International Journal of Multimedia and Ubiquitous Engineering
Vol.10, No.6 (2015), pp.131-140
http://dx.doi.org/10.14257/ijmue.2015.10.6.13

# A Cache Design of Load Balancing System for Object-Based Storage

Shan Ying[1+], Liu Dan[1] and Nian-min Yao[2]

*1 College of Information and Computer Engineering, Northeast Forestry University, Harbin 150040, China; 2 College of Computer Science and Technology, Harbin Engineering University, Harbin 150001, China*
*shanyingsc@163.com*

***Abstract***

*In design of the object-based storage system, cache design of the server becomes an important guarantee for improving service quality. This paper presents a cache design of load balancing system for object-based storage, which consists of three main aspects. First, the cache update policy is proposed. The strategy considers the overall cache as a pool, consistency and accuracy of updating is ensured through the locking mechanism, while using Bloom Filter combined with the updated time series to achieve updating simplification. Secondly, the cache replacement strategy is introduced which achieving the cache replacement policy using cache spanning tree combined with cache stack. Finally, cache design model is proposed based on energy consumption. Experimental results show that the proposed cache design can improve the efficiency of cache operation in object-based storage system, while reducing energy consumption of cache operation.*

***Keywords:*** *object-based storage system; cache design; updating; replacement; energy consumption*

## 1. Introduction

With the expanding scale of object storage system (OBSS), the need of services is increasing which promote higher performance demanding for the size of the OBSS [1-4].OBSS separates data path, control path and management path in order to provide scalability, high performance, security and data sharing cross platforms of storage service effectively. Due to different customer demand to efficient and accessible of access without regularity, efficient and accurate service is required to provide. However, due to multiple access to MDS and OSD, the efficiency of the service becomes very low, while the unbalanced load also occurs. Therefore, effective caching design of the MDS and OSD is adopted as an effective way to improve access services for OBSS [5-8].

As we know, two types of the main server is responsible for service for the object storage system, one is responsible for providing metadata, namely MDS, the other is responsible for providing storage information, namely OSD [9-12]. In order to improve the difference of efficiency and the performance between the two types of servers and services, a two level structure is adopted, in which the MDS cache in the first-level is called L1 cache, while the OSD cache in the second-level is called L2 cache. Using the two-level caching mechanism, which stored the higher frequency data in the cache and adopted timely and effective management strategies, in order to improve the efficiency of services and reduce the overall efficiency of the MDS and the OSD.

Based on the above analysis, a cache design of load balancing system for object-based storage is proposed. The cache design adopt effective cache scheme through establishing L1 and L2 level cache in MDS and OSD, including updating replacement policy and locking mechanisms of cache, in order to achieve collaborative cache, while

ISSN: 1975-0080 IJMUE
Copyright ⓒ 2015 SERSC

energy consumption cache design is adopted which ensure consistency, isolation, durability and stability of cache operations to the greatest extent to obtain the best performance.

## 2. Related Works

In recent years, there are many research directions on cache design, in which the research for the study of energy consumption and cache collaborative aspects of services has become a hot research and focus.

[13] The two-level structure of the disk cache management and the corresponding management algorithms is proposed to ensure the overall performance of the system under high load conditions. [14] The client and metadata server cache is designed as a whole, while the cooperative caching scheme based on the object's size, access costs and network load, can effectively improve the input-output performance of the system. [15] proposed energy-related storage cache management, offline caching algorithms by proposing energy-related greedy algorithms, online energy-related algorithms, the energy consumption of the disk. [16] proposed a local algorithm and a multi-queue management algorithm with global multi-level buffer cache hierarchy, thereby increasing the actual hit rate of second-level buffer cache.

## 3. Main Ideas of Cache Design

### 3.1. OBSS Architecture

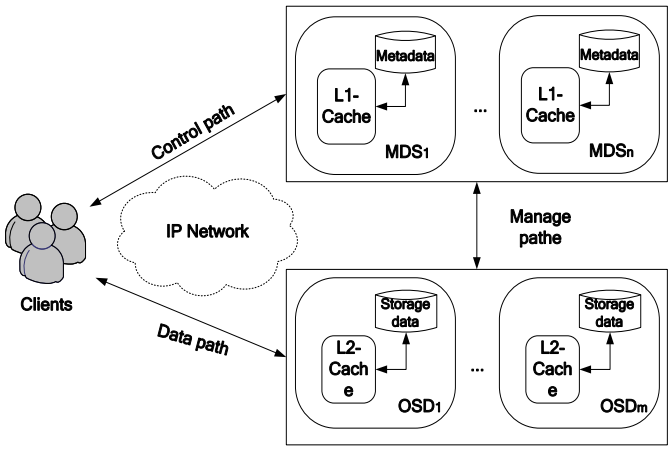


**Figure 1. Architecture Model for OBSS**

The architecture of the model we proposed for OBSS is shown in Figure.1. While the primary parts, relation of the components and the implementation details is described as follows. In this Figure, there are three main components: (1) Clients; (2) OSD (Object Storage Device); (3) MDSs. The three components are attached to the TCP/IP network [4]. OSD provides OSD storage management and data interaction functions. MDS provides three function including data management, database management and storage device management.

### 3.2. Interactive Mode of Data for OBSS

The Interactive model of OBSS has three paths: control path, data path and management path. The three paths are separated.

The Interactive process of OBSS is described as follows:

1. Clients send request to MDS for only once, and then wait for receiving data.

 Copyright ⓒ 2015 SERSC

2. MDS is no longer a transfer station, but a functional entity.
3. Because authorization and authentication of clients are accomplished by MDS, OSD trust MDS which means OSD will not authenticate the requests and commands from MDS.
4. OSD sends response data to client after accomplishing the OSD storage operation.
5. In the whole process, MDS has been a manager of OSD operation.

### 3.3. Cache Updating Scheme

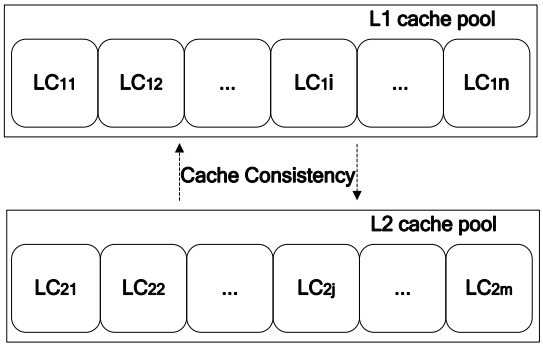


**Figure 2. The Structure of Cache Pool**

Since the update operation of the cache has uncertainty, batch update operation is adopted, namely cumulative updates. Cumulative update periodically updated by frequent operation can reduce the energy consumption caused by updates. Meanwhile, in order to ensure the metadata consistency, the cumulative update operations can be simplified. However, the cumulative update also cause many problems, such as the cumulative update order and the priority issues, which will affect the accuracy of the data in the cache. Therefore, in the update strategy, the locking mechanism is introduced. The operation of the update process is restrict by setting different lock, thus ensuring the accuracy of the data, the mutual coordination between the different locks in order to ensure the stability of the global and local update operation.

Figure.2. shows structure of cache pool for the OBSS. All of cache in MDS construct cache pool, namely L1 cache pool, $LC_1 = \{LC_{11}, LC_{12}, \ldots, LC_{1i}, \ldots, LC_{1n}\}$, all of cache in OSD constitute cache pool, namely L2 cache pool, $LC_2 = \{LC_{21}, LC_{22}, \ldots, LC_{2i}, \ldots, LC_{2m}\}$.

The key issue is the timing of the accumulation update, take the write OSD operation into account to merge the cumulative update and simplification [9,16] is an effective way. Bloom filter matrix storing metadata hash values and unconsolidated update sequence is adopted before Simplification and update operations.

A lock mechanism is adopted during update process. Mutex lock is set during the update operation for the same object, which means multiple update operations to the same object cannot be performed simultaneously on the same time. Update operations are accomplished by update queue, while there is only one update operate effectively at a time. The update exclusive lock is set during the process of consolidation and simplification update operations which writes has the highest priority, while other operations has to wait until the completion of writes operation.

Since the update operations are changed over time, taking number of updates into account only cannot be effectively obtain the equivalent of the update sequence. The algorithm of the literature [14, 16] is improved during the update process, while property update strategy under time sequence is adopted.

Copyright ⓒ 2015 SERSC

**Define 1.** $OB_i$ is expressed as r tuples, $(atr_{i1},\cdots,atr_{ik},\cdots atr_{ir})$, while different sequences behalf different objects, $(ha_{ik},atr_{ik})$ behalf tuples which contains property and hash value, $(atr_{ip},atr_{iq},ctr_i)$ behalf triples of the update sequence, $atr_{ip}\rightarrow atr_{iq}$ behalf a property update operation, $ctr_i$ storage update operations number and serial number, which is a counter.

By definition 1, the same update operation may appear several times during the process of updating, simplification which only using the update sequence number is not effective, but after using the counter, not only the updating maintain the sequence in time, but also prevent multiple update issues arising in the simplification process.

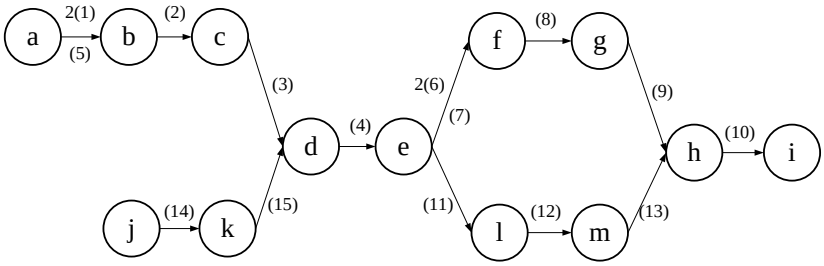


**Figure 3. Updating and Simplification**

Figure 3 shows the updating and simplification process diagram. Symbol in the inner circle represents a update object, the arrows at both ends of sequence represents a updating operation, the updated number of times and the number of the counter object is shown in updates arrow. Updated process is as follows, searching start in order from the beginning of the update object, if the next counter serial number is larger than the current number, then the update remain in effect until it encounters the first object which does not comply this law, while continuous node which meet this rule regarded as a valid update, that is to say, the result of an effective updated is that the start node and end node equivalents the first circle and the last circle for this update, while effective edge is updated and the number of counter is modified. And so forth, until the process diagram does not contain a valid update.

### 3.4. Cache Replacement Strategy

Because of the characteristics of OBSS storage style, L1 cache and L2 cache structure is designed for optimal performance requires. Contents of L1 cache which is regarded as the front end of L1-L2 cache must be consistent with that of L2 cache which is regarded as the back-end storage. In the cache replacement policy, the metadata query process first need to search the nearest MDS for metadata request, if not, then search other MDS of L1 cache pool for reply of the request, while if the entire cache pool is not contain the metadata , then call metadata cache replacement policy.

 Copyright ⓒ 2015 SERSC

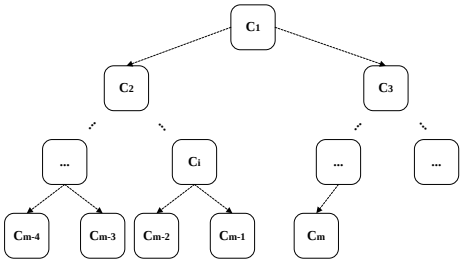


**Figure 4. Cache Pool Topology Structure**

**Define 2.** The cost of the query between different cache $E_{ij}$ can be expressed as

$$E_{ij} = \alpha \cdot cw_i + \beta \cdot cw_j \quad (1)$$

$cw_i$, $cw_j$ is the average of the mean service time delay $L_{1i}$ and network delay $L_{1j}$, $\alpha + \beta = 1$, $\alpha, \beta \in [0,1]$ is adjustment factor.

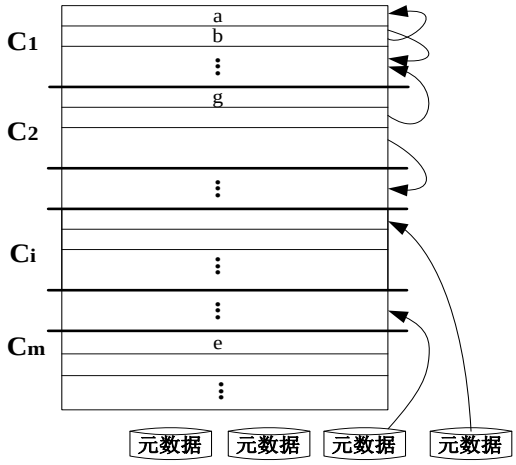


**Figure 5. Cache Pool Stack Structure**

We design collaboration L1 cache pools structure with multiple cache, using B-balanced tree structure as a tree topology between cache of L1 cache pool. The cache which MDS requests is regarded as the root node of the tree topology, namely C1 cache. Topology tree is established based on the query cost between nodes, thus ensuring minimal overall cost of the query.

Figure 4 shows the cache pool topology, the client sends the request to its neighboring MDS, while MDS searches the local cache according to customer request, if the request does not exist in the local cache, the cache topology need to establish topology tree until you find the requested metadata information. If not, the metadata information will be updated.

The topology tree structure is corresponding to a stack buffer pool, the structure determines the replacement policy of cache data.

Figure 5 shows the cache pool stack structure in Figure.4., corresponding to a stack structure of the buffer pool, while the stack is set in the order of the tree hierarchy traversal according to the topology which arranged from top to bottom, *i.e.*, a collection of stack cache. The metadata information stored inside the stack top-down are according to the principle of metadata most recently accessed objects. Adjacent cache stack can be exchanged between the accessed cached data, the data from the

Copyright ⓒ 2015 SERSC

mobile to the arrangement of the principles remain as LRU principle. If metadata which the client requests is not contained in L1 cache pool, then the metadata storage device call the appropriate metadata information stored in the local cache MDS, while metadata information in the bottom of the stack is moved to an adjacent metadata information stack cache. Cache replacement algorithm shown in Table 1.

Input: A collection of CS, request metadata d

Output: Cache stack set after replacement CSA

### 3.5. Evaluation of Energy-Consuming Cache

For above discussion of the update server cache design and replacement policy, we proposed energy consumption in cache design which evaluate energy cost.

Cache in different update and replace operations will cause energy-consuming. The aim of our cache design is for purposes of reducing the energy loss in cache operations.

**Table 1. Cache Replacement Algorithm**

| RepA (CS,d) |
|---|
| 1 if $d$ is in local cache |
| 2 then get the metadata from local cache |
| 3 return success mark |
| 4 else |
| 5 search other cache according to topology structure of CS |
| 6 if the metadata in other cache |
| 7 then store the metadata on an adjacent up level cache, while the metadata on an adjacent up level cache is moved to current cache |
| 8 And the metadata information is returned to service MDS which receives requests |
| 9 return the cache stack collection CSA after replacing |

**Definition 3.** MDS server for collection $MS = \{MDS_1, \cdots, MDS_i, \cdots, MDS_r\}$, $r = |MS|$. Energy-consuming cache operations is measured by time-consuming, namely the cache update process energy consumption and cache replacement process of energy consumption. Energy consumption of cache management is regarded as mark for performance evaluation of OBSS, which is reflected by the average response time for requesting services indirectly.

## 4. Performance Evaluation

In our experiment, we employ a simulation to evaluate the proposed architecture. We first introduce experimental settings. We used NS-2 as the network simulation tool and disksim 4.0 as tool to realize the OSD nodes.

 Copyright ⓒ 2015 SERSC

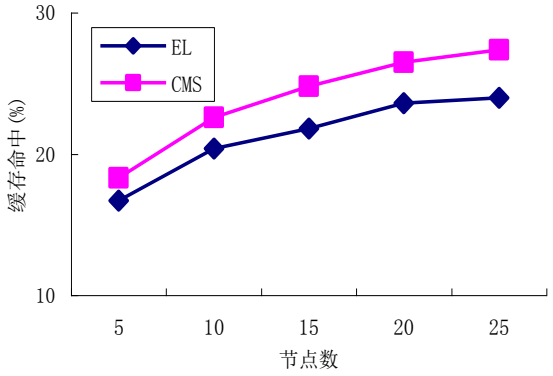


**Figure.6. Cache Hit of Different Node Number**

Experiments were performed to test different ways. First, when testing different nodes in this article (referred to as CMS) for cache hit situation, secondly, testing different situations nodes for the average response time. Again, testing the cache size of the cache hit situations under different circumstances, and finally, testing cache sizes of average response time under different circumstances. CMS used in the experiment with the use of replacement LRU algorithm based on edge updating program (referred to as EL) for comparison.

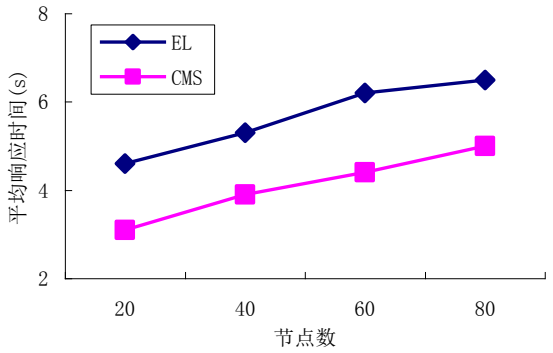


**Figure 7. Average Response Time of Different Node Number**

### 4.1. Nodes Number Scale Simulation

With the increase number of nodes increased in CMS, performance will change accordingly. For this situation, we tested the changes hit rate while the nodes numbers were 5, 10, 15, 20 and 25. The result in Figure 6 shows when nodes number is little, EL and the CMS cache hit has little difference, with the increasing number of nodes, the difference cache hit is increasing, while growth of EL and CMS in cache hits is constantly on the increase. Overall, CMS has been greater than EL cache hit. As we can see from the experimental results, when number of nodes increases, the complexity of the advantages of CMS are apparent.

The average response time is tested with the number of different nodes in Figure 7, The Figure shows the impact of average response time in CMS and EL with the nodes number increasing. As the nodes number in the same circumstances, CMS obtains less average response time than EL. As the number of nodes increased, the average response time is increasing, while the average response time of EL is also constantly increasing. Overall, CMS has less average response time than that of EL.

Copyright ⓒ 2015 SERSC

From the experimental results, when number of nodes increases, performance of CMS affected by the number of nodes is not obvious.

### 4.2. Cache Size Scale Simulation

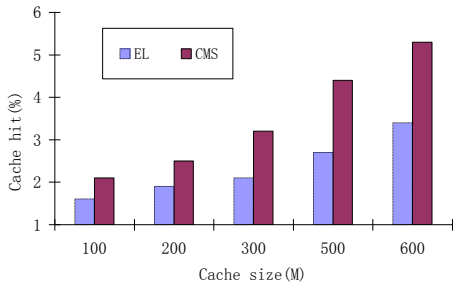


**Figure.8. Cache Hit of Different Cache Size**

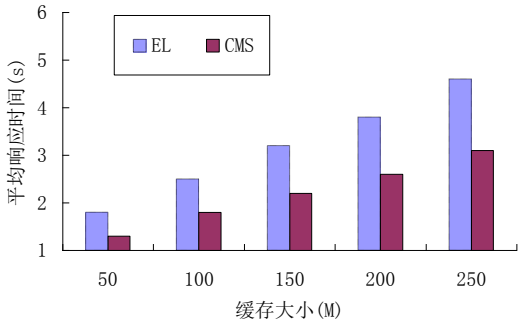


**Figure.9. Average Response Time of Different Cache Size**

As for the different of nodes, different cache sizes are tested. To make test results more clearly, we use the cache size which is not identical to simulate service performance. We compare our strategy (CMS for short) with EL in order to evaluate the impact of different cache size on system performance over time.

In our tests, as taking into account performance of different cache size, we adopted cache hit of cache size as mark. From the result of our tests in Figure.8, CMS continues to show its better than EL with the increasing cache size. From the figure, the trend curves shows that the growth rate of CMS shows faster than that of the EL. Judging from the overall trend, CMS cache hit growth rate is always greater than that of EL, it can also be seen from the figure, with the request arrival rate increases, our scheme has more influence on growth rate of CMS than EL.

The average response time for experimental testing is different for cache size. With the increase in the number of cache size, the average response time will change accordingly. For this problem, we tested the changes of average response time while the cache sizes were 50, 100, 150, 200 and 250. Seen from Figure.9, when cache size of EL and CMS is small, the difference in average response time is not obvious, as the cache size increased, the average response time is increasing, and at the same time, EL growth in the average response time is constantly increases. Overall, CMS has a less average response time than that of EL. From the Figure, the trend curves shows that the growth rate of average response time shows slow growth than that of EL, with the increasing of cache size, the increasing of has little influence on CMS relatively. It can be seen that CMS changes are relatively stable with the increasing of cache size.


Copyright ⓒ 2015 SERSC

## 5. Conclusion

This paper presents a cache design of load balance system for object-based storage, our design takes several aspects. First, the proposed cache update strategy. The policy will be fully considered as a cache pool, ensuring the consistency and accuracy of the update by locking mechanism, while using Bloom Filter and updating time-series to accomplish simplified updates. Secondly, the proposed cache replacement policy, combination of cache spanning and stack structure to achieve cache replacement policy. Finally, the proposed cache design model take energy consumption into account. The cache scheme become an effective way to promote performance of OBSS and reduces energy consumption.

In our immediate future work, we focused on the energy issues of cache for OBSS, consider how to minimum the energy in the cache optimization according to the distribution of the load, thereby using the server cache design to achieve efficiency and improve service performance.

## Acknowledgments

This work is supported by the Fundamental Research Funds for the Central Universities No.DL13BBX03 and Fundamental Research Funds for the Central Universities: HEUCFT1202, HEUCF100609.

## References

[1] Mesnier M., Ganger G. R. and Riedel E., "Object-Based Storage," IEEE Communications Magazine, vol. 41 no. 8, **(2003)**, pp. 84-90.
[2] A. Azagury, V. Dreizin, M. Factor, E. Henis, D. Naor, N. Rinetzky, O. Rodeh, J. Satran, A. Tavory and L. Yerushalmi, "Towards an object store," Proc. 20th IEEE/11th NASA Goddard Conf. Mass Storage Systems and Technologies, 2003(MSST 2003), April 7-10, **(2003)**, pp. 165–176.
[3] M. M. Factor, K. Meth and D. Naor, "Object Storage: The Future Building Block for Storage Systems," Proceedings of the 2nd International IEEE Symposium on Mass Storage Systems and Technologies. **(2005)**, pp. 119-123.
[4] David H. C., "Du and Recent. Advancements and Future Challenges of Storage Systems," Proceedings of the IEEE, vol. 96 no. 11, November **(2008)**, pp. 1875–1886.
[5] T. Gonzalez, "Clustering to minimize the maximum inter-cluster distance," Theoretical Computer Science, vol. 38, **(1985)**, pp. 293-306.
[6] BharadWaj V., "A Weight-based Metadata Management Strategy for Petabyte-scale Object Storage Systems," Proceedings of the fourth international workshop on Storage Network Architecture and Parallel I/Os, **(2007)**, pp. 99-106.
[7] Scott A. B., Ethan L. M., Darrell D. E. L., Xue L., "Efficient Metadata Management in Large Distributed Storage Systems," in Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage System and Technologies (MSS'03), **(2003)**, pp. 290-298.
[8] Liu Q., Feng D. and Wang F., "Research on Metadata Server of High Reliability," Computer Engineering, vol. 34 no.17, **(2008)**, pp. 88-90.
[9] Yifeng Z., Hong J., Jun W. and Feng X., "HBA: Distributrd Metadata Management For Large Cluster-based Storage Systems," IEEE Transactions on Parallel and Distributed Systems, vol. 19 no.6, **(2008)**, pp. 750-762.
[10] Yu H., Yifeng Z., Hong J., Dan F. and Lei T., "Scalable and Adaptive Metadata Management in Ultra Large-scale File System," in proceeding of 2004 IEEE International Conference on Cluster Computing, **(2004)**, pp. 403-410.
[11] Zeng Z. and Bharadwaj V., "On the Design of Distributed Object Placement and Load Balancing Strategies in Large-Scale Networked Multimedia Storage Systems," IEEE Transactions on Knowledge and Data Engineering, vol. 20 no. 3, March **(2008)**, pp. 369–382.
[12] Lin-W. L. and Peter S., "File Assignment in Parallel I/O Systems with Minimal Variance of Service Time," IEEE Transactions on Computers, vol. 49 no. 2, February **(2000)**, pp. 127–140.
[13] Yuanyuan Z., Zhifeng C., and Kai L., "Second-Level Buffer Cache Management," IEEE Transaction on Parallel and Distributed Systems, vol.15 no. 6, **(2004)**, pp. 505-619.
[14] Qin L. J., Feng D., Zeng L. F. and Liu Q., "Dynamic Load Balancing Algorithm in Object-Based Storage System," COMPUTER SCIENCE, vol. 33 no.5, **(2006)**, pp. 88–91.
[15] Qingbo Z. and Yuanyuan Z., "Power-Aware Storage Cache Management," IEEE Transaction on Computers, vol. 54 no. 5, **(2005)**, pp. 587-602.

Copyright ⓒ 2015 SERSC 

[16] Y. Zhu, H. Jiang and J. Wang, "Hierarchical bloom filter arrays (HBA): A novel, scalable metadata management system for large cluster-based storage," in proceeding of 2004 IEEE International Conference on Cluster Computing, **(2004)**, pp. 165–174.

## Authors

**Shan Ying**, she is a lecturer of College of Information and Computer Engineering, Northeast Forestry University. Born in 1981, received Ph.D. degree from College of Computer Science and Technology, Harbin Engineering University. She majors in network storage and cloud storage.

**Liu Dan**, she is a lecturer of College of Information and Computer Engineering, Northeast Forestry University. She received Ph.D. degree from College of Information and Computer Engineering, Northeast Forestry University. She majors in database, Wireless sensor networks.

**Yao Nian-min**, he is a Professor and Ph.D. supervisor of College of Computer Science and Technology, Harbin Engineering University. Born in 1974, member of information storage technology specialty committee in China Computer Federation. He majors in Wireless Sensor Networks and cloud storage system.

 Copyright ⓒ 2015 SERSC