Top40 cache algorithm compared to LRU and LFU

SNE MSc Research Project

A. van Hoof

2 February 2009

Abstract

Describing the workings of a specially written caching algorithm unofficially called Top40 and comparing it to two more common algorithms LRU (Least Recently Used) and LFU (Least Frequently Used) this report tries to show the effectiveness of Top40 algorithm in a media streaming environment. Simulations where done using real data. These simulations show the effectiveness of the Top40 algorithm and its main advantage in the current environment with big media-file sizes: almost no inserts.

Preface

This report was created as part of a four week research project done as part of the System- and Network-engineering master at the University of Amsterdam. The research was mostly done on the NPO-ICT (Nederlandse Publieke Omroep ICT) location at Mediapark Hilversum. I would like to thank the department NPO-ICT for providing resources and time for this research project. A special thanks to Dick Snippe whose guidance and clear explanations gave me the insights needed to do this project.

1 Introduction

To provide streaming media via the internet to the public (uitzeninggemist.nl and others) mediastream-servers are used by NPO-ICT. These mediastream-servers have access to different kinds, with regards to speed, of storage. A set of Perl-scripts has been written by D. Snippe implementing a special for this situation optimized, cache-algorithm. The current production environment uses the cache-algorithm without performance problems. The scripts use text-file input data which is generated every minute and caching is done based on this data. Because the way the created cache-algorithm works resembles the Dutch "Top 40" radio hits chart, the created algorithm is unofficially called: *Top40-caching*.

Research Question

The Top40 cache algorithm has never been compared to other more common algorithms like LRU[4] (Least Recently Used) or LFU[7] (Least Frequently Used). This let to the following research question used for the research in this report:

> *How does the by NPO-ICT created and used cache algorithm compares to other cache algorithms in the same environment.*

The Question has the "in the same environment" part because there are some special demands (or lack of) compared to environments in which the common cache algorithms are used.

2 Storage Cache Setup

As described in Section 1 the mediastream-servers at NPO-ICT use a special setup of their storage to allow caching of their media-files. This section describes the setup. In figure 1 a

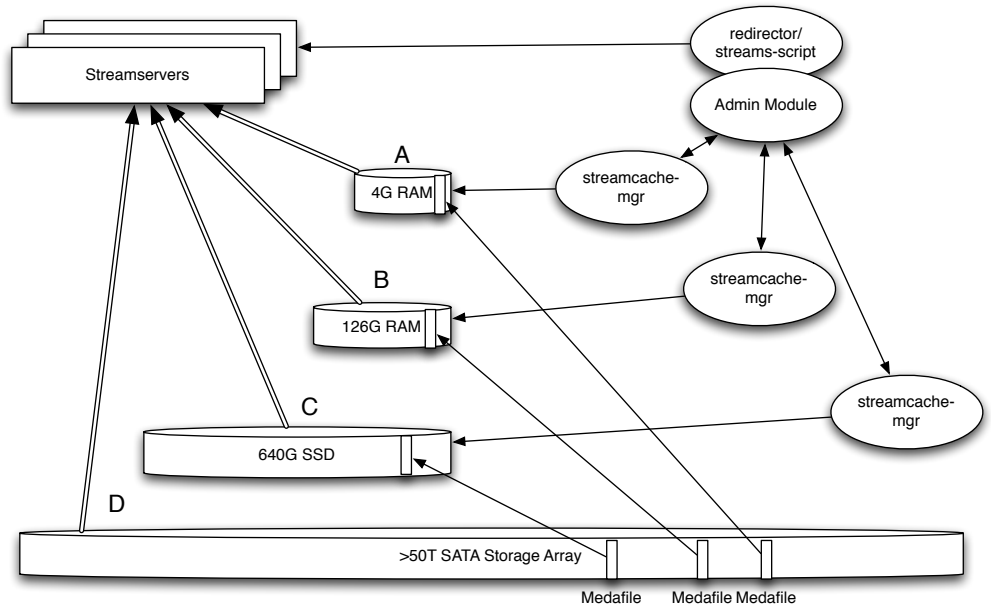


Figure 1: Overview of Storage and Cache available to the mediastream-servers

schematic representation of the setup is given. Cache **A** and **B** are RAM-disks local to the mediastream-servers and therefore the fastest. The same 4B cache **A** is available (mirrored) on every stream-server, the 126GB RAM cache (**B**) is mirrored per two mediastream-servers and therefore less preferred over **A**. The other cache (**C**) and the main storage **D** are shared among all mediastream-servers. All the media-files are initially stored on the SATA Storage Array (**D**). Every cache (**A, B and C**) is filled with media-files from this storage. There is no data copied between the caches **A, B and C**. A process called streamcache-mgr is running for every cache instance (**A, B and C**) deciding how to fill the cache managing based on data received on a per minute basis from the admin module. Because all caches are managed by the same algorithm , cache **A** is a subset of cache **B** which is a subset of cache **C**. The way the the mediastream-servers, the caches and the streamcache-mgr work together create some cache characteristics which are different from more common cache used in mostly hard-disks or CPUs.

Cache	**Read/Write**	**Required Speed**	**Size**	**Cache Miss**	**Cache Inserts**
Common	Read/Write	micro-seconds	MB	Performance impact	Small Size/impact
Stream	Read-Only	seconds	GB	Compensated by storage-hardware	Big size/impact

Table 1: Characteristics of a Common cache compared to the stream cache

3 Simulation Environment

Because it is not possible nor advisable to do tests in a production/live environment, a simulated environment was created. The Top40 cache algorithm is written in Perl[5] and this programming language was used to implement the cache algorithms for comparison. The simulation environment consists of shell and AWK scripts glueing it all together. To avoid access to the real storage environment containing all the files, a Perl Tie BTREE database was used containing filename and file sizes (StorageDB in figure 2) and a same type of database was used the act as a cache (CacheDB in figure 2). The real requests where archived and 11 days of historical data was used to do the simulations. The first day was removed from the results to allow the caches to be filled and and avoid the influence of the initial fluctuations. A modified version of the streamcache-mgr script was used to generate the same output as the LRU an LFU programs and to avoid actual data access and cache updates on the production systems. The plots of the input data characteristics and the results were done using gnuplot[1].

4 LRU, LFU and Top40 algorithm implementation

Both the LRU and LFU algorithm are implemented in a Perl program according to the flow diagram presented in Figure 2. The input data (STATS) is a text file with to columns: requests

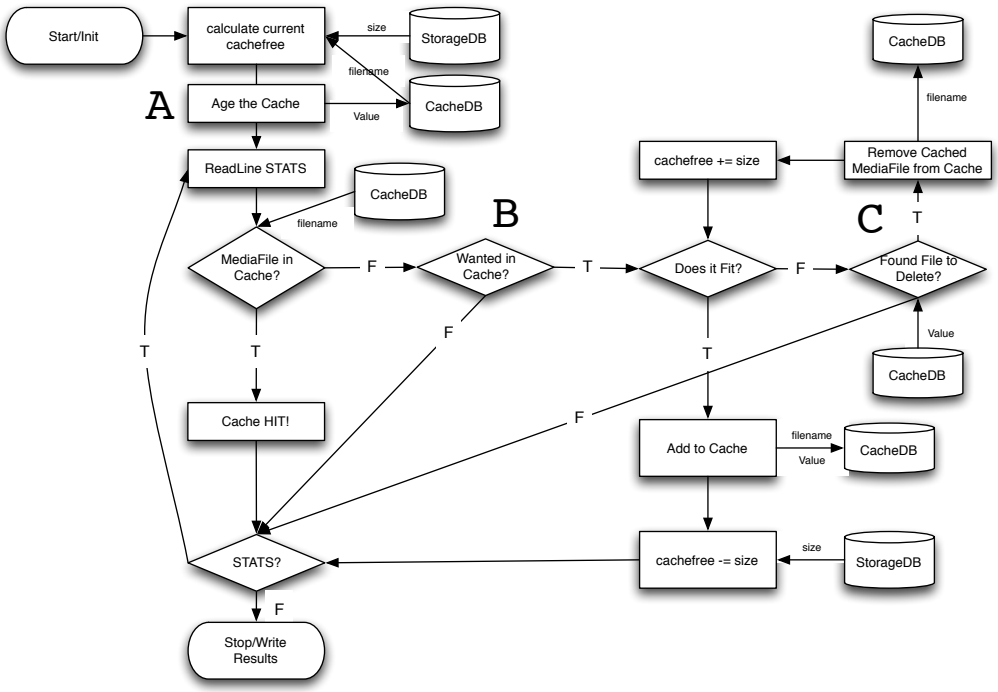


Figure 2: Flow diagram of the LRU and LFU programs

of the pas minute **R** and the filename of the requested file **FILENAME**, ordered by **R**.

The used LRU cache algorithm

With LRU, every FILENAME in cache (CacheDB in Figure 2) has a time-stamp assigned when inserted or when found in cache. This avoids the need for cache aging (**A**). The implemented

LRU algorithm does not make a decision at **B** (FILENAME is always wanted in cache). It selects candidates for removal at **C** finding the oldest files in the cache using the time-stamp stored in the cache with the FILENAME.
The LRU algoritm in pseudo-code:

```
if FILENAME in cache
  value(FILENAME) = timestamp
  EXIT
while cachefree < filesize(FILENAME)
    find and remove candidate in cache with oldest timestamp
    cachefree = cachefree + filesize(candidate)
insert value(FILENAME) = timestamp
```

The used LFU cache algorithm

For LFU when a file is inserted into the cache (CacheDB in Figure 2) the number of requests **R** is associated with the FILENAME. Unlike pure LFU, the implemented LFU algorithm ages the cache (in Figure 2 at **A**) to avoid cache pollution. The cache is aged according to:

$$w(t) = \frac{w(t-1)}{2} \tag{1}$$

When a file is requested (R times) and already in cache its $w(t)$ is recalculated:

$$w(t) = R + \frac{w(t-1)}{2} \tag{2}$$

The LFU algorithm in pseudo-code:

```
Devide every value(FILENAME) in cache by 2
if FILENAME in cache
  value(FILENAME) = R + value(FILENAME)
  EXIT
while cachefree < filesize(FILENAME)
    find and remove candidate in cache with smallest value
    cachefree = cachefree + filesize(candidate)
insert value(FILENAME) = R
```

The Top40 cache algorithm

To manage each cache the streamcache-mgr (figure 1) scripts use a special algorithm to determine what to put in the cache and what can be removed. The algorithm is specially tailored to take advantage of differences with the common cache setups (Table 1). Before a media-file is inserted in the cache some extra choices are made. A media-file requested only once will not be inserted into the cache. Using the input data to determine the weight, the weight value is used in a replacement algorithm, based on Least Frequently Used (LFU). The weight of a file either in or outside the cache is calculated as follows

$$w(t) = w(t_0) * 2^{-\lambda*(t-t_0)} \tag{3}$$

where

$$\lambda = \frac{-log0.5}{decaytime} \tag{4}$$

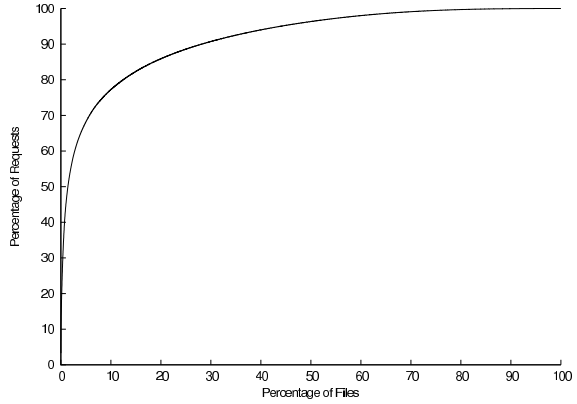
The value of decaytime is a tunable parameter known as time to half value. In the current production environment it is 120 for the 4GB and 126GB caches and 604800 for the 640 GB cache. When a file is requested (R times) its weight is recalculated:

$$w(t) = R + w(t_0) * 2^{-\lambda*(t-t_0)} \tag{5}$$

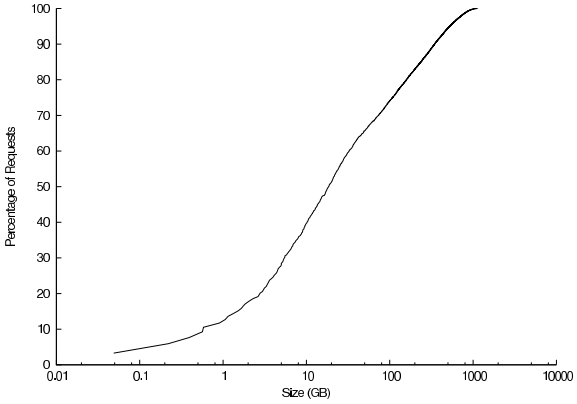
The list containing the weights is ordered by $w(t)$ with the biggest $w(t)$ on top: a "chart". Candidates for removal are files with the smallest $w(t)$ but they are only removed when the file to be inserted has a $w(t)$ bigger than a threshold value and the candidates for removal are below the threshold level. This way media-files becoming very popular (from nothing to number one) will enter the cache and replace files which are on their way down and are in the lower regions of the "chart". This kind of hysteresis also avoids cache inserts of media file which are requested only a few times. This part of the algorithm is the reason for its unofficial name "Top40-caching" because radio music hit charts (Dutch Top-40) act the same way.

5 Characteristics of the Input Data

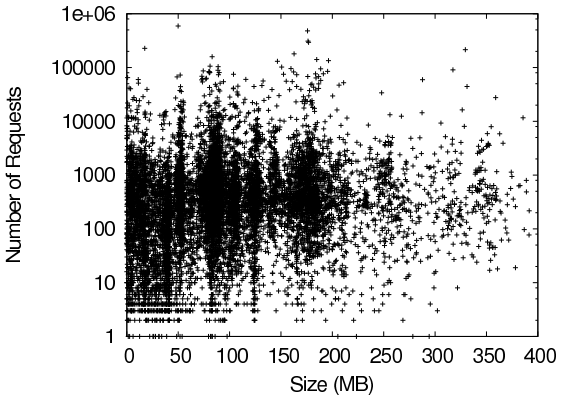
The characteristics of the input data are important for the success of the Top40 algorithm. The input data consists of a media-file name and the number of requests for this file in the past minute ordered from top to bottom. Figure 3(a) shows: a small number of the files are requested many times, while many files are requested only a few times. Because the size of



(a) % of request and % of files



(b) % of request and cumulative file size (log)



(c) Number of requests and file size

Figure 3: Characteristics of the input data

each media-file is available the graphs can be created showing the the cumulative size of the requested files (figure 3(b)), and the size of the files requested (figure 3(c)). This behavior was also found by Abrams e.a. for WWW data [2]. Using the input data a overview of the requests on a per minute basis can be created (figure 4).

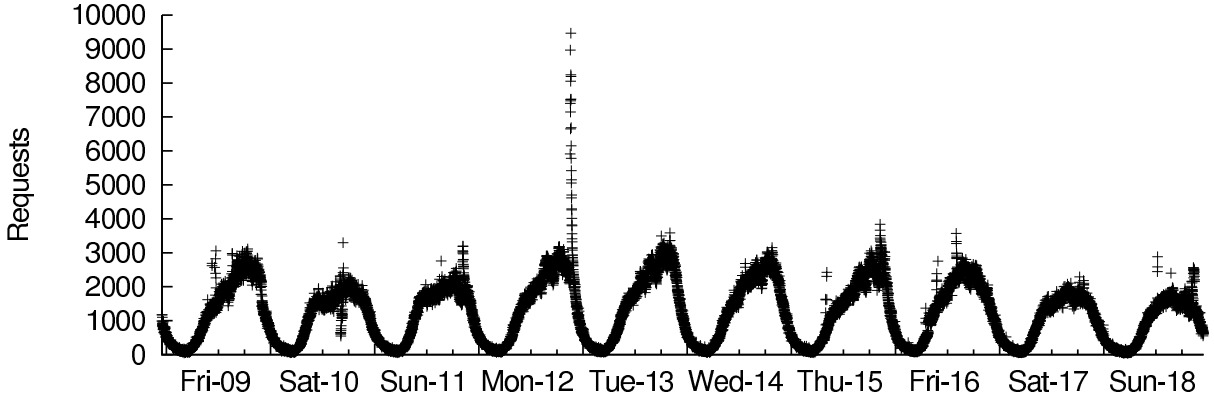


Figure 4: Number of requests within 10 days of January 2009 on a per minute basis

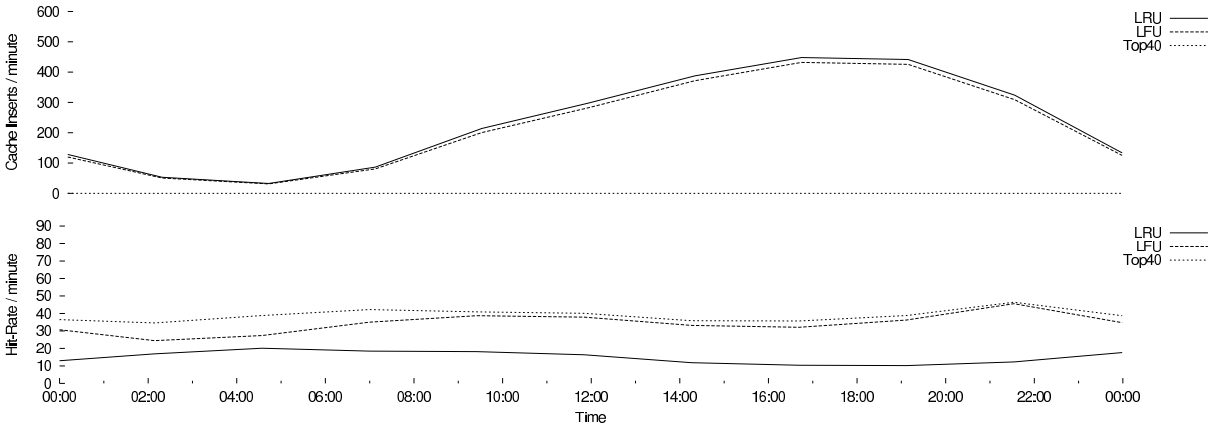
6 Results

All tests used the same input data, so all hit-rates and inserts are based on the same 10 days (11 days minus 1 day for cache settlement). The input data used to create the results consisted of 14.400 files containing a total of 18.061.817 requests for 11.150 different media-files. The Hit-rate

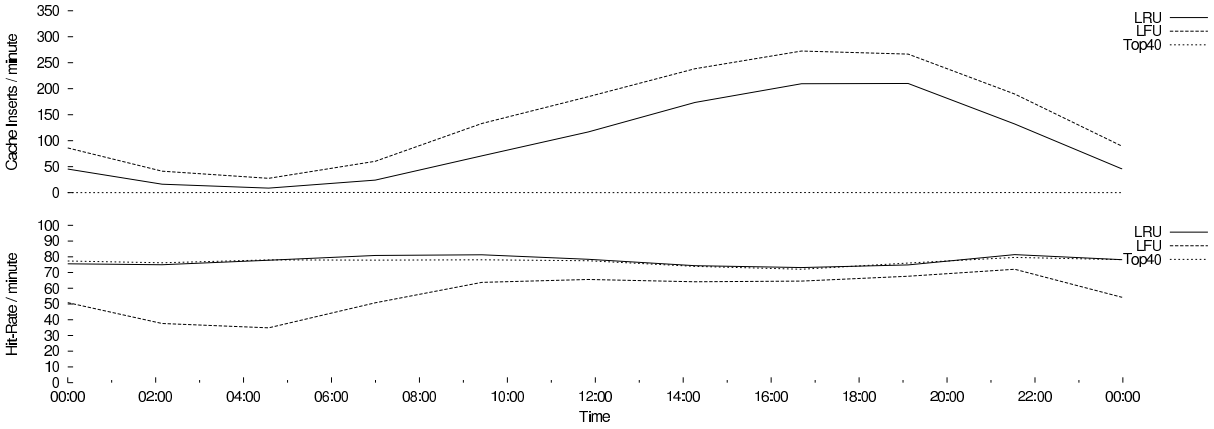
Cache	**4GB**		**126GB**		**640GB**	
	HR	Ins	HR	Ins	HR	Ins
LRU	11.8 σ=7.9	230.3 σ=144.6	75.2 σ=6.3	96.8 σ=70.8	96.7 σ=2.4	12.4 σ=8.8
LFU	33.7 σ=9.7	219.9 σ=139.3	61.8 σ=15.7	147.1 σ=86.6	61.8 σ=15.7	147.1 σ=86.6
Top40	34.9 σ=8.0	0.014 σ=0.14	74.2 σ=6.3	0.16 σ=0.56	92.7 σ=3.3	0.026 σ=0.23

Table 2: Average **H**it-**R**ate (%) and average **Ins**erts of different kind of cache types and sizes

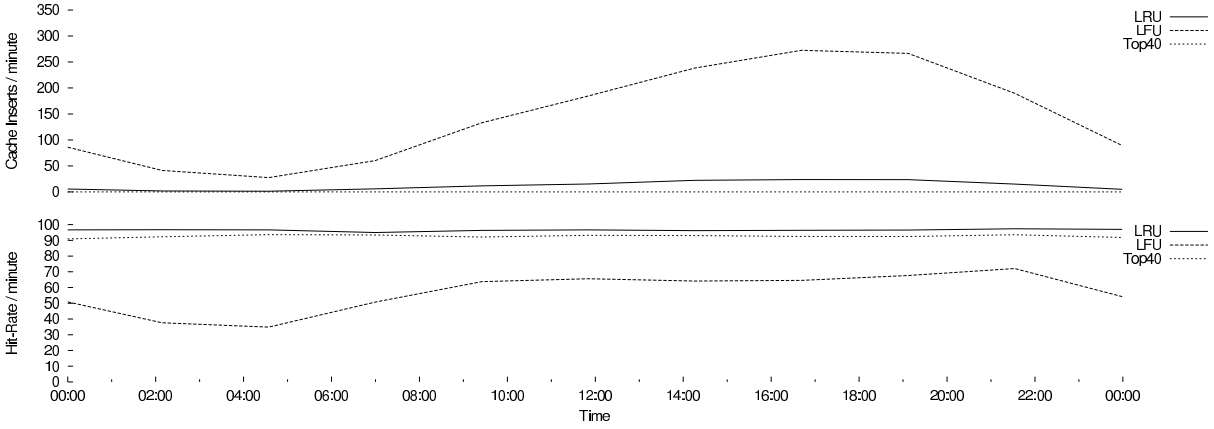
of all caches increases when the size of the cache increases. LFU is already at a maximum with 126Gb cache-size. LRU benefits the most from the biggest cache and LFU performs relatively better in the smallest cache size. In all cache sizes Top40 keeps up with the best performing algorithm for that cache size. Top40 has almost no inserts at any cache size while both LRU and LFU do a lot of cache inserts. A media-file is always inserted from the main storage to a cache and due to the size of the media-file an insert will take considerable time (in the order of seconds) and will have a performance impact. In the production environment the inserts need to be at a minimum. The Hit-rate is an performance indicator, but the number of inserts is even more important. The results shows the effectiveness of the Top40 algorithm, the hit-rate is high while the inserts are minimal. In section A graphs with the results of the simulation are shown. To clearly show the effectiveness of the Top40 algorithm the following graphs show the result for one day (Wednesday 12 January).



(a) 4GB Cache



(b) 126GB Cache



(c) 640GB Cache

Figure 5: Hit-rate/minute and Inserts/minute of LRU, LFU and Top40 on Wed-12 (1 day, 1440 samples) with different cache sizes

7 Conclusion and Future work

Using real input data and two simple but effective caching algorithms as comparison, the specially written Top40 caching (section 4) performs like LRU with big caches and LFU with a small cache size, as shown in table 2), when looking at cache Hit Rate but the cache inserts done by the Top40 algorithm are always far below the cache inserts done by the LRU algorithm making the Top40 far more effective.

The Top40 caching algorithm has tunable parameters like threshold and decay-time (formula 4). Using the test environment it is possible to do simulations with different values of those parameters to show the influence of those parameters on the cache hit-rate and cache inserts. It looks that for the biggest cache (640GB) a better hit-rate is possible, but this may lead to more inserts.

There many more cache algorithms available from simple extentions to LRU and LFU [7] to more complicated like LANDLORD[6] or ARC[3]. Using the created simulation environment these could be compared to Top40-caching.

References

[1] gnuplot homepage, Januari 2009. `http://www.gnuplot.info/`.

[2] Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, Edward A. Fox, and Stephen Williams. Removal policies in network caches for world-wide web documents. In *SIGCOMM '96: Conference proceedings on Applications, technologies, architectures, and protocols for computer communications*, pages 293–305, New York, NY, USA, 1996. ACM.

[3] Nimrod Megiddo and Dharmendra S. Modha. Arc: A self-tuning, low overhead replacement cache. In *In Proceedings of the 2003 Conference on File and Storage Technologies (FAST*, pages 115–130, 2003.

[4] Andrew S. Tanenbaum. *Operating Systems, Design and Implementation.* 1987.

[5] Larry Wall. *Programming Perl.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2000.

[6] Neal E. Young. On-line file caching. In *In Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 82–86. ACM Press, 1998.

[7] Yuanyuan Zhou, James F. Philbin, and Kai Li. The multi-queue replacement algorithm for second level buffer caches. In *In Proceedings of the 2001 USENIX Annual Technical Conference*, pages 91–104, 2001.

A 10 day graphs of the simulation results

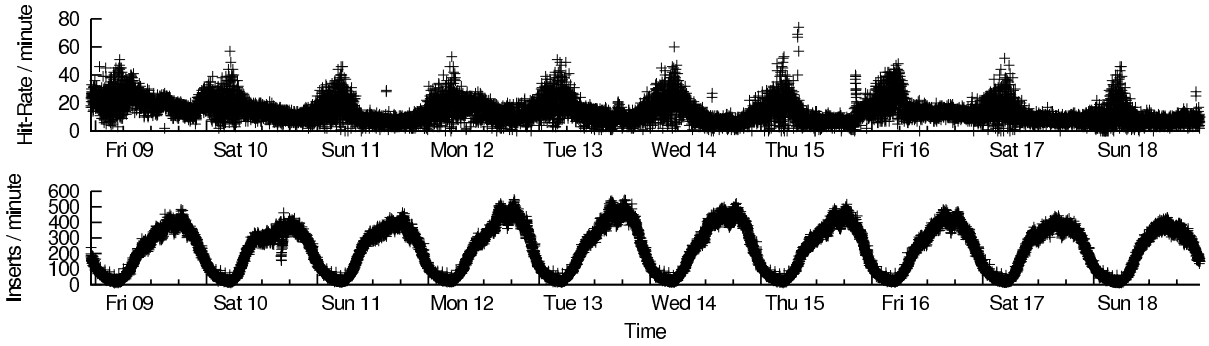


Figure 6: HitRate an Inserts of LRU 4GB Cache

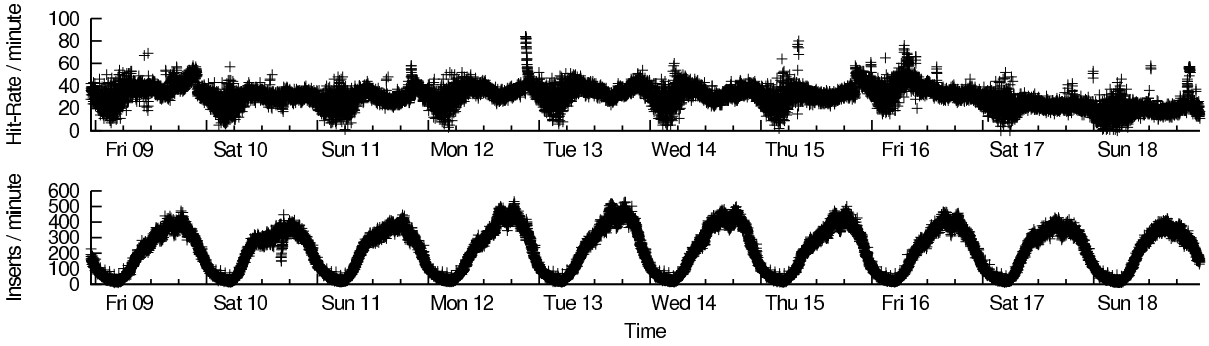


Figure 7: HitRate an Inserts of LFU 4GB Cache

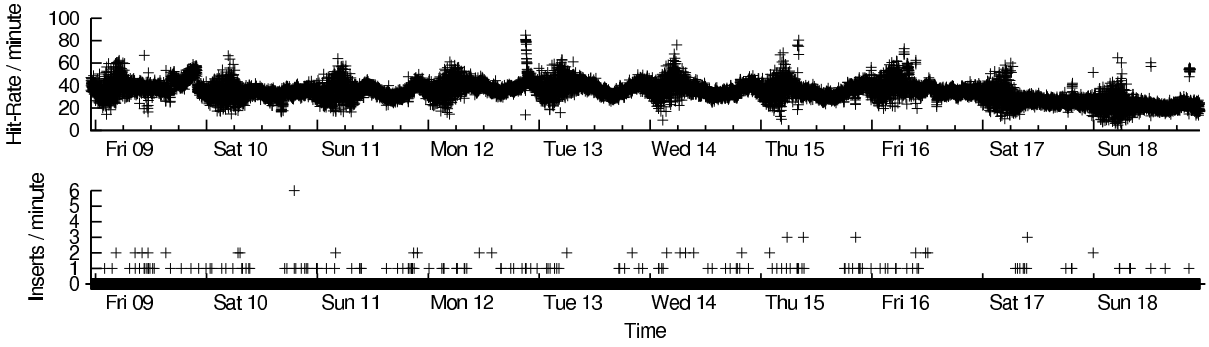


Figure 8: HitRate an Inserts of Top40 4GB Cache

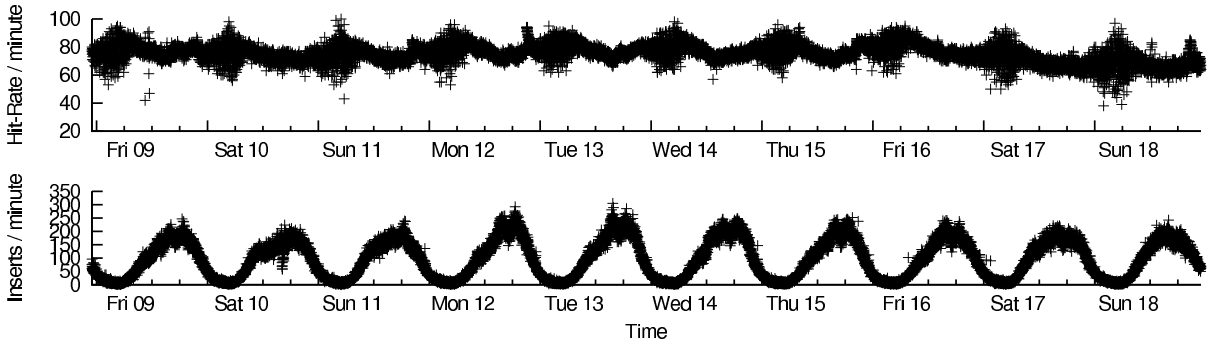


Figure 9: HitRate an Inserts of LRU 126GB Cache

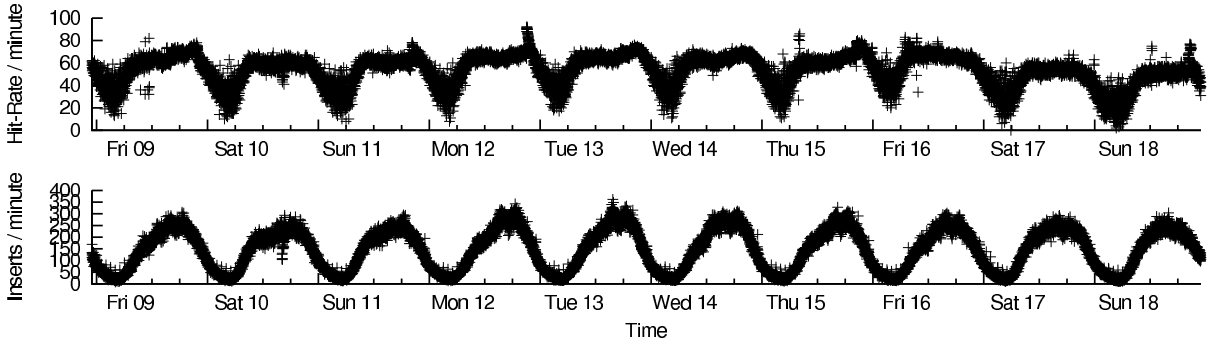


Figure 10: HitRate an Inserts of LFU 126GB Cache

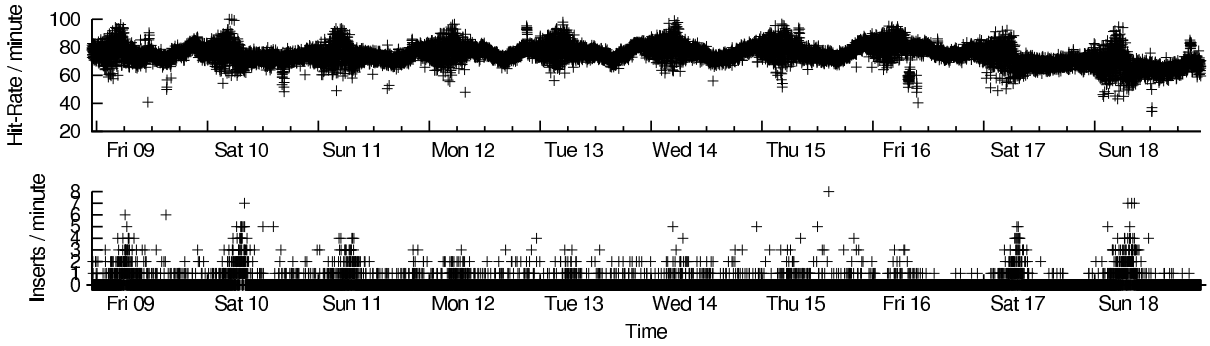


Figure 11: HitRate an Inserts of Top40 126GB Cache

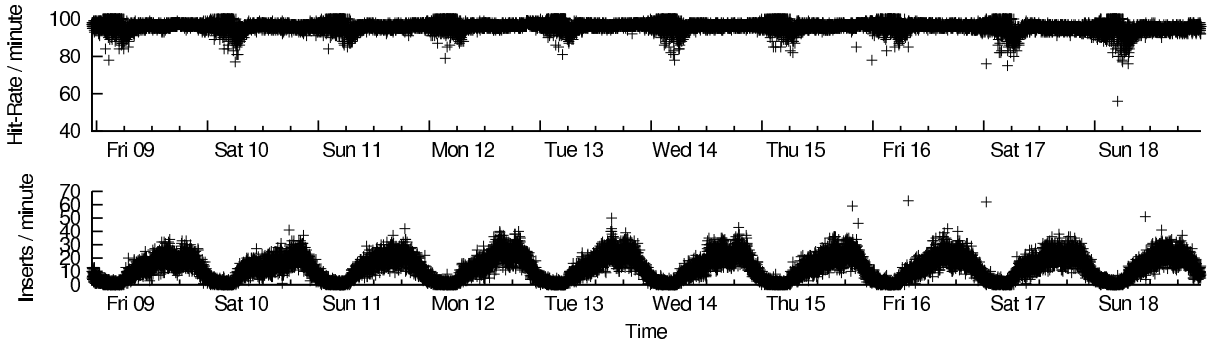


Figure 12: HitRate an Inserts of LRU 640GB Cache

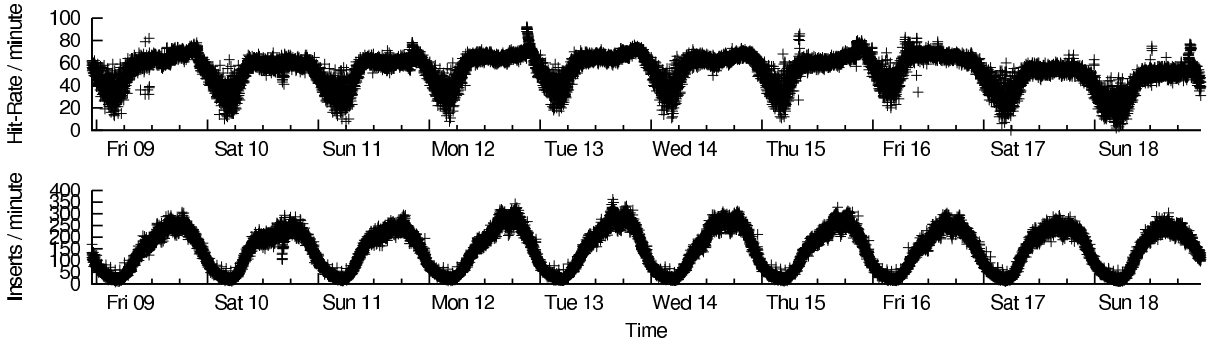


Figure 13: HitRate an Inserts of LFU 640GB Cache

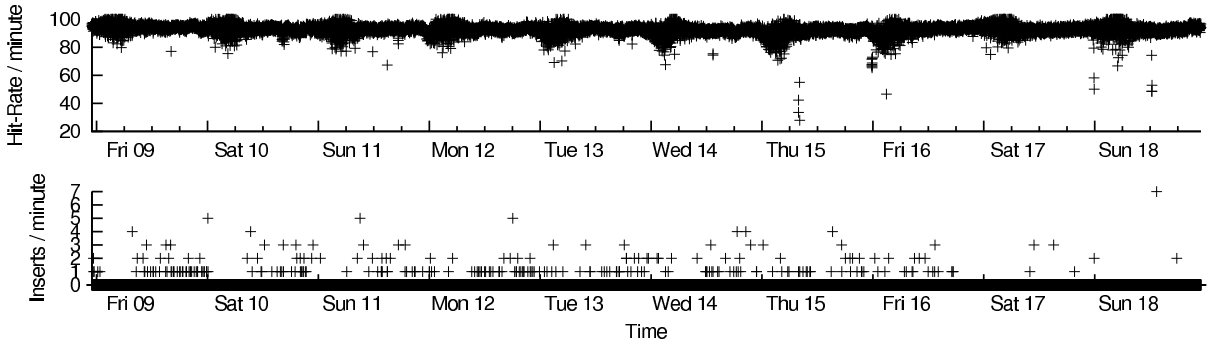


Figure 14: HitRate an Inserts of Top40 640GB Cache