

**A RESEARCH PROGRAM PROPOSAL:
UNIVERSAL CACHE MISS EQUATIONS
FOR AUTONOMIC COMPUTING ***

Y.C. TAY♣

*Departments of Mathematics and Computer Science,
National University of Singapore, Kent Ridge 117543, Republic of Singapore*

♣ *E-mail: dcstayyc@nus.edu.sg
www.comp.nus.edu.sg/~tayyc*

Computer systems in large enterprises and in the outsourcing industry are difficult to manage manually. This difficulty has led to a push for *autonomic computing*, where the goal is to have the system automatically configure its settings, tune its performance and recover from failures.

Such large systems have innumerable caches to enhance performance. One task in autonomic computing lies in sizing these caches and sharing them out among competing workloads. Cache miss equations can help to automate this task.

For the cache miss equations to be helpful, they must be universal: they must fit any real memory reference pattern and cache management policy, through parameters that can be calibrated automatically and dynamically.

This paper proposes a research program to derive such equations. This proposal is for a drastic departure from forty years of cache miss analysis.

Three cache miss equations – one each for main memory, database buffers and processor caches – are presented as evidence for the feasibility of the research program.

Keywords: autonomic computing; cache miss; memory hierarchy

1. Introduction

Autonomic computing^{1,2} is driven by the increasing difficulty in manually configuring, tuning and maintaining enterprise-scale computer systems. As memory is a primary resource, some of this difficulty lies in sizing the many types of caches that are everywhere in every system.

The difficulty in system management has led to outsourcing and the

*This work was supported by National University of Singapore under ARF grant R-146-000-072-112.

growth of storage service providers, Internet data centers and virtual web hosting. These service providers must ensure that the sharing of resources — like caches — among their client systems satisfies their service level agreements. The cache sizing issue is thus compounded by share sizing.

The effectiveness of a cache in reducing latency or increasing throughput depends on the probability that a reference results in a miss. This **miss probability** P^{miss} can be reduced by making more cache available, but that is constrained by the size and share of the cache.

One approach to this cache configuration problem is to treat cache behavior as a black box, and apply control-theoretic techniques³. Black-box control is a general technique that may be necessary when the box is too complicated or poorly understood; the control can only improve if more is known about its behavior. Specifically, for a cache, much more engineering is possible if it can be modeled by a **miss equation** that describes how P^{miss} depends on the cache size or cache share M .

There is a huge literature on deriving miss equations for processor caches, main memory, database buffers, web proxies, etc. However, we are still far from being able to predict P^{miss} for realistic workloads. We believe that this slow maturation in analytic techniques for cache misses and the pressing need for miss equations call for a drastically different approach. We propose here a **research program**:

Focus modeling expertise on a top-down derivation of one universal miss equation for each cache type.

Section 2 explains the keywords in this program statement. In Section 3, we give some examples to illustrate the relevance of the research program to autonomic computing. Section 4 then presents some evidence that points to the feasibility of the program.

2. What the program statement means

One example of **cache type** is main memory, or RAM, i.e. consider RAM as a cache for process working sets. We first explain what the keywords in the program statement mean for RAMs.

A cache miss happens when a referenced page is not in RAM, resulting in a page fault. When considering RAM size M , the miss equation relates P^{miss} for the workload mix to M ; when sizing RAM share for a process, P^{miss} is the miss probability for the process and M is its memory allocation.

The miss equation is determined by a complex interaction between reference pattern and replacement policy, and influenced by myriad factors.

To illustrate: the reference pattern depends on the application mix, whether and how these applications share data, the data layout on disks and whether there is prefetching, the data instance and how it changes over time, how meta-data like indices are organized, when dirty pages get written out, the hardware combination and software versions, and how the whole lot is configured.

Most previous attempts at deriving miss equations are by **bottom-up** analyses that start with a mathematical description of the patterns and policies. For tractability, analysts have to drastically simplify the problem (e.g. run-alone application, a pure-LRU policy) and adopt simplistic assumptions (e.g. independent references, no prefetching). The model is thus far from the reality described above.

The sort of complications that make autonomic computing and outsourcing necessary — complex mix of applications with different characteristics, hardware and software configuration, customization and evolution, dynamic fluctuations and flash crowds, shortage of modeling expertise, etc. — also make bottom-up analysis inadequate and unscalable.

For a miss equation to have a role in autonomic computing, it must be **universal**, in the sense that it holds for all real reference patterns and replacement policies. To attain universality, the miss equation is allowed a small number of **parameters**, whose values depend on the specific pattern and policy. These parameters are to be calibrated automatically, dynamically and adaptively.

For example, our candidate for a universal miss equation for RAM is

$$P^{\text{miss}} = \frac{1}{2}(H + \sqrt{H^2 - 4})(P^* + P_0) - P_0, \quad (1)$$

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

The parameters are M_0 , M^* , P^* and P_0 ; M_0 and M^* are the minimum and maximum M that are necessary (M_0) and sufficient (M^*), P^* is the cold miss probability, and P_0 depends on dynamic memory allocation in the application, prefetching by the paging policy, etc. ⁴.

Therefore, to understand the effect of prefetching, one would analyze how it affects P_0 ; to study the role of a replacement policy, one would examine how it determines P_0 , P^* , M_0 and M^* , and express them in terms of other parameters that describe the policy. This approach of progressively refining the parameterization with increasingly detailed analysis is what we mean by a **top-down** derivation in the research program.

3. Using cache miss equations for autonomic computing

We now illustrate how miss equations can be used for autonomic computing.

In SANBoost, the pooled storage capacity in a storage area network is partitioned among logical units (LUs), and a solid-state disk is used as a cache to be shared among these LUs; LU_{*i*}'s cache share M_i is determined by its IO rate⁵. Given a universal cache miss equation, the virtualization controller can, instead, record cache miss data P_i^{miss} for various M_i values, fit the data dynamically with the equation to determine the parametric values for LU_{*i*}, then use the equations (one per LU_{*i*}) to adaptively decide the cache share M_i .

For example, Lu et al.⁶ enforce QoS guarantees for a differentiated caching service by setting target miss probability ratios ρ_{ij} : if Class k has miss probability P_k^{miss} , their feedback control aims to make $\frac{P_i^{\text{miss}}}{P_j^{\text{miss}}}$ converge to ρ_{ij} . Their scheme considers P_k^{miss} to be an unknown function of M_k . Given the miss equation, such black-box control is unnecessary, and one can use (1) to explicitly compute the M_i and M_j values that satisfy $\frac{P_i^{\text{miss}}}{P_j^{\text{miss}}} = \rho_{ij}$.

Similarly, one can use the miss equation to replace the control schemes in Ko et al.'s QoS architecture for shared storage proxy cache⁷. Their architecture also has a fairness controller and contention resolver to ensure fairness in case cache size is smaller than the sum of desired M_i 's. Their fairness criterion is based on the allocated share M_i ; given a miss equation, one can also take the desired M_i^* into account⁴.

The pooling of resources to be shared by multiple workloads has led to the development of workload management (WLM) middleware that needs operating system support. Franke et al.'s Class-based Kernel Resource Management (CKRM) framework provides such support through differentiated service for all major resources, including memory⁸. For example, their page replacement policy takes into account RAM share specified by the WLM. Given a miss equation, more can be done; for instance, when there is a shortage of page frames, the memory manager can use the miss equation to decide how much to trim from each working set⁴.

4. Feasibility of the research program

Our research program calls for one universal miss equation for each cache type, audaciously targeting the entire memory hierarchy when it is not even clear that universality is possible for any one type.

We now address this feasibility issue by presenting some supporting data for main memory, and candidate equations for database buffers and

processor caches.

4.1. *Cache type = main memory*

For main memory, an equivalent form of the miss equation is the page fault equation

$$n = \frac{1}{2}(H + \sqrt{H^2 - 4})(n^* + n_0) - n_0, \quad (2)$$

where, for a given reference string, n is the number of page faults, n^* the number of cold misses, and n_0 a parameter corresponding to P_0 . Fig. 1 (extracted from previous work⁴) shows how closely the page fault equation (2) fits measurements of real workloads.

Consolidated or outsourced workloads on a system can have very different characteristics. Fig. 1 shows that (2) indeed gives a good fit for (i) compute-intensive, (ii) IO-intensive and (iii) memory-intensive workloads, and for both Linux and Windows; note how, with just parameters M_0 , M^* , n_0 and n^* , the equation can assume very different shapes.

Furthermore, Fig. 1(iv) shows that it can fit different replacement policies. In practice, kernel patches are regular and applications evolve from one version to another; Fig. 1(v) shows that the equation can handle such changes.

This evidence, while limited, suggests universality is possible for a main memory miss equation.

4.2. *Cache type = database buffer*

A database system has hundreds of tuning knobs, many of which concern buffer pools⁹. In preliminary experiments, we see an excellent fit between equation (1) and buffer measurements for TPC-H transactions running on MySQL, as well as for TPC-C transactions running on PostgreSQL — using FIFO, LRU and 2Q buffer replacement — thus demonstrating universality.

In line with our top-down approach, we have refined (1) to get the following candidate miss equation for database buffers:

$$P^{\text{miss}} = \frac{1}{2}(H + \sqrt{H^2 - 4})\left(\frac{n_{\text{record}}}{n_{\text{request}}} + P_0\right) - P_0$$

$$\text{where } H = 1 + \frac{n_{\text{xacn}}n_{\text{record}} - M_0}{M - M_0},$$

M is the buffer size, n_{xacn} is the number of concurrent transactions, and each transaction makes n_{request} requests that are spread over n_{record} records

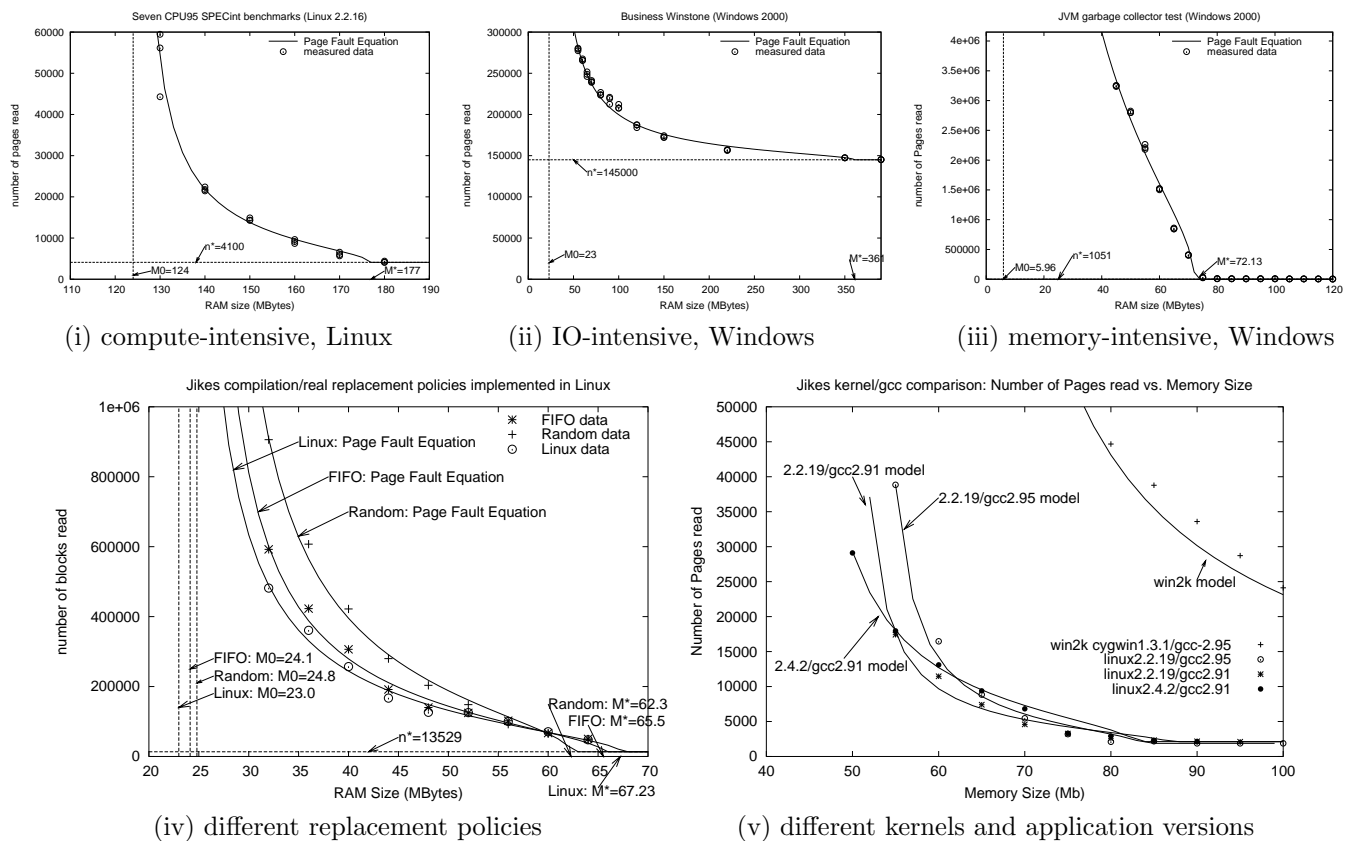


Fig. 1. The page fault equation (2) gives a close fit for measurements with real workloads.

(a transaction may request a record multiple times, e.g. while joining two relations). We are now validating this database-specific miss equation.

4.3. *Cache type = processor cache*

Unlike main memory and database buffers, a processor cache is, in general, two-dimensional — its size is determined by associativity A and number of sets S . Our current candidate ¹⁰ for a universal processor cache miss equation is

$$P^{\text{miss}} = P^* + (1 - P^*) \left(1 - \left(1 - \frac{P_1}{S^f A^h} \right)^g \right),$$

where P^* is the cold miss probability, P_1 is the miss probability if the cache has only 1 block, g the average stack distance in the reference string, and f and h (spatial/temporal) locality parameters. Such an equation can be used to dynamically partition the cache among concurrent processes ¹¹.

We have verified that this equation fits simulation data for many traces available at BYU's Trace Distribution Center (<http://tds.cs.byu.edu>). They include instruction, data and unified cache simulation, and floating point (e.g. scientific) and integer (e.g. gcc) traces, thus demonstrating universality. However, there are few traces there with sufficiently large variation in P^{miss} to stress-test the equation.

5. Conclusion

Caches are everywhere in any computer system. As memory is a primary resource, one major task in autonomic computing lies in sizing these caches and sharing them out among competing workloads. Miss equations can facilitate this task.

However, for them to be useful in automatic and dynamic calculation of cache size or share, the equations must be universal, i.e. work for any combination of real reference patterns and caching policies. We therefore propose here a research program to redirect modeling expertise to the derivation of universal cache miss equations.

As evidence for the feasibility of the research program, we presented candidate equations for main memory, database buffers and processor caches. We are now working on a miss equation for web proxies — their continuously changing workload and continually reconfigured policies ¹² require that the miss equations must be universal to be useful.

References

1. Autonomic computing <http://www.research.ibm.com/autonomic>.
2. Autoadmin <http://research.microsoft.com/dmx/autoadmin>.
3. M. Karlsson, C. Karamanolis and X. Zhu, *ACM Transactions on Storage* **1**, 457 (2005).
4. Y. C. Tay and M. Zou, *Performance Evaluation* **63**, 99 (2006).
5. I. Ari, M. Gottwals and D. Henze, SANBoost: Automated SAN-level caching in storage area networks, in *Proc. Int. Conf. Autonomic Computing*, (New York, NY, 2004).
6. Y. Lu, T. Abdelzaher, C. Lu and G. Tao, An adaptive control framework for QoS guarantees and its application to differentiated caching services, in *Proc. Int. Workshop on Quality of Service*, (Miami Beach, FL, 2002).
7. B.-J. Ko, K.-W. Lee, K. Amiri and S. B. Calo, Scalable service differentiation in a shared storage cache, in *Proc. Int. Conf. Distributed Computing Systems*, (Providence, RI, 2003).
8. H. Franke, S. Nagar, C. Seetharaman, V. Kashyap, H. Zheng and J. Kong, Enabling autonomic workload management in Linux, in *Proc. Int. Conf. Autonomic Computing*, (New York, NY, 2004).
9. D. G. Benoit, Automatic diagnosis of performance problems in database management systems, in *Proc. Int. Conf. Autonomic Computing*, (Seattle, WA, 2005).
10. B. Peng, Y. C. Tay and W. F. Wong, *Manuscript* (2005).
11. G. E. Suh, S. Devadas and L. Rudolph, Analytical cache models with applications to cache partitioning, in *Proc. Int. Conf. on Supercomputing*, (Sorrento, Italy, 2001).
12. F. Ogel, B. Folliot and I. Piumarta, On reflexive and dynamically adaptable environments for distributed computing, in *Proc. Autonomic Computing Workshop*, (Seattle, WA, 2003).