International Journal of Database Theory and Application
Vol.7, No.3 (2014), pp.1-12
http://dx.doi.org/10.14257/ijdta.2014.7.3.01

Effective Metadata Cluster Management for Mass Data Storage

Chen Ningjiang[1], Xiao Zhongzheng[1] and Zhang Wenbo[2]

[1] College of Computer Science and Electronic Information, Guangxi University, Nanning, 530004

[2] Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing, 100190

chnj@gxu.edu.cn, zz.xiao.gx@gmail.com, zhangwenbo@otcaix.iscas.ac.cn

Abstract

An effective decentralized metadata management schema is critical to a reliable and scalable large-scale distributed storage system. With regard to the defects of huge expansion cost and high sensitivity to cluster alteration borne in both the hash-based and subtree-based metadata management schemas, a metadata server (MDS) clustering schema based on consistent hash structure, named DSVL, is proposed. By introducing virtual MDS into the consistent MDS cluster, it can effectively balance MDS cluster load. Besides, the standby mechanism and the lazy update policy are merged and applied to the MDS cluster to accomplish rapid MDS failover as well as zero data migration in case of cluster alteration. The results of prototype system and simulation test prove that DSVL, with the features of balanced metadata distribution, rapid failover, flexible scalability and zero data migration amount in case of node alteration, can meet the demands for flexible and effective management of large-scale data storage systems with ever increasing data amount.

Keywords: *metadata management, metadata cluster, distributed storage*

1. Introduction

The metadata management is the key factors for the good performance and scalability of large-scale distributed file system [1]. The metadata amount takes up a very small percentage of the data storage, however, 50%-80% of the access requests involve metadata operations [2]. The metadata server (MDS) [3] works as an intermediate mapping layer for client's data requests and data files. Decentralized and highly scalable MDS cluster schema [4] can realize the good performance and scalability in data access by sharing loads. Hash-based mapping and subtree partitioning are major metadata partitioning ways. Hash-based mapping has good load balance but its' scalability is terrible. Subtree based schema has huge cost of data migration. Large-scale storage system with periodical input of large amount of data will find its metadata amount increasing dramatically with the gradual growth of data amount. Metadata determines the system's storage capacity. Apparently, a cluster that is flexible in expanding the metadata management nodes can meet the demand for ever increasing data storage.

The paper proposes DSVL (Distributed Schema based-on Virtual MDS and Lazy policy), a distributed dynamic metadata management system based on consistent hash that can accomplish the effective management and flexible expansion of cluster metadata in the large-scale data storage system. Section 2 introduces the work related to metadata management. Section 3 introduces system architecture while Section 4 describes its performance in metadata load balancing, MDS fail-over, scalability and data migration. Simulation tests for DSVL prototype are given in Section 5. Finally, we discuss the advantages of DSVL in Section 5 and conclude in Section 6.

2. Related Work

GFS [5] and HDFS [6] are typical single MDS schema implementation. It is easy to maintain the consistency but it limits the scalability of the file system, and a large number of concurrent writes will dramatically increase the load of the master MDS, which will damage the system stability. In the subtree-based partitioning, the namespace of the whole file system is partitioned to be many subtrees. The static subtree partitioning is relatively easy to fulfill with NFS [7] and Coda [8] as representatives. However, it may give rise to extreme imbalance of the system load. The dynamic subtree takes into consideration the load balance. Ceph [9] typically employs the dynamic subtree partitioning. The dynamic subtree requires all nodes to regularly exchange their load information, resulting in the sharp growth of system load as rehashing is needed in the case of the alteration of the cluster structure. Hash-based metadata partitioning [10] directly determines file location. The metadata requests will be evenly distributed onto the metadata nodes. But it's extremely sensitive to the change of metadata cluster configuration. Therefore, Lazy Hybrid [11] tries to reduce such huge abrupt metadata migration by lazy update policy.

3. Architecture of DSVL

As shown in Figure 1, the global metadata management for DSVL is based on consistent hash structure [12-14]. Consistent hash is usually applied to the design for distributed hash table, as seen in Chord [13] and Apache Cassandra [15]. With its monotonicity, autonomy, decentralization, fault tolerance and scalability, it will not lead to critical changes of element mapping in the case of joining, leaving and failure of nodes.

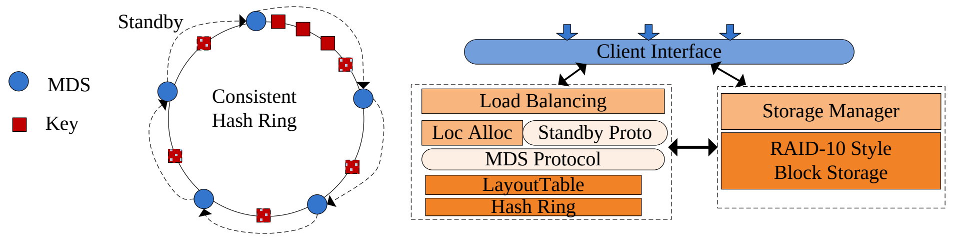


Figure 1. Architecture of DSVL

All MDSs as well as all metadata items in DSVL are mapped to the consistent space by hash functions, and every MDS has its standby MDS called Standby Node which is its next hop MDS in the consistent hash ring. The global data layout table is used to describe the network structure of MDS-cluster, the logic structure of MDS, the distribution structure of metadata and the Lazy-Update Table. Besides, the MDS Protocol is used to regulate the interoperation between MDS instances while the Standby Protocol defines the standby mechanism for the Standby Node that can interact with the target MDS.

3.1. The MDS Layout Structure

Definition 1 Cluster Layout. It describes the network topology of MDS cluster and is marked as T_c:: =[ ]<{MID_i,DC_i,MID_t}>, in which MID_i is the sole identifier of MDS instance,

DC_i is the network connection information of MDS instance, and MID_t is the identifier of the target MDS node served by the Standby Node in the MDS instance.

Definition 2 Metadata Distribution. It describes the mapping of metadata to MDS and is marked as T_d:: $=V_d \cup C_{m,}$, in which C_m=SortedMap<{(K,$vMDS$)→{MID,$host$,$ports$,…}}> (this is a metadata mapping set and an ordered collection, in which K is the Key value of metadata and $vMDS$ is the identifier of virtual MDS mapped to the real MDS.) and V_d=[ ]<{MID→[vM_i, $vM_{i+\gamma}$, $vM_{i+2\gamma}$,...]}> (this is the distribution table of virtual MDS and MID is the MDS identifier mapped to a group of virtual MDSs.)

Definition 3 Lazy-Update Table. It records the local lazy-update data set and is marked as T_u:: =[ ]<{K,MDS_S,MDS_T,$Type$}>, in which K is the metadata identifier, MDS_S is the ID of source MDS, MDS_T is the ID of target MDS, and $Type$ is alteration type. It's the local MDS if MDS_S is empty. $Type$=[rm| mv], in which rm means to delete operation while mv, update operation.

Definition 4 MDS Layout Table. It refers to the compound data structure of every MDS instance that organizes and controls clusters and is marked as LT:: $=T_c \cup T_d \cup T_u$, in which T_c and T_d are shared by every MDS in the cluster while T_u is privately owned by the very MDS.

T_c、T_d and T_u are combined to describe the structure, state and control information of the whole MDS cluster. T_c builds an information table of MDS nodes based on the cluster layout, T_d defines the multi-level mapping of Key→$vMDS$→MDS, and T_u marks the expansion record on Key when metadata is changed (in particular the renaming of directory) so as to conduct lazy processing of data migration in large amount. LT is critical to the building and maintenance of the effective and consistent operation of MDS cluster as it directly determines its logic structure. Figure 2 shows the logic structure of Layout Table in DSVL.

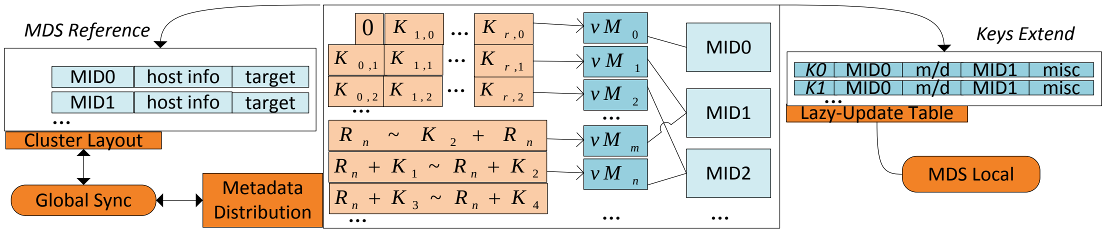


Figure 2. Structure of Layout-Table in DSVL

As shown in Figure 2, $K \in S_h$, $vM_0 < vM_1 < \ldots \leq 2^n$-1, metadata between 0 ~ $K_{r,0}$ are mapped to vM_0, metadata between $K_{0,1}$ ~ $K_{r,1}$ are mapped to vM_1, and so on. vM is mapped to the only MID while one MDS is partitioned into many vMs. T_d table, after a series of mapping, determines the MDS in which metadata reside. The distribution process of vM is also the even distribution process of metadata as well as the provision of matching MDS for client's lookup. Client can, when looking up metadata, send MDS matching requests from any M_i who will first gain via hashing the file's Key value to be input into T_d for matching and return with $M_{t,}$ the target MDS node, then client can be directly redirected to M_t for metadata. T_u, the auxiliary data structure built on T_d, is for marking the altered metadata instead of migrating them immediately so as to reduce the large data migration amount possibly resulting from metadata alteration. T_u are not consistent with each other as every MDS has its local Lazy-Update table that is not shared with other MDSs. Therefore, only relevant local items are marked when alteration of local metadata occurs. Forced data migration will be triggered when the items' size of T_u reaches the threshold, which is apparently helpful for better utilization of storage resources and avoiding the metadata fragmentation caused by large amount of metadata to be migrated but not yet migrated.

3.2. Balanced Metadata Distribution

The uncertainty of mapping MDS nodes to S_h hash ring will lead to the uneven distribution of elements on it and further to the load imbalance of MDS cluster. With regard to the metadata distribution imbalance as a result of MDS mapping randomness and range of data aggregation, the paper proposes to remap numerous Virtual MDSs (vMDSs) evenly to S_h for every MDS so as to ensure equal metadata residing probability of every MDS in every local range.

When a certain MDS instance joins in MDS cluster, the very MDS responsible for its registration will distribute for it the default P vMDSs. As a result, LT's alteration will be directly influenced, so the join request will be broadcast to all other existing MDSs for voting before registration is allowed by MDS via testing to see if the local LT is attached a write lock (*e.g.*, forced data migration is under way). MDS will be successfully registered and distributed with P vMDSs if registration is passed unanimously. As shown in Figure 3, the range of metadata sections of M_1 is far beyond that of M_2 and M_3, noticeably impacting the system's storage and computation performance and utilization. Now let's map M_1 respectively to the space within the same range as $[0, 2^n-1]$, we can get $[vM_{1,0}, vM_{1,1}, vM_{1,2}, vM_{1,3}]$. These vMDS will partition S_h into P equal parts (P =4 as shows in Figure 3 and Figure 4) and the metadata items and vMDS comprise the basic elements of hash space. The mapping between MDS and vMDS is thus finally described as V_d in Definition 2. Such equal partition in enclosed space allows the data distribution and procedure of every $1/P$ to be consistent.

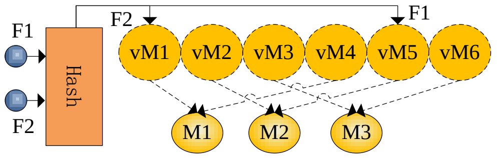


Figure 3. MDS-vMDS Mapping

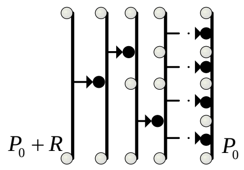


Figure 4. Procedure of vMDS Division

Figure 4 shows the dynamic procedure of vMDS distribution in $1/P$ aliquot part of S_h. The space between P_0 and P_0+R is one aliquot part itself when there is only 1 MDS in the beginning. In the later process of vMDS joining in continuously, if $[P_0,P_0+R)$ is partitioned into the aliquot collection of CV and CV_2, and all elements in CV_2 are 2 times of those in CV, the aliquot block in CV_2 needs to be further bisected until CV_2 is empty. The distribution algorithm of vMDS is described as follows.

```
Algorithm ALLOC-VIRTUAL-MDS
Input: hash factor: $\beta(\beta>1)$; vMDS number: $\rho(\rho>0)$;  vMDS recycling list: $\zeta = List\ \{ vMDS$
}
Output: vMDS set $A$
BEGIN
   (1) Def $N=2^{\beta}-1$, $N_\gamma = N/\rho$, $\Delta = N_\gamma$, $P_\gamma = 0$, $A = [\ ]$
   (2) IF $\zeta \in ø$ :  $A_1 \leftarrow P_\gamma + \Delta/2$
   (3) FOR i = (1:1: ρ] DO: $A_i \leftarrow A_{i-1} + N_\gamma$  ENDFOR
   (4) $P_\gamma \leftarrow P_\gamma + \Delta$
```

```
    (5) IF $P_\gamma \rightarrow N_\gamma$:  $P_\gamma \leftarrow 0$,  $\Delta \leftarrow 0$  ENDIF
    (6) ELSE:   $A \leftarrow rm(\zeta_0)$,  FOR  j = (1:1: ρ]  DO:  $A_j \leftarrow A_{j-1} + N_\gamma$  ENDFOR
    (7)ENDIF
END
```

Δ keeps rotating within $[0,N/\rho)$ with the existing distribution of V_d, so that vMDSs are relatively evenly distributed. The distribution of vMDSs allows metadata to be indirectly mapped to MDSs via vMDSs, creating a balanceing mechanism between MDSs and metadata that contributes to the better response performance of MDS cluster.

Fact 1. MDSs and metadata items are mapped to the same hash space, and metadata K resides in M_x when and only when $((\nexists\ M_i,\ MK_i>MK_{x)} \wedge\ MK_x>K) \vee\ (MK_x=0 \wedge\ (\nexists\ M_{j,}\ MK_j>K))$.

Theorem 1 The consistency of MDS cluster will not be influenced any MDS joins in.

Proof: Allow the existing cluster C to have m MDSs, namely M_0, $M_1,\ldots,M_m$ that are mapped to S_h, and whose *Key* value is MK_0, $MK_1,\ldots,MK_m$ respectively. According to Fact 1, metadata items $\{K \mid K \in [MK_m,MK_0)\} \Rightarrow M_0$, $\{K \mid K \in [MK_0,MK_1)\} \Rightarrow M_1$, …, $\{K \mid K \in [MK_i,MK_{i+1})\} \Rightarrow M_{i+1}$, …,$\{K \mid K \in [MK_{m-1},MK_m)\} \Rightarrow M_m$. If new node M_x with the *Key* value of MK_x is to join in C, $MK_x \in [MK_i, MK_{i+1})$ (MK_i and MK_{i+1} are the *Key* value of M_i and M_{i+1} respectively.), the metadata deployment within $[MK_i, MK_{i+1})$ must be adjusted to fulfill the constraint of Fact 1 $\{K \mid K \in [MK_i,MK_x)\} \Rightarrow M_x$, $\{K \mid K \in [MK_x,MK_{i+1})\} \Rightarrow M_{i+1}$. There will still be $\{K \mid K \in [MK_m,MK_0)\} \Rightarrow M_0$, $\{K \mid K \in [MK_0,MK_1)\} \Rightarrow M_1$, …, $\{K \mid K \in [MK_{m-1},MK_m)\} \Rightarrow M_m$ after adjustment to meet the constraints of Fact 1. To sum up, M_x does not have any influence on the consistency of C, or new node(s) can join in MDS cluster freely. □

Theorem 2 The metadata server interruption time in MDS cluster during the process of new MDS joining in is 0.

Proof: Allow M_d to be the MDS newly joining in cluster C, M_r to be the MDS responsible for registration, and M_d to be distributed between M_a and M_b. The MDS-JOIN algorithm described in this section is now followed. First, M_r asks for votes from all MDSs to see if new MDS is allowed to join and, if yes, records will be newly made for M_d in the local LT of M_r. Second, M_b is the next hop node of M_d, and $\{K \mid K \in M_a \wedge K<M_d\}$ needs to be migrated to M_d and written to T_u based on the lazy policy. Marking the indices of first *Key* and last *Key* is enough as they are in order on LT. Third, M_r broadcasts the alteration of LT. Apparently, the update marking process and the alteration broadcasting process by M_r and the voting process are asynchronously executed with metadata servers, and these three processes don't damage the consistent layout of metadata whose services are still available normally. The broadcasting process of LT alteration will attach write locks to the metadata of M_b, but the lookup services can still operate normally. Other MDSs don't have this request. To sum up, in the process of a new MDS joining in the cluster, the lookup service interruption time is 0. Only one MDS will stop its write service in the alteration broadcasting process while other MDSs can be normally accessed for write. Therefore the system service interruption time is 0.

Corollary 1 DSVL has good MDS scalability.

Algorithm MDS-JOIN
Input：MDS for MDS registration：M_r ; New MDS; M_d
Output：*LT of* M_d

```
BEGIN
  (1) $T \leftarrow FetchLayout(M_r)$       /* gain $LT$ of $M_r$*/
  (2) IF $vote(T.c, M_r) = size(T.c)$    /* executed only after all MDSs allow $M_d$ to join */
  (3)   $Global\_LockLayout(T, M_r)$    /* MDSes execute in parallel*/
  (4)   FOR $M_x$ in $T.c$ PARALLEL-DO:
  (5)     $AddMDS(T_{this}, M_d, M_r)$
  (6)     IF $M_d \rightarrow M_x$    /* lazy update and forced update of the next hop of $M_d$ */
  (7)       $A_l \leftarrow split(T_{this}.m, M_d)$
  (8)       FOR $K$ in $A_l$ DO: $Log\_Migrate(K, M_x, M_d)$  ENDFOR
  (9)       $A_m \leftarrow find(T_{this}.l, M_d)$
  (10)      FOR $K$ in $A_m$ DO: UPDATE-KEY-ITEM ( $K, K, T_{this}$, 'mv')   ENDFOR
  (11)    ENDIF
  (12)  ENDFOR
  (13) $Global\_UnlockLayout(T, M_r)$
  (14) ENDIF
END
```

4. MDS Failover

Persistence of in-memory data is an important issue for storage systems. DSVL addresses this issue with Standby Node [16-18].

Definition 5 M_0 is the standby node of M_t in the MDS instance and marked as M_0 ш M_t when and only when $(\nexists\ M_x, K_x \in [K_0, K_t)) \vee (K_0 = 0 \wedge K_t = N)$, in which K_x, K_0, and K_t are the *Key* value of M_x, M_0, and M_t respectively and N is the size of hash space.

DSVL adopts the checkpoint mechanism. Every MDS must have a Standby Node whose distribution mechanism is shown in Fig. 5. Hash space grows anticlockwise while the distribution of Standby Node is clockwise, M_2 ш M_1, M_3 ш M_2, ….. When M_3 fails, its metadata and the backup data of M_2 all fail. M_4, the Standby Node of M_3, will first discover M_3 failure and quickly retrieve the backup data in M_3 into the MDS instance of M_4. The Standby Node of M_4 will be redirected to M_2 and download the latest memory dump of M_2.

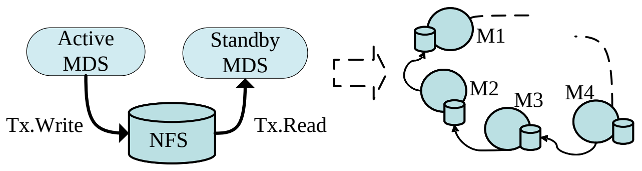


Figure 5. Standby-Node Mechanism

Theorem 3 All metadata of any MDS after its failure will not be lost but taken over by other MDSs.

Proof: It can be inferred from the DSVL architecture described in Section 3.1, the description of Standby Node mechanism in this part, Definition 5 as well as the checkpoint mechanism that every MDS must have a Standby Node, in which the log update of this MDS and the latest checkpoint data will be backed up. Hypothesize the failed MDS is M_f, the failure time is T_f, and the last checkpoint creation time of M_f is T_c, it's obvious that $T_c < T_f$. Allow the memory dump of M_f at T_c to be I_c, then the update record of I_c during $T_f \sim T_c$ is $SEQ_{log} = \{OP_0, OP_1, \ldots, OP_k\}$, in which OP_i is one updated log containing target data items, original data, new data, alteration type and timestamp. Obviously, I_c describes the latest state of the memory dump while SEQ_{log}, all the dynamic behavior of memory dump during $T_c \sim T_f$. As the information recorded by SEQ_{log} contains the original data, this dynamic process can be re-done. In other words, the dynamic alteration process of memory dump can be re-executed by redoing the log record as long as I_c and SEQ_{log} are both available. The latest correct memory dump can be shown as $I_n = I_c \bumpeq SEQ_{log}$. Therefore, the correct state of M_f can be recovered through latest checkpoint and log files.

Theorem 4 The failure of a certain MDS will have no influence on the service state of other MDS instances.

Proof: Allow the failed MDS to be M_f, M_s ш M_f, M_f ш M_t. It can be learned from Definition 2 that the metadata set of M_f, M_s and M_t is respectively $SK_f=\{K \mid K \in [K_f, K_s)\}$, $SK_s=\{K \mid K \in [K_s, K_x)\}$, $SK_t=\{K \mid K \in [K_y, K_t)\}$, and $K_y<K_t<K_s<K_x$, so $SK_f \cap SK_s \cap SK_t = \varnothing$. Computation for all MDSs accordingly can lead to $SK_i \cap SK_{i+1} \cap \ldots SK_n = \varnothing$, in which n is the number of MDSs. In accordance with the checkpoint mechanism, there is backup for log files and the latest checkpoint of M_f on M_s, while the backup for M_t is on M_f. After M_f fails, the backup data of M_t will be lost, but M_t can immediately create the latest checkpoint locally to be downloaded by the new Standby Node as it doesn't fail. Therefore, the failure of a certain MDS will have no influence on the metadata server state of other MDSs.

Theorem 5 The network load in the MDS failover is 0.

Proof: Assign M_f as the failed MDS and M_s ш M_f. M_f fails at time t. According to Definition 5 and the checkpoint mechanism, all updated logs before t and after last merging of checkpoints has been backed up on M_s, and the latest checkpoint is also on M_s. According to the DSVL consistency principle described in Section 3.1, the original metadata of M_f must reside in M_s after M_f fails, so the target MDS for failure recovery is M_s. On the other hand, there are already the latest checkpoint and log files of M_f in M_s, which, according to the checkpoint mechanism, can completely recover the memory dump of M_f. Therefore, failover takes place totally in M_s without causing any network load. The theorem is thus proved.

Corollary 2 DSVL has good MDS fault tolerance and failure recovery capability.

5. Experimentation

Prototype system is developed based on the Hadoop Project[6]. The default minimum MDS number in prototype system is 3, and the data sets used in the experiment are Windows XP Professional SP3, the metadata of Ubuntu-Desktop 12.04 LTS.

5.1. Lookup

Figure 6 shows the relevance between metadata amount and lookup performance with different MDS count. It clearly shows that metadata amount has very little influence on lookup latency. Figure 7 shows that the bigger the MDS count is, the smaller the lookup latency is, despite different concurrency. The DSVL architecture determines that every MDS, loaded with different metadata section, can serve simultaneously with high concurrence. Therefore, S is capable of rapid direct positioning with fast lookup speed as DSVL is based on numerous MDS and hash.

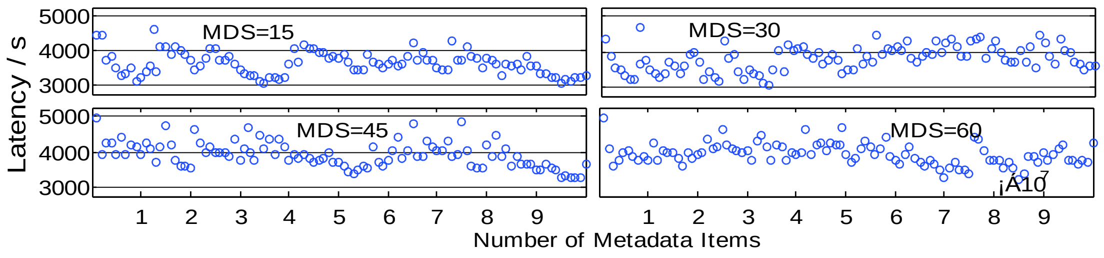


Figure 6. Metadata Amount VS. Average Lookup Latency

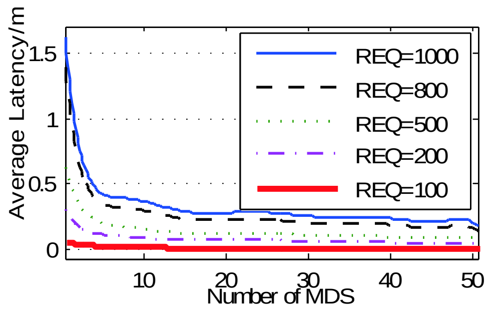


Figure 7. MDS Count VS. Average Lookup Latency

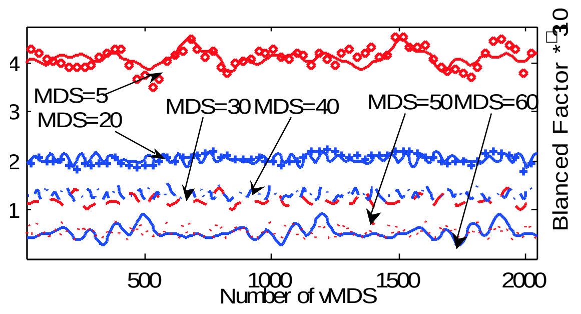


Figure 8. vMDS Count VS. Metadata Distrbution Evenness

5.2. Metadata Distribution Balance

Figure 8 shows the relevance between vMDS count and metadata distribution evenness with different data amount, in which vMDS Count refers to the number of virtual MDS nodes, and STDEV, metadata distribution evenness (standard deviation). Tests with different metadata amounts are done in the experiment, so the metadata distributed to MDSs need to be normalized. It can be seen that STDEV is at 10^{-3} magnitude, indicating good metadata balance in DSVL. STDEV fluctuates periodically but stays within a small range with the growth of vMDS count. The bigger the sample amount (metadata amount) is, the closer the experiment result is to reality because the *Key* value computed by the metadata items is random. STDEV reduces with the increase of metadata amount. The bigger the metadata amount is, the more dense and more overlapped the images are, indicating gradual stability. Figure 9 shows such relevance even more clearly: STDEV reduces gradually with the increase of metadata amount. STDEV decreases more quickly when the metadata amount is smaller than 0.8×10^6, and gradually stabilizes afterwards. In conclusion, big metadata amount is conducive to distribution evenness.

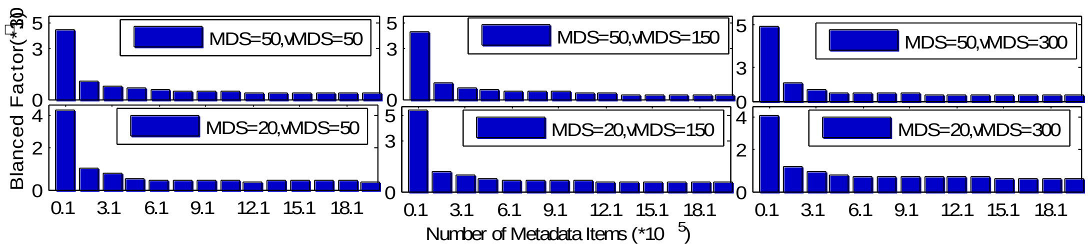


Figure 9. Metadata Amount VS. Metadata Distribution Evenness

5.3. Migration Overhead

As shown in Figure 10, the failure of a certain node causes big data migration when MDS count is small, while such data migration gradually drops and flattens out after 30 as MDS count grows. The data migration gradually increases with rising data amount even when MDS count remains small. Such gap, gradually shrinks when MDS count is 30 to 40. Therefore, the failure migration efficiency is highly enhanced when MDS count is 30 given the conditions of this experiment. It's also shown that the percentage of data migration amount in total metadata amount is smaller than 1:1000, or even 1:100000 when MDS count is 30, indicating extremely low logic failure migration in DSVL. Likewise, Fig. 11 shows the relevance between MDS count and data migration amount caused by new MDS node joining in cluster. However, no data migration will actually occur in the process of MDS failure and joining after latency strategy is adopted in DSVL. Therefore, the results of the experiment are based on the computation of logic metadata migration.

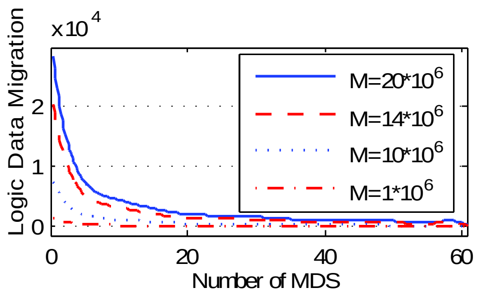


Figure 10. MDS Count VS. Logic Migration on Failure

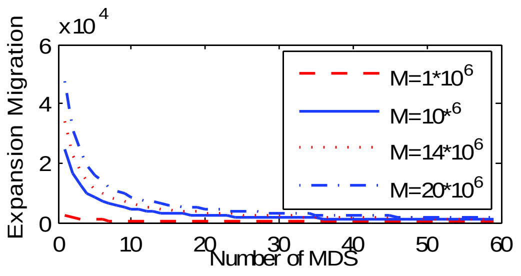


Figure 11. MDS Count VS. Logic Migration of Expansion

6. Discussions

Table 1 compares DVSL with other typical metadata management schemas. Hash-based schema performs well in load balance and metadata lookup but has to pay big cost for cluster alteration, so it's not applicable to clusters needing flexible scalability. Subtree-based schemas have good directory operation because its data structure is familiar with file system hierarchy semantics also experience big cost for scalability. Static-tree-based schema appears better than dynamic-tree-based schema in data migration. Hash-based DSVL comes with good load balance and quick lookup response as well as, because of some relevant improvements, low metadata migration amount and favorable scalability. In the meantime, however, DSVL still doesn't perform as well as sub-tree-based schemas in terms of directory operation, which needs to be further worked on.

Table 1. Comparations of Typical Metadata Management Schemas

	Schema Impl	Load Balance	Lookup Time	Migrate Cost	Scalability	Memory Overhead	Dir Operation
Single MDS	HDFS[6]	No	$O(\log n)$	0	No	huge	$O(1)$
Hash-based	zFS[10]	Yes	$O(1)$	Huge	High Cost	0	Low
Static Tree	NFS[7]Coda[8]	No	Large Latency	0	Much Migration	$O(1)$	$O(1)$
Dynamic Tree	Ceph[9]	Yes	$O(\log d)$	Huge	Much Migration	O(d)	$O(1)$
DSVL	DSVL	Yes	$O(1)$	0	Flexible Low cost	$O(1)$	General

7. Conclusion

The paper proposes DSVL based on consistent hash, a decentralized metadata management schema with good scalability and failover, low data migration, and balanced MDS load. Virtual MDS structure is added onto the consistent hash structure, enhancing the work load balance of the whole MDS cluster. The failure recovery and metadata migration mechanism based on Standby Node and latency strategy enable DSVL to have better MDS fault tolerance and zero-migration in the MDS cluster alteration process, greatly enhancing the system's scalability and reducing the cluster's upgrading cost. Future work will be: introducing auxiliary nodes to enhance dir operations and solving the potential problem that both standby node and master node fail.

Acknowledgement

This work is supported by the Natural Science Foundation of China(No. 61063012, No. 61363003), Guangxi Natural Science Fund (No. 2012GXNSFAA053222), Guangxi university talents support project(No. [2011]40), Guangxi Scientific and Technical Development Program(No. 1348020-7) and Nanning Scientific and Technical Development Program(No. 201109016A).

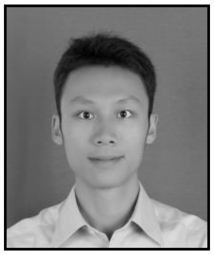
References

[1] A. Traeger, E. Zadok, N. Joukov and C. P. Wright, "A nine year study of file system and storage benchmarking", ACM Transactions on Storage (TOS), vol. 4, no. 2, **(2008)**.
[2] D. S. Roselli, J. R. Lorch and T. E. Anderson, "A Comparison of File System Workloads", USENIX Annual Technical Conference(General Track), San Diego, CA, USA, **(2000)** June 18-23.
[3] S. A. Weil, K. T. Pollack, S. A. Brandt and E. L. Miller, "Dynamic metadata management for petabyte-scale file systems", Proceedings of the 2004 ACM/IEEE conference on Supercomputing, Pittsburgh, PA, USA, **(2004)** November 6-12.
[4] Y. Hua, Y. Zhu, H. Jiang, D. Feng and L. Tian, "Scalable and adaptive metadata management in ultra large-scale file systems", The 28th International Conference on Distributed Computing Systems (ICDCS'08), Beijing, China, **(2008)** June 17-20.
[5] S. Ghemawat, H. Gobioff and S. T. Leung, "The Google file system", ACM SIGOPS Operating Systems Review, vol. 37, no. 5, **(2003)**.
[6] K. Shvachko, H. Kuang, S. Radia and R. Chansler, "The hadoop distributed file system", IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), Lake Tahoe, Nevada, USA, **(2010)** May 3-7.
[7] B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel and D. Hitz, "NFS Version 3: Design and Implementation", USENIX Summer 1994 Technical Conference, Boston, MA, USA, **(1994)** June 6-10.
[8] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel and D. C. Steere, "Coda: A highly available file system for a distributed workstation environment", Computers, IEEE Transactions on. Los Alamitos, vol. 39, no. 4, **(1990)**.
[9] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system", Proceedings of the 7th symposium on Operating systems design and implementation, Seattle, WA, USA, **(2006)** November 6-8.
[10] O. Rodeh and A. Teperman, "zFS-a scalable distributed file system using object disks", Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, San Diego, CA, USA, **(2003)** April 7-10.
[11] S. A. Brandt, L. Xue, E. L. Miller and D. D. Long, "Efficient metadata management in large distributed storage systems", Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage.
[12] D. R. Karger and M. Ruhl, "Simple efficient load balancing algorithms for peer-to-peer systems", Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures, Barcelona, Spain, **(2004)** June 27-30.
[13] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications", ACM SIGCOMM Computer Communication Review, vol. 31, no. 4, **(2001)**.
[14] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web", Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, El Paso, Texas, USA, **(1997)** May 4-6.

[15] V. Mateljan, D. Cisic and D. Ogrizovic, “Cloud database-as-a-service (DaaS)-ROI”, Proceedings of the 33rd International MIPRO Convention, Opatija, Croatia, **(2010)** May 24-28.
[16] D. Borthakur, J. Gray, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg and H. Kuang, “Apache Hadoop goes realtime at Facebook”, Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, Athens, Greece, **(2011)** June 12-16.
[17] Systems and Technologies, San Diego, San Diego, CA, USA, **(2003)** April 7-10.
[18] W. Ji-yi, F. Jian-qing, P. Ling-di and X. Qi, “Study on the P2P Cloud Storage System”, Acta Electronica Sinica, vol. 35, no. 5, **(2011)**, pp. 1100-1107.
[19] L. Ou, C. Engelmann, X. He, X. Chen and S. Scott, “Symmetric active/active metadata service for highly available cluster storage systems”, Proceedings of the 19th IASTED International Conference on Parallel and Distributed Computing and Systems, Anaheim, CA, USA, **(2007)**.

Authors

Chen Ningjiang, born in 1975. He received the Ph.D. degree in China from Institute of Software, Chinese Academy of Sciences in 2006. He is a professor at Guangxi University. His research interests include software engineering, distributed computing, etc.



Xiao Zhongzheng, born in 1988. He is a master candidate at College of Computer, Electronic, and Information, Guangxi University. His research interest is software engineering and parallel distributed computing.

Zhang Wenbo, born in 1976. He received the Ph.D. degree in China from Institute of Software, Chinese Academy of Sciences in 2007. He is an associate researcher at China from Institute of Software, Chinese Academy of Sciences. His research interests include software engineering, distributed computing, etc.