Electronic Preprint for Journal of Information Processing Vol.20 No.2

Regular Paper

Autonomous L3 Cache Technolgy for High Responsiveness

Hironao Takahashi[1,2,a)] Khalid Mahmood Malik[2,b)] Kinji Mori[1,c)]

Received: May 30, 2011, Accepted: November 7, 2011

Abstract: Web services and cloud computing paradigms have opened up many new vistas. The data intensive cloud applications usually require huge amounts of data to input and output from secondary storage systems. The outstanding progress in area of network communications has enabled high speed networks and therefore, communication latency bottleneck in cloud and other web applications has been shifted to node/storage level. Moreover, existing cloud solutions focused mainly on the efficient utilization of computing resources through virtualization and issues of storage bottleneck did not receive much attention. Moreover, virtualization based implementation ensures equal priority to all hosted applications, thus, real time applications in cloud environment can't meet their requirements. To meet the demand of overall low latency in cloud and other web services; and particularly to reduce I/O bottleneck at storage level, novel idea of autonomous L3 cache technology is proposed. Autonomous L3 cache technology utilizes local memory space as dedicated block device cache for certain specific application, thus prioritizing it over rest of hosted ones. Evaluation shows performance improvement of 5–8 times in terms of timeliness in given setup.

Keywords: autonomous L3 cache technology, low latency

1. Introduction

Cloud Computing is normally considered as collection of virtualized computational and memory resources [1]. Based on this virtualization, the cloud computing paradigm allows tasks to be deployed and scaled-out quickly through the rapid provisioning of virtual machines or physical machines [2]. While virtualization enables a more efficient utilization of existing computing resources, it's the automated self-provisioning aspect that really makes clouds run. More specifically, virtualization mainly focuses on efficient utilization of processors and does not bring into consideration application characteristics for storage management. Therefore, due to data intensive nature of most of today's enterprise and business applications in the cloud environment, it leads to broken business processes and huge revenue loss. Such real-life applications of cloud mostly belong to insurance industry and investment banks [4], [5], [23]. These data intensive cloud applications usually require huge amounts of data to input and output from secondary storage systems [3], [7]. Conventional technologies like Unified Buffer Cache (UBC) and virtualization based implementation of cloud don't consider application characteristics/ prioritization and evenly distribute the traffic to all available resources which results in poor timeliness for data intensive applications. In order to regulate the system resources with respect to requirements of application approaches like centralized load balancer systems in cloud data center can be employed. However, they not only will prove to be single point of failure but also don't guarantee timeliness [6].

To achieve low latency for application requiring high I/O, autonomous L3 cache technique is proposed. To prioritize few applications over others, the L3 cache for more than one application at each node can be created. Each server node in Autonomous Decentralized Multi-Layer Cache (ADMLC) [18], [19] system in cloud data center, makes all decisions and controls all operation autonomously. Moreover, each node in ADMLC system supports two logical data fields to communicate with other two nodes in trio-configuration architecture. Overall, Trio-node based ADMLC architecture not only ensures the timeliness but also the reliability.

The rest of paper is structured as follows: Section 2 discusses the related works while Section 3 describes the basic concepts of the ADMLC. Section 4 narrates the architecture of ADMLC. Section 5 describes autonomous L3 Cache technology in detail while evaluation is outlined in Section 6. Section 7 concludes the paper.

2. Related Works

There have been a number of efforts to improve I/O performance of the memory hierarchy at various levels of abstraction in computing systems. In Ref. [8], the author proposed Unified Buffer Cache (UBC) with the focus to unify the file system and virtual memory data to improve I/O transactions. In Ref. [9] I/O-Lite caching system tries to unify all buffering and caching in the system. UBC and I/O-Lite has their own APIs, which need to be explicitly called by the application requiring high I/Os.

[1] Department of Computer Science, Tokyo Institute of Technology, Meguro, Tokyo 152–8550, Japan
[2] DTS, Inc., Taitou, Tokyo 110–0015, Japan
[a)] Hiro@dts-1.com
[b)] Malik@dts-1.com
[c)] mori@cs.titech.ac.jp

Application-Buffer Cache described in Ref. [10] makes cache at the application-level; and it achieves high I/Os for a particular application that uses "Application-Buffer Cache API".

The concept of Ramdisk for Linux/windows is quite prevalent but it is simply virtual drive on top of local memory and does not involve any cache management technique [11]. In contrast, the proposed platinum cache is a block-layer cache and any application can use it [12]. To gain performance, applications just need to be installed on top of L_3 cache. L_3 cache employs 64 MB internal cache and reserves disjoint memory area from kernel. The Autonomous L_3 Cache Technology employs the concept of Ramdisk with efficient caching algorithm at memory level. This paper uses L_3 cache at level 3 (L_3) for the realization of the proposed autonomous L3 cache node architecture.

The importance of cache on top of HDD has also been recognized somewhat late and the size of the cache on current HDD is about few megabytes [13], [14], [15]. Here the focal point is to improve the performance and power of Windows Vista through built-in cache on HDD. In Ref. [16], requirement for large size cache of the size of few Gigabytes has been emphasized. The current small size cache on HDD does not provide noticeable performance improvement. L_4 cache is a block level cache on HDD. Our focal point is the L3 cache memory computing architecture in autonomous node with the off-chip large size cache with the order of gigabytes.

Modern microprocessors support at least two level caches above the main memory but the gap between CPU and memory hierarchy has been increasing over the past four decades [3], [17]. Reference [3] presents the results for memory hierarchy performance measurement of commercial dual-core desktop processors. The results show the significance of on-chip caches and their impact on the processors' performance. The size of the on-chip and off-chip caches on HDD is relatively very small as compared to the data size of data intensive applications, and cannot meet the I/O requirements. In the above literature survey, the focus is on file level cache or application level cache to improve the performance of specific applications. On-chip multilevel cache improves processor performance mainly. Similar to the concept that processor needs multilevel cache to improve its performance, this paper emphasizes that HDD needs multilevel and overall large size cache to improve its performance. None of the above papers have focused on the optimization of the various parameters at system level such as number of cache levels, large size cache on HDD, and cache size at various levels.

This paper addresses these issues and contributes in following ways 1) Firstly, it shows balanced cache computing system model based on two levels off-chip cache. 2) Secondly, it analyzes the behavior of off-chip cache on top of HDD, and its impact on performance of the computing system. 3) Lastly, it investigates the off-chip cache behavior in the computing system, and presents the performance analysis of autonomous L3 cache node technology. In proposed technology cache is implemented as block device and it lies at the block layer of OS [18]. File system layer works above the block layer, and it receives data request from the applications. File system layer keeps track of data: from which application the read or write request is initiated and whether the request is of read-type or write-type. The file system layer separates the read and write request. After extracting application specific information, the file system layer forwards the request to block layer on which our proposed L3 cache technology works.

3. Multi-Level Cache System: Concepts

Cache system design space optimization requires reduced hit time, miss penalty and miss rate. Multilevel caches reduce miss penalty while large cache size and large block size reduce miss rate [13]. The basic thesis of this paper is to design balanced computing system architecture through multilevel cache, with an ample overall cache storage space to achieve these goals. In the broader context of the L_3/L_4 cache, every low level device is cache for higher-level device, and higher-level cache is mounted on low level device. The cache at level "*i*" enhances the performance of the computing system that can be calculated according to Amdahl's law as follows:

$$G_i = \frac{1}{1 - C_i + \frac{C_i}{X_i}} \prod_{j=1}^{m} L_{i,j} \quad (1)$$

Where
G_i = *Gain due to cache at level i*
C_i = *Cache size ratio* (*hit rate*) *at level i*
$L_{i,j}$ = *overhead factor j, at cache level i,*
X_i = *Cache speed ratio of lower level storage media to higher-level storage media at level i*

For instance if access time for HDD T_{d4}=9 ms, and cache access time at local memory T_{d3}=0.045 ms, then X_3=200. Here value of *i* is 3 to represent cache at level 3. Considering m=2, i.e. j=1, 2 $L_{i,j}$ means that this paper considers two overhead factors, namely cache search overhead and synch data communication overhead. In general Amdahl's law ignores overhead factors but in order to have more realistic model, this paper incorporates these in the system (Eq. (1)). Cache misses are a large cost for modern processors, therefore to improve miss penalty and miss rate, the paper proposes two levels cache in the computing system and overall large cache size improves the hit rate significantly.

Memory hierarchy needs to provide high IO transactions in addition to their capacity and availability requirements in the data-intensive computing systems [10], [11], [13]. Caches at the various levels of the memory hierarchy have been introduced in the computing systems to bridge the gap but latency for the CPU has been increasing for the past four decades [10]. Also performance of caches at the various layers of the memory hierarchy in computing systems does not vary much with its corresponding adjacent lower level of storage devices. Generally large and medium size storage systems consist of disk arrays and associated caches to improve the IO performance. The memory layers consist of hierarchical storage media starting from L1 cache to HDD (**Fig. 1**). Top level caches in the hierarchical memory systems are faster than adjacent lower one but are small in size compared to the adjacent lower level of storage devices. As shown in the Fig. 1, the DTS (Data Transmission System) theme is to bridge data transmission performance gap of the lower layer through multi level caches in the computing systems. DTS cache concept is based on

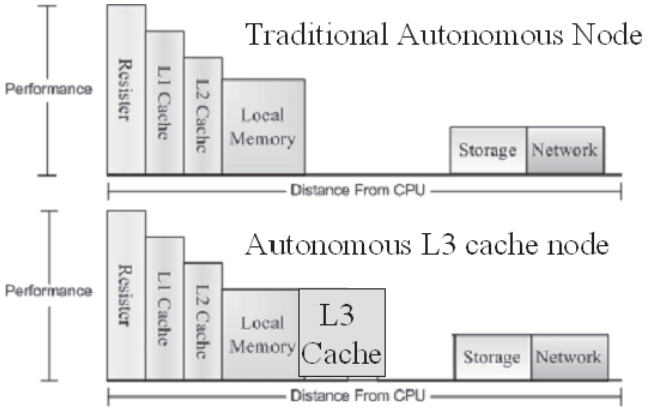


Fig. 1 Concept: L3 cache node vs. Normal node.

layered memory architecture for computing systems [18], [19]. In the broader context of DTS cache, every higher layer is cache for the lower layer in the computing systems (Fig. 1).

This paper proposes a novel concept of exploiting local memory as a block IO device to be used as cache, called DTS cache. The proposed concept uses local memory as managed cache for important data to be maintained for read and write in close proximity to the CPU. Actually part of local memory is managed as level 3 (L3) cache, assuming that there are two levels built-in cache in the system. From Operating System (OS) perspective, the cache device is a transparent layer; therefore it looks like a device mounted on OS. In the context of distributed system, the proposition is to use local memory on each computing system as cache device mounted on remote storage through fast transmission block device protocols.

4. Autonomous Decentralized Multi Layer Cache System Architecture

Autonomous L3 cache node runs on Autonomous Decentralized Multi-Layer Cache System Architecture (ADMLC) [19], [22], that has dual logical data fields and trio node model. The system architecture is shown as **Fig. 2**.

The key feature of ADMLC system is its trio node configuration: each node at least belongs to one group and each group consists of three nodes (Fig. 2). Each node in the system has two storage partitions and each partition of node is mirrored to partition of another node of its group. Each node at data center processes the data to be written on target and thus called P-Node. A P-Node may be attached to another node called C-Node (final target node) whose one or more partitions are dedicated for storage. Of course, it is also possible that storage is also part of P-Node. In that case, P-Node also performs function of C-Node. In both cases, due to trio-node configuration, availability of system is not compromised; because if one of P-Node fails, the alternative node in trio-group have mirrored partitioned, that acts as a backup. It is important to note that, in the proposed configuration, after constant interval C-node shares the updated state with its corresponding node in group by broadcasting the data in Content Data Field (C-DF). Due to its cache write back policy, Autonomous L3 cache node ensures the high I/O response for the write request.

To meet the demand of the high I/O performance, ADMLC has

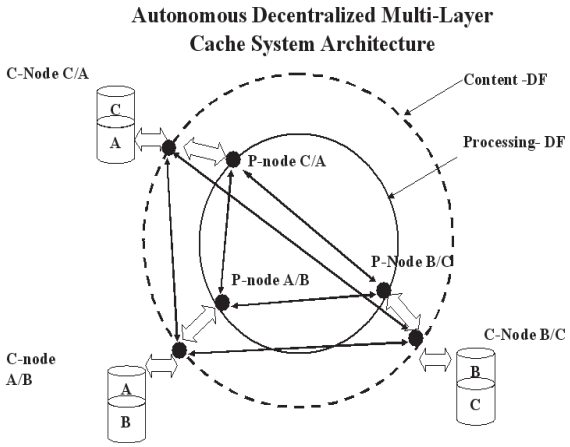


Fig. 2 Autonomous Decentralized Multi-Layer Cache System Architecture.

dual data fields with dedicated function node. The key features of ADMLC can be summarized as follows:

- There are two Data Fields in ADMLC: one is Processing Data Field (P-DF) while the other is a Content Data Field (C-DF).
- Processing node (P-Node) communicates in trio group using P-DF while the Content node (C-Node) uses C-DF.
- Data availability is achieved by dual storage disk partitions of Content Nodes in the trio-group.
- To manage the high demand I/O, dedicated block cache is implemented on node.
- Write speed is achieved by implementing Write Back Cache policy on Processing Node.
- To execute Content Node application program, Processing Node is always required. Therefore, the group creation process is required initially.

5. Autonomous L3 Cache Technology

An Autonomous L3 Cache node has dedicated block I/O cache space in its local memory. Unlike OS local memory and Unified Buffer Cache UBC [13], [20], L3 cache space is storage block address cache and it generated by cache control software with RAM disk] driver [21]. Therefore, L3 cache is not visible from OS: the OS can't reallocate that memory to other programs until RAM disk is un-installed. The main purpose of L3 cache is to ensure timely write I/O t execution using Write Back Cache policy. Each autonomous L3 Cache node mounts two partitions on content node. The write-back L3 cache holds write-data until dirty-data is synchronized on target drive (on content node).

A. Main Operations: Summary

L3 cache technique firstly involves interception of the incoming OS request and breaks it into our block size (currently 8 K) parts. We then write all these parts of the request to L3 cache that is RAMDISK residing in the system's physical memory. This request is marked as PENDING, queued in our internal list, and STATUS_PENDING status is sent to kernel. Note that when write request arrives, P-node implements L3 write back cache with no data flushing policy. However, the other node with mirrored partition in trio-group does not implement write-back policy. The intercepted request is dealt by L3 cache technique in following

way:

Firstly it removes the queued requests from the List and checks for the request type (Read/Write). If the request is read then it calculates the number of blocks involved in the request. Next, for each block it searches the block in INDEX LIST. If it is found then the request is fulfilled. Otherwise, in case of cache miss, we allocate a new block & update the INDEX LIST and fetch the data from target to fulfill the request. Conversely, if the request type is write then we calculate the number of blocks involved in the request. Next, for each block we search the block in Index-List. If it is found then data is written in the block. Otherwise, new block is allocated, Index-List is updated and data is written to the corresponding block.

B. Key Parameters and Functions

The important parameters of L3 cache are as follows:

a) Synch buffer Size Synch buffer size (ψ) is mainly dependent on the difference of speed of L3 and target drive. The bigger the difference between speed of source and target, the significance of larger size of synch buffer increases. Note that, this parameter is useful when write back policy is employed. For write through this parameter is non-effective.

b) Synch Timeout Value Upon expiry of synch timeout (φ) event the dirty data is transferred from our L3 cache to the target drive on content node. That is if constant n seconds elapsed and no new dirty block is added to the dirty list, synching process starts. We employ this parameter to enhance the reliability of data. Note that for write through caching, this parameter is non-effective. Our experimental analysis shows that this parameter affects the read process slightly because occasionally the time based synch may start during the read process.

c) Dirty Transfer Box Percentages These percentages namely red (Upper Watermark) and green (Lower Watermark) of dirty transfer box control the overall operations of L3 cache. We begin our synch process for target drive on red percentage and end it when the green percentage is reached. This parameter is used when the user is using the write back as the writing policy. For write through this parameter is non-effective. The synchronization is a heavy process in terms of processing and it may sometimes result in performance degradation due to handing of the incoming write requests. Note that by synch we mean transferring the dirty block (block on cache only; not written to target yet) from cache to target only. It does not mean that the synched block will be removed from the cache also. The block will be removed according to our flushing policy.

d) Flush Transfer-Box Percentages Similar to the dirty transfer box we have red and green percentages for the flush transfer box. By flushing we mean removing some blocks from the cache to make room for the incoming requests in cache. This parameter is used for both write-through and write-back policies. Flushing the blocks is normally a light weight process which does not involve many operations. A block can be flushed from the cache when it is neither locked nor dirty. The caching algorithm that is used to manage the blocks in cache is important. For this paper we use Least Recently Used (LRU) policy for managing blocks in cache. Normally we set the red and green percentages for this transfer box to be too high in 90 s - say red percentage 95% & green percentage 95% so as to keep most of the blocks in cache to maintain a high hit rate to improve performance of read requests.

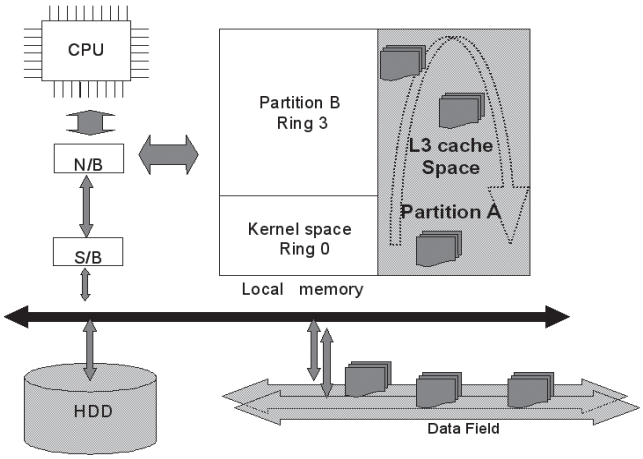


Fig. 3 Autonomous L3 cache node.

e) Pre-Caching in L3 Cache For improving the performance of sequential read cases we implemented the policy of pre-caching in L3 Cache. In this policy we read data of a specific size, depending on the cache size from a specific offset in target. This helps in improving the sequential read of Platinum Cache for a file in proportion to the size of the cache used. The higher the size of the cache, the more data for the file can be pre-loaded from the target to the cache resulting in the better performance when that file is read. If the blocks are not used for reading then we may need to flush out these blocks from the cache. Note this flushing is not a heavy process in terms of processing.

f) L3 Cache Size While it is quite possible to manage creating large L3 cache size from the memory external to the system, this requires restriction of the system memory and a system reboot. Also there might be some conflicts in using this external system memory because this memory is not controlled by the system and it may be possible that the same memory area be used by two different drivers resulting in data corruption. In order to resolve these issues we tried to create a L3 cache from system internal memory. Now the other issue was to increase the size of this internal L3 cache. Using different memory allocation mechanisms we manage to create a L3 cache of about 1 GB from system internal memory. The greater the size of the cache the greater there is a chance of cache hit resulting in better performance for read requests.

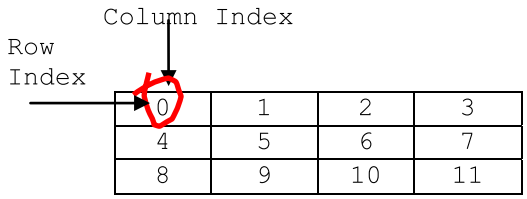
g) Merge block size

Whenever there is request for read data from the target or write data to the target we employ merging technique. For example if during a read process some blocks are missing from cache and if the block size currently is 8 K then instead of generating three to four requests of 8 K each, we make one 32 K read request to the target if possible. Similarly when the synch process is going on, we transfer the data in big chunks like 1 MB etc. The overall structure of Autonomous L3 cache is shown in **Fig. 3**.

C. Key Features

There are three key features of autonomous L3 cache node.

Table 1 Hash based quick search.



a) Firstly, it converts part of local memory into I/O block device cache, and mounts this on Content Node/target device. b) Second important feature is choice of appropriate cache policy as per the application requirement. Since the local memory is transformed to a block I/O device to behave as a cache layer, it ensures high I/O transactions and significantly improves so much so that IO speed is equal to the speed of local memory for the application for which L3 layer is created. c) The third prominent feature of the L3 block cache technology is search algorithm to look for data on the L3 block cache efficiently. A cache search algorithm quickly searches required block of data for I/Os to further reduce the latency of time. Unlike linear searching that searches for each element, the L3 block cache consisting of blocks is divided into groups where each group contains number of blocks as shown as **Table 1**. The jumping algorithm first points to the required group for which that sector corresponds and then finds out the required block of data in the group. If there are total 'n' elements then the cost of linear searching is O(n) while jumping algorithm is a special form of Hash based (Open Hashing Data) searching. The time complexity of hash based jumping algorithm is O(1+n/D) where 'n' is the total number of elements and 'D' is the number of buckets. In best case (n=D) the complexity is O(1). Experimental analysis reveals that on average jumping algorithm enhances 30% search time than standard sequential search technique for sequential requests. If the requests are randomly distributed, then more advantage is expected than standard search and average reaches 60%. Investigation for the value of number of column for block I/O data is carried out. Currently we set the value to be 32, but If CPU performance is higher, then 64 or 128 may be selected as this tradeoff depends upon the CPU specifications. In order to elaborate searching mechanism, consider following sample code:

1. IRP Offset = 0;
2. IRP length = 2,048;
3. NrBlocks = ((offset + NrSectors − 1)>> *(targetDisk ->SectorPerBlock2)) + 1;
4. NrBlocks = ((0 + 4 − 1)>> 3) + 1;
5. Index = (ULONG) ((blockSector.QuadPart) % (targetDisk ->NrBlocks));
6. Index = 1 % 100 = 1;
7. RowIndex = (Index / (targetDisk->index Columns));
8. ColumnIndex = 0 % 32 = 0;
9. Block = _SearchBlock (targetDisk, blockSector, RowIndex, Column Index);
10. Block = _SearchBlock (targetDisk, 0, 0, 0);

Hashing technique is useful for large size data but it is weak for small set of data. Also when the data address has conflict, open address method is used to resolve the issue.

6. Evaluation

This section describes the evaluation of L3 cache technology. We evaluate the system with respect to IOPS, data rate both at single node as well as in network. In future we will extend the evaluation for trio-node based configuration.

A. *Theoretical Evaluation*

The information access behavior follows principle of locality on computing systems and Internet applications such as Web services. Assuming that this principle of information usage follows "standard normal distribution" model, then probability distribution can be considered as standard distribution as follows:

$$f(x) = \left(\frac{1}{\sqrt{2\pi\sigma}}\right)\int \exp\left(\left(\frac{-(x-m)^2}{2}\right)\sigma^2\right)dt$$

Let us consider the value of σ as 1 mainly because present financial based cloud services require high IO demand. Once the behavior of the Autonomous L3 cache reaches stable stage of read operation in L3 cache, the response time will be equal to SDRAM based L3 cache.

In other words following the principle of locality, if majority of the users/requests, say 80% users, will access only 20% of the total data then probability of average existing data in one hour on L3 cache is 20%/24 hours = 0.83% of total storage capacity. Let us consider that the capacity of total mounted storage is 250 GB. In this scenario, necessary L3 cache size is 250 GB * 0.83% = 2.07 GB. Therefore, 2.07 GB data passes on system such as Web server of data center, within one hour. As IO request processing time consists of CPU time plus storage IO access time. Hence if,
Total execution/process time = T (dts)
If local SDRAM memory is L3 cache device, and target storage is HDD, access time of storage is as follows:
T(*SDRAM*) = 45 μs = 0.045 ms
T(*HDD*) = 9 ms (seek time + overhead)
Ratio between them (X) = 200.
Therefore, expected cache hit ratio is (R) = 20%.
According to Amdahl's Law:
T(*dts*) = T(*non cache*)/T(*cache*)
= *Part* T(*non cache*)/(Part T(*non cache*) + Part T(*cache*)/*x*)
= 1/(1 − R + R/x) =>1/(1 − 0.8 + 0.8/200)
= 4.90 times of T(*non cache*)
The expected performance scenario for L3 cache is T(*L3 cache*) = 1.92 ms. Once L3 cache is in stable phase, the IO performance should reach equals SDRAM one:
Ratio of Speed = T(*HDD*)/T(*SDRAM*) = 200.
Therefore, the total ratio of expected speed gain with L3 cache over HDD is 200 times faster IO transactions. This is best-case scenario ignoring different overheads.

B. *Evaluation Parameters for Experimental Setup*

Table 2 highlights values for all parameters used in autonomous L3 cache technology. The values remain same for all experiments unless otherwise stated.

Table 2 Parameters for Experimental Setup.

Parameter	Symbol	Values
Sync Buffer Size	ψ	1 MB
Sync timeout	φ	5 sec
Sync (S)and flush (F) Transfer Box percentages Red (Upper Watermark)	S_R, F_R	70% 70%
Sync (S)and flush (F) Transfer Box percentages Green (Lower Watermark)	S_G, F_G	70% 0%
L3 Cache Size	ρ	1 GB

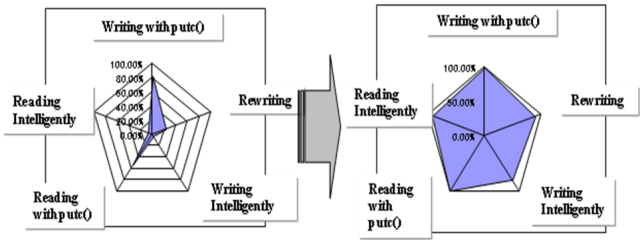


Fig. 4 Autonomous L3 cache node Bonnie benchmark IO performance evaluation results.

C. CPU Utilization

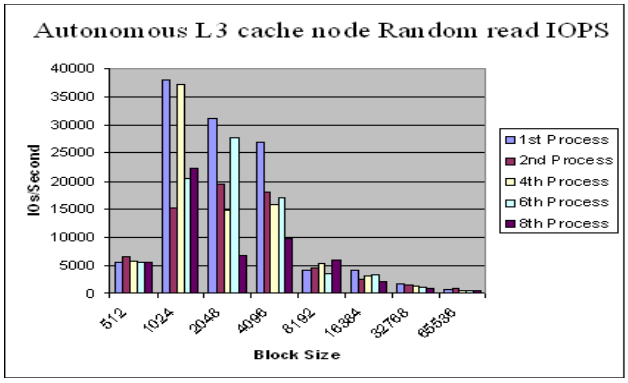
Autonomous L3 cache node evaluation has been carried out on Linux Kernel 2.4.25, Redhat-9 based server by Bonnie benchmark test program. In this test, standard HP Blade CPU with Intel Pentium CPU 2.7 GHz, 4 GB local memory, 1 Gbps Ethernet network and 2 Gbps FC storage system has been used. In this setup, 2 GB local memory is used as Autonomous L3 cache and it is mounted on Fiber Channel RAID storage system as target storage device. The results of the experiments using Bonnie benchmark tool are shown in **Fig. 4**. The experiment has been carried out using Bonnie benchmark. Bonnie command is first executed for the Normal autonomous node and then for autonomous L3 cache node:

#./Bonnie −d /home −s <data size

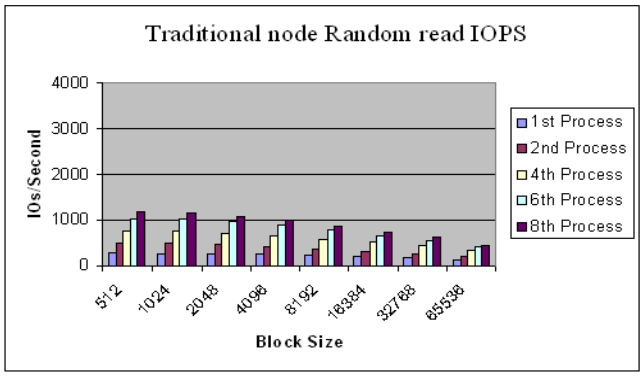
The results show how to manage local memory as effective cache for computing systems. Normal node memory is unified memory space, which is managed by OS. It does not keep particular IO block data in managed way on this memory area even when memory capacity is available. However Autonomous L3 cache node transforms local memory into block device to be used as cache, it enhances IO request on the local memory space and introduces efficiency for both read and write operations. It is observed that write back local memory cache technology is effective to enhance CPU utilization. This means, computer resources can be used more effectively in highly IO intensive applications. The behavior of cache exhibits locality of reference, therefore if block data is larger than the cache size, the performance decreases. The most likely size of data is very sharp normal distribution profile, where most of the access hit is in the L3 cache.

D. IO Performance on single node

In this section evaluation has been carried out using IOmeter with the setup consisting of Linux RedHat AS EL 3, Write Back policy, 2.5 GB L3 cache and 1.5 GB OS memory. The experiment has been performed for various IO data size (blocks) for Autonomous L3 cache node and Normal node.



a: Autonomous L3 cache node IOPS



b: Normal node IOPS

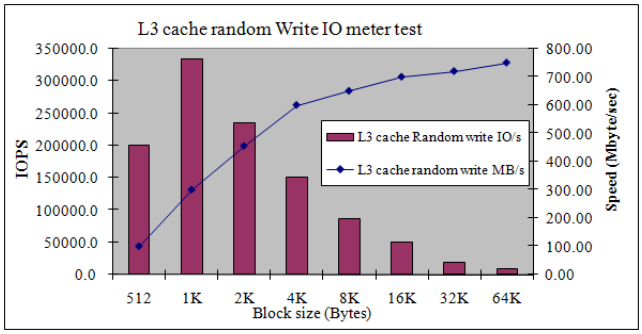
Fig. 5 IOPS experimental performance evaluation.

Initially, the performance in **Fig. 5** (a) and Fig. 5 (b) show the effect of the OS and device characteristics. Device characteristics means when there is no cache hit the read Performance solely depends on the device characteristics provided by the device manufacturer since no effect is visible at this point. The effect of our L3 cache is most appealing for the case 1,024 block of data so much so that it performs 100 times faster IO random read as compared to Normal node IOs using conventional HDD (Fig. 5 (b)). The results reveal that IOPS are inversely proportional to the block size of the test. As the block size of the test increases, fewer requests of that block size can be fulfilled resulting in the deterioration of IOPS. However, the results for 512 block size is not following the above principle due to built in behavior of Unified Buffer Cache (UBC) in Linux OS.

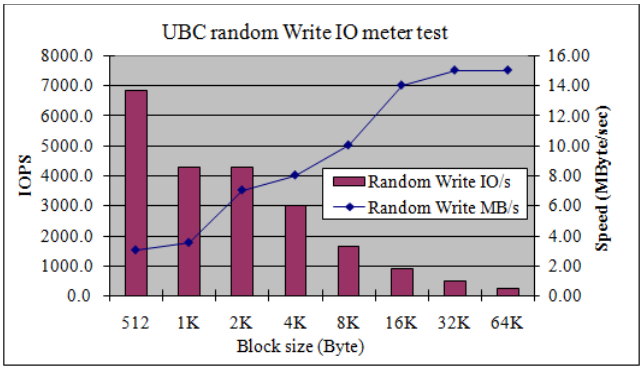
For further analysis, the test was performed for random write case. **Figure 6** compares the IOPS and data transfer rate for the case of random write for both L3 cache node and normal node. Performance analysis clearly depicts that L3cache performs almost 7–8 times better than normal node.

E. Network performance evaluation

In this section we evaluate the performance, in terms of data rate/network speed, considering trio configuration of autonomous L3 cache nodes. The nodes are connected using NFS. The performance of the cache is checked using different block sizes. **Figure 7** shows the maximum transfer size from 4 KB to 128 MB. Transfer performance is measured from the time taken by the packet to reach from source L3 cache node to the destined L3 cache node. In case of relatively small packets there is no big



(a)



(b)

Fig. 6 Data transfer rate and IOPS experimental performance evaluation.

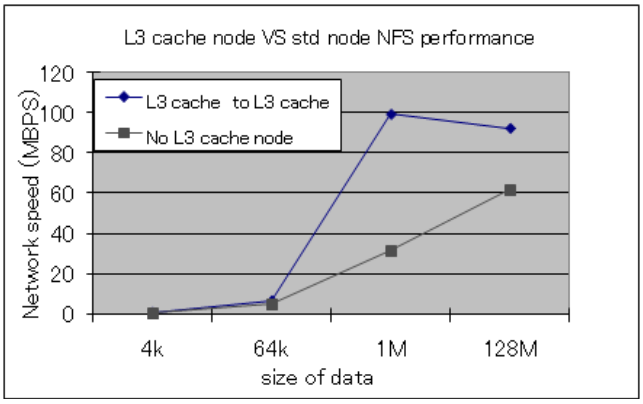


Fig. 7 NFS network transfer rate evaluation.

impact on the overall performance. For example at 4 KB, the network speed is 1 MBPS with L3 cache node to L3 cache node while it is 0.479 MBPS with no L3 cache node scenario. Likewise at 64 K, the network speed is 6.8 MBPS with L3 cache node to L3 cache node while it is 4.9 MBPS with no L3 cache node situation. With large file transfers the performance difference is very much apparent. For example at 1 MB data size, the network speed is 99.25 MBPS with L3 cache node to L3 cache node while it is 31.3 MBPS with no L3 cache node situation. The visible performance difference here is about three times.

In second experiment, 701 MB file was transferred 4 times and data rate was measured in each case. **Figure 8** shows the result. It is obvious that, in case of L3cache trio configuration, performance is continuously improving due to caching effect. Also, in all four cases, data transfer speed between two nodes using L3 cache is significantly better than data transfer speed between two non-L3 cache nodes. In case of file transfer between

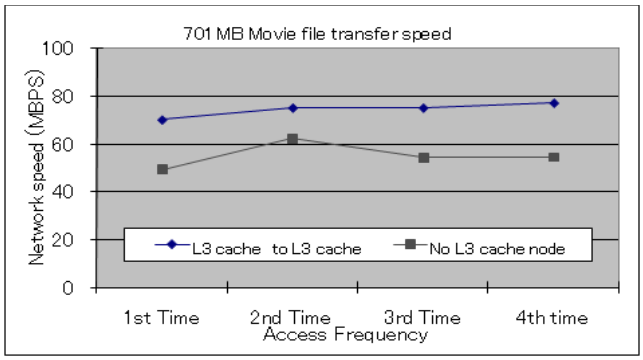


Fig. 8 Autonomous L3 cache node transfer speed vs. Normal node.

L3 cache nodes the data rate varied between 70 MBPS to almost 80 MBPS; whereas in non-L3 cache scenario, it even goes as low as 48 MBPS. The performance of non-L3 cache in 2nd attempt is better than other three cases due to the effect of the OS buffer cache. However, this effect is not only sporadic and is much less effective than L3 cache one. Thus, the L3 cache technology enables to take full advantage of available bandwidth in fast networks.

7. Conclusion

To achieve low latency, data storage systems require very high IOPS for highly responsive cloud applications. This paper proposed autonomous L3 cache node and contributes in following ways: Firstly, it has presented the technique to transform part of local memory into block IO device to be used as L3 cache. Next, Autonomous L3 Cache Node architecture and detailed design including cache management technique, and index based searching to manage data on L3 cache were presented in detail. We have carried out performance evaluation of the proposed technology and the experimental results show significant improvement. The improved IO transactions and network speed optimize utilization of the system resources, such as local memory, CPU and network. In case of networks, the relatively small packets do not show a big difference in overall performance. For example at 4 KB, the network speed is 1 MBPS with L3 cache node to L3 cache node while it is 0.479 MBPS with no L3 cache node situation. Likewise at 64 K, the network speed is 6.8 MBPS with L3 cache node to L3 cache node while it is 4.9 MBPS with no L3 cache node situation. With large file transfers the performance difference is very much clear. For example at 1 MB data size, the network speed is 99.25 MBPS with L3 cache node to L3 cache node while it is 31.3 MBPS with no L3 cache node situation. The visible performance difference here is about three times. Thus, proposed autonomous L3 cache technology shows very good potential for systems requiring low latency.

The focus of this paper was to achieve timeliness while future work will focus on availability/resiliency of the architecture in case of failure of node and further analysis of different overheads.

Reference

[1] Leguizamo, C.P., Kato, S., Kirai, K. and Mori, K.: Autonomous Decentralized Database System for Assurance in Heterogeneous e-Business, *Proc. COMPSAC*, pp.589–595, IEEE (May 2000).
[2] IDC: The Diverse and Exploding Digital Universe, white paper (Mar.

2008), available from ⟨www.emc.com/digital_universe⟩.
[3] Kouzes, R.T., Anderson, G.A., Elbert, S.T., Gorton, I. and Gracio, D.K.: The Changing Paradigm of Data-Intensive Computing, *Computer*, pp.26–34 (Jan. 2009).
[4] Mori, K. and Shiibashi, A.: Trend of Autonomous Decntralized System Technologies and Their Application in IC Card Ticket System, *IEICE Trans.*, Vol.E92-B, No.2 (Feb. 2009).
[5] Ahmad, H.F. and Mori, K.: Autonomous Information Service System: Basic Concept for Evaluation, *IEICE Trans. on Fundamentals of Electronics and Computer Sciences*, Vol.E83-A, No.11, pp.2228–2235 (2000).
[6] Mori, K.: Autonomous decentralized systems: Concept, data field architecture and future trends, *Proc. ISADS*, pp.28–34, IEEE (1993).
[7] Wilkins-Diehr, N., Gannon, D., Klimeck, G., Oster, S. and Pamidighantam, S.: TeraGrid Science Gateways and Their Impact on Science, *Computer*, pp.32–41 (Nov. 2008).
[8] Gorton, I., Greenfield, P., Szalay, A. and Williams, R.: Data-Intensive Computing in the 21st Century, *Computer*, pp.30–32 (Apr. 2008).
[9] Chen, Y.-K. and Kung, S.Y.: Trend and Challenge on System-on-a-Chip Designs, *Journal of Signal Processing Systems Archive*, Vol.53, Issue 1-2, pp.217–229 (Nov. 2008).
[10] Varki, E., Merchant, A., Xu, J. and Qiu, X.: Issues and Challenges in the Performance Analysis of Real Disk Arrays, *IEEE Trans. on Parallel and Distributed Systems*, Vol.15, No.6, pp.559–574 (2004).
[11] Peng, L., Peir, J.-K., Prakash, T.K., Staelin, C., Chen, Y.-K. and Koppelman, D.: Memory hierarchy performance measurement of commercial dual-core desktop processors, *Journal of Systems Architecture*, Vol.54, pp.816–828 (2008).
[12] Kobayashi, D., Watanabe, A., Uehara, T. and Yokota, H.: A high-availability software update method for distributed storage systems, *Research Articles, Systems and Computers in Japan*, Vol.37, Issue 10, pp.35–46 (2006).
[13] Silvers, C.: UBC: An efficient Unified I/O and Memory Caching Subsystem for NetBSD, *Proc. FREENIX Track: 2000 USENIX Annual Technical Conference*, San Diego, California, USA (June 2000).
[14] White Paper, Using Real-Time I/O Signature Analysis to Identify Performance Improvement Options for Database Applications, Solid Data Systems Inc. (July 2006), available from ⟨http://www.soliddata.com/pdf/WP_IOSignatures_v2.pdf⟩.
[15] Wade Tuma, Comparisons of Drive Technologies for High-Transactions Databases, Solid Data Systems, Inc. (Aug. 2007), available from ⟨http://www.soliddata.com/pdf/WP_Drive_Comparison_v2.pdf⟩.
[16] Marburger, III, J.H. and Kvamme, E.F.: Leadership Under Challenge: Information Technology R&D in a Competitive World, An Assessment of the Federal Networking and Information Technology R&D Program, President's Council of Advisors on Science and Technology (PCAST) (Aug. 2007).
[17] Bovet, D.P. and Cesati, M.: Understanding the Linux Kernel, O'reilly Press, pp.422–498 (2001).
[18] Takahashi, H., Ahmad, H.F. and Mori, K.: Layered Memory Architecture for High IO Intensive Information Services to Achieve Timeliness, *11th IEEE High Assurance Systems Engineering Symposium* (*HASE 2008*), Nanjing, China, pp.343–349 (Dec. 2008).
[19] Takahashi, H., Ahmad, H.F. and Mori, K.: Balanced Memory Architecture for High I/O Intensive Information Services for Autonomous Decentralized System, *The 9th International Symposium on Autonomous Decentralized Systems* (*ISADS 2009*), Athens, Greece (Mar. 2009).
[20] Pai, V.S., Druschel, P. and Zwaenepoel, W.: IO-Lite: A unified I/O buffering and caching system, *ACM Trans. on Computer Systems* (*TOCS*), Vol.18, No.2, pp.37–66 (2000).
[21] RAMDISK, available from ⟨http://www.vanemery.com/Linux/Ramdisk/ramdisk.html⟩ (accessed 2009-07-20).
[22] Takahashi, H., Ahmad, H.F. and Mori, K.: The Advantage of Block level L4 cache for NAND Flash SSD in Web Application Environment, IEICE Technical Report Web DB forum Nov 2009 in Japan, pp.31–36 (2009).
[23] available from ⟨http://gridgaintech.wordpress.com/2011/08/08/real-time-a-new-era-of-cloud-applications/⟩.

Hironao Takahashi received his M.S. degree in MOT in 2006 Tokyo University of Science and received Ph.D. of Computer Science from Tokyo Institute of Technology in 2010. He is researching High speed I/O system architecture on Autonomous Decentralized System. He also invented Data Transmission System technology and he is holding nine patents of this field. He is a member of IEEE, IEICE, IEE and IPSJ.

Khalid Mahmood Malik received his Ph.D. degree in Computer Science in 2010 from Tokyo Institute of Technology. He is researching Autonomous Decentralized Mobile Network system, Data Transmission System architecture at DTS, Inc. IEEE member.

Kinji Mori received his B.S., M.S. and Ph.D. degrees in Electrical Engineering from Waseda University, Japan in 1969, 1971 and 1974, respectively. From 1974 to 1997 he was in System Development Laboratory, Hitachi, Ltd. In 1997 he joined Tokyo Institute of Technology, Tokyo, Japan as a Professor. His research interests include the distributed computing, the fault tolerant computing and the mobile agent. He is a Fellow of IEEE and IEICE and a member of IPSJ and SICE, Japan.