Study of cache performance in distributed environment for data processing

Dzmitry Makatun[1,3], Jérôme Lauret[2] and Michal Šumbera[3]

[1]Faculty of Nuclear Physics and Physical Engineering, Czech Technical University in Prague
[2]STAR, Brookhaven National Laboratory, USA
[3]Nuclear Physics Institute, Academy of Sciences, Czech Republic

E-mail: makatun@rcf.rhic.bnl.gov

Abstract. Processing data in distributed environment has found its application in many fields of science (Nuclear and Particle Physics (NPP), astronomy, biology to name only those). Efficiently transferring data between sites is an essential part of such processing. The implementation of caching strategies in data transfer software and tools, such as the Reasoner for Intelligent File Transfer (RIFT) being developed in the STAR collaboration, can significantly decrease network load and waiting time by reusing the knowledge of data provenance as well as data placed in transfer cache to further expand on the availability of sources for files and data-sets. Though, a great variety of caching algorithms is known, a study is needed to evaluate which one can deliver the best performance in data access considering the realistic demand patterns. Records of access to the complete data-sets of NPP experiments were analyzed and used as input for computer simulations. Series of simulations were done in order to estimate the possible cache hits and cache hits per byte for known caching algorithms. The simulations were done for cache of different sizes within interval 0.001 - 90% of complete data-set and low-watermark within 0-90%. Records of data access were taken from several experiments and within different time intervals in order to validate the results. In this paper, we will discuss the different data caching strategies from canonical algorithms to hybrid cache strategies, present the results of our simulations for the diverse algorithms, debate and identify the choice for the best algorithm in the context of Physics Data analysis in NPP. While the results of those studies have been implemented in RIFT, they can also be used when setting up cache in any other computational work-flow (Cloud processing for example) or managing data storages with partial replicas of the entire data-set.

1. Introduction

Efficient usage of available cache space is important for transferring and accessing data in computational grids. Though, a great variety of caching algorithms is known, a study is needed to evaluate which one can deliver the best performance in data access considering the realistic demand patterns.

Cache cleaning algorithms can be applied to keep in the cache of data-transfer tools files that may be reused. The size of those cache is small (several percent of the entire dataset) and the clean up has to take place regularly to make space for further transfers. Another task can, for example, be to delete a part of local data replica if no longer in use or requested. The problem posed by cache cleanup is to select and delete files which are the least likely to be used again. An investigation to find the most appropriate algorithm is required.

In this study, all the caching algorithm were implemented following the concept known as "water-marking". Water-marking is an approach where thresholds are set for the cache cleanup starts and stops. It considers the current disk space occupied by the data in cache and the high-mark and the low-mark for cache size are externally set up and can be adjusted as needed. When the used cache size exceeds the high-mark, the cache clean-up starts, and files are deleted until the used cache size gets below the low-mark. The time interval between clean-ups depends on combination of high/low marks, cache size and data-flow. With watermarking concept more computational demanding algorithms can be implemented as the cleanup may be independent of the data transfers.

2. Access patterns

Several data access patterns were extracted from log files of data management systems at sites of HEP/NPP experiments in order to simulate caching. Three different access patterns were used as input for our simulations:

STAR1: the pattern was extracted from Xrootd [8] logs taken from the STAR experiment's Tier-0 site of RHIC Computing Facility at Brookhaven National Laboratory (RCF@BNL), it consist of records made during a 3 months period (June-August 2012) of all datasets available in STAR.

STAR2: extracted from the same data management system, but of records made during a 7 months period (August 2012 - February 2013) under similar conditions.

GOLIAS farm is part of regional computing center for particle physics at the Institute of Physics (FZU) in Prague, and is part of a Tier-2 site for the CERN/ATLAS experiment. The facility also performs data processing for another experiment - AUGER, which makes less than 1% of the total requests. The pattern was extracted from DPM [9] logs for a 3 months period (November 2012 - February 2013).

The usage of access patterns corresponding to different time periods and experiments helps to verify the results of our simulations. As input of our simulations, we tried to focus on a few characteristic access patter. The key parameters we came up with for the three access patterns are given in Table 1. Both STAR access patterns have similar parameters. It is noteworthy to mention that the first one was taken right before the Quark Matter 2012 conference and the second one, right after. This is important as the access to data is intensified before a conference and without verification, it would be doubtful if our findings would be stable across time. The number of files requested only once during the period, is less than 10% in both cases.

The FZU/GOLIAS access pattern is taken from another experiment with different data-storage structure, DPM is used here within a Tier-2 data access context (user analysis). This access pattern is much less uniform and differs from the other two: the size of files is not explicitly limited and can reach 18 GB, the number of requests for a file varies from 1 up to 94260, with an average 5, while it varies from 1 to 203 for the STAR patterns. 44% of files were requested only once

Our analysis is not sensitive to the particular Data Management system, (XRootD or DPM), further we explain differences in the access patterns with specifics of experiment and role of the cluster in the system (tier level).

3. Cache simulation

The efficiency of caching can be estimated by two quantities, the cache hits H and cache hits per megabyte of data H_d from here on referred to as *cache data hits*:

$$H = \frac{N_{cache}}{N_{req} - N_{set}} \tag{1}$$

Table 1. Summary of three user access patterns used as input for simulations. The selection of two time sequence in STAR and one from a different experiment aims at verifying stability of our result and findings.

		STAR1	STAR2	GOLIAS
Time period	months	3	7	3
Number of requests	$\times 10^6$	33	52	21
Data transferred	PB	50	80	10
Maximal number of requests for one file	–	192	203	94260
Average number of requests per file	–	19	15	5
Number of unique files	$\times 10^6$	1.8	1.7	3.8
Total size of dataset	PB	1.45	2	1
Maximal file size	GB	5.3	5.3	18
Average file size	GB	0.8	1	0.3

$$H_d = \frac{S_{cache}}{S_{req} - S_{set}} \tag{2}$$

where N_{req} is the total number of requests, S_{req} -the total amount of transferred data in bytes, N_{set} -the number of unique files witch were requested at least ones during the considered period, S_{set} - the total size of those unique files in bytes, N_{cache} - the number of files transferred from cache, S_{cache} - the amount of data transferred from cache in bytes.

By maximizing the cache hits H one reduces the number of files transferred from external sources and thus reduces the overall make-span due to transfer startup overhead for each file. By maximizing the cache data hits H_d one reduces the network load, since more data is reused from the local cache.

If the access pattern is completely random, the expected cache hit and cache data hits would be equal to *cache size/storage size*, so it can be useful to compare the actual cache performance to this estimation.

Altogether 27 different caching algorithms were simulated. But the majority of studied algorithms did not bring any improvements over the simplest one (FIFO). Only the algorithms that appeared to be the most efficient are discussed in this paper:

- **First-In-First-Out (FIFO)**: evicts files in the same order they entered the cache. Performance of this trivial algorithm provide a good comparison benchmark against more sophisticated ones which can demand significant computational resources.

◯ **Least-Recently-Used (LRU)**: evicts the set of files which were not used for the longest period of time.

● **Least-Frequently-Used (LFU)**: evicts the set of files which were requested less times since they entered the cache.

★ **Most Size (MS)**: evicts the set of files which have the largest size.

\+ **Adaptive Replacement Cache (ARC)**[5]: splits cached files into two lists: L1 - files with *access count* = 1, and L2 - files with *access count* > 1. LRU is then applied to both list, the self adjustable parameter $p = cache\ hits\ in\ L1/cache\ hits\ in\ L2$ defines the number of cached files in each list. The general idea is to invest more into the list which delivers more hits.

∗ **Least Value based on Caching Time (LVCT)**[4]: Deletes files with the smallest value of the Utility Function:

$$UtilityFunction = \frac{1}{CachingTime \times FileSize} \tag{3}$$

where **Caching Time** of a file F is total size of all files accessed after the last request for the file F.

▽ **Improved-Least Value based on Caching Time (ILVCT)**[3]: Deletes files with the smallest value of the Utility Function:

$$UtilityFunction = \frac{1}{NumberOfAccessedFiles \times CachingTime \times FileSize} \tag{4}$$

where **Caching Time** is the same as for LVCT and **Number Of Accessed Files** is a number of files requested after the last request for selected file.

4. Results

Three series of simulations with three access patterns were performed for each algorithm (90 simulations in total for each algorithm):

- 10 simulations with cache size 1-90 % of storage with fixed low-mark 75% and high-mark 95%. Those simulations aim to understand what would happen if we have large storage cache. Those cases are aligned with a DPM and Xrootd use where most (if not all) the dataset(s) are in the system.
- 10 simulations with cache size 1.2 - 0.0025% of storage with fixed low-mark 75% and high-mark 85%. This high-mark was selected in order to leave enough margin (15%) in case when a large file is requested at the moment when the cache is almost full. We used those simulations to understand the behavior of cache cleanup if the cache size is by several orders less than the dataset size. This is in fact a most common case for transfer cache on data transfer nodes.
- 10 simulations with fixed cache size 10% of storage,fixed high-mark 95% and variable low mark within 0-90%. We performed those simulations to better understand the effect of data retention on cache (delete the least in hope of re-use).

In order to compare one algorithm against another an average improvement can be calculated in a following way:

$$Average\ improvement = \frac{\sum_{i=1}^{n} \frac{value2_i - value1_i}{value1_i}}{n}, \tag{5}$$

where n is the total amount of simulations with equal parameters for both algorithms, i is the number of the simulation,$value1$ - cache hits or cache data hits of a reference algorithm (FIFO), $value2$ - cache hits or cache data hits of a compared algorithm.

Table 2 contains the results of comparison of all the algorithms represented in this paper against FIFO. 60 values for each algorithm were taken from results of simulation series 1 and 2 in order to calculate the average improvement. According our results, the LFU algorithm does not bring any improvement over FIFO due to its well known flaw - it accumulates files which were popular for a short period of time, and those files prevent newer ones from staying in cache. The ARC algorithm was developed as an improvement to LRU, and not surprisingly, it outperforms LRU by ∼5% in cache hits and ∼7% in cache data hits. Therefore, LFU and LRU algorithms could be excluded from the further analysis in our case studies.

Table 2. Average improvement of algorithms over FIFO.

Algorithm	cache hits	cache data hits
MS	**116** %	-20 %
LRU	8 %	5 %
LFU	-27 %	-18 %
ARC	13 %	**11** %
LVCT	**86** %	2 %
ILVCT	28 %	2 %

The graphical detailed results of simulations for all 3 series are given at Figures 1-3. The performance of FIFO and 3 algorithms appeared to be the most efficient (MS, ACR and LVCT) is presented at the plots.

Difference between Tier-2 and Tier-0 access patterns leads to distinct cache performance. Only the data dedicated for the ongoing analysis is placed at the Tier-2 site, while at the Tier-0 site all the experimental data is stored. As a result – the access pattern at the Tier-2 site has stronger access locality. STAR1 and STAR2 access patterns correspond to Tier-0 site and GOLIAS to a Tier-2 site. Thus, any particular algorithm at the plots delivers higher cache hits and cache data hits for GOLIAS access pattern than for STAR1 and STAR2.

The behavior of algorithms is similar within each dataset that is, their respective performance ordering is the same. This observation implies that if one of the algorithm appears to be most efficient for one of the datasets it is also the most efficient for the other datasets. This statement is also true for the rest of simulated algorithms not present on our figure. Though the communities represented by the STAR and GOLIAS access patterns are somewhat similar, this result is slightly surprising as our case studies represent two time sequence within the same usage and totally uncorrelated experiments. It would be interesting to study those algorithms in a different experimental context (outside the HEP/NPP communities) but this study is outside the scope of our paper.

The MS algorithm has shown outstanding cache hits, but the lowest cache data hits. At the same time the LVCT has cache hits comparable to the MS while cache data hits are 2% improved over the FIFO. This algorithm could be an optimal when the cache hits is the target optimization parameter. The ARC algorithm has shown the highest cache data hits for studied access patterns.

The dependence of algorithms performance on low mark is presented at Figure 3. With higher low mark the number of clean-ups increases and that is why the difference between algorithms becomes more notable. Performance of efficient algorithms (FIFO, LRU, ARC and LVCT) increases steadily with the low mark. One should be careful when setting up a cache low mark at a particular site, since a higher low mark can increase cache performance significantly, but at the same time it can result in running cache clean-ups too often, consuming significant computational resources (and potentially increasing the chance to interfere with data transfers hence, degrading transfer performances if delete/writes/read overlap).

5. Conclusion

Performance of cache algorithms implemented with watermarking concept was simulated for a wide range of cache sizes and low marks. Three access patterns from Tier-0 and Tier-2 sites of 2 different experiments were used as input for simulations. Regardless of the cache size, Tier-level and specificity of experiment the LVCT and ARC appeared to be the most efficient caching algorithms for the communities we investigated. While we found the stability of relative

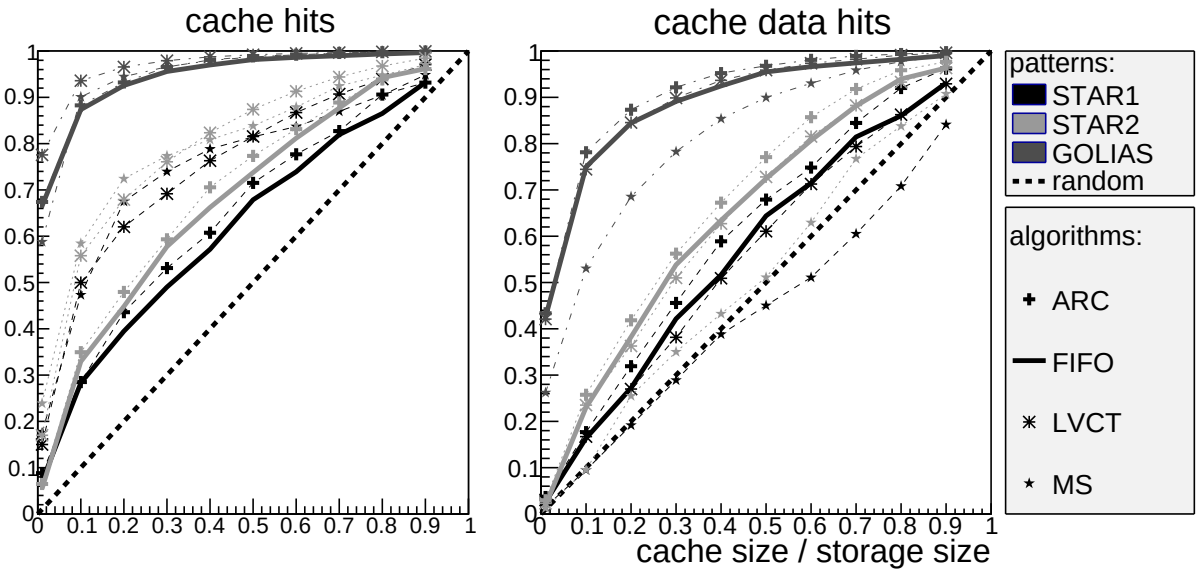

Figure 1. Results of simulation. Performance of algorithms for cache of larger size can be compared. For all of the simulations on this plot the following parameters were fixed: low mark = 0.75, high mark = 0.95

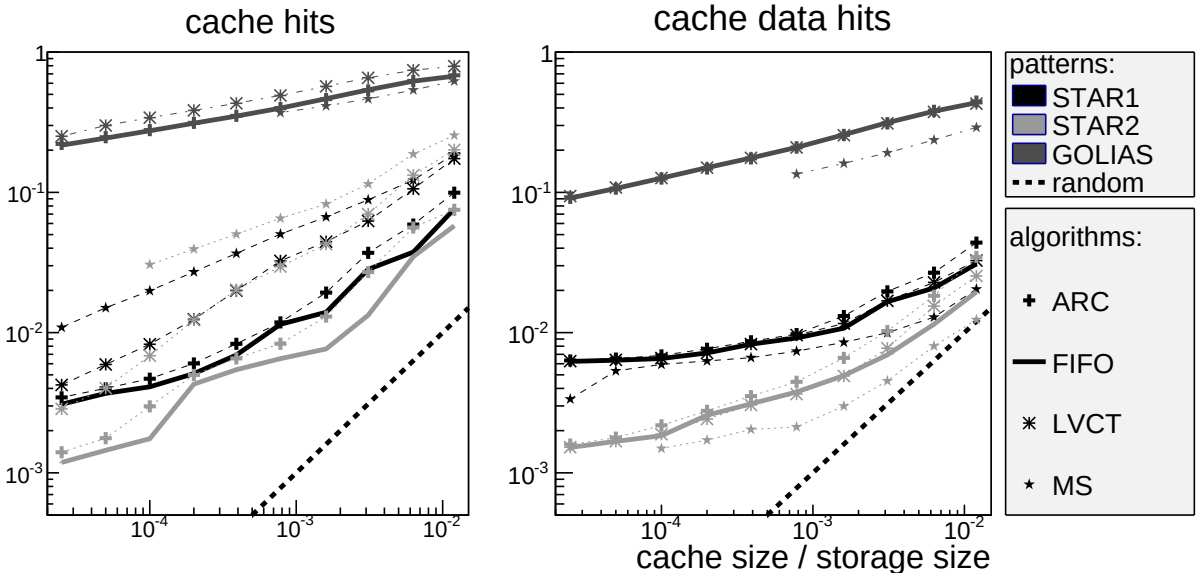

Figure 2. Results of simulation. Performance of algorithms for cache of smaller size can be compared. For all of the simulations on this plot the following parameters were fixed: low mark = 0.75, high mark = 0.85

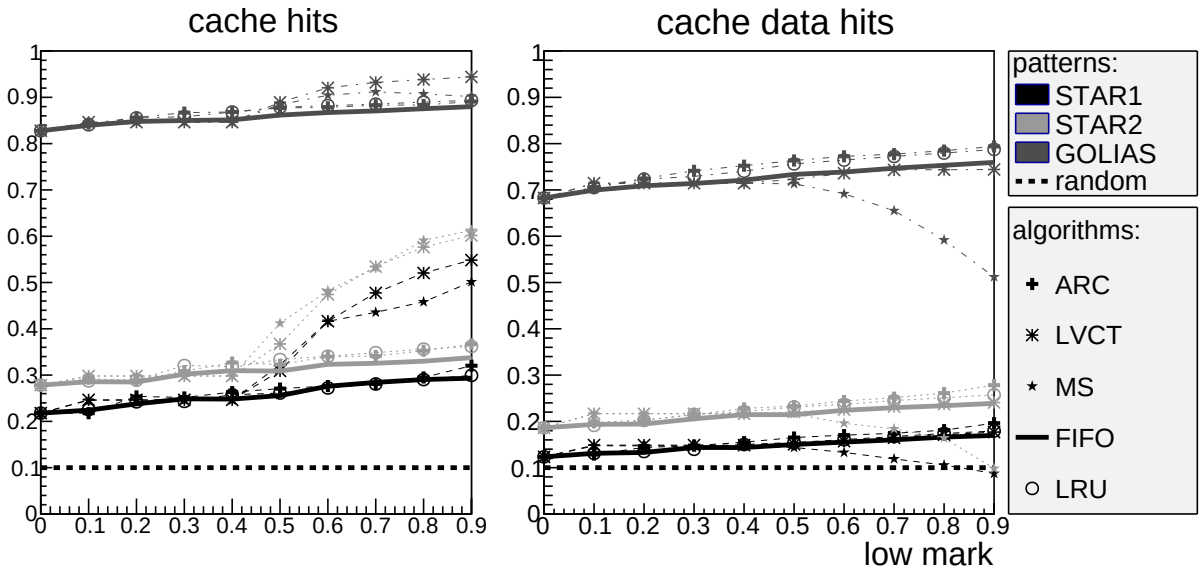

Figure 3. Results of simulation: dependence of cache performance on low mark. For all of the simulations on this plot the following parameters were fixed: cache size / storage size = 0.1, high mark = 0.95

algorithms' performance surprising at first, we attribute this result to an access pattern which is intrinsically similar in nature. An extension of this work could be the investigation of this result in a different experiment context which is a work beyond our initial goal. LVCT and ARC could certainly be safely implemented in tools such as RIFT.

- If the goal is to minimize makespan due to a transfer startup overhead the LVCT algorithm should be selected.
- If the goal is to minimize the network load the ARC algorithm is an option.

Acknowledgments
The work has been supported by the grant 13-02841S of the Czech Science Foundation (GACR) and the US Department of Energy. The support of the Visegrad Fund is gratefully acknowledged. Furthermore the authors would like to acknowledge with much appreciation the help of the staff of GOLIAS computer farm for kindly providing them with the log files and access to the facility for running simulations.

References

[1] Makatun D, Lauret J and Sumbera M 2012 Distributed Data processing in high-energy physics *Proc. of PhD students workshop at FNSPN CVUT* (Prague) ISBN 978-80-01-05138-2
[2] Zerola M, Lauret J, Bartak R and Sumbera M 2012 One click dataset transfer: toward efficient coupling of distributed storage resources and CPUs *J. Phys.: Conf. Ser.* **368**
[3] Achara J P, Rathore A, Gupta V K and Kashyap A 2010 An improvement in LVCT cache replacement policy for data grid *Proc. of the 13th Int. Workshop on Advanced Computing and Analysis Techniques in Physics Research* (Jaipur) p 44
[4] Song Jiang and Xiaodong Zhang 2003 Efficient Distributed Disk Caching in Data Grid Management *Proc. of the IEEE Int. Conf. on Cluster Computing (CLUSTER03)* (Hong Kong) pp 446-51
[5] Megiddo, Nimrod and Modha D S 2004 Outperforming LRU with an adaptive replacement cache algorithm *Computer* **37** 58-65
[6] Fast Data Transfer *Project web-site: http://monalisa.cern.ch/FDT/*
[7] High Performance Storage System *Project web-site: http://www.hpss-collaboration.org/*

[8] Xrootd *Project web-site: http://xrootd.slac.stanford.edu/*
[9] Disk Pool Manager *Project web-site:* https://svnweb.cern.ch/trac/lcgdm/wiki/Dpm