CMPT 815 Assignment 2: Experimentation via Simulation

Wenguang Wang

March 21, 2000

1 Introduction

Distributed systems consisting of workstations and shared file servers typically have main memory disk block caches at both the workstations and the file servers. If a request is satisfied at the client cache, no request will be sent to the server side. Therefore, the existence of client cache may change the characteristics of the reference string and impact the performance of the cache on the server side. Because the file I/O request streams have temporal locality, LRU is often used to manage the cache. However, whether or not LRU is still suitable in the server cache when the file I/O request stream is modified by the client cache is a question. This report investigates this problem through trace-driven simulation.

2 Cache Replacement Algorithms

There are a number of cache replacement algorithms which could be used at a file server cache. This study investigates the performance of three of them: Least Recently Used (LRU), Least Frequency Used (LFU), and RANDom replacement (RAND).

LRU replaces the block in the cache which has not been used for the longest period of time. The basic premise behind the LRU algorithm is that blocks which have been referenced in the recent past will likely be referenced again in the near future. This property, know as *temporal locality*, is a characteristic of disk block access pattern of applications.

LFU requires that a reference count be maintained for each block in the cache. When a replacement is necessary, LFU chooses the block which has the lowest reference count. In my implementation, a tie is resolved by the LRU algorithm, i.e., by replacing the least recently used blocks. The motivation for LFU and other frequency based algorithms is that the reference count can be used as an estimate of the probability of a block being referenced. Blocks with high counts are more likely to be referenced than blocks with low counts. The problem of LFU is that certain blocks may build up high reference counts and never be replaced.

RAND chooses among all blocks in the cache with equal probability. For RAND to work well, all disk blocks should be accessed with equal probability. This is not likely to be the case. RAND provides a kind of lower bound on performance, that is, there is no reason to use any policy which performs worse than RAND.

3 Simulation Methodology

3.1 Trace Characteristic

The trace is recorded at the client side. It includes three kinds of file I/O requests: Read, Write, and Erase. The device id, inode number of the file referenced, and the logical block number are recorded for each request. These (device id, inode #, logical block #) triples can be used to represent a unique disk block within the cache simulator. The Erase events record the device and inode number of the file being removed. All blocks belonging to the erased file will be deleted from the cache when the Erase event is processed. Table 1 shows the characteristics of the trace. There are about 1340 distinct blocks in this trace. Therefore, the length of the trace is very short compared to the number of distinct blocks.

Duration	Number of requests			
(Hours)	Total	Reads	Writes	Erases
1.7	17185	11304	5498	383

Table 1: Trace Characteristics

3.2 Simulation Model

A trace-driven simulator was developed to simulate the operation of both the client cache and the server cache. On the client cache, LRU is always used during all experiments. LRU, LFU, and RAND are simulated on the server side. Reading each event in turn, the simulator first simulates the actions that it would cause at the client cache, followed by those that would be caused, if any, at the server cache.

Because the write-through policy is applied, all write requests have to be propagated to the server and the disk on the server. Therefore, the client cache cannot reduce the number of writes to the server. Therefore, the read miss ratio is the only metric considered in this report. The read miss ratio is measured on the server side, that is, the number of server cache read miss in the percent of the number of read requests sent to the server cache.

3.3 Selection of Warmup

At the beginning of the simulation, the cache is empty, and every new page will cause a cache miss. This initial phase is called *warmup*. The read miss should not be measured in the warmup period. If the warmup period is too short, the accuracy of the result is impacted. If the warmup is too long, the time for measurement is too short and the confidence level is impacted. When the cache size is small, the cache can be fill up quickly, and the warmup period is short. A longer warmup period is needed for a bigger cache. In order to compare the miss ratio on different server cache sizes, the same warmup period should be used. Therefore, the warmup is only computed for the largest client cache size (800) and server cache size (1000) which are considered in this report.

The batch means method [Jain, 1991] is applied to determine the warmup period. First, the read miss ratio for every 100 requests sent to server is computed. The result of the LRU algorithm is shown in Figure 1. The same figure for LFU and RAND is not shown in this report because they are nearly the same as Figure 1. Assume the total number of miss ratios is N. These N miss ratios are divided into m batches of size n each, where $m = \lfloor \frac{N}{n} \rfloor$. Let x_{ij} denote the jth miss ratio in the ith batch.

1. For each batch, the batch mean is computed:

$$\bar{x}_i = \frac{1}{n} \sum_{j=1}^{n} x_{ij}, \quad i = 1, 2, \ldots m$$

2. The overall mean is then computed:

$$\bar{\bar{x}} = \frac{1}{m} \sum_{i=1}^{m} \bar{x}_i$$

3. The variance of the batch means are computed:

$$Var(\bar{x}) = \frac{1}{m-1} \sum_{i=1}^{m} (\bar{x}_i - \bar{\bar{x}})^2$$

The above steps is taken for $n = 1, 2, 3, \ldots$, and the result is plotted in Figure 2. After the last peak at 2400, the system enters steady state. Because the 2400th event sent to server is the 6794th event in the original trace, the warmup for all experiments is 6794, which is 39.5% of the total length of the trace. Therefore, the trace is not very long compared to the warmup period.

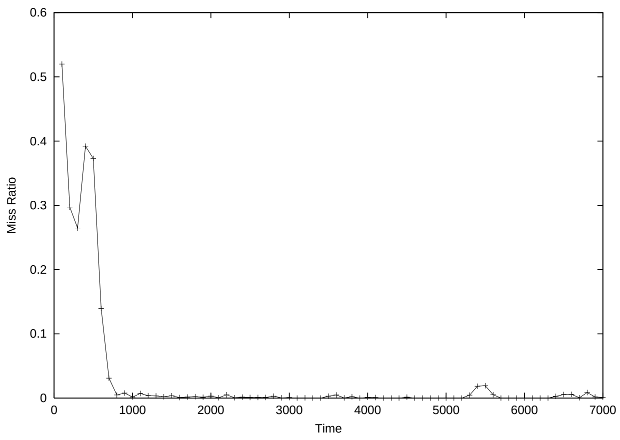

Figure 1: Server Miss Ratio of LRU algorithm

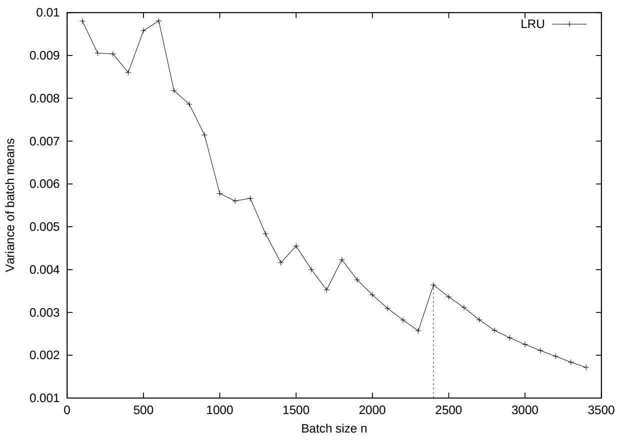

Figure 2: Warmup determination by batch means

3.4 Random Replacement Algorithm

In the random replacement algorithm, different random seeds should not affect the read miss ratio much. Thirty different random seeds are tested for the RAND algorithm. The results show that the maximum relative error to the average value is only 6.7%, which is acceptable.

3.5 Structure and Design of Simulator

The simulator was developed in Visual C++. It contains about 1,800 lines C++ source code. The program is very well organized by the Object-Oriented Programming (OOP) principle. The application of the Standard Template Library (STL) greatly reduces the complexity and length of the program.

3.5.1 Classes

Important classes of the simulator are discussed in this subsection. Each class implements a simple concept of the simulator. Most functions are less than 10 lines C++ code, which makes it very easy to understand and maintain.

- *Simulation* is the main control class, which reads the config file, creates all experiments, run experiments, and outputs results.
- *Experiment* contains parameters for one experiment (i.e. one server cache size and one client cache size). This class calls the client cache algorithm, and passes all missed events to server cache.
- *Cache* is an abstract class which represents the common properties of the three cache algorithms (LRU, LFU, and RAND). All logics of the cache algorithm except how to evict a page and how to update a page when it is referenced are implemented in this class.
- *CacheLRU*, *CacheLFU*, and *CacheRand* are subclasses of the class *Cache*. They implement the three cache algorithms by defining the way to evict a page and update a page.
- *CacheLRUStackDepth* and *CacheLFUStackDepth* are subclasses of the class *CacheLRU* and *CacheLFU*, respectively. They do not change the replacement algorithms, but record the stack depth of each read hit. The stack depth data are generated by these two classes.
- *Hash* implements a hash table for the pages in the cache. The hash table accelerates the speed of searching a page in the cache.
- *RandGen* generates pseudo random numbers which are used in class *CacheRand*.
- *Config* contains common routines to manipulate the config file.

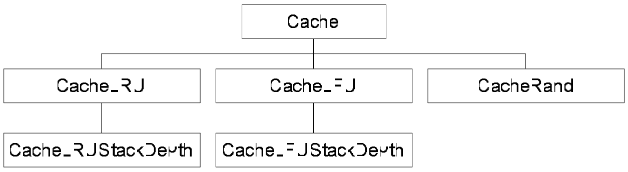

Figure 3: Class Hierarchical Structure

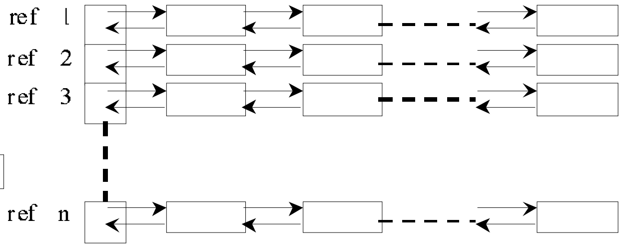

Figure 4: Data Structure

The hierarchical structure of classes of the cache algorithm is shown in Figure 3.

The data structure used for class Cache is a vector of pointers to a doubly linked list, which can be illustrated in Figure 4. All pages with the same reference count are on the same list. A page with reference count i is always on the *ith* list. Because all pages in LRU and RAND always have the reference count 1, only one list is used in LRU and RAND. A hash table (not shown in Figure 4), which is also a vector of pointers to a doubly linked list, is used to locate the page in the cache. Each page is hashed to an index in this vector. The list is used to store pages that have the same hash value.

3.5.2 Config file

A config file is used as the input of the simulator. The size of client cache and server cache, the warmup parameter, and the random seed are all included in the config file. The full factorial method is used for all client cache size, server cache size, and server cache algorithms. Part of the config file is shown in Figure 5. There are 11 client cache sizes, 10 server cache sizes, and 3 server cache replacement algorithms. Therefore, the simulator will run $11 \times 10 \times 3 = 330$ experiments on the trace. The pseudo code of the main loop in class *Simulation* is shown in Figure 6. The trace is only read once for all experiments. Only 2.5 minutes are needed for finishing all these 330 experiments on a computer with PII 333MHz CPU and 64MB RAM.

```
ClientBuffer    0 50 100 150 200 300 400 500 600 700 800 -1
ServerBuffer    100 200 300 400 500 600 700 800 900 1000 -1
ServerAlgorithm LRU LFU RAND EOD
```

Figure 5: Config File

```
while (trace not finish)
  for (all experiments)
    process this request
```

Figure 6: The Pseudo Code for the Main Loop

4 Simulation Results

This section presents and discusses the simulation results obtained. Figure 7 shows the miss ratio of the single level cache. The client size is set to 0, and the server size ranges from 100 to 1000. This figure shows that when the server size is small, LFU yields the most miss ratio, and LRU is slightly better than RAND. It is clear that the LFU policy is inappropriate. As the server cache size increases, the miss ratio of all algorithms decreases, and the difference among all algorithms becomes smaller. When the server cache size is less than 600, LFU is much worse than LRU and RAND, which clearly shows that the LFU algorithm is inappropriate. The strange thing is that when the server cache size is more than 600, the miss ratio of LFU becomes much smaller than LRU and RAND, and the LRU becomes worse than RAND. The reason may be that the trace is too small.

To investigate the locality of the trace, the LRU stack depth and the LFU stack depth data are computed. The stack depth data are generated by running the simulation with the specified client cache size and an infinite server cache. Each stack depth indicates how many "read hits" occur at each LRU/LFU stack position. The LFU stack position is computed in the same way as described in [Willick et al., 1990]. Because so many "read hits" concentrate on very small stack depths, the stack depth histogram spans many scales and is not clear. Therefore, only the Cumulative Density Function (CDF) is shown in Figure 8. For LRU stack depth, 85% hits occurs on stack depth less than 57, which indicates much locality in the trace. For the LFU stack depth, 85% hits occurs on stack depth less than 472. Therefore, the LRU is more suitable than LFU when the client size is 0.

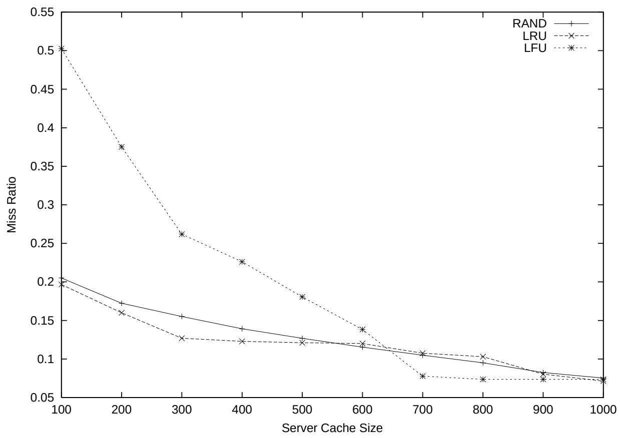

Figure 7: Miss Ratio (Client Cache=0)

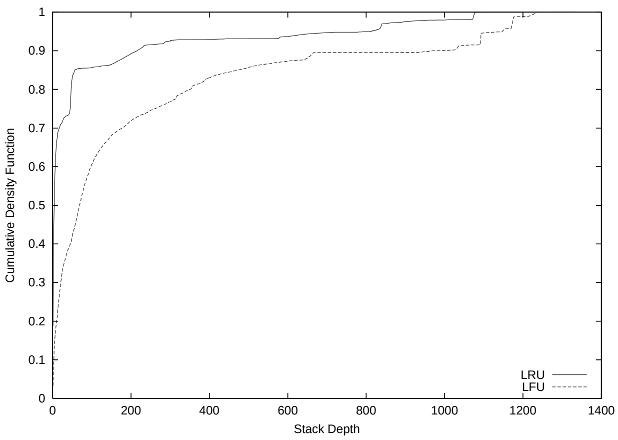

Figure 8: Stack Depth (Client Cache=0)

Figure 9 shows the server read miss ratio when the client cache size is 100. From this figure, we cannot say which one is better. Figure 10 shows the stack depth of LRU and LFU for client size 100. Compared to the stack depth of client size 0, the average stack depths of the two algorithms become bigger. This means that the miss ratio will increase for the same server size. However, the LRU stack depth CDF changes much bigger than LFU. In this figure, for the LFU stack depth, 70% hits occurs on stack depth less than 467. For the LRU stack depth, 70% hits occurs on stack depth less than 836. Therefore, much of the temporal locality in the trace is removed by the client cache. However, the existence of client cache does not affect the stack depth of LFU much. Therefore, the LFU should be more suitable than LRU when a client cache exists. This is not the result shown in Figure 9. The reason may be that the trace is too short.

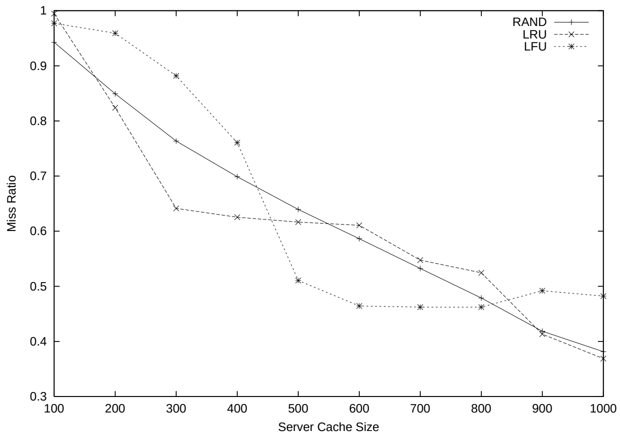

Figure 9: Miss Ratio (Client Cache=100)

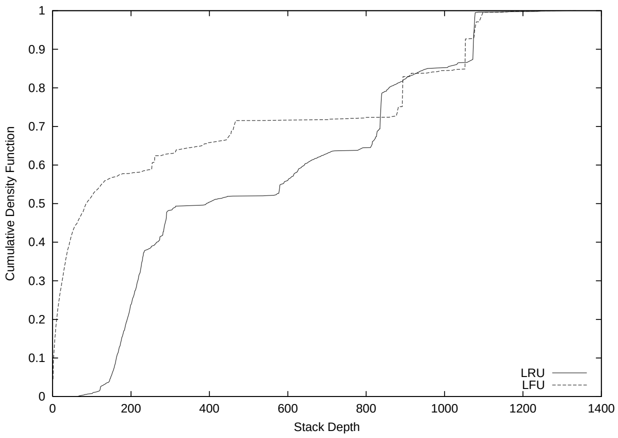

Figure 10: Stack Depth (Client Cache=100)

Figure 11–18 show the miss ratio on other client cache sizes. As the client cache size increases, the stack depth of the LRU algorithm changes more than that of the LFU algorithm (see Figure 12, 14, 16, and 18). This indicates that the LFU should get less miss ratio than LRU. However, the miss ratio does not show the same result (see Figure 11, 13, 15, and 17). When the client buffer size is very big, the RAND algorithm is superior to LRU and LFU, which cannot be explained. The reason may be that the trace is too short.

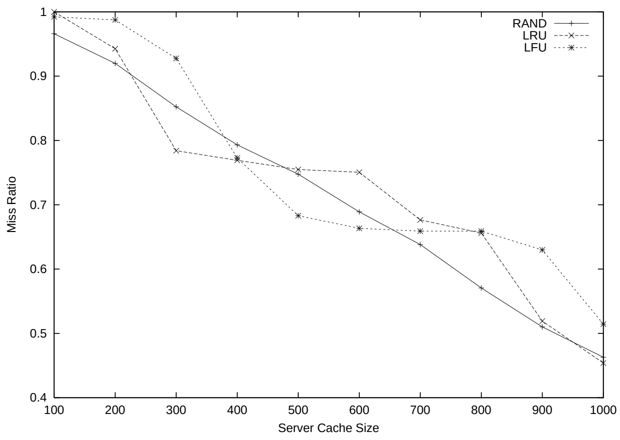

Figure 11: Miss Ratio (Client Cache=200)

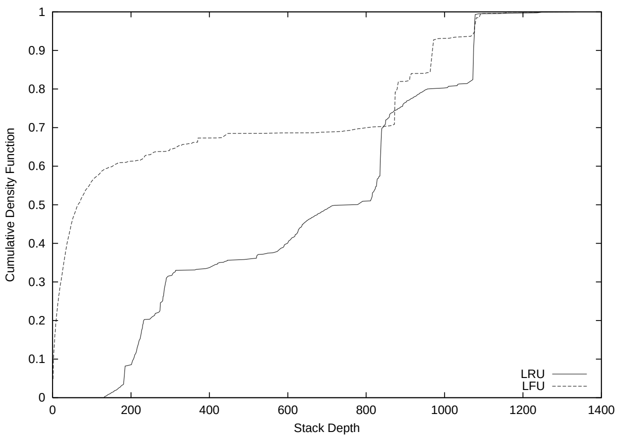

Figure 12: Stack Depth (Client Cache=200)

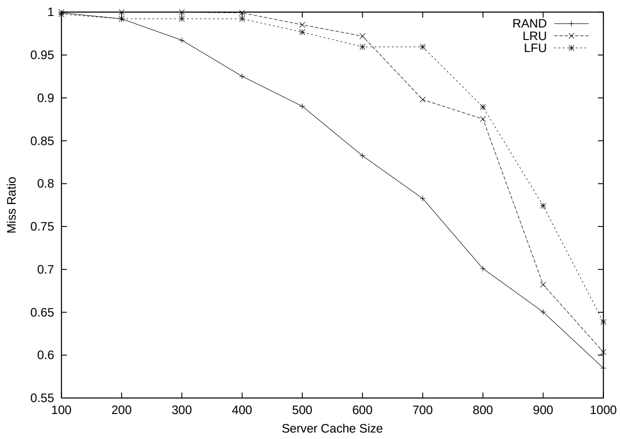

Figure 13: Miss Ratio (Client Cache=400)

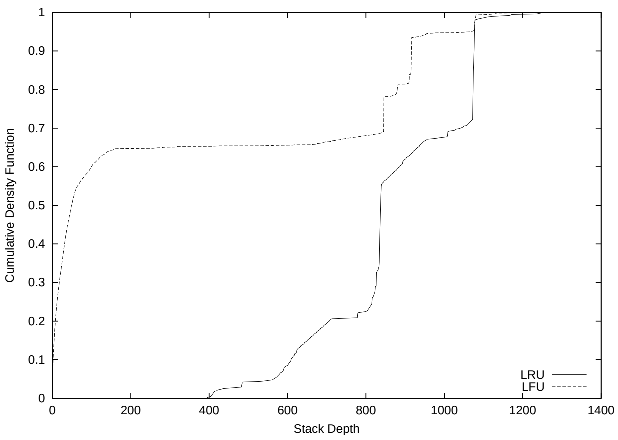

Figure 14: Stack Depth (Client Cache=400)

Figure 19, 20, 21, 22, 23, and 24 shows the change of miss ratio when the client cache size changes. A same trend is that when the client cache size increases, the server cache miss ratio also increases, although at a few points, the server cache miss ratio decreases a little for LFU algorithm.

5 Conclusion

The goal of this report was to investigate the suitability of various management strategies in the context of a network file server. Three algorithms were considered: LRU, LFU, and RAND.

Because the trace for study is too small, not much useful results can be obtained from the server read miss ratio. However, the stack depth data of LRU and LFU suggests that the LRU is not a good replacement policy for a file server cache, and the LFU should outperform LRU.

References

[Froese and Bunt, 1996] Froese, K. W. and Bunt, R. B. (1996). The effect of client caching on file server workloads. In *Proc. 29th Hawaii International Conference of System Sciences*, Kihei.

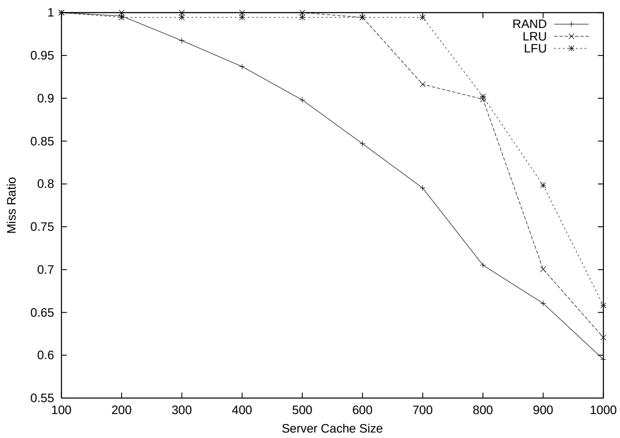

Figure 15: Miss Ratio (Client Cache=600)

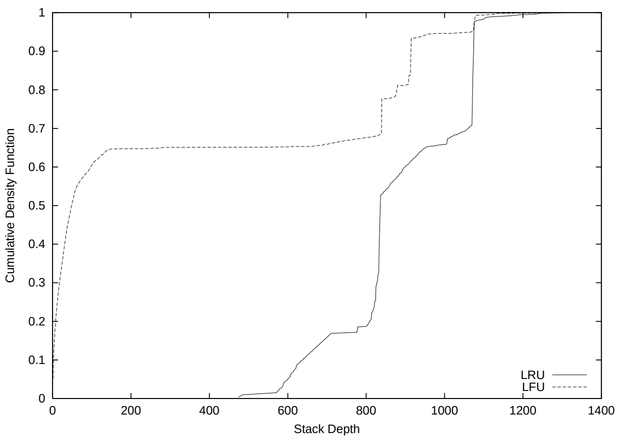

Figure 16: Stack Depth (Client Cache=600)

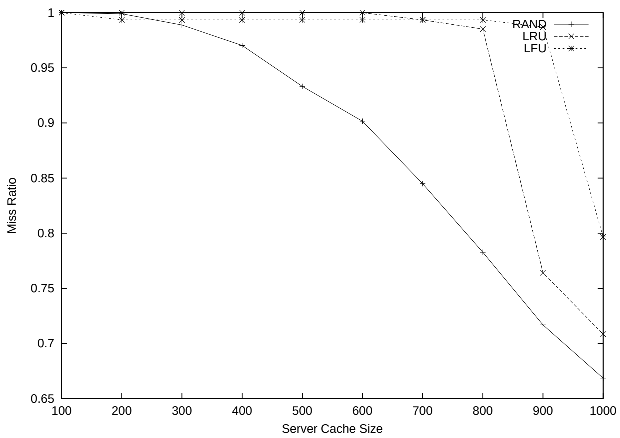

Figure 17: Miss Ratio (Client Cache=800)

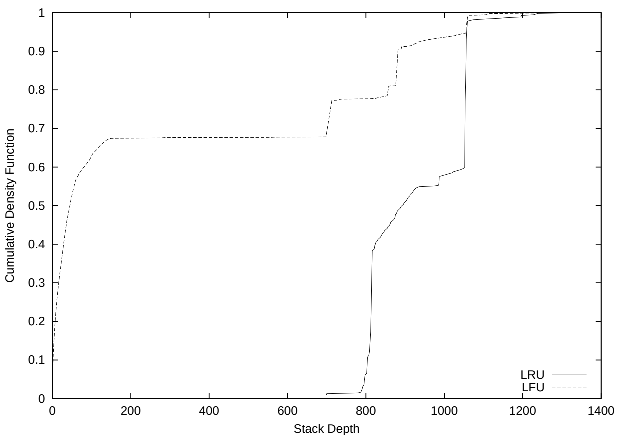

Figure 18: Stack Depth (Client Cache=800)

[Jain, 1991] Jain, R. (1991). *The art of computer systems performance analysis*. John Wiley, New York.

[Willick et al., 1990] Willick, D. L., Eager, D. L., and Bunt, R. B. (1990). Disk cache replacement policies for network fileservers. In *Proc. 10th International Conference on Distributed Computing Systems*, pages 212–219, Paris, France.

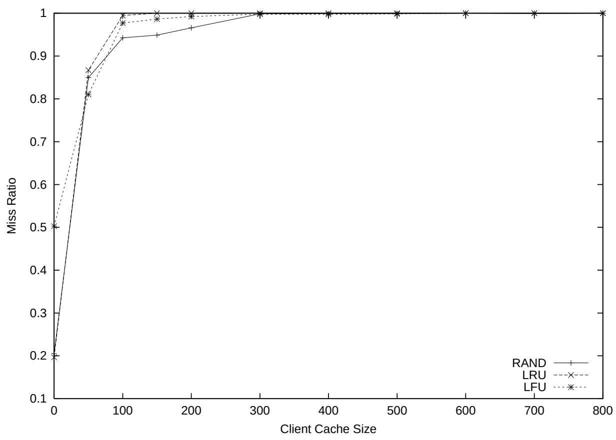

Figure 19: Miss Ratio (Server Cache=100)

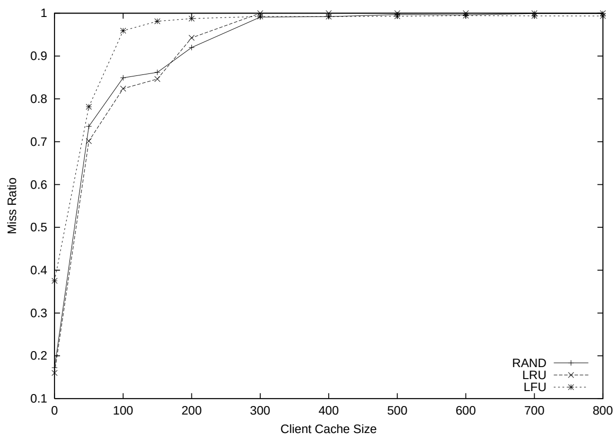

Figure 20: Miss Ratio (Server Cache=200)

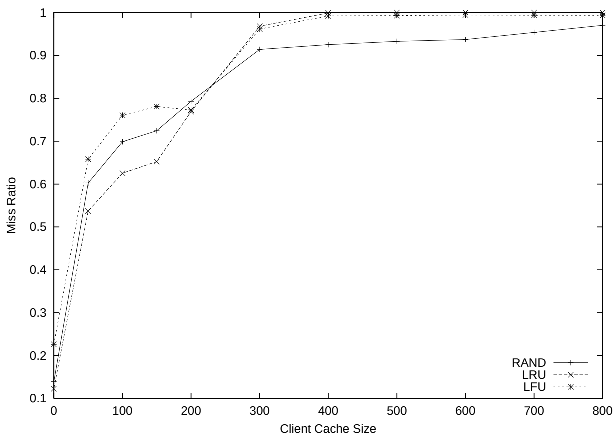

Figure 21: Miss Ratio (Server Cache=400)

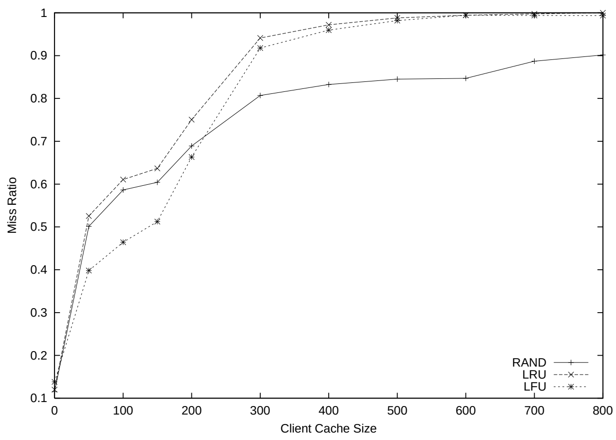

Figure 22: Miss Ratio (Server Cache=600)

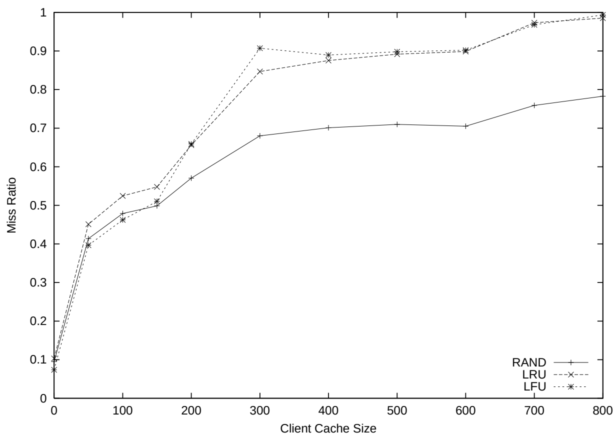

Figure 23: Miss Ratio (Server Cache=800)

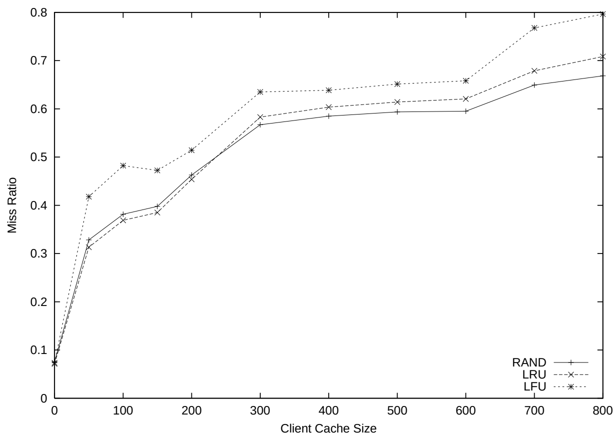

Figure 24: Miss Ratio (Server Cache=1000)