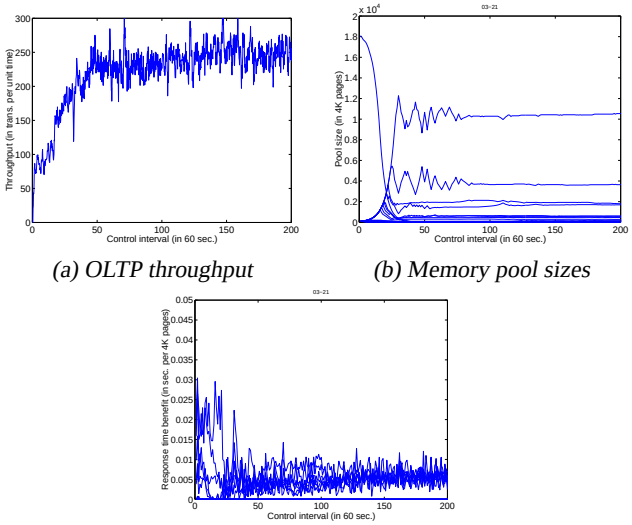
$$\begin{bmatrix} \mathbf{I} - \mathbf{A} & -\mathbf{B}\mathbf{K} \\ -\frac{1}{N}\mathbf{1}_{N-1,N-1} + \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{y_{ss}} \\ \mathbf{e_{I,ss}} \end{bmatrix} = \begin{bmatrix} \mathbf{B}\mathbf{d}_I(k) \\ (\frac{1}{N}\mathbf{1}_{N-1,N-1} - \mathbf{I})\mathbf{d}^O(k) + \frac{1}{N}\mathbf{1}_{N-1,1}(y_N(k) + d_N^O(k)) \end{bmatrix}$$

Note that $\begin{bmatrix} \mathbf{I} - \mathbf{A} & -\mathbf{B}\mathbf{K} \\ -\frac{1}{N}\mathbf{1}_{N-1,N-1} + \mathbf{I} & \mathbf{0} \end{bmatrix}$ can be designed to be non-singular since $-\frac{1}{N}\mathbf{1}_{N-1,N-1} + \mathbf{I}$ is non-singular.

We handle the trade-off between short settling time and overreacting to random fluctuations by choosing control gains based on a cost function, i.e., linear quadratic regulator (LQR) design. LQR finds the control gains that minimize the quadratic cost function

$$J = \sum_{k=1}^{\infty} \begin{bmatrix} \mathbf{e}^\top(k) & \mathbf{e}_I^\top(k) \end{bmatrix} \mathbf{Q} \begin{bmatrix} \mathbf{e}(k) \\ \mathbf{e}_I(k) \end{bmatrix} + \mathbf{u}(k)^\top \mathbf{R}\mathbf{u}(k) \quad (3)$$

The cost function includes the control errors $\mathbf{e}(k)$, the integral of errors $\mathbf{e}_I(k)$, and the control inputs $\mathbf{u}(k)$. We



(a) OLTP throughput *(b) Memory pool sizes*

(c) Response time benefits

Fig. 5. *Load balancing performance for MIMO control under an OLTP workload. The memory pool sizes and response time benefits for 20 buffer pools are indicated by 20 solid lines, respectively.*

choose the matrices $\mathbf{Q}$ and $\mathbf{R}$ based on empirical studies of the effect on performance of the control error and the variability of the control input. In the case where having a response time benefit distribution with a standard deviation of 0.01 causes throughput to decrease by 50 transactions per second and having a memory pool allocations with a standard deviation of 100 causes throughput to decline by 50 transactions per second, we set $\mathbf{Q} = 50^2/0.01^2\mathbf{I}$, and $\mathbf{R} = 50^2/100^2\mathbf{I}$.

IV. Testbed Experiments

This section describes experiments done to assess the MIMO control approach to load balancing.

A. Testbed Setup

The main components of the testbed are the database server and the workload generator. For the database server we use IBM's DB2 version 8.1, a system provides programmatic access to changing the sizes of memory pools such as the database buffer pools and sort heap.

We use two industry standard benchmarks to provide workload generation: an OLTP workload for on-line transaction processing and a DSS workload for decision support. The OLTP workload consists of a large number of concurrent requests, each of which has very modest resource demands. We use 20 buffer pools to contain data and index for the database tables and 50 database clients to generate load. A DSS workload has a small number of long-running, resource intensive requests that are highly variable in their demands.

B. Controller Assessments

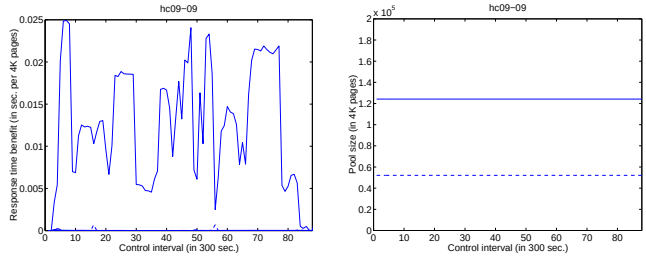
We first study the capabilities of the MIMO load balancing controller for managing 20 buffer pools under the OLTP workload. We also study the performance impact of buffer pool re-sizings by verifying the OLTP throughput (measured in transactions per unit time). Figure 5 (a) shows the throughput, and Figure 5 (b) and (c) displays the memory allocations and response time benefits for the controlled 20 buffer pools (as indicated by 20 solid lines in the plot). Initially, the database memory is not properly allocated: most of the memory has been allocated to one buffer pool, while the other buffer pools are set at the minimum size. The controller adjusts the size of buffer pools so as to equalize the response time benefits of the pools. Put differently, the controller seeks buffer pool sizes that equalize the marginal value of adding memory to a pool. We see that the controller converges around 80 intervals. Also, observe that the effect of the controller's actions is to increase throughput by a factor of three.

Next, we consider the DSS workload. Unlike the OLTP workload, the DSS workload consists of a small number of queries whose durations and resource consumptions are highly variable. In addition to buffer pool memory, allocating sort memory is crucial since a significant amount of sort operations are involved in the long-running queries. Initially, we allocated memory in the buffer pools by consulting with database experts to obtain the typical settings of DB2 memory pools. It turns out that these settings are a poor choice for the DSS workload, as is evident by the measurement results displayed in Figure 6. Observe in the figure that the benefit values are much larger for the data buffer pool (with a solid line) than those for the other pools (e.g., sort memory with a dashed line). (Several other memory pools such as package cache are also included, but they are much less important than the buffer pool and sort memory so that their sizes and benefits are invisible in this figure.) This imbalance strongly suggests that a better allocation of memory could result in substantial performance improvements.

Figure 7 contains the results of using the MIMO controller with the same DSS workload as in Figure 6. Note that the MIMO controller produces benefit values that are consistently smaller and closer to each other than those in Figure 6. This is because the MIMO controller allocates considerably more memory to the data buffer pool represented by the solid line. This results in a significant reduction in total query time, which is the sum of the completion times of all completed queries (the standard metric used for the DSS workload). The total query time for the experiment with static settings is 26342 seconds; for the MIMO controller, the total query time is 10680 seconds. That is, the MIMO controller reduces total query time by 59%.

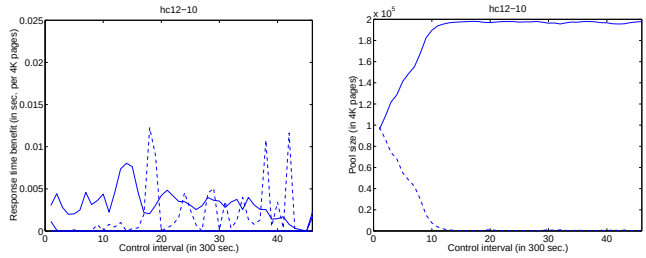
V. Conclusions

Load balancing is widely used to provide scalable growth of computing systems. To date, this has been done without consideration of dynamics such as disturbances and the cost of control actions. Although traditionally an optimization



(a) Response time benefit *(b) Memory pool sizes*

Fig. 6. *Load balancing performance for expert-suggested constant memory pool settings under a DSS workload. Two memory pools are indicated by the solid line and dashed line, respectively.*



(a) Response time benefit *(b) Memory pool sizes*

Fig. 7. *Load balancing performance for MIMO control under a DSS workload. Two memory pools are indicated by the solid line and dashed line, respectively.*

problem, we approach load balancing using a MIMO linear quadratic regulator and appropriate control loop design (e.g., specification of reference inputs). We show how the LQR parameters can be interpreted as control costs, and the MIMO LQR controller can yield significant performance improvement, as evidenced by studies of a DB2 Universal Database Server using industry standard benchmarks. In particular, the controller obtains a factor of three increase in throughput for an OLTP workload and a 59% reduction in response times for a DSS workload.

Acknowledgements

The authors thank Matthew Carroll, Lee Chu, Jerome Colaco, Frank Eskesen, and Steven Froehlich for their assistance with DB2 testbeds.

References

[1] E. Lassettre, D. W. Coleman, Y. Diao, S. Froehlich, J. Hellerstein, L. Hsiung, M. R. T. Mummert, G. Parker, L. Russell, M. Surendra, V. Tseng, N. Wadia, and P. Ye, "Dynamic surge protection: An approach to handling unexpected workload surges with resource actions that have lead times," in *Proceedings of Distributed Systems Operations and Management*, Oct. 2003.

[2] 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.

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

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

[6] T. Lau and E. Tsang, "The guided genetic algorithm and its application to the generalized assignment problem," in *Proceedings of the Tenth IEEE International Conference on Tools with Artificial Intelligence, Taipei, Taiwan*, pp. 336–343, 1998.

[7] P. Dasgupta, P. Chakrabarti, A. Dey, S. Ghose, and W. Bibel, "Solving constraint optimization problems from clp-style specifications using heuristic search techniques," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 2, pp. 353–368, 2002.

[8] M. Athans and P. Falb, *Optimal Control: An Introduction to the Theory and its Applications*. McGraw-Hill Book Company, New York, 1966.

[9] S. Lyashevskiy, "Control of linear dynamic systems with constraints: optimization issues and applications of nonquadratic functionals," in *Proceedings of the 35th IEEE Conference on Decision and Control, Kobe, Japan*, pp. 3206–3211, 1996.

[10] A. Matveev, "Application of linear-quadratic control theory to the solution of special nonconvex problems of global constrained optimization," in *Proceedings of the American Control Conference, Seattle, WA*, pp. 3928–3932, 1995.

[11] J. Rossiter, B. Kouvaritakis, and J.R.Gossner, "Feasibility and stability for constrained stable predictive control," in *Proceedings of the Third IEEE Conference on Control Applications, Glasgow, UK*, pp. 1885–1890, 1994.

[12] S. Mascolo, "Classical control theory for congestion avoidance in high-speed internet," in *Proceedings of the 38th Conference on Decision & Control*, Dec. 1999.

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

[14] Y. Lu, A. Saxena, and T. F. Abdelzaher, "Differentiated caching services: A control-theoretic approach," in *International Conference on Distributed Computing Systems*, Apr. 2001.

[15] B. Li and K. Nahrstedt, "Control-based middleware framework for quality of service applications," *IEEE Journal on Selected Areas in Communication*, 1999.

[16] Y. Diao, N. Gandhi, J. L. Hellerstein, S. Parekh, and D. M. Tilbury, "Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache web server," in *Proceedings of Network Operations and Management*, 2002.

[17] S. Parekh, N. Gandhi, J. L. Hellerstein, D. M. Tilbury, and J. P. Bigus, "Using control theory to achieve service level objectives in performance management," in *Proceedings of IEEE/IFIP Symposium on Integrated Network Management*, 2001.

[18] Y. Diao, J. L. Hellerstein, and S. Parekh, "A business-oriented approach to the design of feedback loops for performance management," in *Distributed Systems Operations and Management*, 2001.

[19] L. Zhang, Z. Zhao, Y. Shu, L. Wang, and O. Yang, "Load balancing of multipath source routing in ad hoc networks," in *International Conference on Communications*, 2002.

[20] M. Mitzenmacher, B. Prabhakar, and D. Shah, "Load balancing with memory," in *Proceedings of IEEE Symposium on Foundations of Computer Science*, 2002.

[21] J. Hyun, I. Jung, J. Lee, and S. Maeng, "Content sniffer based load distribution in a web server cluster," *IEICE Transactions on Information and Systems*, vol. E86-D, no. 7, 2003.

[22] H. Marteins, E. Rahm, and T. Stohr, "Dynamic query scheduling in parallel data warehouses," *Concurrency Computation Practice and Experience*, vol. 15, no. 11-12, 2003.

[23] S. Desic and D. Huljenic, "Agents based load balancing with component distribution capability," in *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 327 –331, 2002.

[24] V. Cardellini, M. Colajanni, and P. S. Yu, "Request redirection algorithms for distributed web systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 4, pp. 355–368, 2003.

[25] H. Kameda, E.-Z. S. Fathy, I. Ryu, and J. Li, "A performance comparsion of dynamic vs. static load balancing policies in a mainframe – peronal computer network model," in *Proceedings of the 39th IEEE Conference on Decision and Control*, IEEE, 2000.

[26] F. H. Walters, J. L. R. Parker, S. L. Morgan, and S. N. Deming, *Sequential Simplex Optimization*. CRC Press, 1991.