A New Approach to Dynamic Self-Tuning of Database Buffers

DINH NGUYEN TRAN
New York University
PHUNG CHINH HUYNH
Microsoft Corporation
and
Y. C. TAY and ANTHONY K. H. TUNG
National University of Singapore

3

Current businesses rely heavily on efficient access to their databases. Manual tuning of these database systems by performance experts is increasingly infeasible: For small companies, hiring an expert may be too expensive; for large enterprises, even an expert may not fully understand the interaction between a large system and its multiple changing workloads. This trend has led major vendors to offer tools that automatically and dynamically tune a database system.

Many database tuning knobs concern the buffer pool for caching data and disk pages. Specifically, these knobs control the buffer allocation and thus the cache miss probability, which has direct impact on performance.

Previous methods for automatic buffer tuning are based on simulation, black-box control, gradient descent, and empirical equations. This article presents a new approach, using calculations with an analytically-derived equation that relates miss probability to buffer allocation; this equation fits four buffer replacement policies, as well as twelve datasets from mainframes running commercial databases in large corporations.

The equation identifies a buffer-size limit that is useful for buffer tuning and powering down idle buffers. It can also replace simulation in predicting I/O costs. Experiments with PostgreSQL illustrate how the equation can help optimize online buffer partitioning, ensure fairness in buffer reclamation, and dynamically retune the allocation when workloads change. It is also used, in conjunction with DB2's interface for retrieving miss data, for tuning DB2 buffer allocation to achieve targets for differentiated service.

Categories and Subject Descriptors: C.4 [**Computer Systems Organization**]: Performance of Systems; H.4.m [**Information Systems Applications**]: Miscellaneous

This work was supported by National University of Singapore ARF grant R-146-000-072-112.
Authors' addresses: D. N. Tran, New York University, 547 LaGuardia PI., New York, NY 10012; P. C. Huynh, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052-6399; Y. C. Tay (corresponding author), A. K. H. Tung, National University of Singapore, 21 Lower Kent Ridge Road, Singapore 119077; email: dcstayyc@nus.edu.sg.

General Terms: Design, Performance

Additional Key Words and Phrases: Buffer allocation, miss probability, autonomic computing

ACM Reference Format:

Tran, D. N., Huynh, P. C., Tay, Y. C., and Tung, A. K. H. 2008. A new approach to dynamic self-tuning of database buffers. ACM Trans. Storage 4, 1, Article 3 (May 2008), 25 pages. DOI =10.1145/1353452.1353455 http://doi.acm.org/ 10.1145/1353452.1353455

1. INTRODUCTION

Most business processes require access to some databases. Competitive pressures require these accesses to be quick, so the database systems must be well tuned.

Traditionally, database tuning is done manually by experts (e.g., database administrators), but this is becoming increasingly infeasible: With the large drop in hardware prices and huge increase in capacity, database systems have grown bigger and more complicated. Furthermore, the workload may be heterogeneous (e.g., server consolidation), dynamic (e.g., Web-driven), or unknown (e.g., outsourcing service). Human expertise for tuning such systems can be hard to find, or prohibitively expensive. This has led major vendors to offer software tools for automatic and dynamic tuning: Automatic Database Diagnostic Monitor for Oracle [Dias et al. 2005], Resource Advisor for SQL Server [Narayanan et al. 2005], and Self-Tuning Memory Manager for DB2 [Storm et al. 2006].

As evident from the tuning aids provided by these three tools, a key issue is the allocation of memory to the *buffer pool* (or simply *buffer*) for caching data and disk pages. Buffer tuning determines P^{miss}, the probability that some referenced object is not in the cache, thus degrading performance.

This article, addresses three issues in buffer tuning as described next.

(1) *Buffer Size M*. Despite the dramatic price drop for memory, buffer size remains an issue, as there are always competing demands for memory space from code, working storage (for sorting, hashing, etc.), metadata, networking, etc. [Lightstone et al. 2002]. There has also been recent interest in powering down excess memory [Cai and Lu 2005].

(2) *Buffer Share M_i*. A large enterprise may run concurrently several workloads with diverse characteristics (short interactive transactions, long queries for decision support, etc.) and different performance goals (e.g., response time versus throughput). Similarly, an outsourcing service provider may host (on the same machine) multiple client databases, with different service-level agreements. For such reasons, a system that runs k different workloads may partition a buffer of size M into smaller sizes $M_1, \ldots, M_k$ (so $M = M_1 + \cdots + M_k$), one for each workload. By *buffer allocation*, we refer to both size M and share M_i.

(3) *Dynamic Self-Tuning*. Workloads are constantly changing, by the month (system upgrade), hour (e.g., daily work cycle), and minute (e.g., flash crowds). Hence, buffer size and share need to be tuned automatically and dynamically.

The impact of buffer allocation on miss probability P^{miss} is through a complex interaction between the workload's reference pattern and the buffer management policy, and affected by innumerable factors.

To illustrate, the reference pattern depends on how data contention is resolved by the concurrency control, whether multiquery optimization pipelines the subexpression evaluation, the design and usage of indices, whether there is disk striping and prefetching, how the database changes over time, what the hardware architecture and software versions are, and how the whole lot is configured.

These myriad factors and their complex interaction make it difficult to predict how a change in buffer allocation affects P^{miss}. Automatic and dynamic buffer tuning is thus a hard problem.

1.1 Current Techniques

Present tuning techniques use simulation, black-box control, gradient descent, and empirical equations.

Trace simulation is used in current tuning software for commercial systems to estimate I/O costs for different buffer allocations [Dias et al. 2005; Narayanan et al. 2005; Storm et al. 2006]. However, traces may not be available and simulation code is hard to modify (e.g., when there is a change in index organization).

Black-box control is an iterative loop that treats the buffer as an unknown function from buffer allocation to P^{miss} [Ko et al. 2003; Lu et al. 2002]. Convergence may not be guaranteed, or may be slower than dynamic changes in the workload.

Gradient descent is a popular iterative technique [Chung et al. 1995; Ko et al. 2003; Suh et al. 2004; Thiébaut et al. 1992; Tian et al. 2003] that uses a measured gradient $\frac{\Delta P^{\text{miss}}}{\Delta M}$ to direct the change in M. However, P^{miss} fluctuation can cause large gradient errors, thus destabilizing the convergence.

Empirical equations have no theoretical basis; they are chosen for their ability to fit data [Hsu et al. 2001; Storm et al. 2006; Tsuei et al. 1997]. One widely-used example is Belady's power law [Chung et al. 1995; Tian et al. 2003], but that is known to give a poor fit for database workloads [Brown et al. 1996]. Moreover, although there is a limit to how much buffer space is needed by any workload, these equations do not identify such an upper bound.

We will say more about these techniques in later sections.

1.2 Our Approach

The main contributions of this article are as follows:

(1) We present a new approach to buffer tuning that is based on a *buffer miss equation*. Our technique is to fit available data with the equation, then use the equation for tuning calculations. Unlike empirical equations, the buffer miss equation is derived from an analytical model (see Appendix), and it identifies an upper bound M^* on buffer size that is useful for tuning.

(2) We validate the equation with four buffer replacement policies (Section 2.2) and twelve reference traces from mainframe commercial databases in large

corporations (Section 2.3). This ability to fit multiple policies and patterns is nontrivial.

(3) We demonstrate how the equation can be used to:
   (a) estimate I/O costs from P^{miss} data when no reference trace is available for simulation (Section 3.2);
   (b) dynamically partition a buffer to minimize total misses (Section 4.1);
   (c) dynamically and fairly reclaim and reallocate buffer space when there is memory pressure or a change in workload (Section 4.2);
   (d) rapidly retune a buffer partition (Section 4.3);
   (e) improve the gradient-descent method (Section 4.4); and
   (f) dynamically adjust buffer allocation to reach a P^{miss} target under differentiated service (Section 4.4).

The experiments either use instrumented PostgreSQL, or DB2's interface for retrieving P^{miss} data.

1.3 Overview

In the following, Section 2 introduces the buffer miss equation and validates it for different replacement policies and several commercial reference traces, as well as explaining the intuition for its efficacy. Section 3 uses static allocation to demonstrate some basic ideas in using the equation. Section 4 then demonstrates dynamic self-tuning to minimize miss probability, and shows how memory can be reclaimed and reallocated in a fair way. The final experiment shows how our approach can help, or replace, the widely-used gradient-descent method. Section 5 relates this article to previous work, and Section 6 concludes with a summary. The Appendix describes the underlying mathematical derivation.

Note that this article is not about the buffer miss equation itself, but on its *application* in buffer tuning. To draw an analogy, Floyd et al.'s paper on equation-based congestion control [Floyd et al. 2000] is not about the TCP equation (that was derived from a previous paper [Padhye et al. 2000]), but on applying that equation to control network traffic.

2. EQUATION AND VALIDATION

The buffer tuning techniques in this article are based on a buffer miss equation that we introduce in Section 2.1. We validate it for different buffer replacement policies in Section 2.2 and with various commercial workloads in Section 2.3, before explaining it in Section 2.4. The equation requires P^{miss} data to determine its parametric values, and Section 2.5 discusses possible sources for this data.

2.1 Buffer Miss Equation

The buffer miss equation is

$$P^{\text{miss}} = \frac{1}{2}(H + \sqrt{H^2 - 4})(P^* + P_c) - P_c, \tag{1}$$

$$\text{where } H = 1 + \frac{M^* + M_b}{M + M_b}, \quad \text{for } M \leq M^*.$$

Table I. Glossary of Notation

M	buffer size
M_i	size of buffer share for workload i, $M = M_1 + \cdots + M_k$
P^{miss}	probability of a buffer miss
P^*	probability of a cold miss (i.e., first reference to a page)
M^*	smallest buffer size for which $P^{\text{miss}} = P^*$
M_b	size of non-buffer memory occupied by database (Figure 11)
P_c	parameter that controls convexity of P^{miss} curve
μ	target P^{miss} for differentiated service

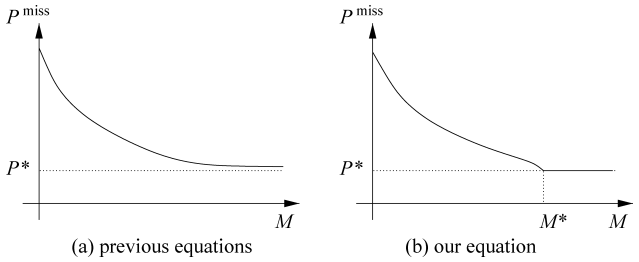


Fig. 1. Comparing our equation to previous equations.

($P^{\text{miss}} = P^*$ for $M > M^*$.) P^*, P_c, M^*, and M_b are parameters that depend on the transaction workload, buffer management, database instance, hardware configuration, etc. These four parameters are minimal in the following sense (Table I lists some notation used in this article).

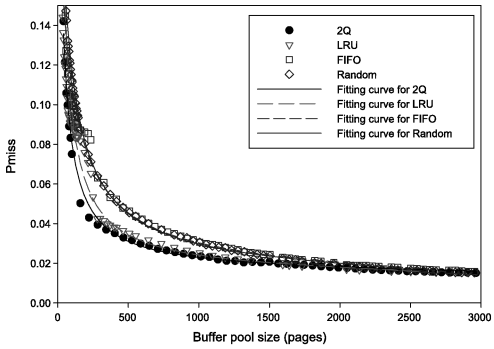
—P^* is the probability of a *cold miss* (i.e., the first reference to a page). It is an inherent characteristic of every reference pattern, and any equation for P^{miss} must account for it.

—When P^{miss} is plotted against M, they generally trace a decreasing curve, as illustrated in Figure 1. Previous equations for P^{miss} models this decrease as continuing forever [Belady 1996; Hsu et al. 2001; Storm et al. 2006; Tsuei et al. 1997]. This cannot be so: There must be some $M = M^*$ at which P^{miss} reaches its minimum P^*, as illustrated in Figure 1(b). Hence, our equation is the first to model this feature of P^{miss} behavior for database workloads.

—Memory for a database system is used for various purposes. Other than the buffer pools for caching data and disk pages, space is also needed for sorting, hashing, locks, code, etc. [Storm et al. 2006]. The memory used for such purposes is modeled by M_b (see Figure 11 in the Appendix).

—P^{miss} is the result of interaction between the reference pattern and buffer management; the latter covers replacement policy, prefetching, record creation, etc., and these are modeled by P_c. In effect, P_c controls the convexity of the P^{miss} curve.

Suppose we have a set of (M, P^{miss}) data points. If P^* is known (the case for Figure 2 in Section 2.2), we use regression [Tay and Zou 2006] to find values for M^*, M_b, and P_c to give a best fit between the points and the curve defined by the



$Policy$	P^*	P_c	M_b	M^*	R^2
2Q	0.0153	−0.0119	42.53	2693	0.992
LRU	0.0126	−0.0112	28.85	2596	0.973
FIFO	0.0152	−0.0087	97.63	2764	0.978
Random	0.0152	−0.0086	74.13	2631	0.995

Fig. 2. The buffer miss equation holds for different replacement policies. (TPC-C uses randomization in generating its transactions, so different experiments may have different P^* for cold misses.) Note the correspondence between convexity and P_c.

buffer miss equation. If P^* is not known (the case for Figure 3 in Section 2.3), we search iteratively for a P^* value that gives a best fit.

Other analytical (nonempirical) models for cache misses in the literature all impose some restrictions on either the replacement policy (e.g., LRU [Dan and Towsley 1990]) or the reference pattern (e.g., independence [Dan et al. 1995]). In contrast, our equation has no such restriction: Changes in replacement policy or reference locality only change the parametric values, not the equation. We next validate this claim.

2.2 Validation: Different Policies

We modify TPCC-UVa [Llanos 2006], an open-source implementation of the TPC-C benchmark [2006], to generate workloads to PostgreSQL version 8.0.0. The experiments are run on a Pentium 2.8 GHz Linux workstation. Our database size is 520MB, stored in an IBM SCSI 10000RPM hard disk. The default page size of PostgreSQL is 8KB.

We also implement LRU (least recently used), FIFO (first-in-first-out), and random replacement polices for PostgreSQL. For each policy, including the original 2Q policy of PostgreSQL-8.0.0, we run the TPC-C workload with different buffer-pool sizes.

Figure 2 plots measurements for a workload of 5 terminals, each generating 15 NEW-ORDER transactions to 1 warehouse. For each policy, the equation provides a smooth curve that gives a close fit for its data points.

Statisticians measure the quality of a regression fit by the coefficient of determination R^2; a perfect fit gives $R^2 = 1$. The R^2 values in Figure 2 exceed 0.97, thus indicating an excellent fit for each curve.

The interaction between replacement policy and reference pattern is very hard to model; previous analyses of LRU-K and GCLOCK, for example, were based on the independent reference model, which is a strong assumption that does not take into account temporal and spatial locality of references [O'Neil et al. 1999; Xi et al. 2001]. It is therefore significant that the buffer miss equation

is able to model various replacement policies by simply changing its parametric values.

2.3 Validation: Commercial Workloads

Many validation studies in the literature are based on simulation with TPC benchmarks, like we have done in Figure 2. However, these benchmarks are synthetic.

To stress test the equation's ability to fit different workloads, we use Hsu et al.'s data [2001]. That data was generated from trace simulation, where the traces were recorded on IBM mainframes with industrial-strength DB2/MVS for 12 commercial workloads from "ten of the world's largest corporations" (including aerospace, banking, consumer goods, direct mail marketing, financial services, insurance, retail, telecommunications, and utilities).

We can see from Figure 3 that our equation gives very good fit for most of the workloads. It also works well with the nonsmooth shape of measured data such as bank and retail. Most of the R^2 values exceed 0.95.

Note that by changing the parametric values, the equation is able to assume very different shapes to fit the data.

2.4 Intuition and Derivation

How is it possible that one equation with just four parameters is able to fit such a variety of replacement policies and reference patterns? To understand this, we briefly present here its underlying intuition.

We derive Eq. (1) from the following page fault equation [Tay and Zou 2006].

$$P^{\text{fault}} = \frac{1}{2}(H + \sqrt{H^2 - 4})(P^* + P_c) - P_c,$$

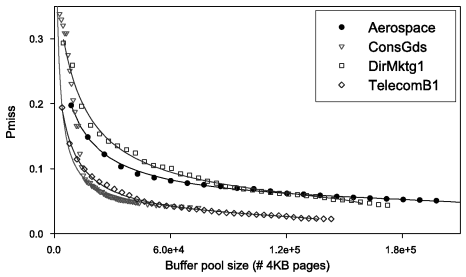
$$\text{where } H = 1 + \frac{M^*_{\text{RAM}} - M_0}{M_{\text{RAM}} - M_0}, \quad \text{for } M \le M^*_{\text{RAM}}$$

Here, P^{fault} is the probability of a page fault, M_{RAM} is the size of random access memory (RAM), and P^*, P_c, M^*_{RAM}, and M_0 are parameters. The page fault equation is based on the following *references+replacement invariant* [Tay and Zou 2006].

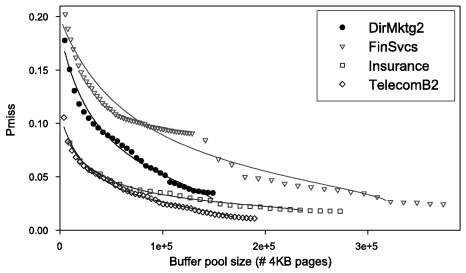
$$\left(1 - \frac{1}{r}\right)\left(1 + \frac{t_{\text{RAM}}}{t_{\text{disk}}}\right) \approx 1, \tag{2}$$

where r is the average number of times a page is read from disk, t_{RAM} is the average time a page stays in memory before eviction, and t_{disk} is the average time between a page's eviction and its reentry into main memory.

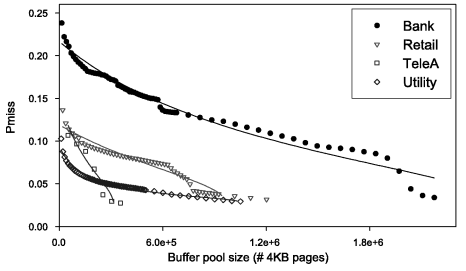
The invariant (2) captures the following intuition (see Figure 4): It is likely for r to be small and $\frac{t_{\text{RAM}}}{t_{\text{disk}}}$ to be large (as when there is little memory pressure), or for r to be large and $\frac{t_{\text{RAM}}}{t_{\text{disk}}}$ to be small (as when there is much memory pressure); and it is unlikely for r and $\frac{t_{\text{RAM}}}{t_{\text{disk}}}$ to be both small or both large. This is a general observation that should hold for any workload and any replacement policy, except for some worst-case scenarios.



Trace	P^*	P_c	M_b	M^*	R^2
Aerospace	0.0460	-0.0329	12393	212849	0.996
ConsGds	0.0402	-0.0259	1165	54647	0.831
DirMktg1	0.0455	-0.0245	4997	163234	0.994
TelecomB1	0.0210	-0.0100	3302	139994	0.997



Trace	P^*	P_c	M_b	M^*	R^2
DirMktg2	0.0242	0.0109	42023	150597	0.990
FinSvcs	0.0244	0.0038	51006	331107	0.969
Insurance	0.0178	-0.0102	11797	242887	0.975
TelecomB2	0.0110	0.0022	21678	172019	0.989



Trace	P^*	P_c	M_b	M^*	R^2
Bank	0.0338	0.0338	1255320	2472528	0.966
Retail	0.0317	0.0300	1141689	954140	0.939
TeleA	0.0275	0.0275	203255	312154	0.924
Utility	0.0293	-0.0192	220075	995467	0.993

Fig. 3. The buffer miss equation holds for various commercial workloads. Note how by changing the parametric values, the same equation can assume very different shapes.

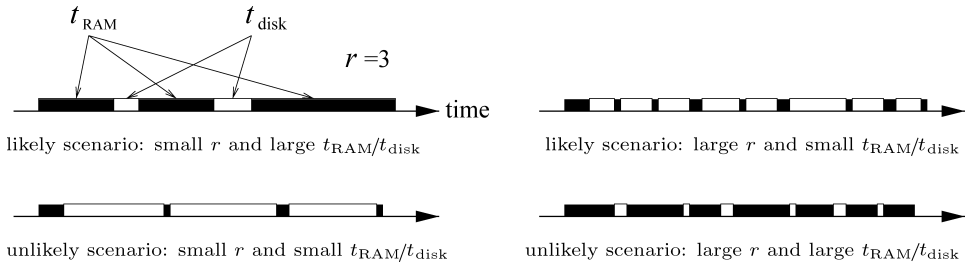


Fig. 4. Intuition for the references + replacement invariant: Likely and unlikely scenarios when workload reference pattern interacts with page replacement policy.

Tay and Zou then derived the page fault equation from the invariant (2) by applying Little's law [Jain 1991] to the page flow into and out of memory, and we derive the buffer miss equation from the page fault equation by focusing on the buffer pool (see Appendix A.1).

The equation's ability to fit a variety of patterns and policies thus comes from the preceding general observation in the invariant.

2.5 Sources of P^{miss} Data

The buffer miss equation relies on a set of (M, P^{miss}) data to determine its parametric values. Where do these data points come from?

Case I. (*There are Previous Measurements*). The system may be running a familiar production workload, for which there is a record of P^{miss} data. Some systems may also phase in a large change in buffer allocation through incremental changes [Storm et al. 2006], and each such small adjustment in M can provide a data point.

Case II. (*There are No Previous Measurements*).

(a) *The Replacement Policy Has the* Inclusion Property [*Mattson et al.* 1970]. This property says that the buffer contents for M also includes those for M' if $M \geq M'$. For example, LRU and LFU (least frequently used) have this property (but FIFO does not). For such policies, one can construct a Mattson stack [Zhou et al. 2004].

Briefly, this is how it works: The stack orders all references made so far in an epoch by recency; there are also two counters: one for cold misses and one for the number of references to each stack position. At the end of the epoch, the counters can be used to calculate P^{miss} for any M (in effect, this is a simulation).

Since the Mattson stack can be used to calculate P^{miss} for any M, it might seem like there's no longer any need for our equation. This is not so, for the following reasons.

(1) By fitting the data with our equation, we can do tuning calculations that cannot be done with the Mattson stack alone. For example, the fairness criterion for buffer reallocation in Section 4.2 (Eq. (5)) is stated in terms of the equation's parameters.

(2) P^{miss} values calculated with a Mattson stack can be nonconcave. Concavity is a useful property for optimization problems [Brown et al. 1996; Zhou et al. 2004]. We can use our equation to fit a concave shape to the data (Figure 3), and thus facilitate tuning optimizations (see Section 5).

(3) The equation provides a concise record (just four parameters) of the P^{miss} data, and the stack and counters can be discarded. The next time this workload is run, we can start with this equation (rather than from scratch, with no data). The equation can also be archived for offline analysis of how the P^{miss} curve is affected by changes in hardware, software, and workload.

(b) *The Replacement Policy Does Not Have the Inclusion Property.*
In practice, most replacement policies do not have the inclusion property. For example, LRU requires locking and manipulation of the stack, so a lower-cost approximation like CLOCK is often preferable [Bansal and Modha 2004]. Current versions of PostgreSQL and DB2 use variations of 2Q and GCLOCK, which do not have the inclusion property.

For such policies, one can do on-the-fly simulation to generate (M, P^{miss}) data points, as follows: The buffer manager contains a simulator for the replacement policy [Dias et al. 2005; Narayanan et al. 2005; Storm et al. 2006], and logs the references while the system is running at $M = B$, say. Once the log is deemed sufficiently long, the manager can run a background simulation of the policy against the reference trace to get P^{miss} data for any $M \neq B$.

Cost consideration may limit the number of simulated (M, P^{miss}) points, but, as we shall see in Section 4.4, our equation can work effectively with just a few data points. Besides, the simulation can be scaled back progressively as more P^{miss} measurements become available.

3. STATIC ALLOCATION

We first consider how the buffer miss equation can be used to determine the buffer size M and to partition it into $M_1, M_2, \ldots, M_k$. To simplify the introduction of ideas, we assume these are done statically, using previously collected P^{miss} data (Case I in Section 2.5). We will consider dynamic adjustments in Section 4.

3.1 Buffer Size *M*

One contribution of our equation is in identifying the buffer size M^* where P^{miss} is reduced to cold misses (see Figure 1). This property can be used for buffer sizing.

Several papers on buffer allocation start with a given M, then focus on how much to give to each query [Ng et al. 1995; Yu and Cornell 1991]. But how big should M be? Memory space is needed for many other purposes: for the operating system, network connections, metadata, sorting, hashing, etc.

Since any memory beyond M^* is not used by the workload, M^* is a natural choice for buffer size M.

Given a set of P^{miss} data, one can fit it with the equation to determine M^*. For example, if we use just the data points for $M < 1200$ in Figure 2, curve fitting gives $M^* = 2341, 2559, 2818$, and 2494 for 2Q, LRU, FIFO, and random, which are within 13%, 1%, 2%, and 6% (respectively) of the values in Figure 2. This accuracy is despite the predicted M^* values being far from the data points at $M < 1200$.

3.2 Reducing *M* for a Buffer in a Black Box

In contrast to the previous example of extrapolating from the data to estimate M^*, one may sometimes need to extrapolate backwards to determine the impact of *reducing* from $M = M_{\text{big}}$ to $M = M_{\text{small}}$ (e.g., server consolidation).

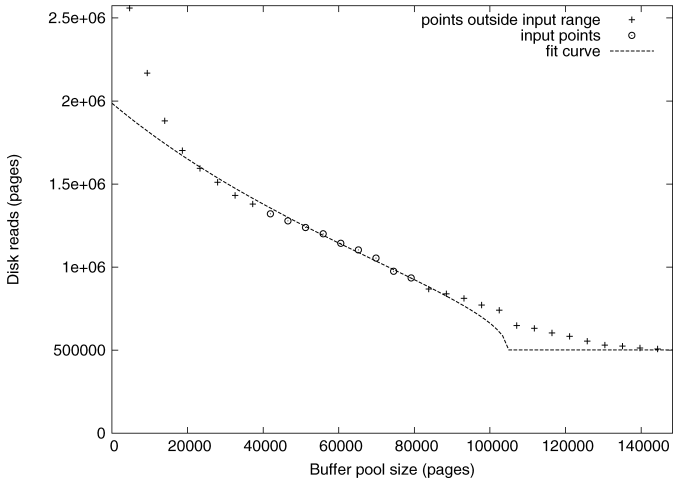


Fig. 5. Only the points (from Hsu's DirMktg2 dataset) inside the range $40000 < M < 80000$ are used for regression to get the curve. Backward extrapolation gives accurate predictions for $13000 < M < 40000$.

This problem may be made more difficult if the performance analyst has access to P^{miss} measurements but not the reference trace: In practice (e.g., for proprietary reasons), the database server may be a black box that contains the buffer, and only the buffer size and the misses that appear as disk reads are observable.

In particular, the hits are invisible, so it is impossible to tell what misses at $M = M_{\text{small}}$ may be generated by the references that are hits at $M = M_{\text{big}}$.

Nonetheless, our equation can be used to predict the number of misses at $M = M_{\text{small}}$. Multiplying Eq. (1) by the (unknown) reference length, we get

$$n^{\text{miss}} = \frac{1}{2}(H + \sqrt{H^2 - 4})(n^* + n_c) - n_c, \tag{3}$$

where n^{miss} is number of misses, n^* the cold misses, and n_c the counterpart of P_c.

By observing the misses from outside the server black box, one can measure n^{miss} for known M values, as well as n^*. Once this data is fitted with Eq. (3), we can use the equation to extrapolate backwards.

Figure 5 illustrates this for Hsu's DirMktg2 dataset. We use only the n^{miss} measurements for $40000 < M < 80000$ for curve fitting. The plot shows that the equation accurately predicts (less than 8% error) the n^{miss} values for $13000 < M < 40000$.

3.3 Buffer Partition $M_1, \ldots, M_k$

An enterprise may run heterogeneous workloads (e.g., batch and interactive) on one server, and a service provider may use one machine to host workloads from different clients. To protect the performance of each workload (so it satisfies some service-level agreement, say), the buffer size M may have to be partitioned into buffer pools of size $M_1, \ldots, M_k$.

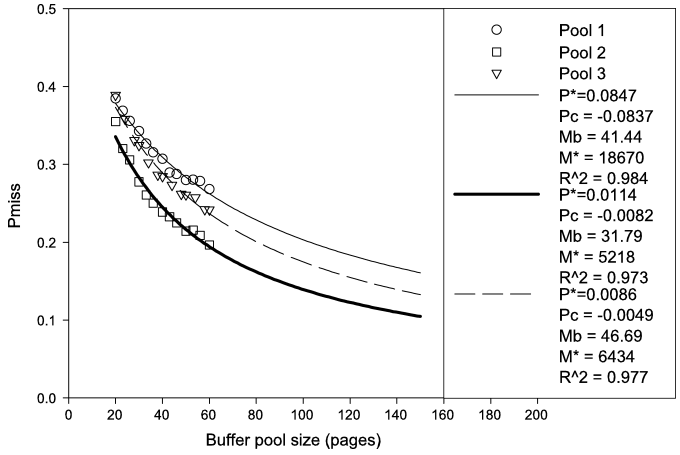


Fig. 6. P^{miss} prediction for buffer partitioning. By fitting miss probability data for each workload, the equation can be used to predict overall P^{miss} for any buffer partition.

Suppose we want to partition M into M_1, M_2, and M_3 for three workloads. Given three sets of miss probabilities (one for each workload and without the constraint $M_1 + M_2 + M_3 = M$), we can fit set i by our equation to get a miss probability function $f_i(M_i)$. We can then predict the aggregate P^{miss} by

$$P^{\text{miss}}(M_1, M_2, M_3) = w_1 f_1(M_1) + w_2 f_2(M_2) + w_3 f_3(M_3), \tag{4}$$

where w_i is the probability that a data reference belongs to pool i.

To test this idea, we implement multiple buffer pools in PostgreSQL, with each pool using its own LRU replacement policy. We run an experiment with three workloads, each with its own database and buffer pool.

The three workloads have 10, 20, and 35 terminals, respectively. Each workload has a different mix of TPC-C transactions (NEW-ORDER, PAYMENT, STOCK-LEVEL, ORDER-STATUS, and DELIVERY) and each terminal commits 30 transactions.

For each workload, we first measure miss probabilities for $M_i \leq 60$ pages and fit the data with the equation to get $f_i(M_i)$, as shown in Figure 6. We also measure the number of references L_i for workload i, and estimate w_i by $w_i \approx L_i/(L_1 + L_2 + L_3)$, without considering how a change in the partition affects w_i.

We then run the three workloads concurrently under the constraint $M_1 + M_2 + M_3 = 150$, measure the aggregate P^{miss} for the buffer, and compare it to the value predicted with Eq. (4). The results are plotted in Figure 7; among the 55 comparisons, the maximum relative error is 6.4%, and the average relative error is 1.9%. This accuracy is achieved despite our severe restriction in using only data from $M_i \leq 60$ for fitting the equation.

One can thus accurately locate the partition that optimizes a given objective function of the workloads' miss probabilities. For example, our prediction correctly locates $(M_1, M_2, M_3) = (20, 50, 80)$ as the partition that minimizes P^{miss} for the buffer. Note that we have no data for $M_3 > 60$, thus demonstrating extrapolation with $f_3(M_3)$.

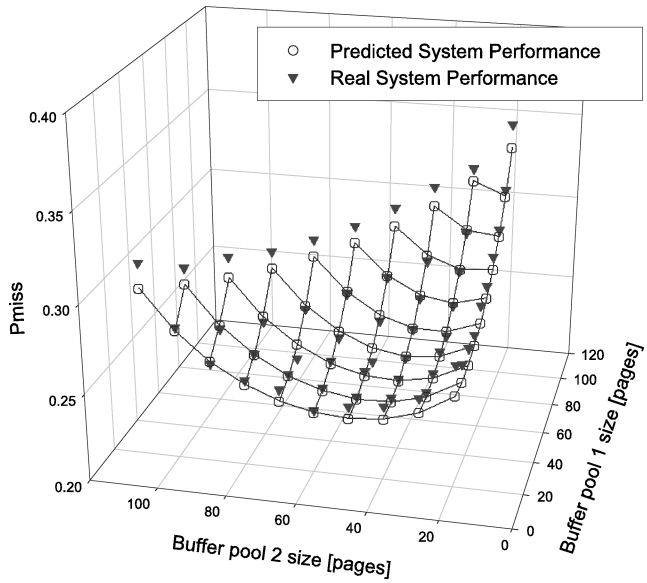


Fig. 7. Comparison between predicted and measured P^{miss} for buffer partition. For the 55 comparisons, the maximum relative error is 6.4%, and the average relative error is 1.9%.

4. DYNAMIC SELF-TUNING

Having laid the groundwork, we now focus on dynamic self-tuning of the buffer allocation. Section 4.1 first considers using epochs to make the static allocation of Section 3 online. Section 4.2 examines how memory can be fairly reclaimed or reallocated when there is a change in the workload, and Section 4.3 further considers how this can be done quickly when there are insufficient P^{miss} measurements. Section 4.4 then compares our tuning approach to the widely-used gradient-descent method.

4.1 Epochs and Online Allocation

A basic idea for dynamic tuning is to divide time into rounds [Ko et al. 2003], intervals [Brown et al. 1996], or epochs [Zhou et al. 2004], collecting data for each epoch and (when there is sufficient data) using measurements from previous epochs to retune in each epoch.

For example, in the buffer sizing problem of Section 3.1, the data for $M < 1200$ may have been obtained in previous epochs, when buffer allocation was changed as some concurrent workloads terminated or started up. Some system may also impose a limit on transferring memory from one workload to another [Storm et al. 2006], so that the transfer takes multiple epochs, each thus generating another data point.

If the workload is changing continuously, then a standard technique is to define a sliding window of recent epochs, and to discard old measurements that slide out of the window.

Table II. Using the Buffer Miss Equation to Minimize Aggregate P^{miss}

Epoch	Partition	P_1^{miss}	P_2^{miss}	P_3^{miss}	P^{miss}
1	(24,24,24)	0.437	0.429	0.416	0.434
2	(40,40,40)	0.347	0.358	0.360	0.350
3	(50,50,50)	0.311	0.347	0.353	0.320
4	(60,60,60)	0.280	0.328	0.330	0.292
5	(30,30,30)	0.377	0.397	0.393	0.381
6	(70,70,70)	0.252	0.319	0.302	0.266
7	(80,80,80)	0.236	0.308	0.301	0.253
8	(85,85,85)	0.228	0.295	0.282	0.243
9	(75,75,75)	0.237	0.315	0.323	0.256
10	(110,110,110)	0.209	0.291	0.256	0.225
11	(100,100,100)	0.217	0.287	0.271	0.232
12	(95,95,95)	0.213	0.284	0.291	0.231
13	(90,90,90)	0.223	0.284	0.285	0.237
14	(65,65,65)	0.254	0.309	0.313	0.268
15	(134,29,32)	0.207	0.395	0.377	0.249
16	(134,29,32)	0.215	0.380	0.379	0.254
17	(134,29,32)	0.213	0.401	0.377	0.254
18	(134,29,32)	0.207	0.401	0.383	0.251
19	(134,29,32)	0.213	0.386	0.401	0.252
20	(134,29,32)	0.209	0.386	0.390	0.251

Each set of P_i^{miss} data from the first 14 episodes is fitted with the equation at the end of epoch 14; the three equations are then used to find the optimal partition $(134, 29, 32)$. This partition of $M = 195$ gives a P^{miss} that is similar to the equipartition $(80, 80, 80)$ of $M = 240$ (saving $240 - 195 = 45$ pages).

For dynamic buffer partitioning, we use epochs of length 2 minutes each: long enough for a pool to reach steady state if its size has changed. We run a workload mix similar to that in Section 3.3, but with each terminal generating transactions indefinitely.

The buffer size M is changed at the beginning of each of the first 14 epochs, but the partition remains $M_1 = M_2 = M_3$. The miss probabilities P_1^{miss}, P_2^{miss}, and P_3^{miss} are recorded in Table II.

At the end of the 14th epoch, the buffer miss equation is used to fit the data for each pool. The three equations are then used in a nested iteration to find a partition to minimize the aggregate P^{miss} (Eq. (4)), while keeping M constant and $P_i^{\text{miss}} < 0.4$ (modeling a service-level constraint).

The calculated optimal partition is (134,29,32). Table II shows an immediate reduction in P^{miss} from 0.268 for the equipartition (65,65,65) in epoch 14 to 0.249 for partition (134,29,32) in epoch 15. This P^{miss} reduction appears small, but is significant for the two reasons next given.

(1) With equipartitioning, a similar reduction in P^{miss} requires $80 + 80 + 80 = 240$ pages, instead of $134 + 29 + 32 = 195$ pages.
(2) The P^{miss} reduction results in a throughput increase from 489 transactions/min. at epoch 14 to 517 transactions/min. thereafter.

4.2 Fair Reclamation and Reallocation

With static buffer allocation, overprovisioning is necessary to accommodate peak workloads. One compelling reason for dynamic self-tuning is that it allows overcommitment (instead of overprovisioning) of memory, and facilitates greater database server consolidation.

However, overcommitment requires rules and mechanisms for buffer reclamation when memory is short [Waldspurger 2002], as when a new job arrives to join the workload mix. We now demonstrate how our equation can be used to calculate the amount of space to reclaim from competing workloads.

Suppose the buffer is partitioned into pools of size $M_1, \ldots, M_k$. We want to reclaim amount Δ_i from M_i, so the ith pool has $M_i - \Delta_i$ after reclamation.

Let $\Delta = \Delta_1 + \cdots + \Delta_k$ be the target total to reclaim from the workloads. For example, some $\Delta > 0$ may be needed for assignment to a new arrival. Conversely, a terminating workload may release its buffer space for distribution to the other workloads, so $\Delta < 0$. If one workload has passed its peak, the buffer may be repartitioned by setting $\Delta = 0$.

It follows that $\Delta_i \geq 0$ and $\Delta_i < 0$ are both possible; $\Delta_i < 0$ just means the ith pool is enlarged after reclamation. What Δ_i should be depends on the reclamation criterion, such as minimizing the aggregate P^{miss}, equalizing the miss probabilities P_i^{miss} among the workloads, etc.

We illustrate with a fairness criterion: After reclamation, $\frac{P_i^{\text{miss}}}{P_i^*}$ should be the same for all i, where P_i^* is the cold miss probability for workload i. (In a reference string of length L, $n_i^* = P_i^* L$ is the number of distinct references and $n_i = P_i^{\text{miss}} L$ is the number of misses, so $\frac{P_i^{\text{miss}}}{P_i^*} = \frac{n_i}{n_i^*}$ is the number of misses per distinct reference.)

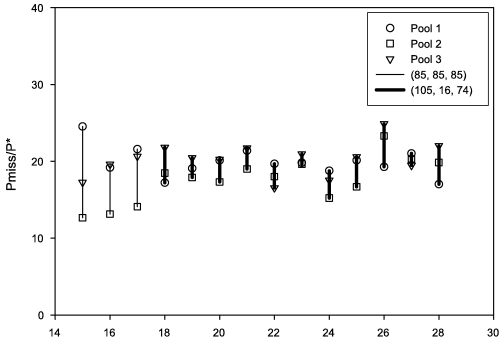
Suppose the P_i^{miss} data has been fitted with the buffer miss equation, giving parameters M_i^*, M_{bi}, and P_{ci}. One can show (Appendix A.2) that $\frac{P_i^{\text{miss}}}{P_i^*} = \frac{P_j^{\text{miss}}}{P_j^*}$ for all i and j after reclamation if

$$\Delta_i = (M_i + M_{bi}) - \beta_i \left(\sum_{r=1}^{k} (M_r + M_{br}) - \Delta \right)$$

$$\text{where } \beta_i = \frac{\left(\frac{P_{ci}}{P_i^*} + 1\right)(M_i^* + M_{bi})}{\sum_{r=1}^{k} \left(\frac{P_{cr}}{P_r^*} + 1\right)(M_r^* + M_{br})}. \tag{5}$$

For validation, we conduct an experiment with three pools, like the one in Table II. For the first 14 epochs, we collect P_i^{miss} data and fit it with the equation, giving the parameters in Figure 8. In epochs 15 to 17, the buffer of $M = 255$ pages is equally partitioned. At epoch 18, $\Delta = 60$ pages are reclaimed, using Eq. (5), which gives $(\Delta_1, \Delta_2, \Delta_3) = (-20.4, 68.6, 11.8)$; after reclamation, the partition is $(105, 16, 74)$.

Figure 8 shows that $\frac{P_2^{\text{miss}}}{P_2^*}$ is always smallest before reclamation, which is unfair. After 60 pages are reclaimed using our fairness criterion, the range in $\frac{P_i^{\text{miss}}}{P_i^*}$ is narrower, and no $\frac{P_i^{\text{miss}}}{P_i^*}$ is consistently smallest.



Parametric values for 3 workloads from epochs 1 to 14:

Pool	P_i^*	P_{ci}	M_{bi}	M_i^*	R^2
$i=1$	0.0133	-0.0121	116	44615	0.98
$i=2$	0.0262	-0.0244	97	30097	0.91
$i=3$	0.0174	-0.0156	112	31265	0.93

Fig. 8. Partition (85, 85, 85) for epochs 15 to 17 and repartition to (105, 16, 74) thereafter. Before reclamation, $\frac{P_2^{\text{miss}}}{P_2^*}$ is significantly lower than $\frac{P_1^{\text{miss}}}{P_1^*}$ and $\frac{P_3^{\text{miss}}}{P_3^*}$; after reclamation, the three are similar.

4.3 Fast Retuning

So far, we assume there are sufficient P^{miss} measurements for regression to calibrate the parameters. If there is a sudden change in workload (e.g., an increase in the number of terminals), one might want to immediately repartition without waiting through several epochs of data collection.

If the buffer manager is using a replacement policy that has the inclusion property, fast adaptation can be done by constructing a Mattson stack to calculate P^{miss} for various M (Case II(a) in Section 2.5).

To test this idea, we use LRU in PostgreSQL. We start three different workloads with 20, 16, and 14 terminals and a buffer size $M = 240$ partitioned into (80, 80, 80). The references in epoch 2 are used to construct a Mattson stack for each workload. In epoch 3, each stack is used to calculate 72 P_i^{miss} data points, which are then fitted with the buffer miss equation.

In Figure 9, there is an unfair spread in $\frac{P_i^{\text{miss}}}{P_i^*}$ values for the first three epochs. When epoch 4 begins, we use Eq. (5) and $\Delta = 0$ to repartition the buffer into (121, 89, 30). Figure 9 shows an immediate narrowing in the range of $\frac{P_i^{\text{miss}}}{P_i^*}$.

This exercise is repeated: At epoch 10, workload 1 reduces from 20 to 16 terminals and workload 2 increases from 16 to 20 terminals, while workload 3 remains unchanged. Figure 9 shows a bigger spread of $\frac{P_i^{\text{miss}}}{P_i^*}$ in epoch 10.

In epoch 12, we construct a Mattson stack, and the buffer is repartitioned at the beginning of epoch 14 into (102, 121, 17). After a 1-epoch time lag, the $\frac{P_i^{\text{miss}}}{P_i^*}$ range narrows again in epoch 15.

We thus see how the Mattson stack and buffer miss equation can be used together to quickly retune buffer allocation as workload changes. However, as the stack is constructed from the references in a single epoch, this procedure's effectiveness depends on how representative the epoch.

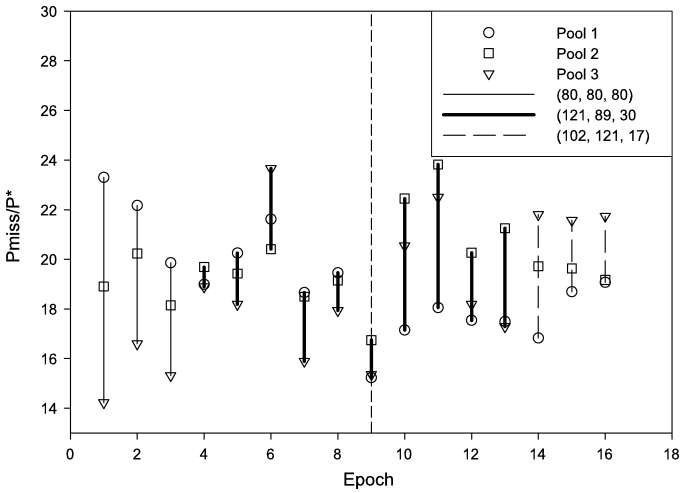


Fig. 9. Quick retuning with Mattson stack. Partition is (80, 80, 80) for epochs 1 to 3; Mattson stack is constructed in epoch 2; repartition to (121, 89, 30) in epoch 4; workload changes in epoch 10 (note the range increase in $\frac{P_i^{\text{miss}}}{P_i^*}$); Mattson stack is constructed in epoch 12; repartition to (102, 121, 17) in epoch 14.

If the replacement policy does not have the inclusion property, fast retuning can be done similarly: Instead of (in effect, a simulation with) a Mattson stack, we can run a background simulation of the policy against logged references (Case II(b) in Section 2.5). The number of points one can generate may be limited by simulation cost and epoch length, but we see in the next subsection that a few points may suffice for the equation to work effectively.

4.4 Comparison With Gradient Descent: Service Differentiation

Finally, we compare our technique to the oft-used gradient-descent method [Chung et al. 1995; Ko et al. 2003; Suh et al. 2004; Thiébaut et al. 1992; Tian et al. 2003], using an experiment that demonstrates the use of DB2's interface for P^{miss} retrieval (publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp); Oracle has a similar interface (www.dbspecialists.com/presentations/buffercache.html).

Gradient descent is used by Ko et al. to drive P^{miss} to a target miss probability μ [Ko et al. 2003]. To reach this target, they use the iteration

$$M(n+1) = M(n) + \frac{\epsilon}{g(n)}(P^{\text{miss}}(n) - \mu), \tag{6}$$

where $M(n)$ and $P^{\text{miss}}(n)$ are the buffer allocation and miss probability, respectively, in epoch n, ϵ is a weight, and $g(n) = \frac{P^{\text{miss}}(n) - P^{\text{miss}}(n-1)}{M(n) - M(n-1)}$, namely $g(n)$ estimates gradient $\frac{dP^{\text{miss}}}{dM}$ of the P^{miss} curve. If all $(M(n), P^{\text{miss}}(n))$ lie on a decreasing curve, then a negative ϵ will guarantee that $M(n)$ converges to μ.

In reality, $(M(n), P^{\text{miss}}(n))$ data can fluctuate and may be nonmonotonic (e.g., transient effects from user arrivals or departures), possibly causing $M(n)$ to diverge from μ. Moreover, the convergence rate remains an issue.

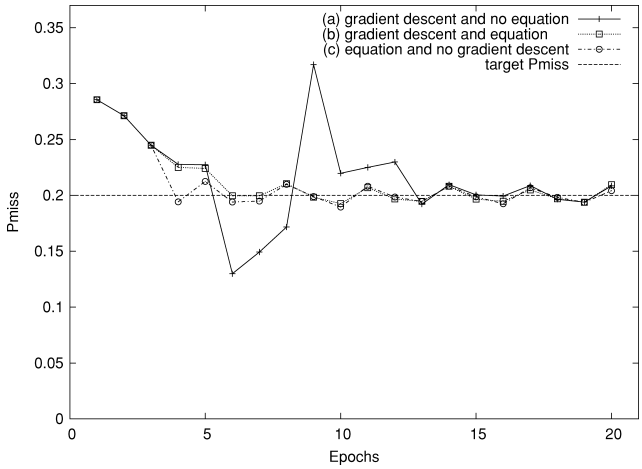


Fig. 10. Convergence comparison of three methods to bring P^{miss} to a target μ: (a) gradient descent, with gradient estimated from successive epochs; (b) gradient descent, with gradient calculated with fitted equation; and (c) M calculated with Eq. (7). For (a) and (b), $\epsilon = -0.5$ (following Ko et al.).

To illustrate, we run a TPC-C workload (mixture of 5 transaction types) on a DB2 database with 8 warehouses. The number of terminals in each epoch is constant, but changes cyclicly between epochs: 50,40,30,50,40,30,... This simulates the departure and arrival of users.

One can view the fluctuations as perturbations of a single workload. We set target $\mu = 0.2$ and (following Ko et al.) $\epsilon = -0.5$, and start with $M = 500$ in epoch 1 and $M = 700$ in epoch 2; these two data points suffice for gradient descent to proceed. Figure 10 shows that $P^{\text{miss}}(n)$ suffers big deviations from μ before reaching the target in epoch 13.

The big deviations are due to the poor gradient estimates that are caused by data fluctuation. One can use our equation to solve this problem, as follows: Suppose we have epochs $n = 1, 2, \ldots, k$ so far;

(i) fit $(M(1), P^{\text{miss}}(1)), \ldots, (M(k), P^{\text{miss}}(k))$ with the equation;
(ii) use the equation to evaluate $g(k) = \frac{dP^{\text{miss}}}{dM}$ at $M(k)$;
(iii) use Eq. (6) to determine $M(k+1)$.

Thus, the fitting smooths away the fluctuations, and the curve's gradient gives a better estimate of the P^{miss} gradient. Figure 10 shows that with this technique, $P^{\text{miss}}(n)$ converges steadily in six epochs to the target μ.

We can, in fact, replace the gradient descent entirely by using the equation to calculate M, as follows:

(i) fit $(M(1), P^{\text{miss}}(1)), \ldots, (M(k), P^{\text{miss}}(k))$ with the equation;
(ii) determine $M(k+1)$ by solving, from Eq. (1),

$$M = \frac{r}{r^2 - r + 1}(M^* + M_b) - M_b, \quad \text{where } r = \frac{\mu + P_c}{P^* + P_c}. \tag{7}$$

As there are four parameters (M^*, M_b, P^*, P_c), we start with $k = 3$ and search for a P_c value to give a best fit for 3 points. Figure 10 shows immediate

convergence: $P^{\text{miss}}(4) = 0.194$, which is 3% from the target $\mu = 0.2$. We see here that the equation can work effectively with a minimal number of data points.

After convergence, all three methods oscillate about μ, together with the cyclical change in workload.

5. RELATED WORK

We now survey some related work.

Lightstone et al. [2002] have described the technological and manpower issues that motivate the industrial push for self-designing, self-administering, and self-tuning systems [Autoadmin 2006; Autonomic Computing 2006]. Benoit [2005] singles out the buffer pool as one of the most important targets for self-tuning (another important memory allocation issue being the space reserved for operators like sorting and hashing [Dageville and Zait 2002]).

In fact, recent tuning software from major vendors all address the issue of buffer tuning [Dias et al. 2005; Narayanan et al. 2005; Storm et al. 2006]. However, they use trace simulation (instead of equations) to estimate the impact of buffer allocation on I/O costs. The buffer miss equation can be used to complement this approach, as described in Case II(b) of Section 2.5. If trace simulation is infeasible (e.g., the trace is not available, like in Section 3.2) or unrealistic (e.g., the simulator is outdated), then the equation can work with the raw data, like we demonstrated with DB2 in Section 4.4.

Many authors have found it useful to base their buffer tuning techniques on an equation relating P^{miss} to M [Chung et al. 1995; Storm et al. 2006; Tian et al. 2003; Tsuei et al. 1997]. Tian et al. [2003] consider the problem of allocating memory to database objects (tables, indices, etc.); this is different from the problem, considered here and elsewhere [Brown et al. 1996; Chung et al. 1995; Ko et al. 2003; Lu et al. 2002; Suh et al. 2004; Thiébaut et al. 1992], of buffer partitioning to suit multiclass workload.

In any case, the equation used by Chung et al. [1995], Tian et al. [2003], and Tsuei et al. [1997] is the power law that Brown et al. [1996] find a poor fit for database workloads. The power law is a rational function of M, whereas Storm et al. [2006] use an exponential function, but without validation.

Brown et al. avoid the need for a specific equation, and rely only on the function being concave. They use the line joining two neighboring data points (i.e., gradient descent) to direct the way towards a target miss probability μ. Concavity guarantees that their method converges. Unfortunately, real data often does not satisfy concavity (see Figure 3).

Nonetheless, one can still apply their idea by first fitting the raw data with the buffer miss equation. It is easy to show that the resulting curve has only one change in convexity, located near M^* (see the end of the curve in Figure 5). Since P^{miss} only changes significantly for $M \ll M^*$, the important part of the equation's curve is always concave.

Concavity is a helpful property for optimization problems [Zhou et al. 2004]. For example, concavity guarantees that P^{miss} (averaged over all workloads i) is minimized when the partition yields $\frac{\partial P_i^{\text{miss}}}{\partial M_i} = \frac{\partial P_j^{\text{miss}}}{\partial M_j}$ for all i and j, and there are algorithms that use this property [Suh et al. 2004; Thiébaut et al. 1992]. As

in the case of gradient descent (Section 4.4), raw data can give poor estimates of these derivatives, and cause convergence to be erratic and slow. Again, the buffer miss equation can be used to smooth the data (Section 4.4), or to calculate the minimizing partition (see Section 4.1).

However, for service differentiation, the objective is not to minimize average P^{miss}, but to satisfy class-specific service goals. Like Brown et al., Ko et al. [2003] translate the problem of satisfying workload-specific latency goals into target P_i^{miss} values. They consider P_i^{miss} to be an unknown function of M_i, and apply black-box control loops; one of these uses gradient descent. Their simplest controller for reaching target μ takes the form

$$M(n+1) = M(n) + \alpha(P^{\text{miss}}(n) - \mu),$$

where $M(n)$ and $P^{\text{miss}}(n)$ are the buffer allocation and miss probability, respectively, in epoch n, and α is a weight parameter. Given sufficient data for our buffer miss equation to fit, one can replace such a control algorithm with the simple calculation in Eq. (7). Figure 10 shows that this can provide faster convergence than gradient descent and black-box techniques. Quick and effective response is also demonstrated in Table II, Figure 8 and Figure 9.

For Lu et al., service differentiation is formulated as a target ratio ρ_{ij} for the hit probabilities of workloads i and j; they then design an adaptive controller on $\frac{M_i}{M_j}$ to drive convergence of $\frac{1-P_i^{\text{miss}}}{1-P_j^{\text{miss}}}$ towards ρ_{ij}. Again, given the buffer miss equation, one (can like in Eq. (7)) replace the control algorithm by a calculation of the desired $\frac{M_i}{M_j}$.

The buffer miss equation is derived from Tay and Zou's page fault equation [2006], which makes two modeling breakthroughs.

(1) It identifies a memory size M_{RAM}^* at which page faults reach a minimum; and
(2) it is derived from an analytical model that does not require restrictive assumptions.

In this article, we exploit these properties through the buffer miss equation, which works as follows.

(1) M_{RAM}^* locates the buffer size M^* at which P^{miss} reaches its minimum (see Figure 11). In contrast, previous equations (e.g., power law) all model P^{miss} as decreasing forever [Belady 1996; Hsu et al. 2001; Storm et al. 2006; Tsuei et al. 1997].

    Buffer management studies often start with a given buffer size M, and focus on dividing this buffer among queries [Ng et al. 1995; Yu and Cornell 1991]. They do not examine what M should be. In our context, the queries constitute a workload. The buffer management policy results in a P^{miss}-vs-M curve for this workload, thus defining an M^* that is a candidate for the buffer size.

    It is a natural candidate: It corresponds to the *knee point* (in the throughput-vs-M graph) that Tsuei et al. [1997] consider to be optimal, and is analogous to the *cache point* used by Dageville and Zait [2002] to size memory for SQL operators.

There is increasing interest in reducing energy costs for data centers, and memory is a prime consumer of electricity [Lefurgy et al. 2003]. Therefore, M^* can be used by various energy-saving algorithms to identify excess memory for powering down [Cai and Lu 2005].

(2) P^{miss} is defined by an intricate interaction, between workload reference pattern and buffer management policy, that is very hard to analyze mathematically. Previous analytical models all impose strong assumptions on the access pattern (e.g., independent references) or replacement policy (e.g., pure LRU) [Dan and Towsley 1990; Dan et al. 1995; O'Neil et al. 1999; Xi et al. 2001]. As for the empirical equations, they have no theoretical justification at all.

The strong underlying assumptions or empirical nature of current equations for buffer tuning leave one in doubt over their ability to model real P^{miss} data. In contrast, Tay and Zou's model is based on their general but intuitive references+replacement invariant (Figure 4). This gives us some confidence that the buffer miss equation is robust, as demonstrated in Figures 2 and 3.

6. CONCLUSION

Current methods for buffer tuning are based on simulation, black-box control, gradient descent, and empirical equations. This article presents a new approach to dynamic buffer self-tuning, using a buffer miss equation that is validated with different replacement policies (Figure 2) and commercial reference traces (Figure 3).

Unlike empirical equations, the buffer miss equation is based on an analytical model (Section 2.4). The data points for calibrating the equation may be from previous measurements (Case I in Section 2.5). If the replacement policy has the inclusion property, the data can also be generated on-the-fly (like a simulation) with a Mattson stack, but the equation remains useful for tuning calculation, performance optimization, and measurement records (Case II(a)). If the replacement policy does not have the inclusion property, the data points can be generated by a background simulation with logged references (Case II(b)); this simulation can be scaled back progressively as more P^{miss} measurements become available.

The equation identifies a natural candidate M^* for buffer size (Section 3.1). It can be used to extrapolate I/O costs when buffer size is changed (Figure 5), without need for reference traces. For multiclass workloads, by fitting P^{miss} data from each workload, we can predict miss probability for every partition (Figure 7), and optimize the partition dynamically (Table II). By restricting the range of data points, we demonstrate the accuracy of the equation in predicting P^{miss} outside the range (Section 3.1, Figures 5, 6, and 7).

By analyzing the equation (Appendix A.2), we determine how memory can be fairly reclaimed or reallocated (Eq. (5)) when workloads change, and demonstrate such a dynamic alteration for the case where P^{miss} data is already available (Figure 8), as well as with on-the-fly measurements (Figure 9).

The equation can be used to smoothen and speed-up convergence of the popular gradient-descent method for reaching a workload-specific P^{miss} target, or can replace that method entirely (Figure 10). This experiment also shows that the equation can work effectively with minimal P^{miss} data.

While the other experiments are done by instrumenting PostgreSQL, the final experiment uses only DB2's interface for retrieving miss probability.

APPENDIX

A.1 From Page Fault Equation to Buffer Miss Equation

This section derives the buffer miss equation from the following page fault equation [Tay and Zou 2006].

$$P^{\text{fault}} = \frac{1}{2}(H + \sqrt{H^2 - 4})(P^* + P_c) - P_c, \tag{8}$$

$$\text{where } H = 1 + \frac{M^*_{\text{RAM}} - M_0}{M_{\text{RAM}} - M_0}, \quad \text{for } M \le M^*_{\text{RAM}}.$$

($P^{\text{fault}} = P^*$ for $M \ge M^*_{\text{RAM}}$.)

Here, P^{fault} is the probability of a page fault, M_{RAM} is the size of random access memory (RAM), and P^*, P_c, M^*_{RAM}, and M_0 are parameters. In particular, M_0 measures the RAM space occupied by nonreplaceable pageframes, like those belonging to the kernel.

Tay and Zou have shown that this equation gives an excellent fit for page-fault data from a wide variety of workloads involving Windows, Linux, different replacement policies, multiprogramming, dynamic allocation, garbage collection, interactive applications, etc. The equation's robustness suggests that it should fit database workloads as well.

For this article, our focus is on M (the size of database buffer), so we need a change of variables. The amount of active RAM is

$$M_{\text{RAM}} = M_0 + M_b + M, \tag{9}$$

where M_b measures RAM space occupied by the buffer manager, metadata, indices, sorting, hashing, network connections, etc. This is illustrated in Figure 11.

If most of the paging activity is caused by buffer misses, then we have

$$P^{\text{fault}} \approx P^{\text{miss}}. \tag{10}$$

In particular, as M_{RAM} increases, P^{fault} and P^{miss} reach the cold miss P^* at the same point, so

$$M^*_{\text{RAM}} = M_0 + M_b + M^*. \tag{11}$$

Substituting Eqs. (9), (10), and (11) into Eq. (8) gives the buffer miss equation.

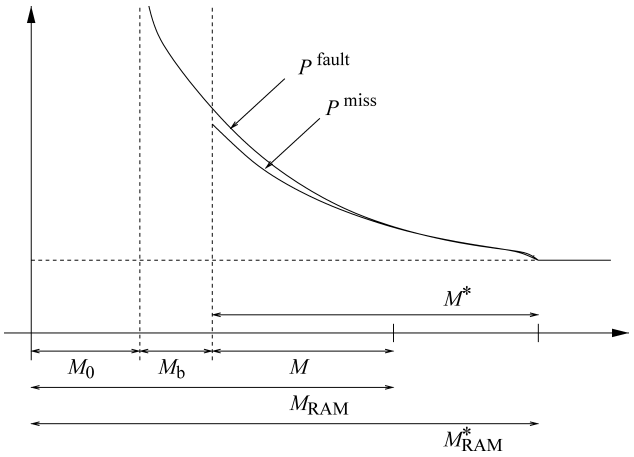


Fig. 11. Relationship between the page fault equation and the buffer miss equation. M_0 models memory occupied by the kernel, and M_b models memory occupied by application code as well as data outside of the database buffer.

A.2 Fair Reclamation

This section shows the derivation of Δ_i for fair reclamation. The buffer miss equation has an equivalent form

$$\left(r - 1 + \frac{1}{r}\right)(M + M_b) = M_a,$$

$$\text{where } r = \frac{P^{\text{miss}} + P_c}{P^* + P_c} \quad \text{and} \quad M_a = M^* + M_b.$$

Reclaiming Δ_i from pool i results in

$$\left(r_i - 1 + \frac{1}{r_i}\right)(M_i - \triangle_i + M_{bi}) = M_{ai}.$$

Let

$$\gamma_i = r_i - 1 + \frac{1}{r_i} = \frac{M_{ai}}{M_i - \Delta_i + M_{bi}}.$$

For the interesting case of $P_i^{\text{miss}} \gg P_i^*$, we have $r_i \gg 1$, so

$$\gamma_i \approx r_i - 1 = \frac{P_i^{\text{miss}} + P_{ci}}{P_i^* + P_{ci}} - 1.$$

This gives

$$\frac{P_i^{\text{miss}}}{P_i^*} = 1 + \left(\frac{P_{ci}}{P_i^*} + 1\right)\gamma_i = 1 + \frac{\left(\frac{P_{ci}}{P_i^*} + 1\right)M_{ai}}{M_i - \Delta_i + M_{bi}}.$$

For equal $\frac{P_i^{\text{miss}}}{P_i^*}$, we get (using $\Delta = \sum_{r=1}^{k} \Delta_r$)

$$\frac{\left(\frac{P_{ci}}{P_i^*} + 1\right)M_{ai}}{M_i - \Delta_i + M_{bi}} = \frac{\sum_{r=1}^{k}\left(\frac{P_{cr}}{P_r^*} + 1\right)M_{ar}}{\sum_{r=1}^{k} M_r - \Delta + \sum_{r=1}^{k} M_{br}},$$

which is equivalent to Eq. (5).

ACKNOWLEDGMENTS

The authors would like to thank Hsu and Smith for sharing data from their article Hsu et al. [2001].

REFERENCES

AUTOADMIN. 2006. AutoAdmin: Self-Tuning and self-administering datdbases. http://research.microsoft.com/dmx/autoadmin.

AUTONOMIC COMPUTING. 2006. Autonomic computing homepage. http://www.research.ibm.com/autonomic.

BANSAL, S. AND MODHA, D. S. 2004. Car: Clock with adaptive replacement. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies (FAST)*. USENIX Association, Berkeley, CA, 187–200.

BELADY, L. A. 1996. A study of replacement algorithms for virtual storage computer. *IBM Syst. J. 5, 2*, 78–101.

BENOIT, D. G. 2005. Automatic diagnosis of performance problems in database management systems. In *Proceedings of the 2nd International Conference on Automatic Computing (ICAC)*. IEEE Computer Society, Washington, DC, 326–327.

BROWN, K. P., CAREY, M. J., AND LIVNY, M. 1996. Goal-Oriented buffer management revisited. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM Press, New York, 353–364.

CAI, L. AND LU, Y.-H. 2005. Joint power management of memory and disk. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. IEEE Computer Society, Washington, DC, 86–91.

CHUNG, J.-Y., FERGUSON, D., WANG, G., NIKOLAOU, C., AND TENG, J. 1995. Goal-Oriented dynamic buffer pool management for data base systems. In *Proceedings of IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, Fort Lauderdale, FL. IEEE Computer Society, 191–198.

DAGEVILLE, B. AND ZAIT, M. 2002. SQL memory management in Oracle9i. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, Hong Kong, China. Morgan Kaufmann, 962–973.

DAN, A. AND TOWSLEY, D. 1990. An approximate analysis of the LRU and FIFO buffer replacement schemes. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*. ACM Press, New York, 143–152.

DAN, A., YU, P. S., AND CHUNG, J. Y. 1995. Characterization of database access pattern for analytic prediction of buffer hit probability. *The VLDB J. 4*, 1, 127–154.

DIAS, K., RAMACHER, M., SHAFT, U., VENKATARAMANI, V., AND WOOD, G. 2005. Automatic performance diagnosis and tuning in Oracle. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, Asilomar, CA. 84–94.

FLOYD, S., HANDLEY, M., PADHYE, J., AND WIDMER, J. 2000. Equation-Based congestion control for unicast applications. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*. ACM Press, New York, 43–56.

HSU, W. W., SMITH, A. J., AND YOUNG, H. C. 2001. I/O reference behavior of production database workloads and the TPC benchmarks—An analysis at the logical level. *ACM Trans. Database Syst. 26*, 1, 96–143.

JAIN, R. 1991. *The Art of Computer Systems Performance Analysis*. John Wiley, New York.

KO, B.-J., LEE, K.-W., AMIRI, K., AND CALO, S. 2003. Scalable service differentiation in a shared storage cache. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society, Washington, DC, 184–193.

LEFURGY, C., RAJAMANI, K., RAWSON, F., FELTER, W., KISTLER, M., AND KELLER, T. W. 2003. Energy management for commercial servers. *Comput. 36*, 12, 39–48.

LIGHTSTONE, S. S., LOHMAN, G., AND ZILIO, D. 2002. Toward autonomic computing with DB2 universal database. *SIGMOD Rec. 31*, 3, 55–61.

LLANOS, D. R. 2006. TPCC-UVa: An open-source TPC-C implementation for global performance measurement of computer systems. *SIGMOD Rec. 35*, 4, 6–15.

LU, Y., ABDELZAHER, T., LU, C., AND TAO, G. 2002. An adaptive control framework for QoS guarantees and its application to differentiated caching services. In *Proceedings of the IEEE International Workshop on Quality of Service (IWQoS)*, Miami Beach, FL. IEEE, 23–32.

MATTSON, R. L., GECSEI, J., SLUTZ, D. R., AND TRAIGER, I. L. 1970. Evaluation techniques for storage hierarchies. *IBM Syst. J. 9*, 2, 78–117.

NARAYANAN, D., THERESKA, E., AND AILAMAKI, A. 2005. Continuous resource monitoring for self-predicting DBMS. In *Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (*MASCOTS*). IEEE Computer Society, Washington, DC, 239–248.

NG, R., FALOUTSOS, C., AND SELLIS, T. 1995. Flexible and adaptable buffer management techniques for database management systems. *IEEE Trans. Comput. 44*, 4, 546–560.

O'NEIL, E. J., O'NEIL, P. E., AND WEIKUM, G. 1999. An optimality proof of the LRU-K page replacement algorithm. *J. ACM 46*, 1, 92–112.

PADHYE, J., FIROIU, V., TOWSLEY, D. F., AND KUROSE, J. F. 2000. Modeling TCP Reno performance: A simple model and its empirical validation. *IEEE/ACM Trans. Netw. 8*, 2, 133–145.

STORM, A. J., GARCIA-ARELLANO, C., LIGHTSTONE, S. S., DIAO, Y., AND SURENDRA, M. 2006. Adaptive self-tuning memory in DB2. In *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB)*. VLDB Endowment, 1081–1092.

SUH, G. E., RUDOLPH, L., AND DEVADAS, S. 2004. Dynamic partitioning of shared cache memory. *J. Supercomput. 28*, 1, 7–26.

TAY, Y. C. AND ZOU, M. 2006. A page fault equation for modeling the effect of memory size. *Perform. Eval. 63*, 2, 99–130.

THIÉBAUT, D., STONE, H. S., AND WOLF, J. L. 1992. Improving disk cache hit-ratios through cache partitioning. *IEEE Trans. Comput. 41*, 6, 665–676.

TIAN, W., MARTIN, P., AND POWLEY, W. 2003. Techniques for automatically sizing multiple buffer pools in DB2. In *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research (CASCON)*. IBM Press, 294–302.

TPC-C BENCHMARK. 2006. TPC-C V5 homepage. http://www.tpc.org/tpcc/.

TSUEI, T.-F., PACKER, A. N., AND KO, K.-T. 1997. Database buffer size investigation for OLTP workloads. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM Press, New York, 112–122.

WALDSPURGER, C. A. 2002. Memory resource management in VMware ESX server. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*. ACM Press, New York, 181–194.

XI, Y., MARTIN, P., AND POWLEY, W. 2001. An analytical model for buffer hit rate prediction. In *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research (CASCON)*. IBM Press, 18.

YU, P. S. AND CORNELL, D. W. 1991. Optimal buffer allocation in a multi-query environment. In *Proceedings of the 7th International Conference on Data Engineering*. IEEE Computer Society, Washington, DC, 622–631.

ZHOU, P., PANDEY, V., SUNDARESAN, J., RAGHURAMAN, A., ZHOU, Y., AND KUMAR, S. 2004. Dynamic tracking of page miss ratio curve for memory management. In *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM Press, New York, 177–188.

Received September 2007; revised December 2007; accepted December 2007