Indonesian Journal of Electrical Engineering and Computer Science
Vol. 14, No. 3, June 2019, pp. 1420~1424
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v13.i3.pp1420-1424

Cluster based mechanism for hot data storage in HBase system

Mohammad Musa Al-Momani
Department of Management Information Systems, Zarqa University, Jordan

Article Info

Article history:

Received Nov 25, 2018
Revised Jan 21, 2019
Accepted Feb 27, 2019

Keywords:

Database systems
HBase
Hot data storage
Substitute regions
Unbalanced accessing

ABSTRACT

The evolution in different areas like e-business contributed to increase the number of data. So, a scalable database is required to accommodate a large number of data. Moreover, this data could be important and the number of accessing for reading or writing operations can make a pressure in some servers more than others, this pressure called unbalanced accessing. This will be achieved if a system that guarantees the distribution of this important data in useful form is used, because the pressure can causes a delay. Accordingly, when we design a multi storage node to keep this data, the pressure will transfer from a server to another. So, the proposed system searched for a satisfied solution to distribute all the users on the servers in a useful way to detect any mistake or repeating updating operation data by using identification feature. This method is applied on Hbase and called Clusterization.

Corresponding Author:

Mohammad Musa Al-Momani,
Department of Management Information Systems,
Zarqa University, Zarqa 13132, Jordan.
Email: dr.mohammadmomani@yahoo.com

1. INTRODUCTION

The database management systems have been used in many information technology domains, and the types of databases depend on the domain which is used. The huge number of data requires a large database with a high storage capacity in order to serve a large number of users [1]. Accordingly, many kinds of database are available and used by large and famous companies. Cassandra [2] is an open source database system (DBS) used in Facebook. The flexibility of this DBS makes the operations like insert, delete and update very easy to do. In addition, a failure to any node occurs will not affect the cluster head node. The second common type is HyperTable [3], it's an open source distributed data storage; this system can include about 1000 nodes. The third type is Mongo database [4], it is between relational and non relational database. CouchDB Apache [5] is a flexible database that is based on ACID features: Atomicity, Consistency, Isolation and Durability. Finally, Hbase [6] is an open source distributed value which is based on Big Tables used for a big amount of data and can deal with complicated cases[7].

When the users are calling a specific data for many times, this will make the server and users face a delay to get the data. This is unacceptable, especially when this hot data (important data) situated only in one server. For that, there have been previous studies tried to solve this kind of problem. One of the studies proposed a new mechanism to avoid the unbalanced data accessing. This mechanism has multi storage nodes that include a copy of a hot data to serve the users without any delay and reduce the load on the server [8]. The system model decides the hot data based on the two factors. The first factor determines the heating of data by a "visit ratio", which means the number of access times in per unit time, and the residence time means the period of the data that still in the memory. Thus, if the data stays for a long time in the memory it will be accessed frequently, and that makes it hot. The other mechanism used for storing data [9] is by copying the hot data record in a multi storage node not as a single node like before in order to allow the parallel access for users at the same time. The first weakness point here is that this method used a Least Recently Used (LRU) policy, but it will be more useful to use a Most Recently Used (MRU) method.

When the user calls the hot data record from node and the memory of this node is very short (which means it does not save all requested data for a long time), it will lead the system to search for this hot data record in the whole server (node). This operation will take a long time and is not useful, so we proposed using (Most Recently Used) MRU policy to make the hot data record stay in the memory as much as possible to reduce the effort of the system and save time. This policy will be fast and will not make the user get exposed to delay [10].

The second weakness point is that the previous method designed multi storage nodes to make the important data available in more than one position to ensure distributing the users and decrease the load on the servers. But this is not a satisfactory and insufficient solution because it will not do a real distribution for the users to avoid the unbalanced accessing. Consequently, the users will face a delay if they are grouped in one node and put it under pressure along with another node without any load (as shown in Figure 1). Accordingly, the goal of this paper is to make a real and satisfied distribution for this hot data and avoid any delay for users and load for server. Thus, this paper proposed a clusterization method to achieve this goal and we will explain this method in the clusterization section. The last weakness point of the previous method does not give any identification number for the hot data record to distinguish this hot data record from the other data. According to that, this study aims to give an accrued identification for each hot data record to distinguish the original copy from other copies. In addition, this id can detect any update that is not transferred from the original data to the other copies. Also, this id can detect if the update message comes from the original copy or not, i.e., it detects any manipulation.

2. DESIGN OF HBASE [8]

In order to solve the unbalanced accessing from the original region or the original node that contains the source of the hot data record. The suggestion of the previous method is to get copies of this hot data record and distributes them in the other regions (servers or nodes) called a substitute region. In this way, the mentioned method will allow the parallel users accessing in the same time and reduce the heating for the original substitute. All these regions including the original and the substituted ones are connected with one region server to ensure that the full data recovery of updating records source is applied to all the copies. But it could not ensure that the update message reach from the source record to the other copies correctly. This method does not also distinguish the hot data record of the source and its copies and does not give any correct number of the updates. So the proposed method give an identification for the original region server and for each hot data copy to distinguish which substitute regions need to update and which one is the source region or source node. The next example will explain the method: Region server WAL

Data record X0 copy 50.0	Data record X1 copy 50.1	Data record X2 copy 50.2

Where 0 is a source data record to make the region server or the node server be distinguished as an original copy. The original copy has reading and updating accesses, while other copies have reading access only. The copies will be from the x1 to xn, This feature will be useful when the source data record needs to update the data. It will send the WAL message that contains the update message to the other copies. When the node or substitute region server received the WAL message coming from the identification of source copy, it will be sure that the updating comes from the original resource, and it will change the old data record. This method helps to update all copies from one place only and distributes all the copies correctly.

To make the updating operation reaches to all the copies but with the new identification that we proposed; the original region that needs to update will send a WAL message to the zoo keeper. Zoo keeper is a table exists in the root region server and saved the root table information with its position, and the WAL message of x0 record. The root region server (the root region table) consists of the whole substituted regions' names that contain the data record names and their id. Every substituted region server will see the WAL message, then the substituted region server checks if it has a copy from the source hot data record x0, and whether it is updated or not. If the copy is not updated, the server will take the WAL and update it. Remark: The source hot data record is the only source who can update the data, while the copies from 1-n have a reading access only.

If there are A, B, C nodes as shown in Figure 1, in order to reduce the heavy load accessing of data, let's take a specific hot data record copy from node A and copy it in node B and C as multi storage node method. This may reduce the heavy load as shown in Figure1. If node B has load more than node C and A, this will not be a useful solution because it will delay the users and make the node under pressure. This means that one other single node becomes hot and the other one stays normal. This will not work usefully.

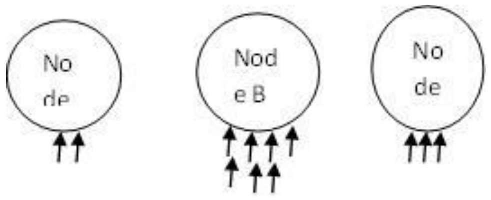

Figure 1. A, B, C Nodes

Accordingly, the solution is in substitute regions method. For example, if there are three nodes A and B and C, the source node here is A that contains the original copy of the data items that's hot. If node A will update this hot data item, it has a written head log file (WAL) to enable it to modify the hot data. After that, node A will send an update message log file to node B and C to modify for their copies of hot data record. Node B and C have a read access only to ensure full data updated recovery. when node B and node C receive the update log file they will do lock operation for the reading accessing operations until they finish the updating operation, and this step will cause delay for the users[11].

3. THE PROPOSED METHOD (CLUSTERIZATION)

Figure 2 shows the cluster nodes like c1, c2, c3, c4 related with nodes called cluster members like x1, x2, x3. The gateway nodes that are related with the cluster head nodes only like Gw1, Gw2.

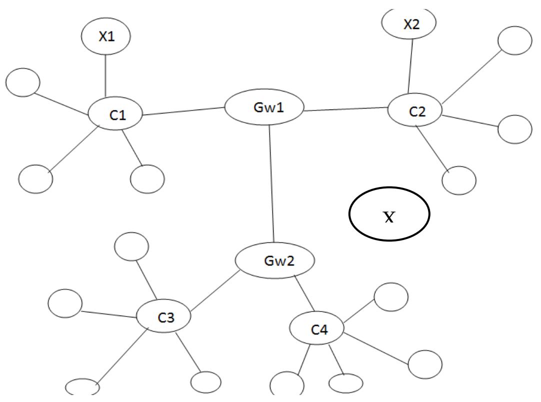

Figure 2. Cluster nodes

Step 1: to reduce the unbalanced accessing distribution (as shown in Figure 1) and reach a logical solution, we proposed after testing the heating based on the two previous factors such as visiting ratio in pair unit time, and the residence time for the hot data record in the memory. We proposed the residence time will be based on MRU policy in order to keep the hot data in memory for a long time. If the hot data is requested when it is outside the memory, the system will take a long time searching for it on the contrary to the previous method that used LRU policy.

Step 2: To solve the problem in step 1, making the hot data record X2, X3 as copies of the original record X1. Each copy of them should be related to the cluster node(c), which means the original X1 related to c1, the copies X2 related to c2 and X3 related to c4 (see Figure 2). According to that, we can distribute the users on the clusters, including the cluster which contains the original hot data record. When the users' access to the hot data x they want, they may get it from c1 or c2 or c4. This means that if the users access the record of the hot data x from c2 like X2, and has a very heavy load, they will expose to delay; while, X3 in c4 has no heavy load or the original record like X1 has no pressure of the accessing operation without being known by the users. Accordingly, the proposed method here is to make all the accessing requests of the hot data x coming from users go directly to the gateway1. This node has log that contains all the links of the main cluster nodes and their related cluster members with all data records. Thus, when the accessing request goes to the gateway 1, it will get the request of hot data record from the user, and then it will check whether the cluster node does exist and find it according to the history which in turn will find all the cluster nodes that have the copies of this hot data record. Then it will bring the source and copies records like X1, X2 and X3. The gateway node will test the heating of these records to decide which one of them has a little access. This means that the gateway node will choose the fewest heat or the fewest load of these three records. Finally, after deciding the fewest load nodes, it will bring this record to the user to avoid the delay problem. For example, if the user calls X1 data record and it has a heavy load of accessing, the gateway node will apply the same steps to give the user the fewest data records, and test the percentage of heating of X1, X2 and X3. Then it discovered that the X3 has the fewest load compared with X1 and X2. So it will send the request of the user to X3. In other words, the gateway node does a heating test for the selected data records and then sorts them according to the fewest load in the first priority and so on[12].

Note: The gateway1 node will call the gateway2 node and check if it has a copy from the selected data record. Then the gateway2 node will check its history, and if it finds it, it will take the data record X3 and send it back to the gateway1 node to do the heating testing procedure. The heating testing levels are classified into:

a) gentle
b) normal
c) hot

Example: for more understanding, suppose the heating ratio = 10 and the user request is from x3, the gateway node will check its log x1 and x2 but x3 will get it from c4 by gateway2, which means it will ask about the whole copies in all linked clusters and gateways. Then, suppose that this node does the heating testing and discover that X1=11, X2=10 and X3=7. These results are classified based on the previous levels as the following:

1) Hot > The heating ratio
2) Normal == The heating ratio
3) Gentle < The heating ratio

So the gateway1 node will sort them ascending from low heating ration to high like X3,X2 and X1 and give the user the fewest load node which is X3 that is a gentle mode. Finally, the proposed method collects all the options and gives us the best one.

Step 3 (updating method): This method proposes that every cluster head node has a substitute region like c1, c2 and c3. When x1 hot data record needs to be updated, the proposed method makes an identification to distinguish the cluster node and its hot data record.

5:10.0.1

This expression means that 5 is the cluster number, 10 is the number of the hot data record that needs to access, and zero is the id to distinguish if it is an original copy or other copies. If it has any number except zero like any number from 1 to n, this means that they are copies. The last number is one, this number is related to the number of updates, which means this is the first update to ensure or guarantee that the update will reach all the copies.

Gateway1 will read this expression and the WAL that includes the update data from X1 will check its log about all the copies of X1 like X2. Then it will ask any linked gateways for the X1 copies. Then it will obtain the X3 copy from the gateway2 node, and then send it back to gatway1.

Now, the gateway1 node will check according to this expression X2 and X3 to ensure if they are A copies and check the last number (in the right) of the id that includes the number of updating times. This means that if WAL has number one (first update message) like WAL==1, X2 and X3 should be sequentially 2.10.1.0, 3.10.2.0. If the last part is not 0, the gateway node will not send WAL to prevent the repeating update messages.

4. CONCLUSION

This proposed method, compared, with many used methods that are mentioned in this paper, reaches the optimal and satisfied solution. Using the clusterization method as a new method could solve the unbalanced accessing distribution and the heavy load of the hot data record problem.

In addition, the proposed method has solved all the mentioned weakness points and helped the users not to expose to any delay. The clusterization method is the appropriate method used to find the best copy of the data without any delay for the users and any pressure in servers.

Finally, this proposed method does an updating operation form that ensure distributing all the update log message or the WAL for all the copies with no mistakes that could happen to the updating times by their identifications to simplify the failure recovery. Thus, this identification makes the nodes distinguish if the updating message comes from the source data record or not to prevent any manipulation on them.

ACKNOWLEDGEMENTS

"This research is funded by the Deanship of Research at Zarqa University/Jordan"

REFERENCES

[1] Han, J., Song, M., & Song, J. (2011). A Novel Solution of Distributed Memory NoSQL Database for Cloud Computing. *10th IEEE/ACIS International Conference on Computer and Information Science.* IEEE.
[2] Lakshman, A., & Malik, P. (2010). Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 35-40.
[3] Khetrapal, A., & Ganesh, V. (2009). *HBase and Hypertable for large scale distributed storage systems.* Purdue University.
[4] Boicea, A., Radulescu, F., & Agapin, L. I. (2012). MongoDB vs Oracle -- Database Comparison. *3rd International Conference on Emerging Intelligent Data and Web Technologies · EIDWT 2012*, (pp. 330-335).
[5] Nayak, A., Poriya, A., & Poojary, D. (2013). Type of NOSQL Databases and its Comparison with Relational Databases. *International Journal of Applied Information Systems (IJAIS)*, 16-19.
[6] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., & Wallach, D. A. (2006). Bigtable: A Distributed Storage System for Structured Data. *7th Conference on USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (pp. 205-2018). USENIX Association.
[7] Stonebraker, M., & Cattell, R. (2011). 10 Rules for Scalable Performance in Simple Operation' Datastores. *Communications of the ACM, 54*(6), 72-80.
[8] Zhuang, H., Lu, K., Li, C., Sun, M., Chen, H., & Zhou, X. (2015). Design of a more scalable database system. *IEEE/ACM International Symposium on Cluster, Cloud Grid Computing*, 1213-1216.
[9] Kallman, R., Kimura, H., Natkins, J., Pavlo, A., Rasin, A., Zdonik, S., ... & Abadi, D. J. (2008). H-store: a high-performance, distributed main memory transaction processing system. *Proceedings of the VLDB Endowment*, 1(2), 1496-1499.
[10] Vora, M. N. (2011, December). Hadoop-HBase for large-scale data. In Computer Science and Network Technology (ICCSNT), *2011 International Conference on IEEE,* Vol. 1, pp. 601-605.
[11] Cattell, R. (2010). Scalable SQL and NoSQL data stores. *ACM SIGMOD Record, 39*(4), 12-27.
[12] Peng, D., & Dabek, F. (2010). Large-scale Incremental Processing Using Distributed Transactions and Notifications. *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation* (pp. 251-264). Vancouver, BC, Canada: USENIX Association.

BIOGRAPHIES OF AUTHORS

	Mohammad Al-Momani graduated from Al-Balqa Applied University in 2006 with a B.S. in Computer Information Systems (CIS) and in M.S. degree in CIS in 2009 from Near East University (Cyprus). In 2014, he received his PhD in Management Information Systems (MIS) from Girne American University (Cyprus). Dr. Al-Momani is now an Assistant Professor at Zarqa University (Jordan) and is currently serving as a head of MIS department. His research interests focus on Networks, Databases, Online learning and Information systems and Sciences.