Journal of Software Engineering

ISSN 1819-4311

Academic
Journals Inc.

www.academicjournals.com

Journal of Software Engineering 8 (3): 203-210, 2014
ISSN 1819-4311 / DOI: 10.3923/jse.2014.203.210

Research on High-availability Distributed Storage Technology Based on K-V Model

YaoFeng Miao and Yuan Zhou
Modern Education Technology Center, Xi'an International University, Xi'an, 710077, China

Corresponding Author: YaoFeng Miao, Modern Education Technology Center, Xi'an International University, Xi'an, 710077, China

ABSTRACT

This study describes a highly available key-value store Dynamo system. The system is the part of Amazon's core business which provides "always-on" support services. Dynamo's main advantage is that it is a fully distributed, no central node of the system and can improve the performance of the system. It provides three parameters (N, R, W) whose value can be set according to users' needs. It has the full membership mode, where each node knows its peer bearer data. In order to provide this service, the Dynamo system using multiple versions of data and conflict resolutions of the application support solves the data consistency eventually.

Key words: Dynamo, distributed storage technology, nodes, K-V model

INTRODUCTION

Dynamo (Charbonneau, 2010) is a fully distributed and has no central node of the storage system. Compared with traditional centralized storage systems (Hill *et al.*, 2012), Dynamo was located in a high-reliability, availability and fault tolerance system early in the designing. Practice shows that the availability and scalability of Dynamo known as a key-value storage platform model are very good and the performance is also good. The response time of the access of 99.9% read-write is within 300 msec. The Dynamo storage platform is composed by many multiple physical machines. The role of each machine is the same, they can be added or removed at liberty which does not require too much human intervention. Each machine can store part of the data, the backup of the data is fully synchronized by the system. The power failure of a single machine or even the data center will not have any effect on the availability of the external system and it is a distributed data storage system with high availability and high scalability.

Now, the size of the network becomes larger and larger (Liang, 2006). The requirements of storage rate, throughput capacity and performance are very high and it is very difficult for the traditional database to take these into account. According to the CAP principle (consistency, availability, partition tolerance) (AlSallut *et al.*, 2013), the traditional database has no partition tolerance and the data is stored centrally. Storage rate may not be a problem but capacity and performance have some problems (Cesarelli *et al.*, 2009; Sedgwick *et al.*, 2005). In order to prevent the single point failure, the data must have backup and copy. Under strong consistency requirements, there must be at the expense of the performance loss. With the rapid increasing of business, there is more and more data and the scalability of the traditional database is also a problem (Li *et al.*, 2005; Lian and Chen, 2010). Hanging hard drives for a powerful machine will not be able to solve this problem, in this case, Dynamo emerges as the times require.

CORE RESEARCHES ON DYNAMO

Dynamo is Amazon's key-value storage platform model and its availability and scalability are very good. Dynamo is mainly used for applications that require the "always-writable" data store and the concurrent write or update operation is rejected by a failure. It is built on the infrastructure whose nodes are considered to be trustworthy. Dynamo is designed for delay-sensitive applications whose response time of the access of 99.9% read-write is within 300 msec. Its several core technology researches are given below.

Distributed hash table: In order to achieve the design without the center, DHT based on key-based routing strategy can find the corresponding storage node:

- A ring is composed of the whole distributed system and it is divided into corresponding regions depending on the number of nodes
- Each region is represented by a token whose value range is (n-1, n]
- Each node is responsible for a region

By using the MD5 hash algorithm, key values are assigned to areas on the ring:

- Let the number of nodes be N
- MD5 algorithm will output a 128 bit integer, to facilitate the representation and the negative is removed, have 0≤token≤2^127
- For any node, its assigned range is (token n-1, token n], i.e., the max previous node is the node value. It will be responsible that the MD5(k) can correspond to all the key value
- There must be an end to end region(wrapping range). The region must include a minimum token value, typically left > right and the assigned value is divided into two parts:
  - MD5 (k) = (0, token 0], i.e., from the minimum value of the node to the minimum of the range (due to removing the negative, so the minimum is 0)
  - MD5 (k) = (token n-1, 2^127]

Membership: For users, all the nodes in the cluster are the same. In fact, seen from the DHT principle, each node is only responsible for the different parts of the entire data set. However, each node has to serve any specific requests. To achieve this transparency of users, each node must maintain some original data:

- All other nodes' lists
- The current status of all nodes
- For a given key, it is mainly responsible for the node and the n-1 node copy (n is a replication factor)

There are three status for a node in the cluster.

A node is added to the flow Token Ring:

- Start the gossip service (communicating with other nodes to obtain the equal information)
- Start the message service
- Launch their loading information of each node

- If there is a new node, do
- If the configuration specifies a token for the new node, use it, otherwise
- Based on the load information of nodes and get the maximum loading current node and get a token from it. The token will share approximately half of its loading
- Local updating and save the token
- Gossip status information to all other nodes and the next boot will do a bootstraping
- Starte the boot process, do:
  - Based on the configuration information of keyspace and get the list of nodes who can be responsible for a portion of the data
  - Initiate pipelineIn request to them and pass this part of the data
- If it is a started node, reading saved token from the configuration table
- Update the new token in the local system information application
- Gossip status information and notify other nodes

A node leaves the flow Token Ring:

- Gossip upcoming is in the leaving node status
- Readjust the range information:
  - Get all of the affected range
  - Get all new nodes who are responsible for these range
- Calculate the data transmitted to other nodes, do:
  - Get all areas charged by departing nodes
  - Obtain these areas of all copies which are responsible
  - Calculate how the token allocated after the node leaves
  - For each Range in step 1, calculate their copies of the address
  - Calculate the difference between the range: The rest range is ultimately needed
- Pipeline out the data to nodes which will replace current nodes
- The current node is removed from the token metadata
- Gossip messages to other nodes
- Stop the gossip of the node
- Stop the message service of the node

Changes of the response members: Different state transitions are somewhat different when exiting nodes receive status changes but do generally the following:

- State inspection to ensure that there is no conflict
- Record the status of the corresponding node for verification
- Update (add, modify and delete) system metadata
- Updates the Range Information, do:
  - Get all of the affected range
  - Get all new nodes which are responsible for these range

Merkle tree: Merkle Tree (MT) is called Hash Tree. It has the Hash tree structure and it is put forward by Merkle (1979). It is very easy to know the data structure of MT. In fact it is simply a hash tree. The values of leaf nodes are hash values and the values of the non-leaf nodes are the values calculated by its child nodes. In Dynamo, each node holds a range of key values and there

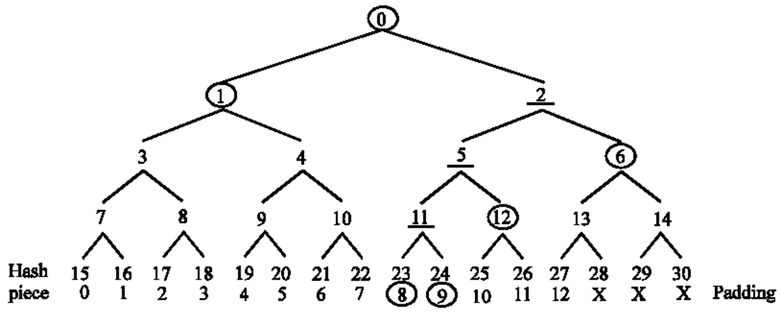

Fig. 1: Data blocks

is a range of overlapping between different node keys. In the operation of removing the entropy, what can be considered is only the two nodes with a range of key values. The leaf node of MT is that is the hash of each key in the shared value range. A MT can be built by the hash leaf nodes from the bottom. Firstly, Dynamo calculates the hash value of the MT root. If they are equal, they are exactly the same. Otherwise, its child nodes will be exchanged and the comparison process has to be continued.

For example, as shown in Fig. 1, let a file have 13 data blocks. We can supplement it to 16 blocks (note that the supplement blank block is only used for assisted verification and it actually does not have the function of data transmission). Each block corresponds to a SHA1 checksum value and repeately pairwise hash until getting a unique root hash value (root hash, H0). This calculation constitutes a binary Merkle hash tree and bottom leaf nodes (H15~H30) correspond to the actual hash value of the data blocks and those internal nodes (H1~H14) are called "path hash value". They constitute a "verification path" between the root hash H0 and the actual hash value. For example, the actual hash value of the data corresponding to the actual block 8 is H23 and SHA1 (((SHA1 (SHA1 (H23, H24), H12), H6), H1) should be equal to H0. Course, the above validation process can be done by n-ary hash tree and the process is similar.

The benefits of MT are nothing but time and space. Under the distributed case, the space can be understood as the corresponding network transmission data. For time, MT whose time complexity is O (lgn) can avoid comparing the linear time by using the tree structure and quickly locate the different key value. For network transmittion, if making the comparison by linear, all hash values must be transferred in the range of the key. But for MT, only a hash value of the checked layer can be obtained which greatly reduces the amount of data transmittion.

Hinted handoff: Anti-entropy permanent mechanism is only used for error recovery, on the contrary, Hinted handoff is used in the status that the node failure is low and short.

If only obeying the compliance with the W+R>N and (W□1) strategy, it means that the system has to make at least a host alive, then the write operation can succeed. To improve the availability, in some specific cases, for a key, all nodes having the primary responsibilities are short-term failures but other parts are normal, such as network segmentations between data center. Relaxing this requirement, system availability can be improved.

Hinted handoff is a short way to handle system short failures. When all the N responsible nodes fail, they attempt to store information in a special non-primary position and they note a hint which contains the real target node information. When the service receives a gossip signal that new node recovers from the failure, it checks whether the data need to handoff. By checking whether the node is referred to hint, if yes, the node that contains the hint will handoff the copy.

Process of hinted handoff: Hinted handoff is implemented as a background thread and a queue can record all hint information:

- Get a pending hinted-handoff from the queue
- Nodes under handover work can get the relevant information through the special position mentioned above
- Based on the above hint receiving information of the target, send the target node via message center
- After the receiving node receives a message, use this hint to local area

Read/write: Any storage nodes in Dynamo are eligible to receive a client's read and write operations to any key. In order to maintain copies' consistency, consistency used by Dynamo is similar to the arbitration and the agreement has two key configuration values: R and W. R is the minimum number of nodes involved in a successful read operation and W is a minimum of successful write operation nodes. In Dynamo, set R and W, such that R+W>N.

In the Dynamo system, N, R, W is configurable tuning. According to the final evaluation results, have (N, R, W) = (3, 2, 2). In other words, there will be three copies of a data. When writing, two nodes returning means that writing is successful and the transaction is successful. As write opreation, reading at least two nodes is considered to be success.

Dynamo's main advantage is that its client application can adjust the value of N, R and W in order to achieve its anticipated performance, availability and durability levels. For example, the value N determines the durability of each object.

For the consistency between the data, Dynamo has three coordination and arbitration mechanisms:

- **Business logic specific coordination:** This is a common case by using the Dynamo system. Each data object is replicated to multiple nodes. When bifurcation occurs, the client application performs its own coordination logic
- **Based on coordination timestamp:** This case is different from the one in the coordination mechanism. If there are many different versions, Dynamo performs simple logic-based coordination timestamp: "Last write wins", that is to say, the object with the largest timestamp is elected to be correct version
- **High performance engine reads:** Although, Dynamo are built into an "always writable" data storage, some services by adjusting the characteristics of its arbitration regard it as a high-performance engine. Typically, these services have a high rate of read requests. In this configuration, have R = 1 and W = N

Replication: In order to achieve high data availability and persistence, all nodes are required to backup. In Consistent Hashing Ring, the subsequent node backups the previous node. For example, B, C and D are backups of A and every backup has three backups. That is to say, the key value K exists on four nodes.

Although, the K value exists on four nodes but the four nodes will be recorded in a persist list. To this example, let list = (A, B, C, D), only the first node is responsible for the operations of value K and copies it to other nodes. When node A fails, subsequent nodes can be handled by the persist list.

Two points need to be considered, one is the virtual node and another is the number of backups. Mentioned above, each node on Consistent Hashing Ring is a virtual node and A, B, C and D may be located on the same physical node. The backup is meaningless. If this node fails, all backups will be lost. When selecting a backup node, the selection of different physical nodes must be ensured. For the number of backups, there is hard to lose data but the inter-backup synchronization overhead increases. In the Amazon system, the number of backup is 3.

Vector clock: By using vector clock, Dynamo can catch causation among different versions of the same object. Vector clock is actually a (node, counter) list. Vector clock is associated with each version of each object. By reviewing its vector clock, an object's two versions can determined whether it is parallel branch or causal sequence. If the object on the first clock counter is less than or equal to all other nodes on the second counter clock object, then the first one is the ancestor of the second one. Otherwise, these two changes are considered to be a conflict and the coordination is required.

When the client updates an object in Dynamo, it must specify which version is going to be updated. By passing the context object receiving from the early read operation, it contains the vector clock' information. When processing a read request, if Dynamo accesses to a plurality of syntactically reconciled branches, it will return all the leaves of an object' the branches which contains the version information corresponding to the context. Using this context update operation is believed to have coordinated the different versions of the update operation and branches are collapsed to a new version.

SPECIFIC REALIZATION

In dynamo, each storage node has three main software components, request coordination membership, fault detection and local persistence engine. All of these components are implented by Java. Local persistence component of Dynamo allows to insert different storage engines, such as Berkeley Database (BDB version), BDB Java edition, MySQL and a persistent backing store with a memory buffer. Designing a pluggable persistence component is to choose the most appropriate storage engine in accordance with the application's access patterns. For example, the object can normally be handled tens of kilobytes of magnitude by BDB and MySQL can handle larger objects. According to the objects' size distribution, the appropriate application of local persistence engine can be selected. In production, Dynamo majority uses BDB transactional data for storing.

Request coordination component is built based on event-driven communications, where the message processing pipeline is divided into several stages. All communications are using Java NIO Channels. Coordinator performs the read and write operations. By collecting data from one or more nodes, or in one or more nodes from the storage data. Each request of the client will cause that the node receiving the client requests will create a state machine. Each state machine contains the logic such as identity a node in charge of a key, send a request and wait for response, possible retry treatment, processing and packaging of the response back to the client. Each state machine instance handles only one client request. For example, a read operation can achieve the following state machine (1) Send a read request to the appropriate node, (2) Wait for the required minimum number of responses, (3) If there are few responses received in a given period of time, the request fails, (4) Otherwise, collect all versions of the data and determine to return version and (5) If enabling the version control, coordinate the implementation of the syntax and generate a write client opaque context which includes vector clocks of all remaining versions. For brevity 's sake, do not contain trouble shooting and retry logic.

After the reading response return back to the caller, the state machine waiting for a short time accepts any pending response. If any response return the obsolete version, the coordinator will be updated with the latest version of these nodes. This process is called read repair because it is used to repair an update and read repair can eliminate anti-entropy operation.

Write request is coordinated by, although choosing the first node of the former N nodes to coordinate is possible, all write serialization approach in a single location will lead to load distribution which led to violation of SLA. To solve this problem, the ranked N nodes in the preferred list allow to coordinate. In particular, since the write operation is usually followed by the read operation, the coordinator of the write operation node serve as the fastest reply. Because this information is stored in the request context (referring to the write operation request). This optimization allows to choose data nodes that have been used which can improve the consistency of "read-your-writes".

CONCLUSION

Dynamo has been used for a long period and proved its high availability (99.9995% of the applications can receive successful response (no timeout). So far, there are no event of data loss. Dynamo is a fully distributed, no central node of the system and can improve the performance of the system. Although, this design will bring the consistency issue, the collection through a variety of techniques can be used which can effectively solve this problem. Dynamo provides three parameters (N, R, W) and users can set the value of them according to their needs. Unlike popular commercial data storage, Dynamo make the logical data consistency and coordination problems expose to developers. At first, people may think the application logic becomes more complex. However, from a historical view point, Amazon platforms are built for high availability. Finally, Dynamo has the full membership mode, where each node knows its peer bearer data. Each node needs to actively work with routing tables of other nodes in the system Gossip. This model in a system with hundreds of nodes works well, however, the expansion of this design is not easy to run thousands of nodes. Because the cost of maintaining the routing table will increase as the size of the system increases. This limitation may be overcomed by the tiered extensions for Dynamo.

REFERENCES

AlSallut, A.Y., H.H. Hejazi and H.A. AbuGhali, 2013. CAP* Theorem vs. CAS** principle in computer networks. Int. J. Adv. Innovative Res., 2: 396-400.

Cesarelli, M., M. Romano, M. Ruffo, P. Bifulco, G. Pasquariello and A. Fratini, 2009. PSD modifications of FHRV due to CTG storage rate. Proceedings of the 9th International Conference on Information Technology and Applications in Biomedicine, November 4-7, 2009, Larnaca, pp: 1-4.

Charbonneau, P., 2010. Dynamo models of the solar cycle. Living Rev. Solar Phys., Vol. 7. 10.12942/lrsp-2010-3

Hill, C.A., M.C. Such, D. Chen, J. Gonzalez and W.M. Grady, 2012. Battery energy storage for enabling integration of distributed solar power generation. IEEE Trans. Smart Grid, 3: 850-857.

Li, B., S. Liu and Z. Yu, 2005. Applying MDA in traditional database-based application development. Proceedings of the 9th International Conference on Computer Supported Cooperative Work in Design, Volume 2, May 24-26, 2005, China, pp: 1038-1041.

Lian, X. and L. Chen, 2010. Ranked query processing in uncertain databases. IEEE Trans. Knowl. Data Eng., 22: 420-436.

Liang, X.B., 2006. Matrix games in the multicast networks: Maximum information flows with network switching. IEEE/ACM Trans. Networking, 14: 2433-2466.

Merkle, R.C., 1979. Secrecy, authentication and public key systems. Technical Report No. 1979-1, June, 1979. http://www.merkle.com/papers/Thesis1979.pdf

Sedgwick, F.G., C.J. Chang-Hasnain, P.C. Ku and R.S. Tucker, 2005. Storage-bit-rate product in slow-light optical buffers. Electron. Lett., 41: 1347-1348.