The Research on Cache Management of Cloud Storage

Si Chengxiang
National Computer Network Emergency Response Technical Team, Coordination Center of China (CNCERT/CC)

Abstract

With rapid development of data intensive application technology, such as scientific computing, data analytical and data mining etc, cloud storage with high availability and scalability can satisfy the application demand for the availability, scalability and performance of shared storage. Cache located on data path of shared storage has an important influence on the performance of system. In cloud storage environment based on the opening storage architecture, due to the inadaptability of the cache management, cache located on access paths provides performance acceleration, but limits availability and scalability of cloud storage system. This paper studies previous cache management technology of cluster or distributed system, and puts emphasis on availability and consistency, which bring down availability and scalability of cloud storage system. Moreover, this paper also analyzes the adaptability of previous cache management technology for cloud storage environment. Finally, we point out the future work which we will concern.

1. Introduction

With rapid growth of the information and data scale, computing ability of IT systems are facing challenges. The emergence of cloud computing may solve the problem. Cloud Computing is a technology that uses the internet and central remote servers to maintain data and applications. It provides user hardware, platform, software and storage services. As is known, data analysis includes data computing and data storage, so storage system is very important. With the explosive growth of data scale, cloud computing system provides high demand for capacity, performance, reliability and scalability of storage system. Cloud storage system based on the opening storage architecture is widely noted. It usually employs general hardware and open source software to construct storage system.

Cache is an effective method to improve IO performance of storage system. It employs data access locality to improve performance and brings down access latency of applications. In the environment of cloud storage, cache located on access paths not only improves storage performance, but makes an important effect on availability and scalability of cloud storage system. To be specific, (1) effect on system availability. In order to improve performance, cache data usually is updated to ended storage asynchronously or periodically. Due to asynchronous update between cache and ended storage, if an error about hardware or software causes system crash, cache data will be lost. It may result in data inaccessible and interruption of application service; therefore, the system availability brings down. (2) Effect on system scalability. In the environment of cloud storage, in order to improve handing ability, it usually concurrently accesses the ended storage through several IO paths. As a result, there are several copies in different paths; however, every copy is managed independently, which will result in cache inconsistency for multiply paths. One modified copy is known by local cache, but it is invisible to other copies in other paths, which results in cache inconsistency.

From the above, with the development of cloud computing, it needs high performance, availability and scalability of storage service. In cloud storage environment based on the opening storage architecture, due to the inadaptability of the cache management, cache located on access paths provides performance acceleration, but limits availability and scalability of cloud storage system. This paper studies previous cache management technology of cluster or distributed system, and puts emphasis on availability and consistency, which bring down availability and scalability of cloud storage system. Moreover, this paper also analyzes the adaptability of previous cache management technology for cloud storage environment.

2. Previous researches on cache management of cluster

In this section, we will study previous cache management technology of cluster or distributed system, and pay attention to availability and consistency. In terms of different usage modes, we will divide different cache management into the following types to illustrate.

2.1 cache management of cluster file system

According to the cooperative relation between cache nodes, we divide into two types—cooperative cache and non-cooperative cache.

2.1.1 Cooperative cache

zFS [1] [2] is developed by the IBM Haifa lab. The entire file system consists of clients, file managers, cooperative cache managers, lease managers, transition managers and oriented-object storage devices. It employed all memory of nodes as cache of the entire file system to improve performance. As illustrated in figure 1, when client A accessed the data on the server, there is no the corresponding cache copy, and then the request will be forwarded to the cooperative cache. The cooperative cache found out the cache copy in the client B through some retrieval operations, and then the request will be forwarded to client B. Client B will send the cache copy to client A; therefore, client A needn't to acquire data from storage device. The cooperative cache in zFS can only buffer read data, and cache consistency is guaranteed by the lease and distributed transaction mechanisms.

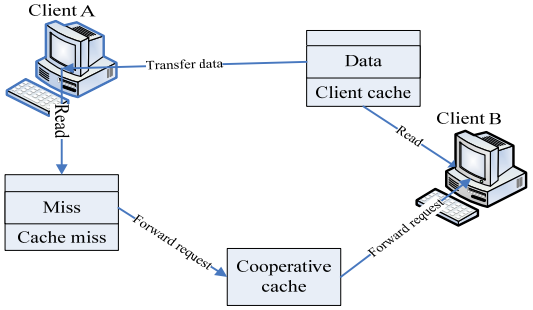

Figure 1. Cooperative cache of zFS

Reference [3] indicated that with the development of networks, employing cache of other clients is feasible for distributed file system. In the reference, remote client memory is regarded as one level of the storage architecture, and client memory, remote client memory, server memory and server storage construct the entire storage architecture. The special manager is responsible for the global cache management and consistency. For the cooperative cache, due to unreliability of write operations, it adopted synchronous-update strategy and only kept read data in cache. In the aspect of cache consistency, it employed the read-write token mechanism, and the central manager is responsible for token allocation and reclamation. When a cache block is modified, it adopted the "write-invalidate" strategy to notify the according clients to invalidate the cache block.

2.1.2 Non-cooperative cache

Storage Tank [4] is a SAN file system designed by IBM. It supports multiple operating systems and its scalability is very high. As illustrated in figure 2, it consists of meta-data clusters, SAN storage network, management server, and client clusters. A client accesses data of storage devices through the SAN storage network. In order to improve system performance, clients of Storage Tank employs client memory as cache. To be noticed, meta-data and data were separated. The separated way enables tiered storage according to importance of data. The cache consistency was guaranteed by the lease and lock mechanisms. If one client needs to modify one local block copy, it must acquire the according lease from the meta-data cluster. In the valid time of a lease, it can modify the block copy. Once a client acquires the lock of a block object, it's no need to communicate with the meta-data server. When a lease is invalid, the client submitted the modified data to the corresponding meta-data cluster. The meta-data server is responsible for copy updates between clients. The lease mechanism effectively reduced cache consistency overhead.

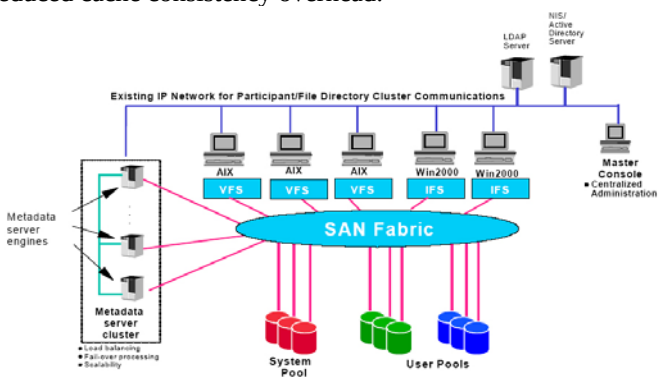

Figure 2. The architecture of storage tank

NFS [5] is widely used in the industry, and has been widely transplanted into all operating systems. The property information of client files or directories is kept in client cache for a certain time. To be noted, the client never keeps file data in its cache. NFS adopts the timeout mechanism to distinguish whether client cache is effective. To be specific, while data staying in cache exceeds a certain time, NFS will refresh the cache data at the following access. The cache reliability of NFS is weak, and the timeout mechanism also doesn't guarantee cache consistency for multiple clients fundamentally.

Reference [6] adopted NVRAM to improve write performance and reliability of the distributed file system. It added NVRAM into client and server sides respectively as temporary cache for write data. Due to the limit of cache consistency, this way is only appropriate for the distributed file system whose write data buffered in clients, such as Sprite and AFS. NVRAM can guarantee performance and reliability of cache effectively, but it needs special hardware which results in weak generality. Moreover, NVRAM is very expensive and low cost performance for a long time. In the aspect of cache consistency, it followed cache consistency protocols of its host file system. For example, Sprite adopted the "concurrent write-sharing" protocol. When

multiple clients open a file concurrently, and one of them modifies the file, the server will notify every client to invalidate its cache.

Clients of coda file system [7][8] adopt local disks as data cache, which can guarantee persistent data access at the case of shared storage failures. When clients fail, due to the non-volatile property of disk medium, data which has been written into cache disks will be reliable. However, data which hasn't been flushed into cache disks won't assure reliability. For the aspect of cache consistency, there are two strategies ---- the positive and the passive. The cache consistency of the former is relatively strong, which prevents concurrent partition writes and limits concurrent read and write for a single partition to solve access conflicts. The latter employs usual lease methods to make cache effective, which assign an upper limit for one lease. When one lease exceeds the limit, clients won't access the according objects.

2.2 cache management of networking applications

This section will introduce cache management strategies of several popular networking applications, such as web, streaming media. Internet is a widely distributed system, and cache technology has been extensively used in network services. There are kinds of cache distributed at access paths from user terminals to network data sources, such as browser, gateway server, the proxy server of the ISP service providers, switches and routers. According to the location of one cache server, there are the following two cases.

2.2.1 Cache cluster for the ended database

As illustrated in figure 3, Oracle Web Cache Cluster [9] provided cache cluster service for upper applications. Every node in the cluster is loosely coupled, which provides a single cache mapping for upper applications. According to the relative cache size of all nodes, Oracle Web Cache Cluster automatically distributed cache content, and every cache object will assigned to one owner node. For example in the figure 2.3, there are three nodes in the cluster. If the relative cache size of node X is 10, the one of node Y is 10, and the one of node Z is 20, the cluster will distribute 25% cache content to node X, 25% to Y, and 50% to Z. For the hot data object, through the way of hot data replications, every node will have one copy. When a node receives one request, which is the owner node of the request data, the node will send the data to upper applications directly. If the node isn't the owner, it will forward the request to the owner node, and the owner node will return the data content to the node. The node will finally return the data content to upper applications. For hot data replicated among several nodes, when one node failed, other members will take over. As a result, upper applications will access the original data, which can guarantee the availability of cache service. For the aspect of cache consistency, when data copy is updated, the owner will send the invalid message about the data copies to other nodes, which will invalidate the according copies. Therefore, the policy of the cache consistency is "Write-Invalidate".

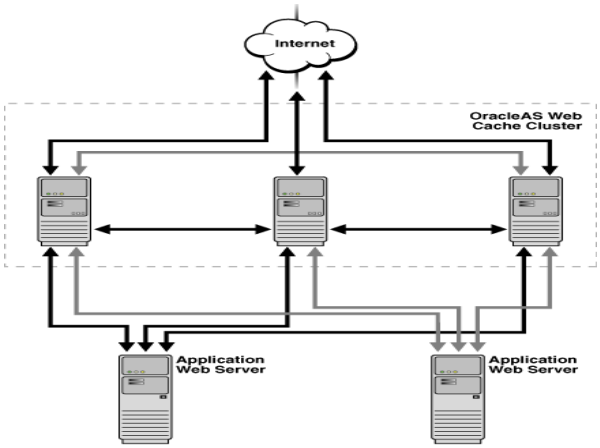

Figure 3. Ended cache cluster of web applications

The type is usually deployed at the gateway or inner of local network. Memcached[10] is a universal distributed cache system, which is used to bring down the load of the ended database. It employed memory of storage cluster to provide cache service for web application nodes. Memcached made fully use of all memory of storage cluster to construct cache objects which are shared and managed centrally. The front web nodes manage all memory of cluster nodes by a global hash table, which can avoid duplication of cache objects in the entire cluster and improve cache utilization of the entire storage cluster system. The goal of Memcached is to improve cache utilization of the entire system without consideration for cache reliability。In addition, every cache object is unique in the system, so cache consistency is no need to be considered. Reference [11] added data persistency mechanism for Memcached to guarantee cache reliability.

2.2.2 Proxy cache cluster

Web Cache is widely used, which is distributed in the entire Internet. The implement of it is usually simple, and its distribution in access paths is similar to the tree. As shown in figure 4, it employed ICP [12] protocol to finish queries for every cache levels. Its basic principle as follows: if a query hit in some level cache, it will return immediately. Otherwise,

the query request will be forwarded to cache nodes at the upper level until the content server. The cooperative work finished by the tree-level proxy cache servers, can reduce the amount of communication with the external network and meet the users' requirements. Since there are multiple paths to reach the web server, a cache node failure can still ensure the availability of caching proxy service. When data of the web server is updated, and the according data in cache is still old, cache inconsistency will appear. How to keep cache consistency has been widely studied. Previous consistency algorithms are divided into two categories: weak consistency mechanism and strong consistency mechanism. Weak consistency technologies include TTL (Time-To-Live), Adaptive TTL and user query. Similar to the timeout mechanism of NFS, TTL is an expired time assigned for a cache copy. If the current time is less than TTL, the cached data is regarded as fresh. Or we regarded the cache copy is gray, namely the copy is old. For strong consistency mechanism, the server will record which client has accessed the data. When the data is modified on the server, it will notify clients, which have cached the according data, to invalidate the cached data. Strong consistency mechanism is relative complicated, and will increase access latency.

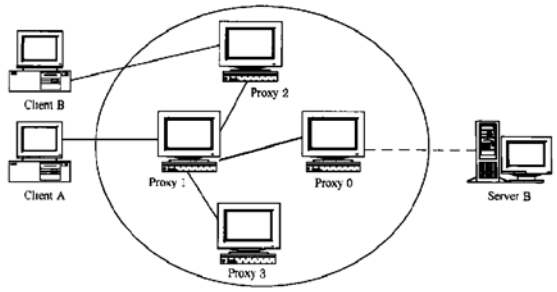

Figure 4. Proxy cache cluster

2.3 memory management of computing cluster

The computing cluster employed idle memory as swap partition to avoid slow disk operations.

[13] illustrated the feasibility of using idle memory between nodes by the analysis of idle memory resource. According to the experimental result, it acquired the following conclusions. (1) There is a large proportion of idle memory for desk-top machine. (2) In a workstation cluster, 60%-68% of memory is idle. (3) For data intensive applications, the idle memory will improve performance largely. (4) Employing remote idle memory is not apparently effect on the host machine. In order to employ remote idle memory, the paper provided a series system call, which was designed for acquiring remote idle memory from central memory manager. What is more, remote idle memory is used for read-only data.

[14] employed remote memory as swap partitions. In order to avoid crashing of the remote node providing remote memory, it adopted the way of parity logging to protect swap-in data. Similar to [13], the paper first analyzed the feasibility of remote idle memory. It implemented the remote swap partition through the way of block devices. It first selected a node with idle memory to employ the idle memory of the node. Once the node is overload, the system will select another node. If there is no appropriate node, it will employ local disk as swap partition. In order to guarantee the reliability of remote swap partitions, it employed the method of parity logging. First, a client computed the parity of S pages, and S is the number of idle nodes. Then, the parity will send to a dedicated parity server. It does not allow rewrite the same page, and only new write is allowed. While the original old pages won't be erased, this makes parity not recalculate and reduces the calculation number of parity. When all data in one parity group is signed as old, it will be erased. It is similar to the mechanism of disk log; moreover, every server keeps multiple versions for the same data, which will occupy a large number of memories.

Similar to [14], [15] also provided remote memory as swap partition. It also employed the mechanism similar to parity logging to avoid the faulty of the swap-in node. The difference is that when the swap-in node is overload, it will forward the swap-out data to other node with idle memory.

According to the above remote swap partition, it can be regarded as the cache of swap partitions. Moreover, due to the uniqueness of cache data, there is no the problem of cache consistency.

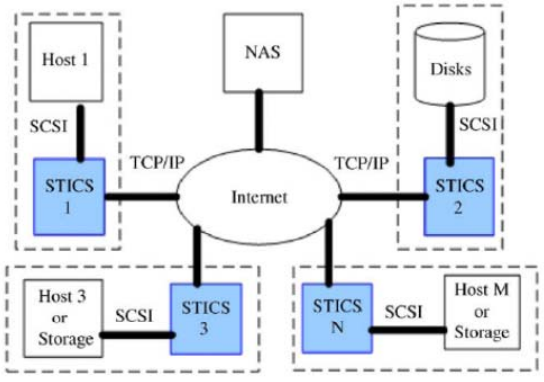

Figure 5. The architecture STICS

2.4 cache management of network storage system

2.4.1 Clients of network storage system

STICS [16] is a cache system for network storage devices, which is designed to finish effective conversion from SCSI protocol to IP protocol. It employed two-level cache to realize the protocol conversion and effectively alleviated performance gap between the two. STICS included two parts---read cache and write cache. The former consisted of a large DRAM, and the latter made up of two-level cache, including a large log disk and a small NVRAM. Modified data is written into the small NVRAM. When the data amount of NVRAM reaches a certain value, the data will be flushed into the large log disk. STICS provide a lock manager for every shared device and maintain a lock status table which record lock status of shared data segment. Moreover, it provides a shared read lock and an exclusive write lock. Therefore, cache consistency of STICS is guaranteed through the lock and the write-invalidated mechanism. The architecture of STICS is shown in figure 5.

2.4.2 Access path of network storage system

Cache located on IO path has a significant impact on the front applications and the ended storage.

SANBoost [17] realized a cache system based solid state disk for a virtual server. Its goal is to solve performance interference between several logic storage volumes and guaranteed quality of service. In order to guarantee cache exclusive from the underlying disk array, it only buffer hot data whose access frequency reached a certain value. From the overall perspective, in fact, SANBoost and disk array cache form the two-level cooperative cache. It employed the distributed lock to synchronize shared access between clients and SANBoost. From the view of cache reliability, solid state disk is persistent, which can ensure that the write cache data is not lost in the event of system failure.

Dm-cache [18] developed by IBM, employed local disk as device cache for network storage system. Its great feature is to customize and manage cache system based on session semantic strategy. Every session is associated with the deployment and execution of every application. The beginning of the session establishes the corresponding cache, and deletes the corresponding cache after the session end. From the view of high availability, if each server has its own independent cache disk, the complexity of data server recovery from failures will be increased. Dm-cache didn't consider cache consistency, and referred to the upper application to solve the problem.

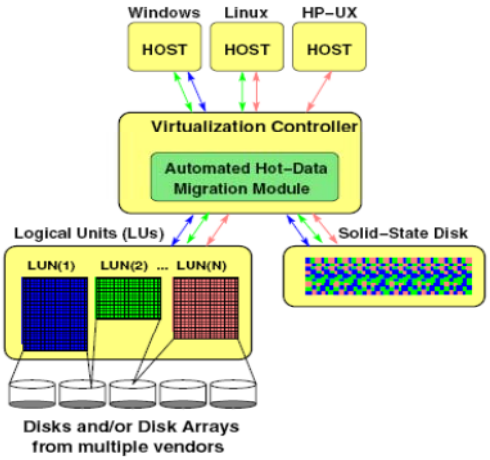

Figure 6. The architecture of SANBoost

ACQ[19] implemented a SAN-level cache in a out-band SAN virtual server. Due to the out-band way, the virtual server is not in data path. It proposed a out-band cache model . In the data path, it added a cache based on solid state disk or disk medium, which could be accessed by all clients. The virtual server is responsible for address mapping and statistics of hot blocks. Through the identification of the hot zone, clients will load the corresponding hot data into the shared cache from the back-end storage. The updated data will first reach the shared cache. Due to the feature of solid state disk or disk medium, when there is some faulty, the reliability will be guaranteed.

3. Summaries

In this section, we summary the related research, and analyze the applicability of the previous technology for cloud storage environment. The summary of related works is shown in Table 1.

3.1 Perspective of availability

As stated above, there are two ways to guarantee cache reliability. ① synchronized update. The data is synchronized to the back-end storage to ensure data consistency, which well supported failover.② the media property. NVRAM or other persistent media can effectively protect the written data, but it can't guarantee continuity of cache service. If data in cache isn't flushed into the ended storage before cache nodes failure, the latest data which is inaccessible may lead to the interruption of business (such as critical data inaccessible). Therefore, the availability of the entire system brings down.

Availability of storage system demands that when parts of the hardware and software failures, the system can still ensure data availability and integrity, and achieve the continuity of applications. Therefore, the redundant technology is employed to eliminate

Table 1. Summary of the related works

Usage	type	Related research	Cache location	medium	Cache object	Method of reliability	Availability	Method of consistency	trait
Cluster file system	Non-cooperative cache	NFS v3	client	memory	file	no	high	Time out	Stateless, simply implement
		Reference [3]	Client/server	NVRAM	file	Property of using medium	low	Concurrent Write-sharing, Write Invalid	High reliability of cache data
		Coda	client	disk	file	Property of using medium	low	Lease, write Invalid	Large cache capacity, high availability
		Storage Tank	client	memory	file	no	low	Lease，Write Invalid	High performance
	Cooperative cache	Reference [6]	client	memory	file	Synchronous update	High	Read-write token、Write Invalid	High cache utilization, high read performance
		zFS	client	memory	file	Synchronous update	high	Lease, distributed transaction	High cache utilization, high read performance
Networking applications	ended database	Memcached	Storage server	memory	Data object	no	no	no	Uniformly managed, high cache utilization
		OracleAs web cache cluster	Cache server	memory	Data object	Multiple data copy	High(multiple paths)	Invalid rules, Write Invalid	Uniformly managed, high cache utilization, load balance
	proxy cache cluster	Web Cache	Web cache proxy	Memory and disk	Data object	Data parity	High(multiple paths)	TTLtime-out,Write Invalid	High flexible
Computing cluster	Remote Pager	[Acharya99]	Computing node	memory	Swap data	no	no	no	High cache utilization
		[Markatos96]	Computing node	memory	Swap data	Parity Logging	High (Parity reconstruction)	No	High cache utilization
		Nswap	Computing node	memory	Swap data	Parity Logging	High (Parity reconstruction)	no	High cache utilization
Network storage system	Client	STICS	SAN client	Memory and disk	Data block	Property of using medium	low	Lock manager、Write Invalid	High cost-effective ratio, large cache capacity
		Dm-cache	SAN client	disk	Data block	Property of using medium	low	Guaranteed by applications	High cost-effective ratio, large cache capacity
	Data path	SANBoost	IO node	Solid state disk	Data block	Property of using medium	low	Distributed locak mechism	High cost-effective ratio, large cache capacity
		ACQ	IO node	Solid state disk	Data block	Property of using medium	low	No	out-band cache managed

single point of failure. When fault happens, failure detection and failover mechanisms will enable work to continue.

3.2 Perspective of consistency

The reason of cache inconsistency is that there are multiple copies for a data block. The technology of cache consistency is to guarantee the consistent view for the same data. There are two ways to maintain cache consistency: write-invalid and write-update. For the former, if one copy has been modified, other consistency, which well supported failover.② the media property. NVRAM or other persistent media can effectively protect the written data, but it can't guarantee continuity of cache service. If data in cache isn't flushed into the ended storage before cache nodes failure, the latest data which is inaccessible may lead to the interruption of business (such as critical data inaccessible). Therefore, the availability of the entire system brings down.

Availability of storage system demands that when parts of the hardware and software failures, the system can still ensure data availability and integrity, and achieve the continuity of applications. Therefore, the redundant technology is employed to eliminate single point of failure. When fault happens, failure detection and failover mechanisms will enable work to continue.

3.2 Perspective of consistency

The reason of cache inconsistency is that there are multiple copies for a data block. The technology of cache consistency is to guarantee the consistent view for the same data. There are two ways to maintain cache consistency: write-invalid and write-update. For the former, if one copy has been modified, other copies will invalidate. When other node demands the according data, it needs to reacquire the latest version from the server. For the latter, the Summary of related works modification for one copy will notify others. Previous studies mainly employ timeout-update or mechanism can guarantee weak consistency, which write-invalidate mechanism to maintain different levels of cache coherency. The timeout-update is suitable for applications with low consistent demand. Write-invalidate mechanism employs the distributed lock to achieve different levels of cache consistency, but semantics of the distributed lock is very complex. The distributed lock mechanism usually needs the lock manager which may be distributed or centralized. The centralized lock manager usually becomes the bottleneck of system performance; therefore, the system scalability brings down. The management complexity of the distributed lock manager is usually high; therefore, the scalability is limited with the increase of system scale.

Aimed at inapplicability of cache management in the environment of cloud storage, this paper studied previous cache management technology. We analyzed design features, management technology, cache protection and cache consistency of several typical cache systems, and summarized the related research from the perspectives of cache availability and consistency respectively. According to the summary, we conclude that previous technologies for cache protection have the shortcomings of low performance, weak support for system availability, cache consistency limiting the scalability of the system. In future work, we will study the efficient cache protection and consistency maintenance techniques for cloud storage environment. On the basis of guaranteeing performance, we will try to enhance availability and scalability of the entire system.

4. References

[1] O. Rodeh, U. Schonfeld, A. Teperman, zFS - A Scalable Distributed File System Using OSD. In proceeding of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies. San Diego, CA, April 2003.

[2] A. Teperman, A. Weit. Improving Performance of a Distributed File System Using OSDs and Cooperative Cache. In Proceeding of the 18th International Conference on Parallel and Distributed Computing Systems. Las Vegas, Nevada, USA, September 2005.

[3] M.D. Dahlin, R. Y. Wang, T. E. Anderson, D. Patterson. Cooperative Caching: Using Remote Client Memory. Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation, Monterey, CA, USA November 1994.

[4] J. Menon, D. A. Pease, R. Rees, L. Duyanovich, B. Hillsberg, IBM Storage Tank - A heterogeneous scalable SAN file system, IBM SYSTEMS JOURNAL, Vol 42, No 2, February 2003.

[5] B. Callaghan, B. Pawlowski, P. Staubach, Sun Microsystems, Inc. NFS Version 3 Protocol Specification, June 1995.

[6] M. Baker, S.Asami, E.Deprit, J.Ousterhout, M. Seltzer. Non-volatile Memory for Fast, Reliable File Systems. In Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems, Boston, MA, USA, October 1992, pp.10~22.

[7] P. J. Braam. The Coda Distributed File System. Linux Journal, June 1998.

[8] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasai, Ellen H. Siegel, and David C. Steere. Coda: A Highly Available File System for a Distributed Workstation Environment, IEEE Transactions on Computers, Vol 39, No 4, April 1990, pp.447-459.

[9]http://download.oracle.com/docs/cd/B14099_19/caching.1012/b14046/clusterconfig.htm#OWCAG010.

[10] B. Fitzpatrick. Distributed caching with Memcached. Linux Journal, August 2004.

[11] Memcachedb, http://memcachedb.org/, 2008.

[12] D.Wessels, K. Claffy. Application of Internet Cache protocol(version2), RFC2187, September 1997.

[13] A. Acharya and S. Setia. Availability and Utility of Idle Memory on Workstation Clusters. In Proceedings of ACM SIGMETRICS Conference on Measuring and Modeling of Computer Systems, New York, NY, USA, May 1999, pp. 35–46.

[14] P. Evangelos, G. Dramitinos. Implementation of a Reliable Remote Memory Pager. In Proc. USENIX 1996 Annual Technical Conference, San Diego, CA, USA, January 1996.

[15] T. Newhall, D. Amato, Alexandr Pshenichkin. Reliable Adaptable Network RAM. In Proceedings of 10th IEEE International Conference on Cluster Computing, Tsukuba, Japan, October 2008.

[16] X. He, M. Zhang, and Q. Yang. STICS: SCSI-to-IP cache for storage area networks. Journal of Parallel and Distributed Computing, Vol. 64, No. 9, September 2004, pp. 1069-1085.

[17] I. Ari, M. Gottwals, D. Henze. SANBoost: automated SAN-level caching in storage area network. In Proceedings of the 1st IEEE International Conference on Autonomic Computing, New York, NY, USA, May 2004, pp. 164 – 171.

[18] E.V. Hensbergen and M. Zhao. Dynamic policy disk caching for storage networking. IBM Technical Report, RC24123, IBM Research Division Austin Research Laboratory, 2006.

[19] D. Xiao, J.Shu, W. Xue, W. Zheng. An Out-of-band Approach to SAN-level Cache Management. 14th NASA Goddard/23rd IEEE Conference on Mass Storage Systems and Technologies, College Park, Maryland, USA, May 2006, , pp.91-96.

[20] The NIST Definition of Cloud Computing . http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

