IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 16, Issue 2, Ver. XI (Mar-Apr. 2014), PP 36-41
www.iosrjournals.org

Improvising Data Locality and Availability in Hbase Ecosystem

Shalini Sharma[1], Satyajit Padhy[2]
[1](M. Tech (CS&E), Amity University, India)
[2](M. Tech (CS&E), Amity University, India)

Abstract: *In this paper, we try to represent the importance of data locality with the HBase architecture. HBase has a dynamic master slave architecture but the emphasis on data locality, i.e. getting the logic or processing near to the data is the major phenomenon followed for better and efficient performance. Data Locality is valid as every region server has the information of every data blocks located in respective regions but what if the region server crashes or the region server is restarted or the regions are randomly re-distributed with all the region servers due to load balancing, then data locality is completely lost during that time. Performance is majorly affected if there is misconfiguration of data locality in the cluster. The HMaster uses [4] .META table to get information about the region server that has its specified regions containing rows. Keeping an eye on this disadvantages and challenges, we propose to improvise data locality by allocating maximum regions to that region server which had the maximum data blocks of that region in it. An algorithm is proposed based on HRegion locality index for deciding the criteria of allocating the regions to region servers for maintaining data locality.*

Keywords: *Data Locality, HRegion, HDFS, locality index, MapReduce, Region servers*

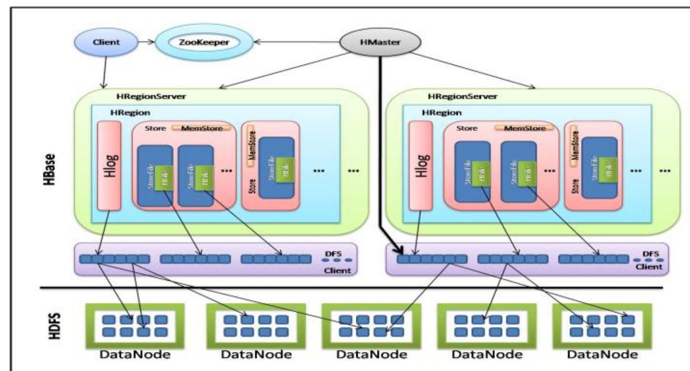
I. Introduction

Hadoop is a project that is meant to store and process large sets of data, probably in petabytes or zetabytes by using MapReduce algorithm. MapReduce algorithm is specifically used for distributed processing on data sets by using mappers and reducers. On the other hand Hadoop Distributed File System [2] is a distributed file system that is used to store large sized files in it. Whenever there is a situation to do effective lookups for records in files in HDFS, it is not possible due to obvious reasons. HBase comes in to picture in this situation where it allows these kind of fast lookup operations. HBase is a NoSQL database that is situated on top of HDFS. It is a column based database which is basically used for storing large volume of data. The main reason why we should prefer HBase instead of a relational database is due to scalability.

HBase is a columnar database [1] that is actually responsible for storing large volume of data in the rows of a table. HBase stores all its data actually in Hadoop Distributed File System. A set of rows from a table is comprised as a region and a set of regions are governed by a region server. The .META table is responsible for saving all the information regarding all the regions that are located in their respective region servers and even of its location in HDFS. A region server is always run in a datanode and most importantly that is one of the important reason data locality is achieved in HBASE. When write request comes to region server, it first writes into memory and commit log and it is only later when it stores the data or flushes the data into HDFS.

It is quite evident that in order to achieve greater performance, we need to take the processing near to the data rather than migrating the data all the way near the processing. For this very fact Hadoop even comply strictly to data locality which helps in locating every datanode at every instant of time from the Meta directory. When a large file arrives at the namenode, the namenode distributes it into many blocks and many of these blocks are stored in a datanode. Every block in turn has two replicated blocks in different datanodes. Similarly, when Hadoop MapReduce has to process the data that resides on HBase [3], data locality is again a key criteria that has to be followed here. As discussed above, the region servers are responsible to keep track of all the data that are located in its contained regions so that every task can be mapped accordingly. Whenever a job has to be mapped to one of the region, then the HMaster looks for the appropriate location of the region that is contained in some of the region server by fetching the information of host name and region information from the HTable [1].

If the region server is restarted, usually avoided and then all the regions are again re-allocated by the HMaster to all the region servers. This cause a huge conflict because the locality information is violated due to the re-distribution of regions on the available region servers. This scenario is even faced if a specific region server crashes and it again raises a question about its regions that is going to be re-assigned [1]. How data locality is supposed to be preserved in these kind of situations? We will discuss about the proper mechanism proposed in order to recover data locality in this type of scenarios and mechanism to resolve this issue.



Overview

HBase has a rigid master slave architecture and its main purpose is to be a scalable and efficient NoSQL database which helps in storing data. HBase [5] has strongly constant read/writes which makes it suitable for high-speed counter aggregation. There is automatic sharding which helps in splitting of regions as the volume of data grows in a particular region. The automatic failover mechanism of HBase allows availability of data to a higher probability as the regions are reallocated among the rest of the region servers. HBase stores all its data in the end in HDFS so that data is permanently stored. HBase even supports many API like Java client API for programmatic access and Thrift/REST API as options for other programmatic options. HBase even supports MapReduce framework for processing parallel with a large number of jobs.

From the above architecture diagram, HBase is located on top of HDFS [1] and it is definitely a fact that HBase uses HDFS [10] as its underlying architecture. The HMaster is responsible for monitoring all Region Server instances in the cluster and is the interface for all the metadata operations required to monitor and track the HBase cluster. The HRegionServer is the actual implementation of Region Servers and they are used to monitor all the regions allocated to it. A region is a collection of rows from the HTable which consists of a collection of data in HBase. The DFSClient [10] is an interface used for flushing the data of HBase memory log and store it in HDFS for storing it permanently. The datanode is HDFS has one Region Server configured and the regions in it holding the data are replicated across other datanodes in the HDFS cluster. As you can see from above the replication factor in HDFS is 3 and hence all the data blocks in HRegion is replicated thrice across the cluster. The HBase client HTable is responsible for finding Region Servers that have the specific row request by the clients. The ROOT and META tables are centrally maintained and are most importantly used for data locality. The –ROOT- keeps track of where the META table is and its table structure consists of a META region key and its corresponding values which states the location of META table**.**

The flow of data is in a top down methodology as you can observe from the architecture diagram. Whenever a client sends a write request to HRegionServer, it first writes changes into memory and commit log; then at some point it decides that it is time to write changes to permanent storage on HDFS [13]. This is the rule of data locality and it states that since the region server and datanode are running on the same physical machine, hence the first replicated copy will be saved in that server only. The remaining two replicas will be written to another datanode in the same rack and to a datanode in a different rack respectively. One replica is written to a datanode in a remote rack and another replication [10] is written in the same rack but in a different datanode or let me put it this way a different HReionServer. As a result RegionServer serving the region will almost always have access to local copy of data.

In typical HBase [7] setup a RegionServer is co-located with an HDFS DataNode and this is what makes the synchronization process really fast. If the regions are not moved between region servers then the data locality persists perfectly. If short circuit reads are enabled then a region server can perform fast read operations from the local disk. So clearly the data locality is maintained and performance is enhanced until there is any violation happening to the regions stored in a particular region server. These violation can be of any method and this creates a problem of data locality in the HBase ecosystem. The violations are discussed in the next section.

II. Violations Of Data Locality

It is very evident that Hadoop Map Reduce [1] is efficient because of the fact the logic or processing is taken near to data rather transferring the data to the logic. Every input job is broken into many smaller sections of it known as blocks. As a matter of fact the default block size of every block in Hadoop distributed file system is really large then expected. Henceforth the default size of a block in datanode is 64MB or 128MB is even chosen. Although larger block sizes can even be chosen when we are sure that the size of data is large enough then the size of the block. Each and every map task is allocated to one block in HDFS [10]. It should be even noted that if the number of blocks are less, then automatically the number of mappers are less too for obvious

reasons. The namenode has a central metadata file which keeps location details of every file and takes the logic always near the data. This is the phenomenon how Hadoop makes sure that data locality is preserved.
There is always a curiosity if HDFS can preserve data locality and can successfully take data near logic, then how will HBase be successful in doing the same. An interesting fact to note is HBase [6] saves its data in HDFS as a permanent basis. The actual data is saved in HFile and even its log files in WAL are saved in HDFS at the end of the process. It uses FileSystem.create(Path path) to create these files in HDFS as per the desired location.
There are two access pattern used in this scenario:-

i. Direct random access of the data blocks in HBase.
ii. MapReduce scanning of tables, you wonder if care was taken that the HDFS blocks are close to where they are read by HBase.

The different kind of violations that disrupts the data locality in HBase are:-

1. The HBase balancer decides to move a region to balance data sizes across RegionServers.
2. A RegionServer dies. All its regions need to be relocated to another server.
3. A table is disable and re-enabled.
4. A cluster is stopped and restarted.

1.1 Migration of Regions by HBase Balancer

The HBase balancer might migrate regions among Region Servers due to the following reasons:-

- To balance the cluster’s load so that each Region Server has equal number of regions allocated to it by volume of data.
- When a Region Server crashes and re-allocation of its regions has to be done in the cluster due to misbalance of regions.

The balancer in HBase can be configured and it is set by making hbase.balancer.period and by default the value is 30000 ms or 5 seconds. If the migration of regions occur due to load balancing across HBase cluster [8], than the regions are re-allocated according to the balancer requirement and this violates data locality since the re-allocation can be anywhere in the cluster. Until there is any compaction the data locality still remains a mystery for HMaster.

1.2 Crashing of Region Server

It was noted that most of the crash scenarios happen when the load on the Region server is high. There might be situation when HBase and zookeeper might undergo a large volume of load and the communication between them can be disturbed. Due to this kind of reasonable failures and loss of communication, the user is thrown an exception stating that the region server is down. It must be even noted that rigorous search queries can even impose heavy load on the cluster and this might be even one of the top contending reasons for region server crash. Practically such an approach is unethical [8] and precautions should be carried for following a strict regime of distributed processing so that load gets divided evenly. When a region server crashes the regions in it has to be reallocated and the regions are reallocated as an automatic failover mechanism that is adopted by HBase. We will take into consideration this criteria as one of the most important one for our solution.

1.3 Disabling/enabling of table

Region servers are picked in round robin fashion when a table is re-enabled. This results in poor data locality. We should use retain assignment when re-enabling tables. HBase does support automatic disaster recovery and to implement a higher availability solution the application has to implement it. For the very sake of HA data is backup or replicated across the same datacenter and might be across data centers too. It must be strictly followed that all the table data and [9] column families should be available in the same cluster after re-enabling a table otherwise data locality will be completely lost.

1.4 Restarting a HBase cluster

This is one of the worse scenario where everyway data locality is violated. It is done in extreme conditions and most importantly usually it is avoided. This is considered as one of the worst destruction scenario. When a cluster [12] is restarted all the regions are re-assigned which violates the entire data locality history and there is a major performance setback. Practically when a cluster restarts, there is no probability that the regions will be allocated to the same region servers as they were because the allocation happens fresh and it is allocated randomly.
The regions can be allocated with any of the below mentioned factors:-

1. An Assignment Manager is started by the Master when the cluster is re-started.
2. The Assignment manager does lookups in META table for finding the allocated region entries.
3. If the region that is assigned is still alive then the assignment manager tries to keep it that way.

4. If there are any regions that are not alive and allocated to any region server, then the LoadBalancerFactory [6] is started to randomly allocate the regions among the region servers.
5. After the assignment of the region to the region server, corresponding updation in META directory is even performed.

As a result of failure of a region server at any time, the regions are no longer available since the regions cannot be reached. The Master will even throw an error stating that the Region Server has crashed. The regions of the crashed region server will again be re-allocated among the live region servers.

2. Negotiating by HRegion Locality Index

The phenomenon of data locality is always achieved between region and region servers by the data replications. Furthermore during the replication process of any block in the Hadoop distributed file system, the following procedures takes place:-

1. The first replication of the block is written to the local datanode.
2. The second replication is written to another datanode but in the same rack.
3. The third replication of the block is written across another rack.

Sometimes due to a flush or compaction data locality can be achieved but at the cost of performance. Hence it is always a tough decision to achieve locality in this kind of situation. Whenever a failover occurs in a Region Server, there would not be any data locality after it is restarted but ultimately due [1] to the incoming of new files data locality can be achieved for them. The HBase cluster [12] is avoided to be restarted since it can create a lot of obvious problems rather it is kept alive every time to monitor the flow of data in the cluster. With heavy flow of data in the cluster, compactions are usually performed to rewrite the files. Furthermore all the data are ultimately written to HDFS and that becomes a permanent method of storage unlikely in memory in HBase. The data is always written to files by the HMaster and whenever the size of the file grows over a limit, [1] compactions are performed to consolidate them to another file. The interesting factor is let whenever the HBase files are written to HDFS, the namenode knows exactly where to write them to maintain data locality because of the obvious FileSystem.create class. The files are written by suggesting a path to the namenode about the location details. The Distribute File System class is used in HDFS to write files to it and implementing this interface is the optimal way of doing it. A DFSClient [13] instance is returned which connects the client and the namenode and ultimately the location of the file is returned by the namenode to the client. This entire process happens smoother if the data is found local.

Conclusively it should be noted that the process of compaction is usually performed on all the tables and normally data is found local in the host. The locality index of a region in a region server returns the factor of probability of finding the blocks local to the host. Let's look at the locality index method proposed in this paper for calculating a way to assign regions to region server.

2.1 Locality Index

There has been motivation drawn from HFile [7] level locality information that did not guarantee that much vividly about the regions in a region server. But in case of load balancer and region server issues, the region level locality information plays a major role and this paper tries to propose a difference in this issue. The locality information of a region in a region server can be defined as below:-

Locality index= (Frequency of HDFS blocks which can be retrieved from local host) / (Number of datanode blocks for any specific region)

The whole idea is to allocate regions to that region server which has the maximum data blocks of that particular region. The locality index states about the fraction of probability of finding data locality for a region on a region server and most importantly the regions with the larger locality index should deserve a higher chance of always getting allocated to the same Region Server. For example, if region A has a locality index of 15.4 and region B has a locality index of 1.4 then it is quite evident that that region A should be allocated to that region server always to preserve data locality then region B.

Basically there are two important steps that are followed during the assignment of regions to region servers based on locality index:-

1. Every time during the startup of a Region Server, the locality index needs the be calculated by screening through HDFS [10] and since it can be overload in a performance it is usually done only during the startup of a region server.
2. The Region Server has a record of the locality index for all its regions and the Master can request for this information always via zookeeper or remote procedure calls.

Region id	Region server id	Block id	Locality Index

Table 1: Table Locality Index

The optimal calculation of allocation of regions to region servers is executed by monitoring the locality index for a region and there must be a global decision made by the Master while allocating regions to a region server. A minimum cost maximum flow algorithm can be followed in this kind of situation for an optimal performance. The minimum cost [8] maximum flow algorithm states for any graph there should be considered maximum flow with the minimum cost of traversal possible. This problem combines maximum flow (getting as much flow as possible from the source to the sink) with shortest path (reaching from the source to the sink with minimum cost).

In this paper the minimum cost denotes the performance resources being utilized which should be minimum and the maximum flow must be towards the region with the highest locality index [4]. Therefore the allocation of regions must be allocated to preserve data locality but keeping in mind about the performance too.

2.2 Optimizing locality index

There are three major factors by which we can benchmark the locality index in the HBase cluster:-

1. Graph: - Let there be a bipartite graph and consider 2 sets of regions and region servers in the left and right side of the graph respectively. Each node in the set of regions [9] is connected to one of the nodes in the source set. Each node in the set of region servers is connected with one of the sink nodes for equal show of connection.
2. Capacity: - Capacity is the connection between any two nodes and the capacity between a source node and region node is one. And the capacity between the region nodes and region server nodes is also one. The purpose of assigning a capacity of one is to state that each region server can have only region assigned to it. Nevertheless to make sure that load is balanced across the HBase cluster [13], regions are accordingly distributed across the region servers and hence the capacity between a sink node and a region server node is the average number of regions that can be allocated to a region server.
3. Cost: - The cost is exactly the opposite of locality index and it is the cost between the nodes in the graph. If a region is allocated to a region server due to higher locality index then its cost of the edge between them must be low. Hence higher the locality index, lower the cost between them.

III. Conclusion & Future Work

Presently the regions are allocated to region servers with respect to a basic loading attribute. There are a lot of factors when data locality plays a major role for enhancing the performance. The fundamental is pretty simple because the region server and datanode practically runs in the same physical machine and data locality can be thought from this very basic fact. At the end all the HBase [5] files (HFile) are stored in HDFS and even the log files called the write ahead log files. There has not been much work done for preserving data locality in HBase during situations that might violate data locality in the cluster. Hadoop prioritizes this feature because the map reduce jobs are processed faster and HBase tries to preserve the same for faster processing. Data locality is violated for many reasons as discussed earlier and this is a problem to look out for. This paper talks about the concept of locality index which helps in allocating regions [1] to region servers especially when the region server is restarted due to failure or crash. This concept of locality index is stored in a table for every region suggesting its probability of being allocated to a region server. Higher the locality index, the more the chance of being allocated to the same region server as it was before. In this paper, we have taken one point into consideration where data locality is lost when a region server restarts and regions are allocated randomly after that after the HMaster checks the META table.

The four violations listed above are more than enough reasons for violating data locality. The locality index method of a region states the index of a region and it helps the HMaster to look at it in the [8] META table to re-assign that region by looking at its locality index. The region server which will be having the highest locality index for a region will be assigned with that region and by this data locality [11] can be preserved even after the cluster is restarted or the region server crashes due to overloading. The future work is supposed to be more focused more on preserving data locality in almost every scenario so that remote reads and write are avoided more often.

References

[1]. HBase: The Definitive Guide by Lars George

[2]. Y. Guo, Z. Pan, and J. Heflin, "LUBM: A benchmark for OWL knowledge base systems." Journal of Web Semantics, vol. 3, no. 2-3, pp. 158–182, 2005.

[3]. S. Nishimura, S. Das, D. Agrawal, and A. Abbadi, "Mdhbase: A scalable multi-dimensional data infrastructure for location aware services," in *Mobile Data Management (MDM), 2011 12th IEEE International Conference on*, vol. 1. IEEE, 2011, pp. 7–16.

[4]. F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: A distributed storage system for structured data," ACM Transactions on Computer Systems (TOCS), vol. 26, no. 2, p. 4 , 2008.

[5]. K. Muthukkaruppan, "HBase @ FacebookThe Technology Behind Messages," April 2012. [Online]. Available: http://qconlondon.com/dl/qcon-london-2011/

[6]. HBase A Comprehensive Introduction by James Chin, Zikai Wang Monday, March 14, 2011 CS 227 (Topics in Database Management)

[7]. Introduction to HBase by Gkavresis Giorgos.

[8]. http://en.wikipedia.org/wiki/Apache_HBase

[9]. D. Han and E. Stroulia, "A three-dimensional data model in hbase for large time-series dataset analysis," in Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2012 IEEE 6th International Workshop on the. IEEE, 2012, pp. 47–56.

[10]. The Hadoop Distributed File System, Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, Yahoo! , Sunnyvale, California USA.\

[11]. Hadoop MapReduce presentation by Casey McTaggart, Object-oriented framework presentation

[12]. Distributed Semantic Web Data Management in HBase and MySQL Cluster, Craig Franke, Samuel Morin, Artem Chebotko †, John Abraham, and Pearl Brazier Department of Computer Science , University of Texas - Pan American.

[13]. Hadoop-HBase for large-scale data, Vora, M.N. ; Innovation Labs., Tata Consultancy Services (TCS) Ltd., Mumbai, India