

**Using Statistical Correlation For Dependency Analysis Of  
Cache Replacement Policies**

Technical Report UCSC-CRL-03-16

Ismail Ari

Storage Systems Research Center  
Jack Baskin School of Engineering  
University of California, Santa Cruz  
Santa Cruz, CA 95064  
<http://ssrc.cse.ucsc.edu/>

February 12, 2004

## **Abstract**

New cache replacement policies are continuously being designed and the old ones are being tuned for specific workloads. Heterogeneous policies are being used together either to complement each other within the cache hierarchies or to manage the cache resources in a single operating system in a cooperative fashion. If we consider cache replacement policies to be “cache experts”, then it becomes crucial to understand whether we are really consulting independent experts or a group of dependent experts. The information gained from a group of experts can be only as good as information gained from one of those experts, in which case there is no need to maintain the whole group. We use statistical correlation formulas to quantify the amount of dependency between a large set of well-known cache replacement policies and find high dependencies.

# 1 Introduction

The motivation behind consulting multiple experts is that a group of experts can provide more information than a single expert [11]. By suitable combinations of these experts better predictions with higher accuracies may be achieved. The disagreement between experts is actually useful, since if the experts never disagreed there would be no need to consult multiple experts [11]. The goal is to be able to find a set of experts who differ in their decisions to complement each other well. However, we must first quantify these differences. In this paper we are using a statistical correlation formula as a tool to measure the relatedness or dependency between a large set of cache replacement policies.

One common approach in systems design is to choose the expert that performs the best [3] on a given task and to fine tune this best expert for a specific system to get highest possible accuracy. However, fine tuning today's complex systems that are servicing complex workloads is a tedious task [5] and making wrong decisions has high performance and monetary costs. Workloads change over time, mix with other workloads or get filtered [19, 4] by peer servers making the analysis even more tedious. Besides there are still patterns that are undetected by even the best expert, while these patterns can be detected by some other experts.

## 2 Methodology

Statistical procedures are used to quantify the *relation* among a set of random variables  $X_1, X_2, \dots, X_n$ . In the caching case the hits and misses of replacement policies within a time period could be used as the random variables and the relation will be the degree to which these variables vary together or *covary*.

The formula for correlation is derived from the formula for *covariance*. The covariance of two random variables  $X$  and  $Y$  is a way of measuring the dependency between  $X$  and  $Y$ . If paired  $X$  and  $Y$  values tend to both be above or below their

means at the same time, this will lead to a high positive covariance. It is defined by:

$$\begin{aligned} cov[X, Y] &= E[(X - E[X])(Y - E[Y])] = \\ &E[X.Y] - E[X].E[Y] = \overline{XY} - \overline{X}.\overline{Y} \end{aligned} \quad (1)$$

Note that if  $X = Y$ , then the covariance is equal to the variance:

$$cov[X, X] = \overline{X^2} - \overline{X}^2 = var(X) = \sigma_X^2. \quad (2)$$

and if  $X$  and  $Y$  are statistically independent, then  $E[X.Y] = E[X].E[Y]$  and  $cov[X, X] = 0$ . The covariance itself must be standardized before it can be useful for comparison purposes with different data sets, since it is sensitive to the standard deviation of  $X$  and  $Y$ . The Pearson Product-Moment Correlation Coefficient,  $r$ , is a simple way to provide this standardization:

$$r = Cov(X, Y) / S_X . S_Y = SP_{XY} / \sqrt{SS_X . SS_Y} \quad (3)$$

where,

$$\begin{aligned} SS_X &= \sigma(X - \overline{X})^2 = \sigma_X^2 - (\sigma_X)^2 / N, \\ SS_Y &= \sigma(Y - \overline{Y})^2 = \sigma_Y^2 - (\sigma_Y)^2 / N, \end{aligned} \quad (4)$$

Correlations range from -1 (perfect negative relation) through 0 (no relation) and to +1 (perfect positive relation). For a pair of random numbers to be uncorrelated the two streams need to vary randomly around their means with respect to each other.

## 3 Simulation Setup

We implemented a cache simulator in C++ [6] and made it easy for new cache replacement policies to be integrated and compared. Broad range of candidate recency-based, frequency-based, and size-based replacement policies have been implemented and added to the pool of policies. In this paper, we compare policies using a single cache space. Our future work includes extending this to multiple cache levels, and cache topologies.

### 3.1 Cache Replacement Policies

Here we give an overview of the implemented policies leaving out the details to the referenced literature.

Different cache replacement policies use different criteria to make local replacement decisions. Random, First-In-First-Out (FIFO) and Last-In-First-Out (LIFO) policies replace the objects suggested by their names and do not require any information about the objects to be replaced. Time, frequency and object size are the most commonly used criteria for local replacement decisions. Least Recently Used (LRU) is the most popular policy and uses recency of access as the sole criteria for replacement. Least Frequently Used (LFU) uses popularity or frequency of access information and replaces the least popular objects. Most Recently Used (MRU) and Most Frequently Used (MFU) are inferior when used alone, since they replace recently requested and popular objects. They may be beneficial in mixtures of policies, therefore we include them for completeness.

SIZE replaces the largest object and GreedyDual-Size (GDS) [13, 10] replaces the object with the smallest key  $K_i = C_i/S_i + L$ , where  $C_i$  is the retrieval cost,  $S_i$  is the size and  $L$  is a running age factor.  $L$  is set to the key value of the objects that are replaced from the cache. GDS with Frequency (GDSF) [8] adds the frequency of access,  $F_i$ , into the same equation and replaces the object with the smallest key  $K_i = (C_i \times F_i)/S_i + L$ . LFU with Dynamic Aging (LFUDA) replaces the object with minimum  $K_i = (C_i \times F_i) + L$  [8]. GreedyDual\* [13] is a generalization of GDS that captures both long-term popularity and short-term temporal correlations. We are currently working on implementations of LRU-K [17, 18], 2Q [14], MQ [21] and Unified Buffer Management (UBM) [15] policies. Other taxonomies of replacement policies are presented in prior work [7, 13].

### 3.2 Workloads

Startup trace [2] is a one day log of HTTP requests to a major proxy cache in the Squid na-

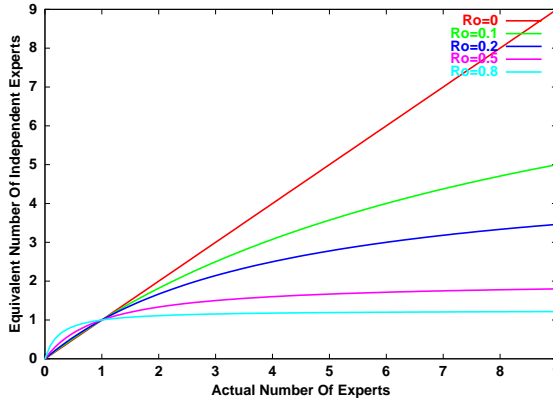


Figure 1: Effective number of independent experts decreases in a set of dependent experts drastically with increased correlation.

tional caching hierarchy by National Lab of Applied Network Research (NLANR). The trace has around 54,000 requests to 144 MBytes of unique data and a hit rate at infinite cache size ( $HR^\infty$ ) of 52%. Digital Equipment Corporation (DEC) web proxy [1] proxy served 14,000 workstations in DEC in 1996. We used trace of date 9/16/96 with approximately 1.2 million requests accessing 6 GBytes of unique data and  $HR^\infty$  of 57%. The Zipf slope of this trace is 0.78.

## 4 Analysis and Results

Equivalent or effective number of independent experts is much smaller than the actual number of experts if the experts are highly correlated. Clemen and Winkler have derived the following formula [11] to provide a feel for the impact of dependence on the value of information from multiple sources:

$$f(x) = x / (1 + (x - 1) \times Ro) \quad (5)$$

where  $x$  is the number of experts and  $Ro$  is the correlation between the experts. Figure 1 plots Equ. 5 for different values of  $Ro$ . The linear line is for independent experts. As the correlation among experts increases the equivalent number of experts on the Y-axis drops drastically.

Figure 2 shows the effect of cache size on a set of replacement policies for the Startup workload. The infinite cache size of the tested workload was around 144 MBytes, which is the unique

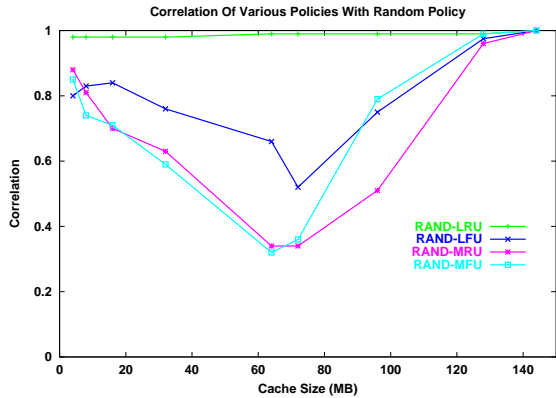


Figure 2: Effect of cache size on the correlation between policies. The effect of *policy* on correlation is much higher in mid size caches and the effect of *cache size* is much higher closer to the infinite and zero cache sizes.

amount of bytes in this request stream. When the cache size is infinite there is no need for replacement, thus the policies have the same hit and miss results. The misses are the compulsory misses and the hits are any re-references. The correlation of all the policies at infinite cache is one signifying a perfect agreement.

When the cache sizes are small, the amount of data and re-references the policies can see within their stack distance is minimal. Therefore, policies can distinguish themselves from each other minimally. So, the correlations are also high at this edge although not as high as the infinite cache end.

The effect of differences of policies on correlation is most significant in the middle region which is around half of the infinite cache size for the two Zipf distributed web proxy workloads tested. In this zone of cache sizes policies have enough space to prove their worth or difference from others. Random and LRU are highly correlated in most of the time, this is probably due to the expected cache residency time [19] during which some re-references are possible if the policy is not attacking the temporal locality. This is exactly what MRU is doing resulting in reduced correlation with the Random policy.

Figure 3 shows the correlation results for the twelve tested policies as a matrix of gray scale

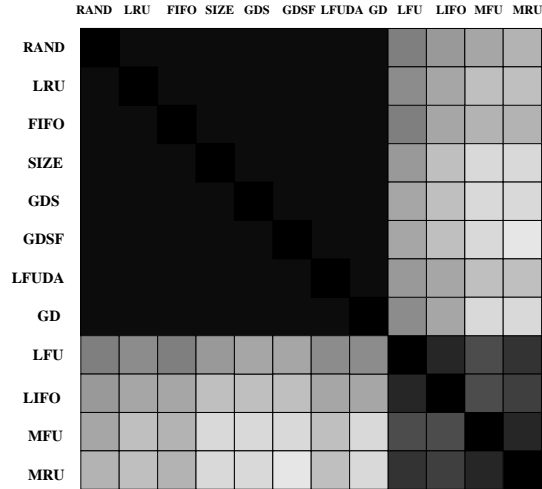


Figure 3: This graph shows the correlation results between policies as a correlation matrix. The density of gray is proportional to the correlation. Black denotes full correlation. The policies tested have resulted in two camps in terms of their correlations for the Startap web proxy trace used.

intensity for the NLANR Startap workload. The cache size is the half of infinite cache size ( $144/2=72$  MBytes). The darker colors signify higher correlations (close to 1). Note that the diagonal is completely black, since the correlation of a policy with itself is one. In the initial matrix the order of the policies were different and the darker zones were more distributed. By putting the policies with higher correlations together we were actually able to find the two major camps of policies that agreed with the policies within their group much more than the policies in the other group as seen by the lighter regions.

Figure 4 shows the correlations for the DEC web proxy workload with a smaller set of policies and for half of the infinite cache size for this workload ( $6\text{ GBytes}/2 = 3072$  MBytes). There is a similar pattern, although LFU was more integrated into the first camp.

## 5 Implications Of The Results

Using the correlation information a cache designer could choose the simplest policy to implement in a set of correlated policies.

If caches in a hierarchy are of the same size

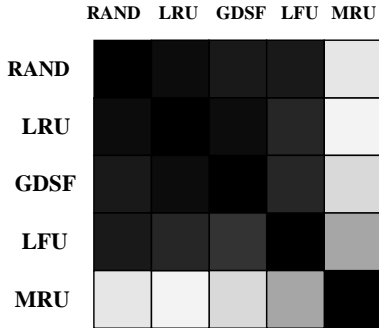


Figure 4: This graph shows the similar correlation results for the DEC trace, with a smaller set of policies and at cache size of 3072 MBytes.

and if they hold exactly the same elements then a miss in one of them will also result a miss in the other ones. This is called *inclusive caching* [20] and makes upper levels useless. This is usually the case when the same or highly dependent cache replacement policies are used at all levels. We would like to achieve as much *exclusive caching* [20] as possible between the collaborating caches, so that the cluster has the effect of a one big unified cache to the users. Using heterogeneous caching policies (policies that are different in nature) has been demonstrated to improve exclusivity in multi-level caches by Busari and Williamson [9], Wong *et al.* [20] and in our previous work [12, 7]. To design good cache hierarchies, collaborative groups, or overlay networks [16] we need to understand the dependencies between various policies; the pairs that complement each other well.

If two policies are known to disagree most of the time, then their agreement on a workload either points to the usage of cache sizes close to the infinite or zero edges as in Figure 2 or to the randomness in the workload. The information gathered by our correlation analysis may be used to understand the nature of various workloads and to reconstruct similar synthetic workloads of adjustable length for trace-driven simulations. These workloads can also be made to change characteristics over time from random, to sequential, to Zipf distributed and so on to favor different policies. Such a workload is very useful for benchmarking the adaptivity of new policies.

## 6 Related Work

To the best of our knowledge nobody has yet quantified the amount of dependency between cache replacement policies. Dependency of experts and value of information from dependent experts has been analyzed in other contexts [11, 3].

The methods for aggregation of information from multiple experts is classified into two; mathematical and behavioral [3]. In the latter approach experts are made to interact to come up with an agreement. Mathematical methods are usually easier to implement. The simplest mathematical aggregation method is the linear opinion pool, a weighted linear combination of experts’ estimations. If experts are considered equal they are initially assigned equal weights, otherwise the experts that are believed to be “better” in the sense of being more precise can be assigned higher weights. In our previous work [12, 7] we have used machine learning techniques to update the weights of experts in the opinion pool.

Voting, cascading, and boosting are other techniques for combining experts’ opinions. Voting constructs a simple linear opinion pool. Boosting tries to construct an expert with small error rates from experts which are just slightly better than random. Learners are consulted (or trained) serially[3]. The third learner is consulted only when the first two learners disagree. In cascading, the simple expert handles majority of the patterns and those that are rejected by it are handled by the complex classifier. A very simplistic version of cascading has been tried in context of caching as “virtual cache management” by Arlitt [8].

## 7 Conclusions

We have used statistical correlation for quantifying the amount of dependency between a large set of cache replacement policies, or “cache experts”. Correlations have high impact on the value of information from multiple sources. We

have found high correlations between many policies.

We are currently using set-based heuristics to measure the differences between policies. In this experiment we look into the commonalities between the objects cache by each policy, where they agree and disagree. We intend to compare the results of the set-based heuristics with the results of the correlation formula. We would like to complete our analysis for a broader set of workloads and then formulize and generalize the results as much as possible.

## Acknowledgements

This research is sponsored by Hewlett-Packard Laboratories, Storage Technologies Department.

## References

- [1] Digital Equipment Corporation, Digital's Web Proxy Traces. <ftp://ftp.digital.com/pub/DEC/traces>, August–September 1996.
- [2] National Lab of Applied Network Research (NLNR). Sanitized access log. Available at <ftp://ircache.nlanr.net/Traces/>, July 1997.
- [3] F. Alimoglu and E. Alpaydin. Combining multiple representations for pen-based handwritten digit recognition. *ELEKTRIK: Turkish Journal of Electrical Engineering and Computer Sciences*, 9(1):1–12, 2001.
- [4] A. Amer and D. D. E. Long. Adverse filtering effects and the resilience of aggregating caches. In *Proceedings of the Workshop on Caching, Coherence and Consistency (WC3 '01)*, Sorrento, Italy, June 2001. ACM.
- [5] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch. Hippodrome: running circles around storage administration. In *Proceedings of the 2002 Conference on File and Storage Technologies (FAST)*, Monterey, CA, Jan. 2002.
- [6] I. Ari. Multicache simulation environment version 1.0 reference guide. Technical Report UCSC-CRL-03-02, University of California Santa Cruz, Santa Cruz, CA, 2003.
- [7] I. Ari, A. Amer, R. Gramacy, E. L. Miller, S. A. Brandt, and D. D. E. Long. ACME: adaptive caching using multiple experts. In *Proceedings in Informatics*, volume 14, pages 143–158. Carleton Scientific, 2002.
- [8] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin. Evaluating content management techniques for web proxy caches. In *Proceedings of the 2nd Workshop on Internet Server Performance (WISP '99)*, Atlanta, Georgia, May 1999.
- [9] M. Busari and C. Williamson. Simulation evaluation of a heterogeneous web proxy caching hierarchy. In *Proceedings of the 9th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '01)*, pages 379–388, Cincinnati, OH, Aug. 2001. IEEE.
- [10] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS '97)*, pages 193–206, Dec. 1997.
- [11] R. T. Clemen and R. L. Winkler. Limits for the precision and value of information from dependent sources. *Operations Research*, 33(2):427–442, 1985.
- [12] R. Gramacy, M. Warmuth, S. A. Brandt, and I. Ari. Adaptive caching by refetching. In *Proceedings of the 2002 Neural Information Processing Systems (NIPS)*, 2002. Accepted for publication.
- [13] S. Jin and A. Bestavros. GreedyDual\* web caching algorithm: Exploiting the two sources of temporal locality in web request streams. In *Proceedings of the 5th International Web Caching and Content Delivery Workshop*, Lisbon, Portugal, May 2000.
- [14] T. Johnson and D. Shasha. 2Q: A low overhead high performance buffer management replacement algorithm. In *Proceedings of the 20th Conference on Very Large Databases (VLDB)*, pages 439–450, Santiago, Chile, 1994.
- [15] J. M. Kim, J. Choi, J. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim. A low-overhead high-performance unified buffer management scheme that exploits sequential and looping references. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 119–134, San Diego, CA, Oct. 2000.

- [16] J. Kubiawicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An architecture for global-scale persistent storage. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Cambridge, MA, Nov. 2000. ACM.
- [17] E. J. O’Neil, P. E. O’Neil, and G. Weikum. The LRU-K page replacement algorithm for database disk buffering. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 297–306, 1993.
- [18] G. Weikum, A. C. König, A. Kraiss, and M. Sinnwel. Towards self-tuning memory management for data server. *Data Engineering Bulletin*, 22(2):3–11, 1999.
- [19] C. Williamson. On filter effects in web caching hierarchies. *ACM Transactions on Internet Technology*, 2(1):47–77, 2002.
- [20] T. M. Wong and J. Wilkes. My cache or yours? making storage more exclusive. In *Proceedings of the 2002 USENIX Annual Technical Conference*, pages 161–175, Monterey, CA, June 2002. USENIX.
- [21] Y. Zhou, J. F. Philbin, and K. Li. The multi-queue replacement algorithm for second level buffer caches. In *Proceedings of the 2001 USENIX Annual Technical Conference*, pages 91–104, Boston, Massachusetts, June 2001. USENIX.