Multi-Size Optional Offline Caching Algorithms

Andrew Choliy
Rutgers University
ayc41@scarletmail.rutgers.edu

Max Whitmore
Brandeis University
whitmore@brandeis.edu

Gruia Calinescu
Illinois Institute of Technology
calinescu@iit.edu

ABSTRACT

The optional offline caching (paging) problem, where all future file requests are known, is a variant of the heavily studied online caching problem. This offline problem has applications in web caching and distributed storage systems. Given a set of unique files with varying sizes, a series of requests for these files, fast cache memory of limited size, and slow main memory, an efficient replacement policy is necessary to decide when it is best to evict some file(s) from the cache in favor of another. It is known that this problem is NP-complete, and few approximation algorithms have been proposed. We propose three new heuristics, as well as a 4-approximation algorithm. We then evaluate each algorithm by the metrics of runtime complexity and proximity to the optimal solutions of many synthetic data sets.

KEYWORDS

offline caching, caching algorithm, distributed storage

ACM Reference format:
Andrew Choliy, Max Whitmore, and Gruia Calinescu. 2017. Multi-Size Optional Offline Caching Algorithms. In *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis, Denver, Colorado USA, November 2017 (SuperComputing 2017),* 3 pages.
DOI: 10.475/123_4

1 INTRODUCTION

The general caching problem involves choosing an efficient replacement policy for a two-level memory system where one is local, fast cache of fixed size and the other is distant, slow main memory. The problem's goal is to minimize access to the slow memory. This problem is well-studied in the online version in the context of CPU caches. In the offline model, the sequence of file requests is known in advance. Irani [6] introduces a variant of the offline problem in the context of web caching where caching a faulted file is optional and the cost of a file fault is equal to its size (bit model). We consider this model. Zhao et al. [7] use the same model when studying distributed storage systems and was the catalyst for our work.

Formally, we are given a finite set M of m unique files. Each unique file i has a size $s(i) \in \mathbb{Z}^+$ equal to its cost $c(i)$. We are also given a finite list R of n requests to the unique files. Each request $r \in R$ corresponds to some file i, denoted $g(r) = i$. We are also given a cache capacity $C \in \mathbb{Z}^+$ and an initial cache Q_1 which may or may not be empty. For each r, if the requested file i is in the cache, it is read at cost zero. If the file requested is not in the cache, a fault occurs and $g(r)$ must be read from slow memory. Because the policy we examine for caching is optional, the faulted file i may or may not be loaded into the cache for later use.

A solution to a given general caching problem has a length-n schedule of caches, where each cache Q_r is a set of unique files stored at location r in the list of requests. To obtain the total cost of the schedule, use $\sum_{r=1}^{n} \sum_{i \in (g(r) \cup Q_{r+1}) \setminus Q_r} c(i)$. Here, $Q_{n+1} = \emptyset$. An optimal replacement policy loads and evicts files at each request in such a way that it minimizes the overall fault cost. A feasible solution is displayed by Figure 1.

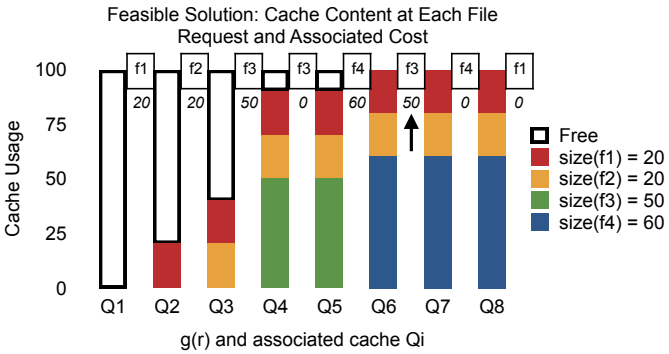


Figure 1: With C = 100, $cost(g(r))$ is shown below each file request. The cumulative cost is: 20, 40, 90, 90, 150, 200, 200, 200. Each file requested not present in the cache is loaded except for f3 at the index denoted by the arrow. It is read from main memory to later read f4 for free and save cost.

2 METHODOLOGY

We consider several offline algorithms. The first was proposed by Zhao et al. [7]. We then present three of our own variants and our adaptation of Bar-noy et al.'s [2] framework in the context of Loss Minimization to optional caching. To evaluate these algorithms, we compare total cost and runtime metrics using a large set of synthetic data samples. We also include results for the well-known LRU algorithm and the optimal solution (when feasible to calculate) as reference points. All the algorithms have a runtime of $O(nm \log m)$ except Overload Reduction, which is $O(n^2)$.

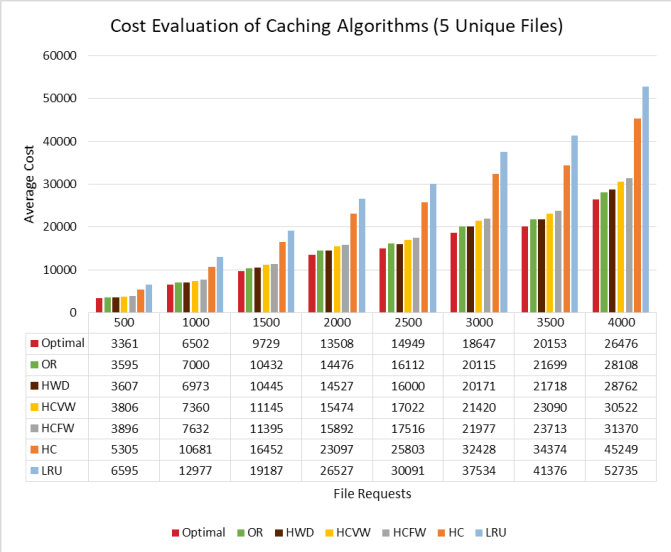
Heuristic Caching from HyCache+ (HC) Proposed by Zhao et al. [7], a cost/gain value is computed for each file equal to its size multiplied by the number of remaining requests for that file. As the algorithm iterates through the file requests, these values dictate if $g(r)$ is added to the cache, and which files will be evicted, if any.

Heuristic Caching with Fixed Window Size (HCFW). HCFW is our first variant of the HC algorithm. Rather than considering all of R when computing the cost/gain value for a file, we only look at a window of $3m$ file requests ahead.

Heuristic Caching with Variable Window Size (HCVW). The HCVW algorithm is similar to HCFW, but the length of the

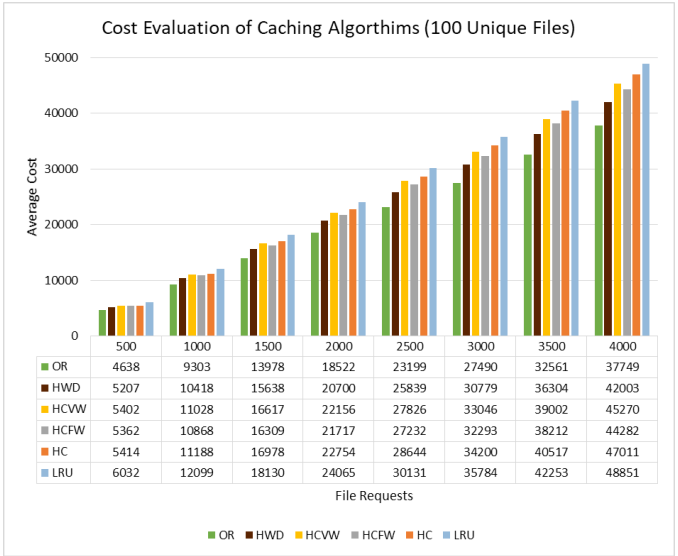
SuperComputing 2017, Denver, Colorado USA

DOI: 10.475/123_4



	500	1000	1500	2000	2500	3000	3500	4000
Optimal	3361	6502	9729	13508	14949	18647	20153	26476
OR	3595	7000	10432	14476	16112	20115	21699	28108
HWD	3607	6973	10445	14527	16000	20171	21718	28762
HCVW	3806	7360	11145	15474	17022	21420	23090	30522
HCFW	3896	7632	11395	15892	17516	21977	23713	31370
HC	5305	10681	16452	23097	25803	32428	34374	45249
LRU	6595	12977	19187	26527	30091	37534	41376	52735

Figure 2: Data for average cost with 5 unique files.



	500	1000	1500	2000	2500	3000	3500	4000
OR	4638	9303	13978	18522	23199	27490	32561	37749
HWD	5207	10418	15638	20700	25839	30779	36304	42003
HCVW	5402	11028	16617	22156	27826	33046	39002	45270
HCFW	5362	10868	16309	21717	27232	32293	38212	44282
HC	5414	11188	16978	22754	28644	34200	40517	47011
LRU	6032	12099	18130	24065	30131	35784	42253	48851

Figure 3: Data for average cost with 100 unique files.

window expands and contracts to include at least one request of each unique file from the current point in the request list to its end. If a file no longer appears in the request list, it is excluded from this rule.

Heuristic Caching Weighted by Distance (HCWD). This algorithm takes the distance of each request into account when computing the cost/gain value for each file. For each request, use the size of its file divided by its distance from the current step in the request list. The sum of these values for a given file becomes that file's cost/gain.

Overload Reduction (OR). This algorithm is adapted from Bar-Noy et al. [2] and also inspired by Albers et al. [1]. The original version was intended for the forced caching version of our problem; we have modified it for optional caching. We can prove this is a 4-approximation, improving [6].

3 RESULTS

Our evaluations were performed on a machine with an i7 4810MQ CPU and 8 GB of DDR3 RAM. We consider cases with m = 5, 10, 100, and 1000 unique files, as well as between 500 and 4000 file requests at increments of 500. File sizes range from 21 to 40 (m = 5 or 10) or 1 to 20 (m = 100 or 1000). C is always 100. For each combination of number of unique files and number of file requests, we generate 30 random trials. Then we average the total cost and runtime metrics for each algorithm across the trials.

Each of our algorithms (HCFW, HCVW, and HCWD) generally perform better than HC in terms of average cost while maintaining HC's fast runtime. With 5 unique files, HC returns on average 168.8% of the optimal cost, whereas OR and HCWD give just 107.3% and 107.6%, respectively. With 100 unique files, OR produces an output that is on average 18.2% lower than HC. HCWD is second best at a 8.6% improvement. When the number of unique files is relatively close to the number of file requests, we see very similar cost metrics across the algorithms, even LRU. This makes sense because it becomes rare that files are repeated often, minimizing the usefulness of caches in the first place. This is best reflected in the cases with 1000 unique files, where we see only OR having any notable reduction in total cost versus HC. Even then, it is only an improvement of 5.6% on average. The OR algorithm generally has the best performance, but at the cost of a significantly longer runtime to the point where it must be measured in seconds. Whether the time and system resources spent on executing the OR algorithm is acceptable will depend on the specific application. If not, the HCWD algorithm is generally the next best option in optimizing cost and runtime.

4 PREVIOUS AND RELATED WORK

In the simplest model with unit-size files and cost, an optimal algorithm exists given by Belady [4]: evict the file that is requested farthest from the current. The problem becomes much harder in the multi-size model, shown by Chrobak et al. [5] to be NP-complete. Irani [6] gives the first solution as an O(log k)-approximation ratio, where k is the ratio of C to the size of the smallest file.

In the offline forced caching model, Albers et al. [1] provide an O(log(M + C))-approximation algorithm where M and C denote the cache size and the largest file fault cost, respectively. They also provide a reduction to the Loss Minimization problem. Bar-noy et al., [2] using the local ratio technique from Bar-Yehuda and Even, [3] provide a 4-approximation algorithm to Loss Minimization and for the Multi-Size Forced Offline Caching problem.

5 CONCLUSION AND FUTURE WORK

Additional evaluation of these algorithms using real-world workload data would provide greater insight to the usefulness of these algorithms in applications.

Acknowledgement. This work was supported by the National Science Foundation under award NSF - 1461260 (BigDataX REU).

REFERENCES

[1] Susanne Albers, Sanjeev Arora, and Sanjeev Khanna. 1999. Page replacement for general caching problems. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms (SODA '99)*. Society for Industrial and Applied Mathematics, Baltimore Maryland, 31–40.

[2] Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, and Joseph (Seffi) Naor. 2001. A Unified Approach to Approximating Resource Allocation and Scheduling. *Association for Computing Machinery (ACM)* 48, 5 (2001), 1069–1090.

[3] Reuven Bar-Yehuda and Shimon Even. 1985. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics* 25 (1985), 27–46.

[4] L. A. Belady. 1966. A Study of Replacement Algorithms for a Virtual-storage Computer. *IBM Systems Journal* 5, 2 (1966), 78–101.

[5] Marek Chrobak, Gerhard J. Woeginger, Kazuhisa Makino, and Haifeng Xu. 2012. Caching is hard - even in the fault model. *Algorithmica* 63, 4 (2012), 781–794.

[6] Sandy Irani. 1999. Page Replacement with Multi-Size Pages and Applications to Web Caching. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing (STOC '97)*, Vol. 3. ACM, El Paso Texas, 701–710.

[7] Dongfang Zhao, Kan Qiao, and Ioan Raicu. 2015. HyCache+: Towards Scalable High-Performance Caching Middleware for Parallel File Systems. *International Journal of Big Data Intelligence* (2015).