Proc. the 17th Int'l Conf. Computer Communications and Networks (ICCCN), St. Thomas, Virgin Islands, Aug. 2008.

Security-Aware Cache Management for Cluster Storage Systems

Mais Nijim, Xiao Qin[†], Ziliang Zong, Xiaojun Ruan, Kiranmai Bellam
Computer Science, School of Computing
The University of Southern Mississippi, Hattiesburg, MS 39406
{Mais.nijim, ahmed.abukmail}@usm.edu
Computer Science and Software Engineering
Auburn University, Auburn, AL 36849
{xqin, zz0003,xruan}@eng.auburn.edu

Abstract

Cluster storage systems have emerged as high-performance and cost-effective storage infrastructures for large-scale data-intensive applications. Although a large number of cluster storage systems have been implemented, the existing cluster storage systems lack a means to optimize quality of security in dynamically changing environments. We solve this problem by developing a security-aware cache management mechanism (or CaPaS for short) for cluster storage systems. CaPaS aims at achieving high security and desired performance for data-intensive applications running on clusters. CaPaS is used in combination with a security control mechanism that can adapt to changing security requirements and workload conditions, thereby providing high quality of security for cluster storage systems. CaPaS is comprised of a cache partitioning scheme, a response-time estimator, and an adaptive security quality controller. These three components help in increasing quality of security of cluster storage systems while allowing disk requests to be finished before their desired response times. To prove the efficiency of CaPaS, we simulate a cluster storage system into which CaPaS, eight cryptographic, and seven integrity services are integrated. Empirical results show that CaPaS significantly improves overall performance over two baseline strategies by up to 73% (with an average of 52%).

1. Introduction

With the advent of powerful processors, fast interconnects, and low-cost storage systems, clusters of commodity PCs have emerged as a primary and cost-effective infrastructure for large-scale scientific and web-based applications. A large fraction of scientific and web-based applications are parallel and data-intensive, because these applications deal with a large amount of data transferred either between memory and storage systems or among nodes of a cluster via interconnection networks.

An increasing number of data-intensive applications [1] include long running simulations, remote-sensing applications [2], and biological sequence analysis[3], to name just a few. Importantly, cluster storage systems have become increasingly popular for data-intensive applications running on high-performance computing platforms, which assure the effectiveness of the nation's critical infrastructures [4][5]. The concept of cluster storage systems was introduced in conjunction with different domains of applications like Internet data caches [6] and video servers [7]. Cluster storage systems are ideal storage candidates for these data-intensive applications running on high-performance clusters, since the applications need extensive computation coupled with access to diverse data repositories.

Cluster storage systems are an important and essential building block for any high-performance cluster computing infrastructures. In next-generation data grids, some of clusters will be dedicated to data storage systems to support distributed data manipulation functionalities[4]. In the proposed research we will build an infrastructure to conduct research on high-performance cluster storage systems. The specific research issues to be addressed in cluster storage systems are security, scalable disk bandwidth, and high I/O performance. In this study we focus on adaptive security issues in cluster-based architectures for storage systems to support data-intensive computing. This is mainly because clusters make use of inexpensive off-the-shelf PC component to provide high-performance and scalable disk I/O support for data-intensive applications [11]. At the current prices of disks, 100TB of disk storage can be integrated to a cluster for less than $50,000. Cluster storage systems can provide scalable disk bandwidth by stripping and storing large files across a number of cluster nodes. Thus, cluster systems can readily achieve over 1 GB/Sec aggregate bandwidth by distributing large files across multiple nodes. Note that this performance is 33 times higher than that of a single disk system.

[†] The work reported in this paper was supported by the US National Science Foundation under Grants No. CCF-0742187 and No. CNS-0713895, Auburn University under a startup grant, and the Intel Corporation under Grant No. 2005-04-070.

Security services that protect data residing in storage systems from talented intruders are important for data-intensive applications that are security-sensitive in nature[12]. In addition, many data intensive applications require disk requests to be completed within a specified response time [7][12]. Consequently, improving quality of security and guaranteeing desired response times are two major concerns when it comes to the development of cluster storage systems. In this paper, we propose a cache management technique (or CaPaS for short) that aims at achieving high security and desired performance for cluster storage systems. The cache management scheme is used in combination with a security control mechanism that can adapt to changing security requirements and workload conditions, thereby providing high quality of security for disk systems in general and for cluster storage systems in particular. A salient feature of our cache management mechanism that it contains a cache partitioning technique that helps in increasing quality of security of cluster storage systems while allowing disk requests to be finished before their desired response times.

The rest of the paper is organized as follows. Section 2 presents architecture for cluster storage systems. In section 3, we propose the CaPaS cache partitioning strategy for security-aware cluster storage systems. Section 4 analyzes experimental results. Finally, Section 5 concludes the paper with future directions.

2. Architecture of Cluster Storage System

In this study we consider a networked cluster storage system, which consists of a cluster storage system, array of security services, volatile or non-volatile memory, an adaptive security service controller, a security-aware cache partitioning mechanism, and an un-trusted network. The architecture of the networked cluster storage system is depicted in Fig.1. It is worth noting that this system architecture is general enough to accommodate a wide range of storage systems like network attached storage and storage area networks. The security-aware cache partitioning mechanism, the adaptive security service controller, and the security service controller are at the heart of the cluster storage system connected to the unsecured network. The array of security services is developed to protect disk requests issued by clients.

In the architecture of networked cluster storage systems, clients issue disk requests to a cluster storage system through an un-trusted network. The entire cache of the cluster storage system is divided into separate partitions, one for each disk, by a security-aware cache
partitioning mechanism. It is believed that if the caches are separately treated, it is easy to control cache miss sequences. Each partition for a disk will be managed separately using the conventional LRU (Least Recently Used) replacement algorithm. The process of cache partitioning is undertaken in a way to maximize quality of security for all disk requests handled by the storage system. A security service controller selects the most appropriate security service to protect data for each disk request.

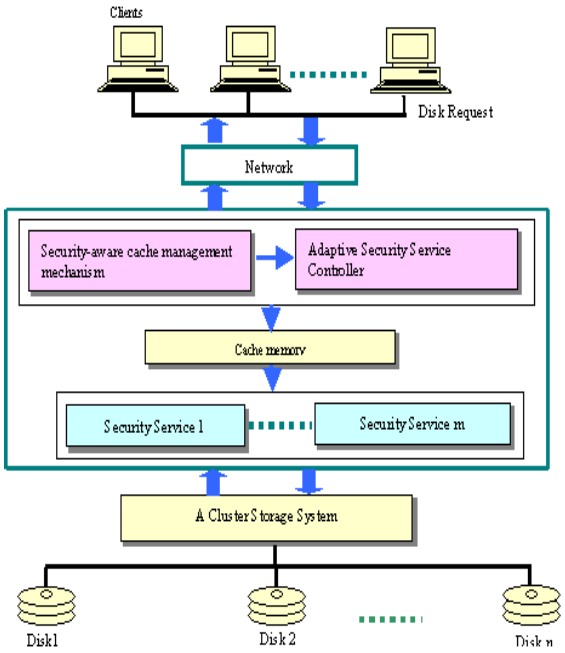

Fig.1. Architecture of networked cluster storage systems.

Note that disk requests issued by clients are read or write requests. It is called a cache hit if a requested data item is found in the caches. When a requested data item is not in the cache, a cache miss occurs. If there is a cache hit for a read request, the requested data item will be retrieved from the cache and returned to the client through the network interconnection. If it is a cache miss of the read request, the requested data item must be retrieved from the disk. We make use of the write back policy when writing to the cache. Thus, all the information is written to the cache in the first place. The modified data items residing in the cache are written to the cluster storage system only when they are replaced.

Cluster storage systems can exploit I/O parallelism in two possible ways: inter-request (inter-operation) and intra-request (intra-operation) parallelism. Inter-request can be achieved if independent requests can be

served by the disk system at the same time. The intra-request allows parallel execution by multiple disks for a single disk request. Our cluster storage system architecture can be applied to address the issues of both inter-request and intra-request parallelism.

3. Security-Aware Cache Management

We present in this section the security-aware cache management scheme, which aims at improving quality of security of cluster storage systems with dynamically changing workload conditions. In what follows, we first outline the main idea of the security-aware cache management scheme. Next, we delineate the security-aware cache management algorithm, where dynamic cache partitioning plays an important role.

3.1 Cache Partitioning

The performance of cluster storage systems can be substantially improved by employing a large memory space as a cache to reduce the number of disk accesses. Cache management policies are of importance in cluster storage systems, because cache management can judiciously cache data blocks for future accesses. In addition to boosting performance by caching data blocks, cache management mechanism can be geared to improve quality of security by affecting the sequence of disk accesses. Specifically, a cache management scheme can shorten the average response time of disk request, making it possible to increase security overheads to improve security while guaranteeing disk requests' desired response time. In this paper, we focus on a security-aware cache management scheme or CaPaS, which is used in conjunction with the conventional cache replacement algorithm (the Least Recently Used algorithm or LRU). Note that our cache management approach does not limit itself to the LRU policy; rather, our approach can be employed in combination with other replacement policies) with the inclusion property. CaPaS dynamically partitions cache blocks in such a way to maximize quality of security without violating desired response times. The entire cache of the cluster storage system is divided into separate partitions, one for each disk, by a security-aware cache partitioning mechanism. The cache partitioning process is motivated by observations that storage systems workloads are not equally distributed among the disks. If the caches are separately treated, it is easy to control cache miss sequences. As such, each cache partition for a disk is managed separately using the conventional LRU replacement algorithm.

To maximize security of a cluster storage system using an optimal cache partitioning, we have to quantify the quality of security experienced by disk requests processed by each disk. Before formalizing the problem, we model a collection of security services required by a disk request r_i as $S_i = (S_i^1, S_i^2, \ldots, S_i^q)$, where S_i^j is a required security level range of the jth security service. The adaptive security service controller is intended to determine an appropriate point s_i in space S_i, i.e., $s_i = (s_i^1, s_i^2, \ldots, s_i^q)$, where $s_i^j \in S_i^j,\ 1 \le j \le q$.

In this study we are concerned with desired response time, which is one of performance requirements. As such, we denote the desired response time of disk request r_i by t_i. We focus on modeling security benefits gained by disk requests with intra-request parallelisms, because disk requests with inter-request parallelisms can be considered as special cases of requests with intra-request parallelisms. Let p_i represent parallelism degree of r_i. The security benefit of request r_i can be expressed as Eq. (1).

$$S(r_i, P_d) = \sum_{j=1}^{p_i} SL(r_{ij}, P_d),\ p_i \le m, P_d \le P, \quad (1)$$

where P is the total cache size, and P_d is the partition size of the dth disk, m is the number of disks in the cluster storage system and SL_{ij} is the security level of a combination of security services chosen for the jth stripe unit of r_i. It is intuitive to show that the parallelism degree p_i can not exceed the total number m of disks in the cluster storage system. The security benefit SL_{ij} (see Eq. 1) experienced by the jth stripe unit of r_i is written as

$$SL(r_{ij}, P_d) = \sum_{k=1}^{q} w_i^k s_{ij}^k (P_d) \text{ where}$$

$$s_{ij} = \left(s_{ij}^1, s_{ij}^2, \ldots, s_{ij}^q\right) s_{ij}^k \in S_i^k,\ 0 \le w_i^k \le 1 \text{ and}$$

$$\sum_{j=1}^{q} w_i^k = 1 \quad (2)$$

where s_{ij}^k is the security level of the kth security service applied to the jth stripe unit of r_i, and w_i^k is the weight of the kth security service for the disk request. Note that users specify in disk requests the weights to reflect relative priorities given to the required security services. Given a sequence $R_d = \left\{r_{1j1}^d, r_{2j2}^d, \cdots, r_{njn}^d\right\}$ of disk requests submitted to the dth disk of the cluster storage system

via the network, we model the security benefits of the disk requests processed by the *d*th disk with Pd partition size as the summation of the security levels of all the requests in R_d. Thus, we have:

$$S(R_d, P_d) = \sum_{i=1}^{n} SL(r_{ij_i}^d, P_d) \quad (3)$$

Now we formalize the following non-linear optimization problem as below

$$\text{Maximize} \sum_{d=1}^{m} S(R_d, P_d) = \sum_{d=1}^{m} \sum_{i=1}^{n} SL(r_{ij_i}^d, P_d)$$

$$\text{Subject a)} \sum_{d=1}^{m} P_d \le P, \text{b)}, \quad (4)$$

$$\text{b)} \ \forall 1 \le i \le n : \max_{1 \le j \le p_i} \{\theta_{ij}\} \le t_i,$$

$$\text{c)} \ \sigma_{ij} \ge s_i \text{ and } p_i \le m$$

where θ_{ij} is the response time for the *j*th stripe unit of request r_i. In an effort to enhance security of the system, we need to guarantee that the following four conditions are met. First, the sum the partition sizes of all the disks must be less than or equal to the total cache size in the cluster storage system. Second, response time of all stripe unit in request r_i must be smaller than the desired response time. Third, the low bound on security level can not be violated. Last but not least, the parallelism degree of r_i has to be smaller than or equal to the number of disks in the system.

3.2 The CaPaS Algorithm

We proposed a novel security-aware cache management algorithm or CaPaS for cluster storage systems. The CaPaS algorithm adaptively determines the most appropriate security levels of security mechanisms applied to each read or write request while guaranteeing the desired response time of the disk request. More specifically, the CaPaS algorithm is carried out in the following three phases: Cache Partitioning (see Section 3.1), response time estimation and quality of security control. Due to the space limitation, response time estimator is not explained in this study.

The pseudo code of the CaPaS algorithm is outlined in Fig. 2. When a disk request r_i is arriving to a cluster storage system, CaPaS inserts the newly arrived request into a waiting queue based on the desired response time policy (see Step 1 in Fig. 2). The first phase is responsible for dividing the cache space into *n* separate partitions (one partition for each disk depending on the workload of each disk). After the cache partitioning phase, CaPaS initializes the security levels of request r_i to the minimal levels (see Step 4 in Fig. 2). If the request is read, CaPaS checks whether the requested data item is residing in the cache. If there is a cache hit for the disk request (i.e., the requested data item can be retrieved from the cache), CaPaS repeatedly increase the security level of the security mechanisms serving the request until the desired response time of the request can not be guaranteed or the security levels of the request are approaching 1.0 (see steps 9-11). If there is a cache miss for the request, the requested data must be accessed from the corresponding disk (see steps 19-20).

Similarly, if there is a cache hit for a write request newly arriving to the storage system, CaPaS adaptively increases the security levels of the selected security mechanisms protecting the write request until its desired response time can not be guaranteed or the security levels exceed the maximal value which is 1.0 in the implementation of CaPaS. In case of a cache miss for a write request, one cached data item must be written back to a disk in accordance with the write back policy (see steps 25-29 in Fig. 2).

4. Experimental Results

To evaluate the performance of the CaPaS strategy in an efficient way, we simulated a networked cluster storage system, in which nine confidentiality and nine integrity services were integrated. Table 1 summarizes important system parameters used to resemble real world cluster storage systems. In addition, we implemented the CaPaS cache management algorithm to optimize the quality of security of the cluster storage system. We evaluate the performance of CaPaS using extensive simulation experiments. To reveal the performance improvements gained by our proposed algorithms. We compare CaPaS with two baselines Alg-rithms, which are briefly described below.

(1) No Cache Partitioning: the cache space of the cluster storage system is not partitioned; all the disks in the system share the same cache.

(2) Uniform Partitioning: the cache space is partitioned equally among the disks.

In our simulation experiments, we made use of the following four metrics to demonstrate the effectiveness of the CaPaS scheme. (1) Satisfied Ratio. It is a fraction of total arrived disk requests that are finished before their desired response times. (2) Average Security Level. It is calculated as a sum of security level values of all disk requests divided by the total number of disk requests. (3) Average Security Overhead. It is the mean time spent in security mechanisms. This metric is measured in seconds. (4)

Overall Performance. It is expressed as a product of the satisfied ratio and the average security level. (5) Cache Miss Ratio (or Miss Ratio for short). It is the number of cache misses divided by the number of cache references.

4.1 Impact of Disk Request Arrival Rate

This set of experiments is aimed at comparing the CaPas strategy with the two baseline schemes with respect to disk request arrival rates.

With various settings of data size , disk bandwidth, and cache size, we study the impacts of arrival rates on cluster storage system performance. To achieve this goal, we increased the arrival rate of disk requests from 0.1 to 0.8 No./Sec. while setting the data size to 300KB, the disk bandwidth to 25MB/Sec., and cache size to 40MB. Fig. 3 shows performance impacts of the confidentiality services. Fig. 3a reveals that CaPaS outperforms the other two algorithms in terms of satisfied ratio significantly. We attribute this improvement to the fact that CaPaS judiciously partitions the system cache space among the multiple disks in the system; therefore, CaPaS allows more disk requests to be finished before their desired response times by lowering down cache miss ratios.

We observe from Figs. 3b and 3c that CaPaS delivers better average security levels and higher security overheads compared with the other two algorithms under a wide range of workload conditions. However, CaPaS always achieves higher average security levels compared with the other two strategies. The results can be explained in part by Figs. 3c and 3d showing that the average security overhead of CaPaS is constantly higher than that of the alternatives. Thus, high security levels of CaPaS are achieved at the cost of high security overheads. Fig. 3d clearly reveals that CaPaS noticeably outperforms the alternative strategies in terms of overall performance. Specifically, CaPaS
obtains an improvement in overall performance over the alternatives by up to 44% for the confidentiality services. The performance improvement can be explained by the fact that CaPaS adaptively enhances security levels of each disk request under the condition that all requests served in the cluster storage system can be finished before their desired response time.

5. Conclusions and Future Work

It is often desirable for next generation cluster storage systems to be highly flexible in order to support varying quality of security at different times during a data-intensive application lifetime. In addition, this trend is especially true for data-intensive applications where disk requests need to be completed within specified response times. As such, high quality of security and guaranteed response times are two major performance goals to be achieved by cluster storage systems. In this paper, we have proposed a security-aware cache management mechanism (or CaPaS for short) for cluster storage systems. CaPaS, which is used in combination with a security control mechanism, can achieve high security and desired performance for cluster storage systems. CaPaS is conducive to providing high quality of security for cluster storage systems by adapting to changing security requirements and workload conditions. Importantly, CaPaS is comprised of a cache partitioning scheme, a response-time estimator, and an adaptive security quality controller. These three components help in increasing quality of security of cluster storage systems while allowing disk requests to be finished before their desired response times. We simulated a cluster storage
system where the CaPaS cache management approach, eight cryptographic, and seven integrity services are implemented. Experimental results demonstratively show that CaPaS significantly improves overall performance over two baseline strategies by up to 73% (with an average of 52%).

References

[1] H. Eom and J.K. Hollingsworth, "Speed vs. Accuracy in Simulation for I/O-Intensive Applications," *Proc. Int'l Symp. Parallel and Distributed Processing Symposium*, pp. 315 – 322, May 2000.

[2] D.B. Trizna, "Microwave and HF Mmulti-Frequency Radars for Dual-Use Coastal Remote Sensing Applications," *Proc. MTS/IEEE OCEANS*, pp. 532 - 537, Sept. 2005.

[3] J. Hawkins and M. Boden, M., "The Applicability of Recurrent Neural Networks for Biological Sequence Analysis," *IEEE/ACM Trans. Computational Biology and Bioinformatics*, vol. 2, no. 3, July-Sept. 2005

[4] J.M. Perez, F. Garcia, J. Carretero, A.Calderon, L.M. Sanchez, "Data Allocation and Load Balancing for Heterogeneous Cluster Storage Systems," *Proc. 3rd Int'l Symp. Cluster Computing and the Grid*, 2003.

[5] P.J., Varman, R.M Verma, "Tight Bounds for Prefetching and Buffer Management Algorithms for Parallel I/O Systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 10, no. 12, pp. 1262 – 1275.

[6] B. Tierney, W. Johnston, H. Herzog, G. Hoo,G. Jin, J. Lee, L. Chen, D. Rotem, "Distributed Parallel Data Storage System: A Scalable Approach to High Speed Image Servers," *Proc. ACM Intl' Conf. Multimedia*,

1994.

[7] T. Iiitamura, Y. Oue, I. Ohnishi, and M. Shimizu, "Dynamic Access Load Balancing on the Parallel Secondary Storage," *Proc. Int'l Symp. Parallel Algorithms/Architecture Synthesis*, 1997.

[8] E. Riedel, M. Kallahalla, and R. Swaminathan, "A Framework for Evaluating Storage System Security," *Proc. the 1st Conf. File and Storage Technologies*, Monterey, CA, Jan. 2002

[9] A.W. Leung and E.L. Miller, "Scalable Security for Large, High Performance Storage Systems," *Proc. 2nd ACM workshop on Storage security and survivability*, pp. 29-40, Alexandria, VA, 2006.

[10] J. Hughes and D. Corcorna, "A Universal Access, Smart-Card-Based, Secure File System," *Linux Showcase*, Oct. 1999.

[11] J. Wu, P. Wyckoff, and D. Panda, "PVFS over InfiniBand: Design and Performance Evaluation," *Proc. Int'l Conf. Parallel Processing*, pp. 125-132, Oct. 2003.

[12] M. Nijim, X. Qin, T. Xie, and M. Alghamdi, "Awards: An Adaptive Write Scheme for Secure Local Disk Systems," *Proc. 25th IEEE Int'l Performance Computing and Communications Conf.*, April 2006.

[13] T. Xie and X. Qin, "Scheduling Security-Critical Real-Time Applications on Clusters," *IEEE Transactions on Computers*, vol. 55, no. 7, pp. 864-879, July 2006.

Parameter	Value
Cache size	128MB
Number of disks	8
Capacity of one disk	184 GB
Average seek time	1136 ms
Average rotation time	837 ms
Blocks revolution per minute	7200 RPM
Transfer rate per disk	30 MB/Sec.
Arrival rate	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8 No./Sec.
Data size	100KB, 150KB, 200KB, 250KB, 300KB, 350KB, 400KB, 450KB
Disk bandwidth	10MB/Sec., 15MB/Sec., 20MB/Sec., 25MB/Sec., 30MB/Sec., 35MB/Sec., 40MB/Sec.

Table 1. Disks parameters in the simulated cluster storage system

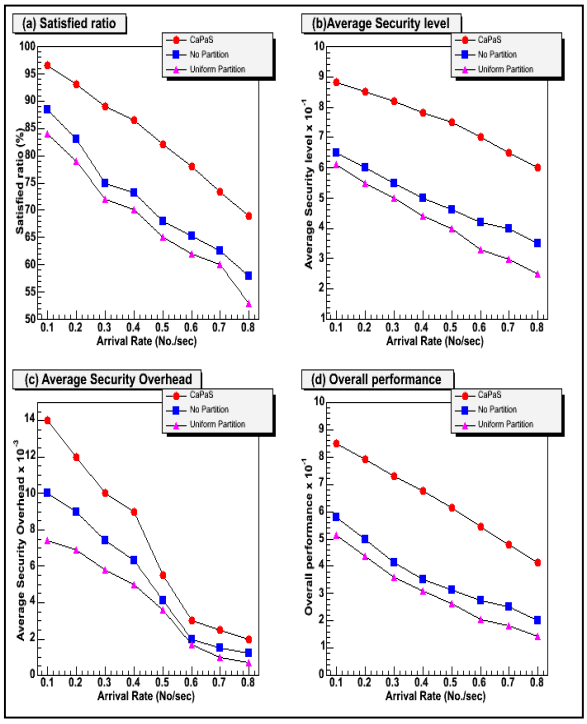

Fig. 3. Performance impact of confidentiality services where data size = 300KB, disk bandwidth = 25MB/Sec. and cache size = 40 MB.

Input: r: a newly arrived disk request
t_i: desired response time of the ith request
s_i: the ith request's lower bound on security level
Q, a waiting queue at the client side

1. Insert r into Q based on the earliest desired response time first policy
/***Phase 1: Cache Partitioning***
2. Partition the cache space based on the disks' workload conditions;
3. **for** each request r_j in the waiting queue Q **do**
/* Phase 2: Response Time Estimation */
4. Initialise the security level σ_{ij} to the value s_i
5. Estimate the response time
6. **if** the request r_j is read
7. **if** the request data is stored in the cache (hit)
/* Phase 3: adaptive security quality control */
8. **while** estimated response time < desired response time t_i **do**
9. **if** σ_{ij} < 0.9 **then /*** σ_{ij} can be further increased */
10. Increase security level σ_{ij} by 0.1;
11. Estimate the response time of the r_j request;
12. **else** break **/*** σ_{ij} can not be further increased */
13. **end while**
14. **if** estimated response time > desired response time t_i **then**
15. Decrease security level σ_{ij} by 0.1; Apply the security service with level σ_{ij} to the rj request
17. Deliver the r_j request through the network subsystem to the client
18. **else** (miss) /* the request ri is not stored in the cache
19. Grab the data from disk i and store it in the cache partition C_i
20. Repeat step 8-17 (Phase 3)
21. **if** the request r_j is write request
22. **if** the request r_j is in the cache (hit)
23. Apply steps 8-16
24. Deliver the data to the disk subsystem
25. **else** (miss)
26. Apply the LRU replacement algorithm to manage the cache
27. Use the write back policy to write the request in the cache
28. Repeat steps 8-16
29. Deliver the request r_j to the disk subsystem
31. **end for**

Fig. 2 The CaPaS Algorithm.