2005 American Control Conference
June 8-10, 2005. Portland, OR, USA

Comparative Studies of Load Balancing With Control and Optimization Techniques

Yixin Diao, Chai Wah Wu, Joseph L. Hellerstein, Adam J. Storm, Maheswaran Surendra, Sam Lightstone, Sujay Parekh, Christian Garcia-Arellano, Matthew Carroll, Lee Chu, and Jerome Colaco

Abstract— **Load balancing is a widely used technique to optimizing distributed computing system performance. System response delays are reduced by equalizing the loads, such as adjusting memory pool sizes to balance disk access demands in a database management system. In this paper we formulate load balancing as a constrained optimization problem and investigate two load balancing controllers based on feedback control theory and optimization theory. We show the difference and equivalence between their design methods and criteria. Furthermore, our studies on a DB2 Universal Database Server reveal their performance difference regarding to system noise and workload variations.**

I. Introduction

Load balancing is essential in distributed computing systems to improve quality of service by managing customer loads that are varying over time. The request demands of incoming requests are optimally distributed among available system resources to avoid resource bottlenecks as well as to fully utilize available resources. Load balancing also facilitates horizontal scaling (e.g., adding computing resources to accommodate increased loads). In this paper we compare two load balancing techniques in the context of managing database server memory, where the goal is to balance the load consumption across multiple memory pools.

There is a vast literature on load balancing, including its use in multiple source routing [1], redirection for web-server systems [2], and memory management [3]. The focus of these efforts, however, is more on load balancing in a specific context, for example, how to classify the memory load through reference characteristics and how to divide the memory into partitions to accommodate them; the general load balancing strategies are seldom formally analyzed. Although heuristic tuning logic may work well in certain circumstances, lack of rigorous convergence analysis can lead to oscillation and unnecessary reallocation overheads for load balancing in a dynamic environment. Recently, analytical approaches such as those employing control theory for computing systems are studied for web server performance, differentiated caching and web service, and multimedia streaming [4] [5] [6]. None of these approaches, however, addresses load balancing or optimization of computing system resources.

Y. Diao, C. W. Wu, 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, C. Garcia-Arellano, M. Carroll, L. Chu, and J. Colaco are with IBM Toronto Lab, Markham, Ontario, Canada.

In this paper we formulate load balancing as a constrained optimization problem and investigate its different properties. Furthermore, we study two load balancing technologies, based on feedback control theory and optimization theory, respectively. The control-based approach uses a model-based feedback controller to equalize the marginal gains from different memory pools so as to maximize the expected saved system time for processing database requests. Not all the components in the feedback loop, however, are strictly analyzed, and the enforcement of the resource constraints is handled by a separate projection logic, which makes it difficult to analyze. In the second approach, we use the optimization-based problem formulation to develop a solution that exploits rigorous constrained optimization techniques. Comparative analysis between these two approaches shows similarities regarding to design methodologies as well as differences in method implementation.

The remainder of the paper is organized as follows. Section II describes the database memory load balancing problem and experiment setup and analytical models for simulation studies. Section III presents the control-based and optimization-based load balancing approaches and compares the difference and similarity between them. Section IV assesses our controllers for a DB2 Universal Database Server. The conclusions are contained in Section V.

II. Load Balancing Problems

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

A. Database Memory Management

Figure 1 shows the architecture and system operations of a database server that provides load balancing across multiple memory pools. The database clients interact with the database server through the database agents which are computing elements that coordinate access to the data stored in the database. Since disk accesses are much slower relative to main memory accesses, database systems use memory pools to cache disk pages so as to reduce the number and time of disk input/output operations needed. The in-memory data are organized in several pools, which are dedicated for different purposes and can be of different types and characteristics. The management of these pools, especially in terms of determining their optimal sizes, is a key factor in tuning and determining database system performance. Increasing the size of a memory pool can drastically reduce

0-7803-9098-9/05/$25.00 ©2005 AACC

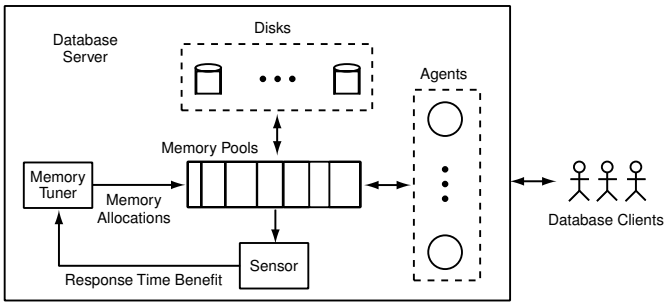
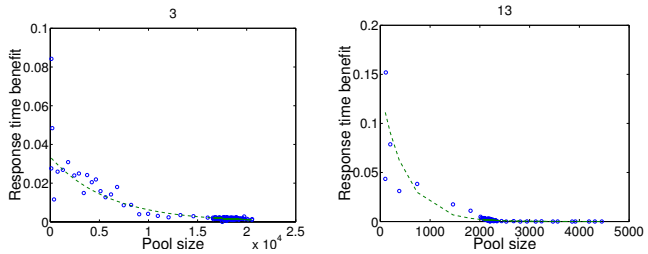


Fig. 1. *Architecture of load balancing for a database server.*



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

Fig. 2. *Empirical model of the response time benefit of DB2 buffer pools for an on-line transaction processing workload.*

its response 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 response time obtained from an increase in memory allocation as the response time benefit, or just benefit, measured in the units of saved 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. The memory tuner adjusts pool allocations with the intent of reducing overall response time for data access. An intuitive approach is to allocate memory to pools so that each has the same benefit, the marginal gain. The motivation for this allocation strategy is that the performance of computing systems is largely affected by the most loaded resource. Thus, by balancing the load (in this memory management case, equalizing the benefits for memory usage in all pools), we hope to optimize the performance.

B. Experiment Environment and System Modeling

In this section we focus on understanding the relationship between memory allocation and response time benefit. We start from describing the experiment environment.

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 which 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 online transaction processing (OLTP) workload and a decision support systems (DSS) workload. 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. The DSS workload has a small number of long-running resource-intensive requests that are highly variable in their demands; we structure the DSS runs so that there are always four transactions executing concurrently.

We conduct experiments with the prototype code running on top of DB2. For dynamic resource allocation in response to workload variations, the response time benefits must be measured or estimated at run-time. One example of estimating the buffer pool benefit is to have ghost buffers (a.k.a., dataless buffers) record the number of buffer hits and disk time for cache sizes larger than the current buffer size [3]. Another approach is to use the Belady's lifetime function to approximate the buffer hit ratio [7]. Estimation accuracy and measurement overhead, certainly, are of great concern for practical usage. While detailed descriptions of benefit estimation mechanisms are outside the scope of this paper, in general, the benefit measure needs to be common for all the memory consumers. For this reason, we define the benefit as "saved response time per unit memory increase." In contrast, other utilization measures may not make sense for some consumers. For example, hit ratio is used in buffer pools but not sort memory.

Figure 2 depicts two example relationships between response time benefits and memory pool sizes, under the OLTP workload. The horizontal axis is the size of the memory pool in the unit of 4K pages; the vertical axis is the response time benefit of adding an increment of memory in the unit of second per 4K pages. The circles indicate results obtained from testbed experiments. We assume an exponential relationship between the pool size and the saved response time, $x_i = p_i(1 - e^{-q_i u_i})$, so that the relationship between the pool size and the response time benefit is $y_i = \frac{dx_i}{du_i} = p_i q_i e^{-q_i u_i}$, 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 p_i, q_i are model parameters that can be obtained empirically. Although simple, the above model results in a reasonable fit as shown in Figure 2 by the dashed lines. This comes from the use of "local" metrics (so that the benefits are only affected by memory/disk access behaviors but not other parts of the database and operating systems). It is also because the inter memory pool effect is not that significant so that per memory pool model can be valid. Intuitively, the response time benefit curve implies that adding memory can be of great benefit when the current memory pool is small, and its benefit will decrease with the increasing of memory pool size; thus, the saved response time will saturate when the memory pool is large enough and no more savings can be earned. This function shape is also consistent with the Belady's lifetime function for approximating the buffer

hit ratio of references that follow the Least Recently Used (LRU) model. However, our models are more general and can be applied not only to buffer pools but also to other memory pools such as sort [8].

C. Load Balancing Problem Formulation

The above memory management problem can be classified into a general class of approaches that equalize a measured output, so-called load balancing techniques. For a load balancing and resource allocation problem, 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). 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. Let $u_1, \cdots, u_N$ be resource allocations for N resource consumers and $y_1, \cdots, y_N$ be their measured outputs. The load balancing problem is to choose u_i such that $y_1 = \cdots = y_N$ subject to the constraint that $\sum u_i = U$ where U defines the total resource. 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.

In the database memory management example, the resource is the memory being managed, 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, and the measured outputs are response time benefits obtained as marginal gains of changing memory allocation. From the models obtained in Section II.B, the optimization problem is to maximize the total saved response time f, or $\max_{u_1,\cdots,u_N} f$, where $f = \sum_{i=1}^{N} x_i = \sum_{i=1}^{N} p_i(1 - e^{-q_i u_i})$, subject to the constraint of the total available memory $\sum_{i=1}^{N} u_i = U$. Since $f = \sum_{i=1}^{N-1} x_i + x_N = \sum_{i=1}^{N-1} p_i(1 - e^{-q_i u_i}) + p_N(1 - e^{-q_N(U - \sum_{i=1}^{N-1} u_i)})$, we can obtain $\max f$ by finding where the gradient is zero, that is $\frac{\partial f}{\partial u_i} = p_i q_i e^{-q_i u_i} + \frac{\partial p_N(1 - e^{-q_N(U - \sum_{i=1}^{N-1} u_i)})}{\partial u_i} = p_i q_i e^{-q_i u_i} - p_N q_N e^{-q_N u_N} = y_i - y_N = 0$ 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 [9].

In fact, the above load balancing property can be derived without the knowledge of specific performance function. Given a scalar performance function with N control variables $u_1, u_2, \ldots, u_N$

$$J = f(u_1, u_2, \ldots, u_N) \tag{1}$$

and the scalar equality constraint

$$g(u_1, u_2, \ldots, u_N) = 0, \tag{2}$$

the constrained optimization problem is to find the values of u_i that maximizes J subject to the constraint. This results in $N-1$ independent components of u_i and reduces the order of the optimization problem by one. The solution of the constrained optimization problem can be found by using direct substitution of the constraint for the performance function (as used above in the memory management example). The direct substitution method, however, may not be easy to use especially for a vector equality constraint. More generally, the constrained optimization problem can be solved using Lagrange multipliers. Furthermore, besides the vector equality constraint we consider the vector inequality constraint

$$h(u_1, u_2, \ldots, u_N) \geq 0. \tag{3}$$

For the database management problem, the inequality constraints are $h_i = u_i - \underline{u}_i \geq 0$, that is, every memory pool has its minimum size.

The solution of this constrained optimization problem can be found using the first order Karush-Kuhn-Tucker (KTT) necessary conditions. Define the Lagrange function

$$\begin{aligned} L &= f(u_1, u_2, \ldots, u_N) + \lambda g(u_1, u_2, \ldots, u_N) \\ &\quad + \mu^\top h(u_1, u_2, \ldots, u_N) \end{aligned} \tag{4}$$

which adjoin the original performance function and the constraints using the Lagrange multipliers λ and μ. The KTT necessary conditions for a solution u to be locally optimal is that the constraints are satisfied, i.e., $g(u) = 0$ and $h(u) \geq 0$ and there exists Lagrange multipliers λ and μ such that the gradient of the Lagrangian vanishes:

$$\frac{\partial L}{\partial u_i} = \frac{\partial f}{\partial u_i} + \lambda \frac{\partial g}{\partial u_i} + \sum_{j=1}^{N} \mu_j \frac{\partial h_j}{\partial u_j} = 0 \tag{5}$$

Furthermore, μ satisfies the complementarity condition $\mu_j h_j = 0$ with $\mu_j \geq 0$. For the memory load balancing problem with the inequality constraint $h_i = u_i - \underline{u}_i \geq 0$ and equality constraint $g(u_1, u_2, \ldots, u_N) = \sum_{i=1}^{N} u_i - U = 0$, we have $\frac{\partial L}{\partial u_i} = \frac{\partial f}{\partial u_i} + \lambda + \mu_i = 0$. Since the complementarity condition implies that $\mu_i = 0$ if $h_i > 0$, and $\mu_i > 0$ if $h_j = 0$. This means that the benefits $\frac{\partial f}{\partial u_i}$ are all equal for the memory pools at the optimal allocation whose sizes are not at the boundaries. For the memory pools whose sizes are at the boundaries, their benefits would be smaller.

A few remarks of the load balancing problem. (i) A locally optimal load not at the boundaries is a balanced load if the measured outputs y_i are the partial derivatives of the performance metric J with respect to the resource allocations u_i, that is, $y_i = \frac{\partial f}{\partial u_i}$. (ii) A balanced load not at the boundaries is a globally optimal load if the measured outputs y_i are the partial derivatives of the performance metric J with respect to the resource allocations u_i and the performance function f is a concave function of u_i.

(iii) For a "weighted" constraint $\sum_{i=1}^{N} w_i u_i = U$, the optimal load is a "weighted"-balanced load, i.e., all y_i/w_i are equal. (iv) Note that $y_i = \frac{\partial f}{\partial u_i}$ may not be easy to measure from the system as f is a global function across all loads/consumers. However, if $f(u_1, u_2, \ldots, u_N)$ can be expressed as $f(u_1, u_2, \ldots, u_N) = q_1 f_1(u_1) + \cdots + q_N f_N(u_N)$, then the measured output $y_i = q_i \frac{df_i}{du_i}$ becomes local measurement which is easier to measure or simulate. This facilitates distributed modeling and control as in the database memory management problem where, for example, the saved disk I/O time is counted separately for each buffer pool. (v) Given $f(u_1, u_2, \ldots, u_N) = q_1 f_1(u_1) + \cdots + q_N f_N(u_N)$ and $q_i > 0$, a balanced load is a global optimal load if all $f_i(u_i)$ are concave functions. For the database memory management problem, the exponential relationship is concave.

The database memory management problem is one application of the load balancing algorithm. We study two other applications in Appendix.

III. Load Balancing Controller Design and Comparison

A. Control-Based Approach

As discussed in Section II, maximizing the total saved response time from all memory pools can be achieved by equalizing the response time benefit. We design a multiple-input multiple-output (MIMO) feedback controller to implement the load balancing function [9]. 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 memory pool resizing). The load balancing model and controller are given as follows.

$$\mathbf{y}(k+1) = \mathbf{A}\mathbf{y}(k) + \mathbf{B}(\mathbf{u}(k) + \mathbf{d}(k)) \quad (6)$$

$$\mathbf{e}(k) = (\frac{1}{N}\mathbf{1}_{N,N} - \mathbf{I})\mathbf{y}(k) \quad (7)$$

$$\mathbf{e_I}(k+1) = \mathbf{e_I}(k) + \mathbf{e}(k) \quad (8)$$

$$\mathbf{u}(k) = \mathbf{K_P}\mathbf{e}(k) + \mathbf{K_I}\mathbf{e_I}(k) \quad (9)$$

The first equation is a local linear approximation of the exponential response time benefit curve as that in Section II.B, where $\mathbf{y}(k)$ is a $N \times 1$ vector representing the measured output (e.g., response time benefit). The $N \times 1$ vector $\mathbf{u}(k)$ represents the computed resource allocation (e.g., memory pool size) from the feedback controller, and the $N \times 1$ vector $\mathbf{d}(k)$ represents the possible adjustments made to enforce the equality and inequality constraints. The $N \times N$ matrices $\mathbf{A}$ and $\mathbf{B}$ contain state space model parameters that can be obtained from system identification.

Equation (7) specifies the $N \times 1$ control error vector $\mathbf{e}(k)$, where $\mathbf{I} = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 1 \end{bmatrix}$ and $\mathbf{1}_{N,N} = \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$ are $N \times N$ matrices. We define the average measured output $\bar{y}(k) = \frac{1}{N}\sum_{i=1}^{N} y_i(k)$ as the control reference. The i-th control error is $e_i(k) = \bar{y}(k) - y_i(k)$. The control objective is to make $e_i(k) = 0$, that is, equalizing the measured outputs (i.e., $y_i(k) = y_j(k)$ for any i and j) so as to maximize the total saved response time. Note that in contrast to having a static value or external signal as the reference input, we specify the reference as a linear transformation of measured outputs.

The dynamic state feedback control law is defined in Equation (9), where $\mathbf{u}(k)$ is a $N \times 1$ control input vector (subject to constraint-oriented adjustment $\mathbf{d}(k)$ prior to be applied to the database server) and $\mathbf{e_I}(k)$ is the $N \times 1$ vector representing the sum of the control error as defined in Equation (8). The above state feedback controller can be designed (specifying the $N \times N$ matrices $\mathbf{K_P}$ and $\mathbf{K_I}$) using linear control systems design techniques such as linear quadratic regulator (LQR). Specific considerations need to be given to maintain the controllability due to the existence of equality constraints [9]. A cost model can also be specified to choose the LQR $\mathbf{Q}$ and $\mathbf{R}$ matrices based on the impact on system performance of changing resource allocations and transient load imbalances [10].

The load balancing model and the feedback controller described above can be simplified to facilitate online modeling when the workload is unknown in advance and can change overtime. It is also useful to manage a large number of memory pools where the number of pools can also vary at run-time and to increase the robustness regarding to measurement uncertainties and uneven control intervals. This simplification is performed by using distributed control strategies. The model is built and the controller is designed locally for each individual memory pool; the only connection between different pools is the control reference signal– the average measured output. Specifically, a single-input single-output model

$$y_i(k+1) = b_i(k)u_i(k) \quad (10)$$

is built online for the i-th memory pool using recursive least squares, and an integral controller

$$u_i(k+1) = u_i(k) - \frac{1-p}{b_i(k)}\left(y_i(k) - \frac{1}{N}\sum_{j=1}^{N} y_j(k)\right) \quad (11)$$

is deployed for the i-th memory pool where p is the only design parameter indicating the desired closed loop pole. Although in general for a real database server the system dynamics may not be negligible (e.g., an increase of buffer pool size may not immediately result in response time benefit decrease because of the time needed to fill up the added buffer space) and the cross memory pool impact does exist (e.g., an increase of sort memory will not only bring down the benefit for sort memory but also that for the buffer pool that stores temporary sort spill pages), our experimental results for several different workload benchmark environments confirms the control performance of this distributed controller. This is also because of our using

relatively large control and sample interval to reduce the effects of dynamics as well as measurement noise.

Note that the control input obtained from the above feedback control approach may not satisfy either the equality constraint on total available memory or the inequality constraint on minimum memory pool size. To enforce the constraint conditions, a projection method is used to adjust the control input $u(k)$. Specifically, we compute

$$d_i(k) = \frac{u_i(k)}{\sum_{j=1}^{N} u_j(k)} \left(U - \sum_{j=1}^{N} \underline{u}_j \right) + \underline{u}_i \qquad (12)$$

where $u_i(k))$ is from the above feedback controller but is further bounded to be non-negative (i.e., set $u_i(k)$ to 0 if it is negative). The new memory pool size is given by $u_i(k) + d_i(k)$ to satisfy both the equality and inequality constraints and is applied to the database server. Since the effect of the projection method is modeled by a disturbance signal $d(k)$ in Equation (6), it will not affect the stability and convergence of the closed loop system.

B. Optimization-Based Approach

As formulated in Section II.C, the load balancing problem is a constrained optimization problem. Instead of solving it using the control-based approach, in this section we use the standard constrained optimization technique, particularly, the active set methods [11]. Although equality constraints are relatively easy to enforce, for their effect is to reduce the dimensionality of the optimization problem by the number of independent linear equality constraints, the inequality-constrained problem is more involved. This is because for the set of inequality constraints, only the constraints *active* (becoming equality) at the solution are significant; this set of constraints that hold with equality at the solution, however, is generally unknown. The active set algorithm works with the following logic steps. (i) Predict the active set by selecting a *working set* of constraints to be treated as equality constraints. (ii) Evaluate the validity of the working set and add or delete a constraint if necessary. (iii) Compute the feasible search direction and step length obeying the above working set, so that the inequality-constrained problem is solved as equality-constrained problem through iterations.

With the active set method, the search direction is constructed to lie in a subspace defined by the working set; the step length is ensured to not violate any constraint that is not in the working set. For the constrained optimization problem defined by Equations (1), (2), and (3), the optimal solution is searched as follows (since we are *maximizing* an objective function, our notation will vary slightly in several places compared to the optimization literature which usually *minimizes* an objective function):

$$u(k+1) = u(k) + \lambda(k)p(k) \qquad (13)$$

where $u(k)$ denotes the $N \times 1$ vector comprising of N memory pool sizes, $p(k)$ denotes the $N \times 1$ projected gradient vector (i.e., the feasible search direction), and $\lambda(k)$ is the (scalar) step length. Specifically, the search direction can be computed using quasi-Newton or Newton's method, and the Newton direction is projected to formulate the feasible search direction [11]

$$\begin{aligned} p(k) &= -(H(k) - H(k)A^{\top}(k) \left(A(k)H(k)A^{\top}(k)\right)^{-1} \\ &\quad A(k)H(k))y(k) \end{aligned} \qquad (14)$$

where $y(k)$ denotes the $N \times 1$ gradient vector comprising of N response time benefit measurements, I is a $N \times N$ identify matrix, A is a $m \times N$ matrix consisting of m linear equality constraints in the working set, and H denotes the inverse of the $N \times N$ Hessian matrix. The inverse Hessian is approximated sequentially by a quasi-Newton update such as BFGS.

The scalar step length $\lambda(k)$ can be computed using line search algorithms. For instance, the step size $\lambda(k)$ is selected when sufficient increase is observed. One way this is done is by decreasing the step size until the Armijo condition

$$f\left(u(k) + \lambda(k)p(k)\right) \geq f\left(u(k)\right) + c_1\lambda(k)\nabla f\left(u(k)\right)^{\top} p(k) \qquad (15)$$

is satisfied. However, the evaluation of the cost function f is required, which is not available in the database memory management problem since the saved response time is unknown or difficult to measure. In line search type optimization algorithms, about the only place where the cost function f is needed is in the scalar line search component to determine $\lambda(k)$. Therefore, we can get an optimization algorithm that does not require knowledge of f if we modify the line search algorithm to use only the gradient information. Specifically, we conduct backtracking line search which decreases the step size $\lambda(k)$ each time by a multiplicative factor $\beta < 1$ until the following condition

$$\nabla f\left(u(k) + \lambda(k)p(k)\right)^{\top} p(k) \geq c_1 \nabla f\left(u(k)\right)^{\top} p(k) \qquad (16)$$

is satisfied. Note that Equation (15) is equivalent to Equation (16) when f is linear. On the other hand, when f is concave, $f\left(u(k)\right) \leq f\left(u(k) + \lambda(k)p(k)\right) - \lambda(k)\nabla f\left(u(k) + \lambda(k)p(k)\right)^{\top} p(k)$. Thus Equation (16) implies the Armijo condition (Equation (15)). In other words, for concave f, the convergence properties associated with line search algorithms using the Armijo condition is preserved when using Equation (16) instead of the Armijo condition.

C. Comparison Between the Control-Based and Optimization-Based Approaches

In this section we compare the control-based and optimization based approaches in the context of database memory management and show their equivalence regarding to design criteria and difference in design implementations.

Consider a situation where all memory pool sizes are not at the boundary, so that the working set only consists of the equality constraint $\sum_{i=1}^{N} u_i(k) = U$, that is,

$$A(k) = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} \qquad (17)$$

is a $1 \times N$ vector. Suppose the benefits (i.e., first derivatives) are available and the memory pools are independent, i.e., f can be written as $f = \sum_i f_i(u_i)$. Then the Hessian is diagonal, and thus applying group partial separability [12] to quasi-Newton updates of the inverse Hessian results in an update which is equivalent to the finite difference method of estimating the inverse Hessian matrix

$$H(k) = \begin{bmatrix} \frac{1}{s_1(k)} & 0 & \cdots & 0 \\ 0 & \frac{1}{s_2(k)} & \cdots & 0 \\ \vdots & & & \vdots \\ 0 & 0 & \cdots & \frac{1}{s_N(k)} \end{bmatrix} \quad (18)$$

where $s_i(k) = \frac{y_i(k)-y_i(k-1)}{u_i(k)-u_i(k-1)}$ is the slope of the benefit curve for memory pool i. Since

$$\begin{aligned} & A^\top(k)\left(A(k)H(k)A^\top(k)\right)^{-1}A(k)H(k) \\ = & \frac{1}{\sum_{j=1}^N \frac{1}{s_j(k)}} \begin{bmatrix} \frac{1}{s_1(k)} & \frac{1}{s_2(k)} & \cdots & \frac{1}{s_N(k)} \\ \frac{1}{s_1(k)} & \frac{1}{s_2(k)} & \cdots & \frac{1}{s_N(k)} \\ \vdots & & & \vdots \\ \frac{1}{s_1(k)} & \frac{1}{s_2(k)} & \cdots & \frac{1}{s_N(k)} \end{bmatrix} \end{aligned} \quad (19)$$

we have

$$p_i(k) = -\frac{1}{s_i(k)}\left(y_i(k) - \frac{1}{\sum_{j=1}^N \frac{1}{s_j(k)}} \sum_{j=1}^N \frac{y_j(k)}{s_j(k)}\right). \quad (20)$$

This gives

$$u_i(k+1) = u_i(k) - \frac{\lambda(k)}{s_i(k)}\left(y_i(k) - \frac{1}{\sum_{j=1}^N \frac{1}{s_j(k)}} \sum_{j=1}^N \frac{y_j(k)}{s_j(k)}\right) \quad (21)$$

where the step length $\lambda(k)$ is computed using the modified Armijo rule as above. Because "successive step size reduction" is used and the initial step length is picked at 1, we have $0 < \lambda(k) \leq 1$.

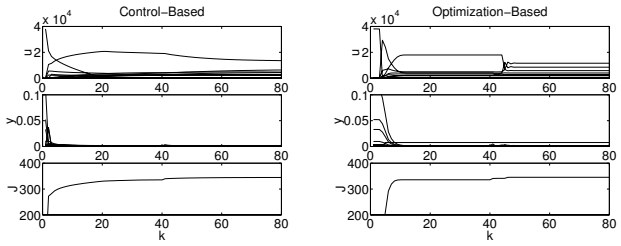
Comparing the above simplified optimization-based control law with the distributed control law in Equation (11), we can see that the search direction are approximately the same when the Hessian is close to a multiple of the identity matrix. (For example, if $s_1(k) = \cdots = s_N(k)$, then $\frac{1}{\sum_{j=1}^N \frac{1}{s_j(k)}} \sum_{j=1}^N \frac{y_j(k)}{s_j(k)} = \frac{1}{N}\sum_{j=1}^N y_j(k)$.) In either approach, the change in u depends on the difference between the benefit and average benefit. Equation (21) is a standard projected method (i.e., projection Newton) satisfying the equality constraint (and inequality constraints if the working set $A(k)$ represents so) and the average benefit is a convex sum of all the benefits. Equation (11) uses "pure" average benefit (i.e., $\frac{1}{N}\sum_{j=1}^N y_j(k)$) and does not enforce the constraints so that additional projection method, such as that in Equation (12), needs to be applied. Unlike the optimization-based method that strictly applies constrained optimization techniques, the control-based approach implicitly considers the constraints to reduce its dependence on the model so as to improve its robustness; meanwhile, this implicit consideration still guarantees system stability and convergence, for its effect can be modeled by the disturbance signal $d(k)$ and compensated by the feedback control scheme.

Furthermore, there exists other differences between the two methods. First, $1 - p$ is used in the control-based approach, where p ($0 < p < 1$) is the desired closed loop pole and is specified as a design parameter (a smaller p indicate a more aggressive control action). In the optimization-base approach, $\lambda(k)$ is used and calculated online with the line search method. Both of them, however, are in the range between 0 and 1. Second, $b_i(k)$ is the steady-state gain in the control-based approach determined online using recursive least squares. Same as $s_i(k)$, it also indicates the slope of the benefit curve. On the other hand, $s_i(k)$ can be quite subject to noise (even if it can also be computed using recursive least squares, a not common practice in calculating the Hessian matrix). Third, we consider the difference in deriving Equation (11) and (21). In the optimization-based approach the computation of the inverse Hessian matrix is greatly simplified by not considering the interactions between memory pools. If these interactions are significant, then the inverse Hessian is not diagonal and the quasi-Newton inverse Hessian update will be more complicated than Equation (18). In the control-based approach, the SISO model (10) is also a simplified version of the dynamic system model (6) which captures the system dynamics as well as the interactions between memory pools.

IV. Controller Assessment

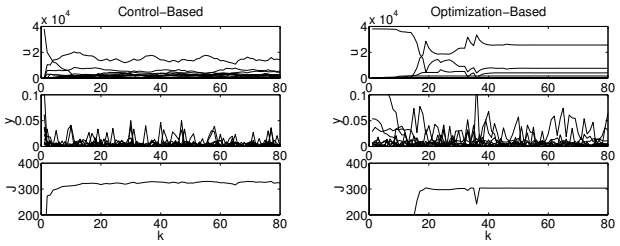
This section describes simulation studies for comparing the control-based and optimization-based approaches in balancing the load of database memory. The simulator uses the models developed in Section II.B, where the model parameters are identified from the data collected in experiments with an OLTP workload. The variabilities in response time benefit data are also emulated using noise models with a normal distribution and the standard deviations matching with those from different memory pools. A total of 20 buffer pools are being managed with a total of 40,000 4K pages. The minimum pool size is 20 4K pages.

We first consider the case without noise. The memory is initially allocated so that all the pools has 100 pages except one with 38,100 pages. The load balancing performance is show in Figure 3 for both the control-based and optimization-base approaches, where the horizontal axis is the control interval in 30 seconds. A workload change occurs at the 40th interval. The top plot shows the sizes of 20 memory pools (some of these 20 lines are not distinguishable), the middle plot shows the response time benefits of these 20 pools, and the bottom plots shows the total saved response time computed from models developed in Section II.B. Both controllers adjust the size of buffer pools so as to equalize the response time benefits from different pools. However, we see that the optimization-based controller is converging much faster (except the initial "inertia" after the



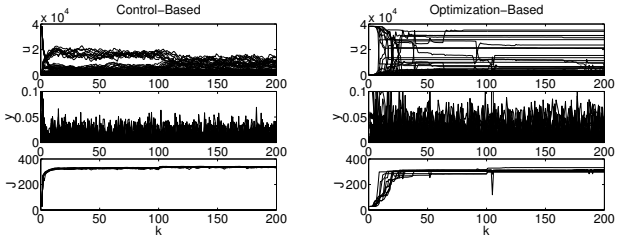
(a) Control-based approach (b) Optimization-base approach

Fig. 3. *Load balancing performance with workload change but without noise for a simulated OLTP workload.*



(a) Control-based approach (b) Optimization-base approach

Fig. 4. *Load balancing performance with noise for a simulated OLTP workload.*



(a) Control-based approach (b) Optimization-base approach

Fig. 5. *Load balancing performance of multiple runs with noise and workload change for a simulated OLTP workload.*

0th and 40th intervals). This can be explained by noting that the optimization-based approach takes full advantage of accurate model information, while the control-based algorithm enforces the constraints implicitly and chooses a larger p for robustness considerations.

Next, we consider the effect of noise. As shown in Figure 4 (without workload changes), the control-based approach is still converging to approximately the same memory allocations as the ones without noise. However, the optimization-based approach does not and thus results in larger and unbalanced response time benefits and smaller total saved response time. To see this noise effect more clearly, we conduct this simulation 10 times with different randomness and also with workload changes at the 100th interval as shown in Figure 5. The control-based approach shows higher robustness with respect to the noise and with more consistent convergence behaviors, while the optimization-based approach seems more sensitive to noise. However, its performance may be improved by using recursive least squares for estimating the Hessian matrix or by adjusting the parameters c_1 and β in the line search procedure. On the other hand, Figure 4 and Figure 5 show that noise causes the allocation in the control-based approach to fluctuate at each interval, whereas in the optimization-based approach, the allocation does not change significantly once an optimal (or steady-state) solution is found. This can be attributed to the fact that the line search makes sure a change in allocation is only done if it result in a significant increase in the saved response time.

Experimental comparison are also conducted for the DSS workload with similar results; in the interest of brevity, however, we do not include those plots here.

V. Conclusions

Load balancing is widely used in computing systems for performance optimization through equalizing loads. One example in database systems is to adjust the memory pool sizes so as to balance the disk I/O demands. In this paper we have formulated load balancing as a constrained optimization problem and studied two load balancing controllers based on feedback control theory and optimization theory. Their difference and similarity have been studied analytically. Furthermore, our simulation studies over a DB2 Universal Database Server have demonstrated better performance and robustness for the feedback control approach with respect to system noise and workload variations.

References

[1] 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.

[2] 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.

[3] J. T. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka, "Informed prefetching and caching," in *Proceedings of the 15th Symposium on Operating System Principles*, pp. 1–16, 1995.

[4] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. John Wiley and Sons, 2004.

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

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

[7] J. R. Spirn, *Program Behavior: Models and Measurements*. Elsevier-North, Holland, 1977.

[8] IBM, *IBM DB2 Universal Database Administration Guide*. IBM Corp, 2002.

[9] Y. Diao, J. L. Hellerstein, A. Storm, M. Surendra, S. Lightstone, S. Parekh, and C. Garcia-Arellano, "Using MIMO linear control for load balancing in computing systems," in *Proceedings of the American Control Conference, Boston, MA*, 2004.

[10] Y. Diao, J. L. Hellerstein, A. Storm, M. Surendra, S. Lightstone, S. Parekh, and C. Garcia-Arellano, "Incorporating cost of control into the design of a load balancing controller," in *Proceedings of the Real-Time and Embedded Technology and Application Systems Symposium, Toronto, Canada*, 2004.

[11] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*. Academic Press, 1981.

[12] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer-Verlag, 1999.