12

Autonomous Decentralized Multi-Layer Cache System to Low Latency User Push Web Services

Hironao Takahashi[1,2], Khalid Mahmood Malik[2] and Kinji Mori[1]
[1]*Department of Computer Science, Tokyo Institute of Technology Tokyo,*
DTS, Inc 3-39-5 Higashi Ueno Taitou-ku Tokyo,
Japan

1. Introduction

Emerging rich interactive Web services require timeliness and high availability. These applications are usually characterized as high I/O intensive service model such as e-commerce services, medical sciences including healthcare and digital imaging. [1,3]. These applications require continuous operation, non-stop service system and timeliness to achieve high assurance to meet Service Level Agreement (SLA). SLA is the explicit requirement of the Quality of Service (QoS), such as reliability and timeliness. SLA requirement in the emerging applications on the Internet needs Autonomous Decentralized System (ADS) [8] characteristics [2,4].

The usage of Web services on the Internet is increasing exponentially and giving rise to very large online community. Web service community behavior shows power functions (from 2.1 to 4) that is called "small world". Therefore, some web sites are much more popular and hence highly I/O intensive. In these websites, there is considerable response time delay due to increasing demand of user push type I/O request and its data coherence. Faded Information Field (FIF) technology supports pull type of read event access enhancement while Autonomous Decentralized System by different class of nodes by its service level. But Web services require interoperable communication for user push type also. Traditional system does not meet the dynamic demand and it doesn't have Multi layer-cache concept. To enhance the user push type I/O performance, there are two approaches. One is the cache node approach and the other is selecting a high performance device. Each of them has pros & cons. High speed device such as NAND Flash SSD has less capacity and very limited write life cycle time. Proxy cache node effects for read event but it doesn't achieve write event on each node dynamically. Thus the interoperable I/O performance is not enhanced by existing approaches. How to solve these issues by the system architecture is proposed by this paper. First is to achieve timeliness user pushing type I/O performance by using write back cache processing node (P-Node). Second is to maintain online property by trio nodes by dual data field configuration. The third is data availability which is achieved by dividing processing node and content node in two data fields and duplicating data storage partitions. This system architecture has two data fields with trio nodes Autonomous Decentralized

Multi Layer cache system. Processing node (P-Node) has L3 cache and it performs low latency of time response. L3 cache is dedicated block cache memory on P-Node. Because operating system managed memory doesn't hold specific block data of application services inside of the memory [10, 11], there have been a number of efforts to improve I/O performance however these are substantially different from our work. In [12], the author proposes Unified Buffer Cache (UBC). The focus is to unify the file system and virtual memory caches of file data to improve I/O transactions. However it is an unmanaged one level cache that is very much different from the L3 cache. Similarly [13], [14] provide solution for high I/O, based on RAM disk memory [18] and solid-state disk, and is altogether different from L3 block cache [16]. Other side of block cache is L4 cache. It is inside block device in the Content Node (C-Node). Pre-fetching read is performed by C-Node L4 cache. Thus, two different cache nodes manage user pushing type services model with low latency time web service.

Section 1 is introduction and section 2 is Autonomous multi-layer cache system architecture. Section 3 is its evaluation and section 4 concludes our work.

2. Autonomous multi-layer cache system architecture

2.1 System architecture

Autonomous decentralized multi-layer cache system is designed to achieve low latency user push web service. The system architecture is shown as Figure 1.

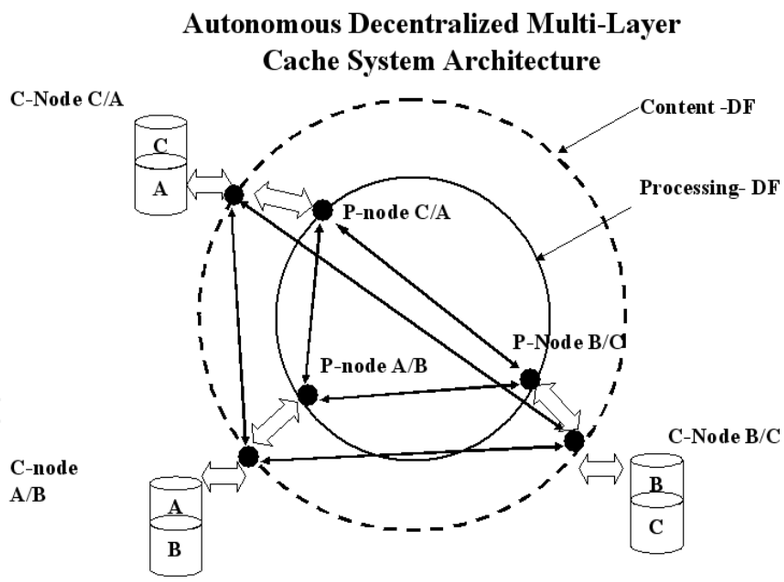


Fig. 1. Autonomous Decentralized Multi-Layer Cache system architecture

The system has following features.

1. Dual Data Fields for Processing and Content.
2. Processing Data Field for Process trio Node group.
3. Content Data Field for Content Trio Node group.
4. P-Node and C-Node are configured Trio Node group on each Data Field.
5. Data availability is achieved by dual storage disk partitions.

6. To achieve the timeliness I/O, the dedicated block cache is implemented on both Nodes.
7. Write event performed by L3 cache on P-Node.
8. Data availability achieved by duplicated storage partitions on each C-Node.

The storage space for all data is at C-node trio group. Each C-Node is connected three of them. To execute C-Node application program, P-Node is always required. Therefore, group creation process is required initially.

2.2 Initialization

Initialization process is shown in Figure-2.

1. C-Node broadcasts its own node status.
2. When other C-Node receives their node status, reply to it with own node status via Content Data Field.
3. Created Trio C-Node group, broadcasts the request of P-Node availability via C-DF.
4. Receive the initialization request from C-Node, P-node reply their node status.
5. C-Node notices P-Node for P-Node trio group creation.
6. P-Node is created by nearest P-Node autonomously.
7. Once P-Node trio group and C-Node trio group is created, P-Node starts to mount C-Node storage partitions.
8. Here, P-Node has two mounting points on C-Node. P-Node mounts one storage partition via L3 cache space and another storage partition is mounted via non L3 cache space.
9. P-Node manages two storage partitions now. The data availability is achieved by the duplicate storage partitions.

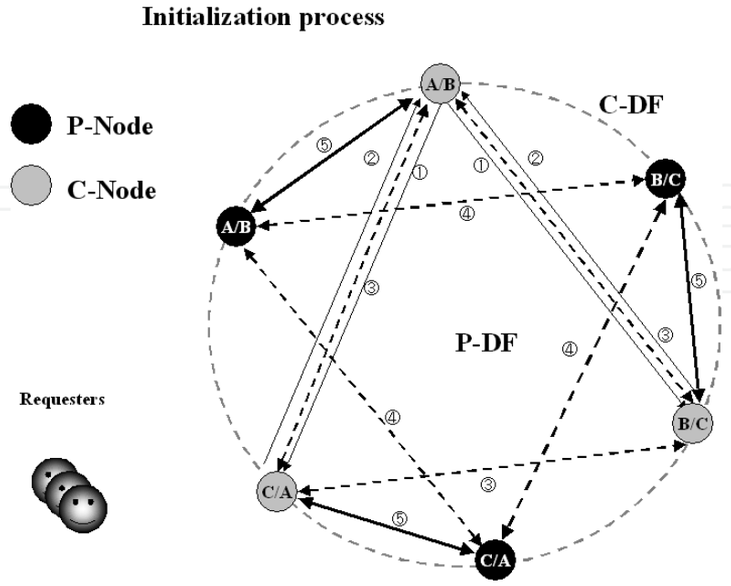


Fig. 2. Initialization process

10. C-Node (A/B), C-Node (B/C) and C-Node(C/A). Therefore, P-Node trio group also has same storage partition as P-Node (A/B), P-node (B/C) and P-Node (C/A).

After mounting of C-Node successfully, application program and service are loaded from C-node. Now P-Node is ready to start web service for C-Node.

2.3 Processing Node (P-Node) autonomous failure sense and replacement process

Processing node (P-Node) is timeliness high I/O response event execution node. To execute timeliness event, P-Node has L3 cache space in its local memory to do one hop write event. Each P-Node in trio group communicates with each other. When, one of P-node fails, other P-node in trio group detects the status. The failure replace process is shown in Figure 3. In this case, P-Node (A/B) is failed and P-Node (B/C) detected this status by the event of co-related storage partition as "B". P-Node (B/C) broadcasted the message for inquiry of available extra P-Node via P-DF (Step-1). There are three P-nodes available and the nearest position P-Node is to be the candidate of new member node. The fastest replier P-Node is selected by this policy (Step-2). P-Node (B/C) and P-Node (C/A) are creating new P-Node connection (Step-3). New P-Node (A/B) starts to mount C-Node (A/B) and load application program from C-Node (A/B). During the creation of new P-Node (A/B), the service for storage partition "A", "B" and "C" are sustaining by P-Node (B/C) and P-Node (C/A). P-Node doesn't have any application program and its data. These data and service program are stored in C-Node. Therefore, P-Node replacement event can be so easy and so quickly for change. The data in L3 cache on failure P-Node (A/B), the possibility of dirty data on storage partition "A". This dirty data "A" is flushed by P-Node (C/A) message. Therefore, write dirty data coherency is protected.

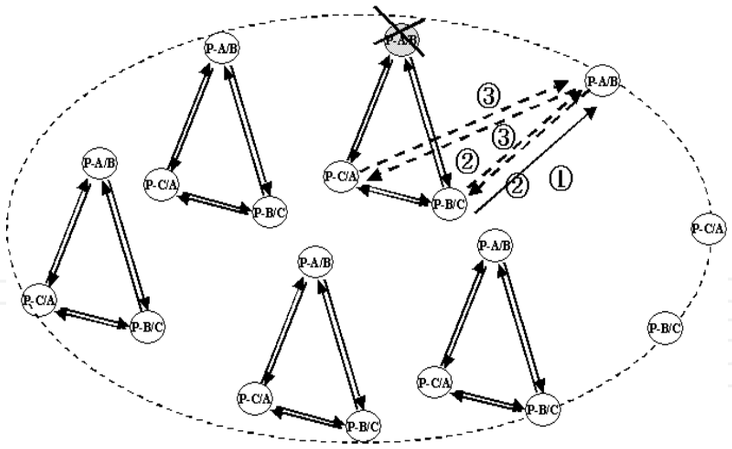


Fig. 3. P-Node (A/B) replacement process

2.4 Processing Node (P-Node)

Processing Node (P-Node) has L3 cache in its local memory space. Unlike OS local memory and UBC, this cache space is storage block address cache and is generated by cache control software with RAM disk driver. Therefore, L3 cache is not visible from OS. The main purpose of L3 cache is timeliness write I/O event execution by write back cache policy. Each P-Node is mounted two partitions on C-Node and one side of partition is mounted through

L3 write back cache. L3 write back cache holds write data until the dirty flush timing comes. At the flush time L3 cache autonomously flush the dirty data. The summarization of P-Node features is following. 1) Three nodes are collaborating as trio node group. 2) Write event request executes by L3 cache write back policy in local memory space. 3) Each P-Node is mounted two storage partitions on C-Node to maintain its data availability. 4) Minimized write I/O latency. 5) Autonomous read cache node expansion. The uniqueness of P-Node is trio node configuration. Each node has two storage partitions and three nodes that have mirroring partitions as mirrored. Therefore, if one P-Node fails, other node can continue the system operation and maintain its availability. But, the main purpose of two partitions are mirrored by three nodes is write back cache with no dirty data flushing policy. Because P-Node achieves timeliness write I/O event without any write speed performance drop. P-Node receives write event request by its L3 write back cache. Creating dirty data flush is done by other P-Node that was mounted on same partition without L3 write-back cache.Figure 4 shows L3 block cache P-Node.

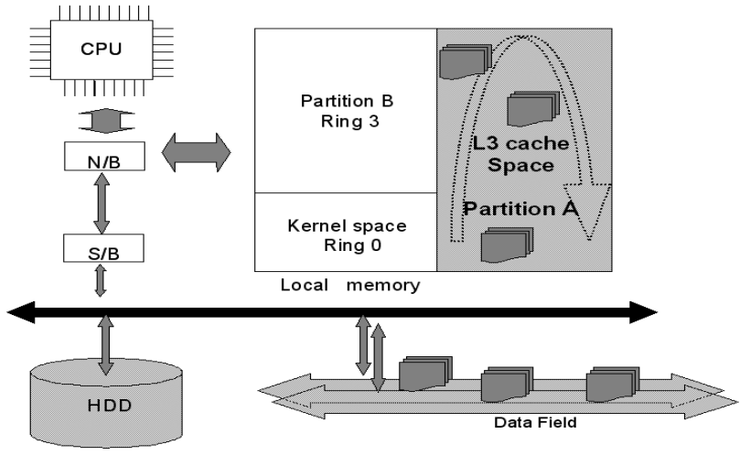


Fig. 4. L3 Block cache P-Node

Other space is OS memory space. Other one of node service runs on this memory space. L3 block cache policy is always write-back cache and other one is no write-back policy. The P-Node performs the high I/O response from the request by L3 block cache and performs write eventually coherency process. Thus, P-Node activates by two nodes minimum and they achieve low latency write event manner.

There are three features of L3 cache. Firstly it converts local memory into I/O block device cache, and mounts this on C-Node target device. Secondly cache policy selection as per the application requirement, and lastly an appropriate technique to search and access the data efficiently in the L3 block cache. The selection of block device is very flexible, for instance if system needs more high I/O speed for I/O and network transactions then the local memory is the ultimate choice. The local memory is transformed to a block I/O device to behave as a cache layer. Consequently it performs high I/O transactions and significantly improves response by utilizing bandwidth equal to the local memory. L3 cache management software detects the block address, which was requested, and keeps this data on the cache device. So, most of the requests are fulfilled from the cache. If the requested block data is not available on the cache, then it is accessed from target storage device. The third function of the L3

block cache technology as shown as Table –1 is search algorithm to look for data on the L3 block cache efficiently. A new cache search algorithm quickly searches required block of data for I/Os to further reduce the latency of time. The L3 block cache consisting of blocks is divided into groups where each group contains equal number of blocks. The jumping algorithm first points to the required group for which that sector corresponds and then search to the required block of data in the group. Jumping algorithm enhances much higher search time than standard sequential search technique for sequential requests. If the requests are randomly distributed, then more advantage is expected than standard search. Investigation for the value of number of column for block I/O data is carried out. In the current paper the value is 32, but If CPU performance is higher, then 64 or 128 may be selected as this tradeoff depends upon the CPU specifications.

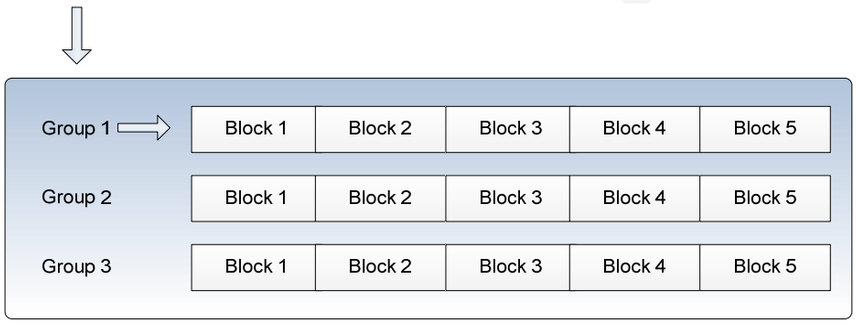


Table 1. P-Node L3 block cache jumping search table

In order to elaborate this mechanism, following is its example. This simple code describes how to search desired block of data effectively on the L3 block cache:

```
IRP Offset =0;
IRP length = 2048;
NrBlocks = ((offset + NrSectors - 1)>> *(targetDisk->SectorPerBlock2)) +1;
NrBlocks = ( ( 0 + 4  - 1)>> 3  ) +1;
Index = (ULONG) ((blockSector.QuadPart) % (targetDisk->NrBlocks));
Index = 1 % 100 = 1;
RowIndex = (Index / (targetDisk->index Columns));
ColumnIndex = 0%32= 0;
Block = _SearchBlock (targetDisk, blockSector, RowIndex, Column Index);
Block = _SearchBlock (targetDisk, 0, 0, 0);
```

Block of data will be searched in the list attached on this index that whether it exists on L3 block cache or not. 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.

2.5 Content Node (C-Node)

Content node (C-Node) has all of service application program and data related to each storage partition. Each C-Node is configured trio group autonomously. The storage partition is encapsulated L4 block cache. The L4 cache is dedicated device block cache and it has write back policy. The request from P-node is entertained in write back cache manner.

Therefore, write event latency of time to be shorter than non L4 cache C-Node model. C-Node is Content Node that has dynamic data availability function for web application services. There is iSCSI base connection from P-Node to C-Node and it enables to connect P-Node for high I/O response execution. The capacity of storage is virtualized value that was shown and it is showing multi Terabyte storage capacity. C-Node has two partitions regularly and they are mirroring by the trio nodes as shown in Figure 5. The disk partitions inside of C-Node are main pool and sub pool such as "A" and "B", "B" and "C" and "C" and "A". Therefore, if "A" partition disk is failed; other duplicated partition can be used for maintaining the service. C-Node is connecting Data Filed and is waiting the event from upper P-Node. When P-Node receives the storage mount point from C-Node, P-Node mounts to C-Node and starts each service from each partition. Once each service is started, C-Node tries to maintain frequently accessed block address for each service on L4 cache to enhance the cache hit availability.

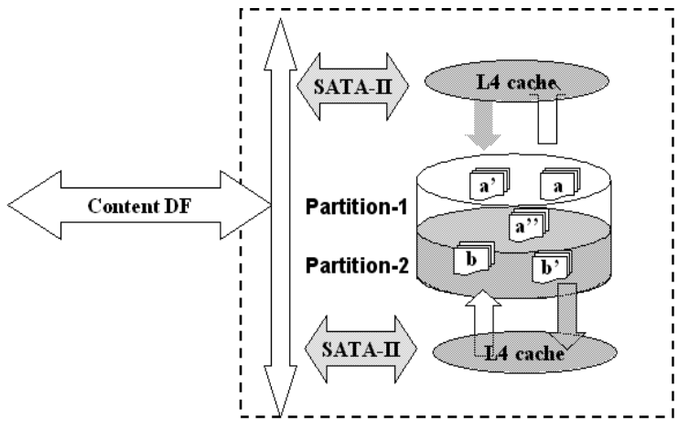


Fig. 5. L4 cache design diagram

Thus, L4 cache on C-Node holds the most used block data on the L4 cache and is utilized for P-Node expansion when service demand is increased. Generally, C-Node can connect other C-Node whenever P-Node in Data Field requests new extra storage capacity. The process of expansion event for C-Node executes by block data analysis monitor in P-Node. The block data analysis tool gets the block address total sum in the sampling time and if the total block address exceeds the threshold value, this P-Node broadcasts the expansion request to extra C-Node autonomously. The threshold value considers the balance L3<L4 to maintain timeliness I/O performance. C-Node has L4 cache that is located inside of each storage drive. The L4 cache implements pre-fetching technology to reduce the first read cache penalty. The accuracy of read cache hit ratio is determined by algorithm. There are many cache pre-fetching techniques available such as Least Recently Used (LRU), First In First Out (FIFO), Hidden Markov and NN+ CBR. In this case, LRU model is selected. C-Node autonomous storage node functions were implemented by following independent modules. 1) L4 cache layer management module. 2) Failure sense and event notice module. 3) Autonomous Local storage duplication module. 4) Pre-fetching read module. L4 cache enhances read / write cache hit ratio by dedicated L4 block cache space. Event the first read cache penalty, C-Node pre-fetches the most possible read data by its pre-fetching technique. Therefore, the almost read penalty is less than direct storage access. The basic features of L4 cache are 1) Loosely connection, transparency 1GB block cache.2) Write-back policy with

pre-read cache implementation and 3) Pre fetching boot up file, 4) Internal UPS that has automatic shutdown function acts whenever un-properly shutdown or upper system power fail re-securing dirty data on the cache table. 5) Can be expandable the total number of L4 cache size by multiple number of L4 cache drive.

This L4 cache space is completely individual autonomous memory space on the device. Today, there are many RAID storage subsystems available. But seeing from the individual device point of view, there are some difference points. RAID controller has buffer cache memory for RAID sets. But, this is the unified block cache space and it for block cache of RAID set and its volume. Also it is not safety idea when RAID subsystem electric power failure occurs. P-Node L4 cache is loosely connection individuality block cache. Only L4 cache device achieves no risk with high I/O speed with assurance benefit. Figure 6 is L4 cache block diagram in C-node. In this general case of design, 1GB DRAM for L4 cache memory with Flash SSD is target mounted device. Internally, 64bit based DRAM assigned and its I/O performance shows more than 270MB/s with 40KIOPS transaction performance under MLC SSD configuration.

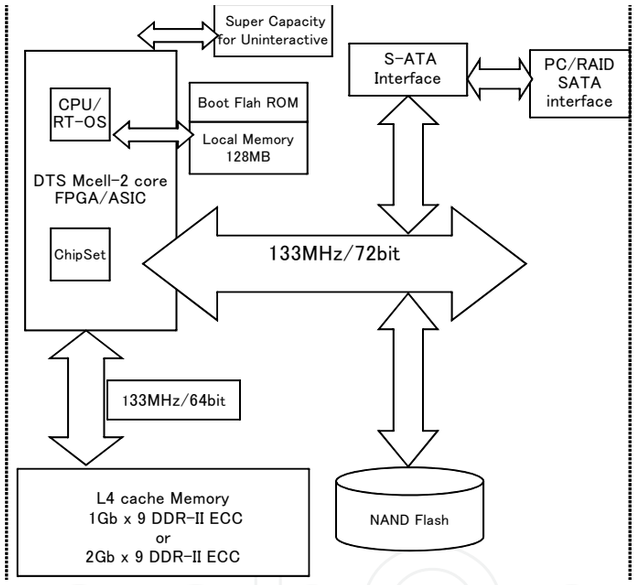


Fig. 6. L4 cache design diagram

2.6 Write data coherency process by P-Node

When write data request received from out side requester, two P-Nodes that are mounted C-Node application program partition execute the write request event by L3 cache P-Node and non-L3 cache P-Node. For instance, If Write event required at storage partition "A", there are two P-Node related this event as P-Node (A/B) and P-Node (C/A). In this case, P-Node (A/B) has L3 cache for "A" and P-Node (C/A) hasn't. The ACK for write event requester replied by P-Node (A/B) using L3 write back cache is by one hop. Then P-Node (A/B) is holding dirty data. Another P-Node (C/A) executes write event to C-Node (C/A). It also does C-Node (A/B) by write event broadcast message via C-DF. Once received write event request from P-Node (C/A). Both of C-Node (A/B) and C-Node (C/A) have done the write event and ACK to P-Node (C/A) that the event is completed. When P-Node (C/A) received it from both of C-

Node, P-node (C/A) signal cast to P-node (A/B) the message of write event was done. Then P-node (A/B) flag off of write dirty and open the memory space for new write event entry. Therefore, always write-data response time is maintaining high I/O response by L3 write-back cache with non-L3 write-through cache without cache memory size limit.

2.7 Read event by L3/L4 cache

L3 cache on P-Node effects read I/O event also. L3 cache is dedicated storage cache on P-Node and it maintains the data inside of L3 cache. The read event process is following. a) When received the read event "A" from requester, there are two P-Nodes that are related partition "A". b) One of P-Node has L3 block Cache space and other is no cache space of dedicated "A" partition. It just mounts "A" partition. c) When executes read event of "A" partition, the one of P-Node which has L3 cache for partition "A" executes the read event if L3 cache space holding the read event data and broadcast its acknowledge via data field as timeliness manner. d) Other one of P-Node executes by traditional read event execution if read data detects inside of memory space. e) The fastest read event executed node broadcasts the result of read request event to other nodes with event message. f) Once received the read event result from any P-Nodes, this node stops the read event execution. g) Eventually, the fastest P-Node result reaches the requester. P-Node read event is very flexible process. If one of node holding data for the event, anyone can be the fastest replier. Other node that has data for read request is stopped by the fastest node action. When the read event access size exceeds more than L3 cache, L4 cache on C-Node has possibility to maintain the next priority data. C-Node manages the size of L4 cache if P-Node L3 cache size exceeds L4 cache, C-Node expands the L4 cache capacity inside C-node disk configuration change and it maintains the ratio "L3<L4".

2.8 Content Node (C-node) failure sense and replace

When the event of C-Node failure occurs, co-related P-Node detects C-Node status by its response behavior. For example, C-Node (A/B) is mounted by P-Node (A/B), therefore, P-Node (A/B) detects C-Node (A/B) failure by no response or reply timeout. P-Node (A/B) single casts the message of call extra C-Node via C-DF. If an unassigned C-node is existing, then this C-node broadcasts the status to related C-Node in trio group. The nearest C-Node becomes the new member of C-Node trio group and starts to communicate with three as new trio group. The application is stored another size of C-Node with latest data, C-Node (B/C) provides "Storage partition "B" and C-Node (C/A) supplies storage partition "A" Thus, data duplication is time consuming work but this event runs on only C-DF. Therefore, seeing from out band requester, it is hidden internal task and less impact for out band service. This is one of the advantages by dual data field design. After duplication event is finished, New C-node (A/B) is released to P-Node (A/B). Even before complete this duplication event; other two of C-Nodes with three P-Nodes can sustain the operation without any service interruption.

3. Evaluation

3.1 Determination of geometrical architecture

Write I/O event requires two L3 cache nodes to manage write-data coherency by Eventually Consistent manner. This process is utilized by two L3 cache nodes with two L4 C-Nodes for

assurance purpose. One of C-Node is mounted one of L3 cache node and other one is mounted no L3 cache as standard drive access way. Therefore, two P-Node with two C-Node partitions are required to maintain high latency of I/O. Thus, two partitions are required physically to achieve low latency of time for the write event. But, in two node configuration, there is no assurance policy against one node failure. Seeing from online function sustainability, three nodes is minimum number for a group. Each P-Node is connecting C-Node for its representative. Therefore, the combinations between P-Node and C-Node require the same number of nodes. To consider the online availability, the number of nodes is determined by Scale-out for the system. The latency of time and its overhead and threshold value are evaluated. Then it is determined about the number of nodes which can be easily managed. About online availability, what is the best number of node on each Data Field is evaluated. Proposing architecture has trio node group and when the event of one of node failed, the node availability of system level is 7/8. If node number n is same as node number n, its redundancy is much higher than Trio node n=3. But there are so many data communication on Processing Data Field and Content Data Field. For example, Dual node is 3/4, Trio node is 7/8 and Quad node is 15/16. 3/4 < 7/8 < 15/16 Therefore, quad node is higher than others. But, IO transaction is n*(n-1)/2 by Metcalfe's law. Therefore, its network overhead is

NP *(dual)* = 2*1/2 = 1
NP *(Trio)* = 3*2/2 = 3
NP *(Quad)* = 4*3/2 = 6

Moreover, management cost of cache coherency and cache size on each node is 1/n size from original cache size. Therefore, ½, 1/3 and ¼ cache size is utilized. It significantly reduces cache size. This matter generates very big impact of cache performance and less cache hit ratio. Therefore, Trio node plus two partitions is much better architecture in comparison with other number of nodes architecture.

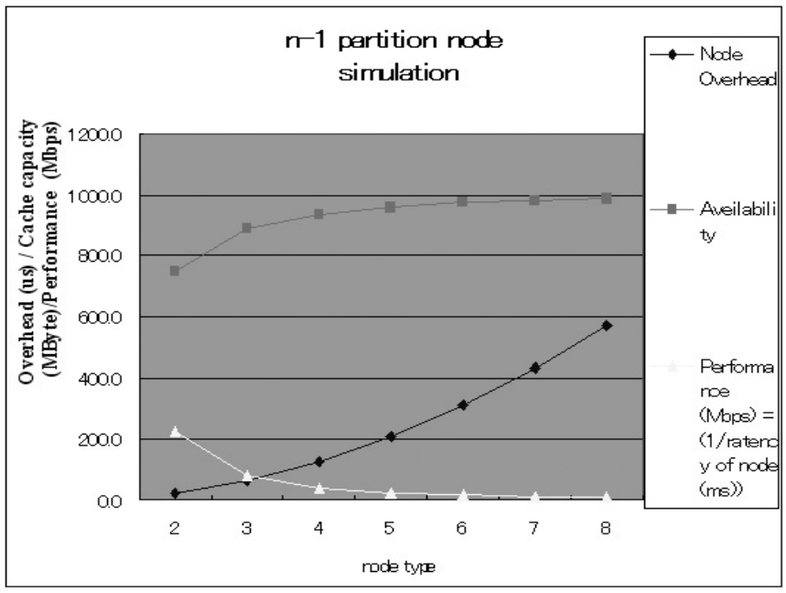


Fig. 7. n-Node simulation

Figure 7 shows n-Node simulation. The availability of Two-Nodes is 0.75 and Trio-Nodes is 0.888 and Quad-node is 0.937. But, node overhead is increasing by number of node. Two-Node is 22msec and Trio-node is 64mSec and Quad-Node is 126mSec. Seeing from balance point of view, two-Node is minimum overhead model but in one node failure, there are no redundancy nodes. It is risky from availability point of view. Therefore Trio-Node is better than other configurations. Also proposing system has dual data fields. It has high adaptability too.

3.2 P-Node + C-Node performance evaluation by XDD bench

In this section we evaluate C-Node L4 cache management policy as L3<L4. The objective of this experiment is to carry out performance evaluation of P-Node + C-Node using L3/L4 cache (combined performance gain due to L3 and L4 caches). This evaluation shows that the high I/O cache node doesn't maintain the I/O performance gain stable. The I/O performance gain calculates by Amdahl's law as shown in equation (1).

$$G_i = \frac{1}{1 - C_i + \frac{C_i}{X_i}} \prod_{j=1}^{m} L_{i,j}. \tag{1}$$

Gi = cache gain,
Ci = cache hit ratio
Xi= cache device speed gain compare target device speed.
Lij = overhead for cache program.

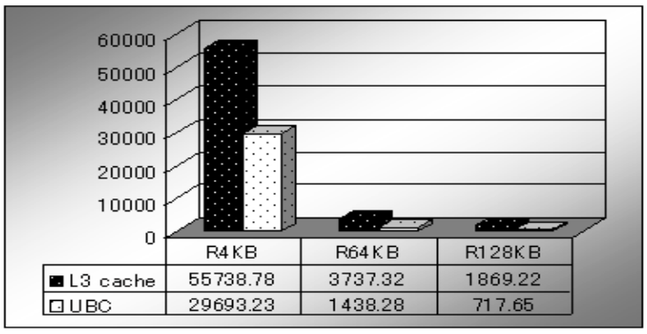
Therefore, if the access doesn't hit the cache, performance drops more than original target drive speed because cache overhead exists. To manage this cache behavior, multi-layer cache system architecture with balanced cache ratio is key factor. This experiment carries out two scenarios under this hardware and XDD benchmark test was utilized as shown in Table 2.

Machine	HP DL-145
CPU	AMD Opteron(tm) Processor 248 2210.364MHz x 2
OS	Linux Redhat 4
System RAM	Total 4 GB
DTS cache L_3	1GB (write Back) / 3GB OS space
DTS cache L_4	L3 < L4 2GB, (by L4 drive x 2) L3 > L4 0.03GB, (by HDD x 2)
Be Benchmark tool	XDD65.013007

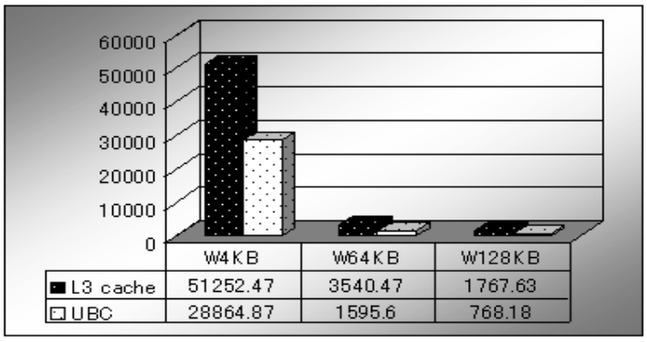
Table 2. System specifications for system configuration

The evaluation of I/O performance of L_3 cache on P-Node against OS UBC cache has been carried out on Linux Kernel 2.6.24 rt27 32bit by XDD65.013007 as shown in Figure 8 (a), (b) and (c). In this scenario sequential read and write data types are utilized. Interoperable (two way) I/O types read/write (50% 50%) are evaluated in this scenario. The test files are 4KB, 64KB and 128KB on each access type for L3 cache in comparison with OS based UBC cache.

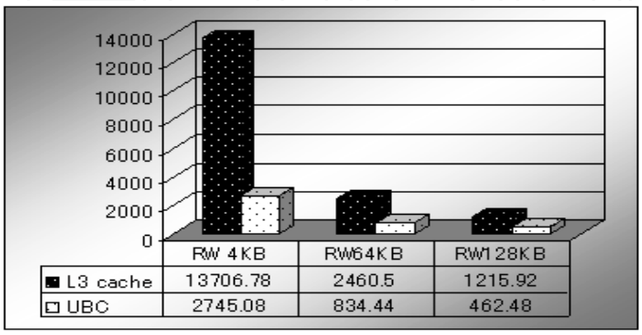
The total memory size is 4GB in the computer and it is assigned 3GB OS area and 1GB L3 cache. UBC utilizes full size of 4GB memory without any limitation. L3 cache policy was designed write back policy with read through. The reason of read through cache on UBC is because it is read buffer cache when read event is on the buffer space. L3 cache dedicates the write event and holds write event data for the read event. The result shows that L3 cache gives double performance than UBC buffer at sequential read and write test. Other result at read/write 50%/50% shows 13706.78 IOPS with L3 cache in comparison to 2745.08 for UBC. It is five times gain at read/write 50%/50% compare UBC cache node system. L3 cache is dedicated storage partition block cache and it manages write back policy by L3 cache and read from L3 cache. Therefore, interoperable I/O performance is very effectiveness event. This result shows the proof of performance effects by L3 cache compare unified buffer space.



(a) Read sequential



(b) Write sequential



(c) Read/write 50% / 50%

Fig. 8. L3 cache VS OS UBC cache I/O performance.

4. Conclusion

Web service is heterogeneously demand model in today. User push type web service is so popular such as twitter and SNS. The demand from them is web evolving target model behavior. Therefore, web service needs adaptability, online expandability and data availability. Increasing user push type web service demand has dynamic behavior. Therefore the management of the coherency of written data is also a big issue. Proxy server is read cache model and it is data duplication model but it doesn't support duplication write data event dynamically. It also doesn't support coherency of write data too. Maintaining the low latency of time, single Data Field model can't achieve these issues with limited size of write cache memory space. The proposed system architecture ensures 1) Adaptability for user push type web service demands and 2) online node sustainability and 3) low latency of write data without write cache size limit and 4) P-node /C-Node cache autonomous contribution. Utilize L3 cache and L4 cache benefits by P-Node/C-Node L3/L4 cache. The write data coherency issues, P-Node plus C-Node write eventually Consistency process is enabled it. The concept is always executed by L3 cache P-Node and it guarantees the write event latency minimum network hops. Therefore, it achieves the low latency of time write I/O event for real time web application. P-Node also performs read data performance by L3 cache. L3 cache on P-Node evaluation shows double performance in comparison with UBC cache at sequential read and write test. Other results in case of read/write 50%/50% shows 13706.78 IOPS compare 2745.08 for UBC. It is five times faster than UBC cache node system. This is the proof of performance effects by L_3 dedicated cache for low latency web service in comparison with unified buffer on autonomous node.Thus, the layer cache node would be utilized under many massive I/O applications by its autonomous decentralized node. Co-related P-Node and C-Node show high I/O advantage with dynamic data availability. Therefore, Autonomous multi-layer cache system architecture is the solution for interoperable communication with low latency of time web service with dynamic data availability. Our next step is variable service application level evaluation and autonomous node community expansion / reduction technology design.

5. References

[1] I.-L. Yen, R.Paul and K. Mori. Towards integrated methods for high-assurance systems. IEEE Computer, 31(4):32-34, April 1998.

[2] K. Mori. Autonomous decentralized systems: concept, data field architecture and future trends. In Proc. of ISADS, IEEE, pages 28-34, 1993

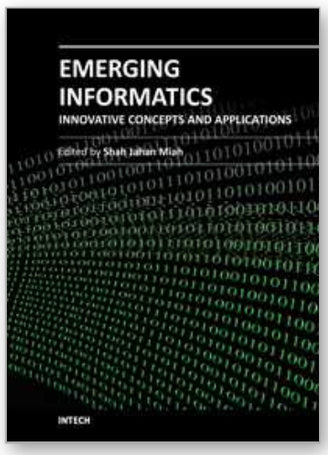
[3] H.F. Ahmad and Mori. Autonomous Information Service System: Basic Concept for Evaluation, IEICE Transactions on Fundamentals of Electronics and Computer Sciences, Vol. E83-A, No.11 pp.2228-2235, November 2000

[4] Leguizamo, C.P. Kato, S. Kirai, K. Mori, K. Autonomous Decentralized Database System for Assurance in Heterogeneous e-Business. Proceeding of COMPSAC, p589-p595, IEEE, May 2000

[5] S. Przybylski,M. Horowitz, J. Hennessy, Characteristics of performance-optimal multi-level cache hierarchies, Proc of the 16th Annual International Symposium on Computer Architecture pp. 114 - 121 (1989).

[6] Moore's law www.intel.com/technology/mooreslaw/ index.htm

[7] Elizabeth Varki, Arif Merchant, Jianzhang Xu, and Xiaozhou Qiu. Issues and Challenges in the Performance Analysis of Real Disk Arrays. IEEE Transactions on Parallel and Distributed Systems, 15(6):559–574, 2004.
[8] ADS protocol specification R3.0 , MSTC/JOP 1101-19999/09/3
[9] Dai Kobayashi, Akitsugu Watanabe, Toshihiro Uehara, Haruo Yokota, 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)
[10] Daniel P. Bovet, Marco Cesati : Understanding the Linux Kernel , O'reilly Press, pp.422-498 (2001).
[11] Maurice J. Bach : The Design of The UNIX Operating system, Bell Labs Press, pp.264-287 (1986)
[12] Chuck Silvers, UBC : An efficient Unified I/O and Memory Caching Node for Netbsd, Proceedings of FREENIX Track: 2000 USENIX Annual Technical Conference, San Diego, California, USA, June 18–23, 2000
[13] White Paper, Using Real-Time I/O Signature Analysis to Identify Performance Improvement Options for Database Applications, http://www.soliddata.com/pdf/WP_IOSignatures_v2.pdf, Solid Data Systems Inc, July 2006.
[14] Wade Tuma, Comparisons of Drive Technologies for High-Transactions Databases, http://www.soliddata.com/pdf/WP_Drive_Comparison_v2.pdf, Solid Data Systems, Inc. (August 2007)
[15] Mohamed Zahran, Kursad Albayraktaroglu and Manoj Franklin, Non-Inclusion Property in Multi-level Caches Revisited, IJCA, Vol. 14, No. 2, pp.1-10, (2007)
[16] Hironao Takahashi, Hafiz Farooq Ahmad, Kinji Mori, Layered Memory Architecture for High IO Intensive Information Services to Achieve Timeliness, 11th IEEE High Assurance Systems Engineering Symposium Nanjing, China, December 3 - 5, 2008
[17] S. Pai, Vivek ; Druschel, Peter ; Zwaenepoel, Willy, "IO-Lite: a unified I/O buffering and caching system", ACM Transactions on Computer Systems (TOCS), Vol. 18, No. 2, 2000, p. 37-66.
[18] RAMDISK browsed at sep 20th 2009 http://www.vanemery.com/Linux/Ramdisk/ramdisk.html
[19] S. Przybylski,M. Horowitz, J. Hennessy, Characteristics of performance-optimal multi-level cache hierarchies, Proc of the 16th Annual International Symposium on Computer Architecture pp. 114 - 121 (1989).
[20] Hironao Takahashi, Hafiz Farooq Ahmad, Kinji Mori, "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, March 23-25, 2009, pp 93-99



Emerging Informatics - Innovative Concepts and Applications

Edited by Prof. Shah Jahan Miah

ISBN 978-953-51-0514-5
Hard cover, 274 pages
Publisher InTech
Published online 20, April, 2012
Published in print edition April, 2012

The book on emerging informatics brings together the new concepts and applications that will help define and outline problem solving methods and features in designing business and human systems. It covers international aspects of information systems design in which many relevant technologies are introduced for the welfare of human and business systems. This initiative can be viewed as an emergent area of informatics that helps better conceptualise and design new world-class solutions. The book provides four flexible sections that accommodate total of fourteen chapters. The section specifies learning contexts in emerging fields. Each chapter presents a clear basis through the problem conception and its applicable technological solutions. I hope this will help further exploration of knowledge in the informatics discipline.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen