Scientific Journal of Impact Factor (SJIF): 4.14
e-ISSN (O): 2348-4470
p-ISSN (P): 2348-6406

International Journal of Advance Engineering and Research Development

Volume 3, Issue 9, September -2016

Repository Framework back of Facebook Messages accepting HBase at Scale

Mr. Divyesh Patel [1] , Dr. Amit Ganatra [2], Mr. Kalpesh Patel [3]

[1]*Asst. Professor , CHARUSAT,* [2]*Dean, FTE CHARUSAT ,* [3]*System Administrator,CHARUSAT*

Abstract: *Facebook messages, combining messages, chat and email in real-time, the first Facebook application for use in the manufacture of HBase. In this article, we will discuss why we choose HBase for this use case, the first improvements made to make ready HBase production, engineering and operational challenges along the way, and continuous work we had to do it again*
in production, to improve the efficiency and reliability of HBase. We will also describe some of the other uses cases of HBase in Facebook.

1. Introduction

In 2009, discussions began about the possible storage solutions for the next version of Facebook messages product. This product release meeting chat, e-mail and the previous version of the message product in an umbrella. Chat messages were about 20 times the number of traditional messages, and e-mail messages were likely to be much larger in terms of raw size. Previously chat messages stored in the memory and stored for a few days, and chat messages are stored in MySQL. The new product requirements means that a storage system that can support effectively requires an increase in the amount 20x Deeds, except store providing cheap and stretch for datasets we expect to grow up 150 TB + / month.

2. Why we picked HBase for Facebook Messages

HBase is a large-scale, distributed database management systems built on top of Hadoop Distributed File System (HDF). it is an open source project Apache code, and model large table. In this article, we describe our reasons for choosing HBase as storage for new messages on Facebook.

2.1 High write throughput

For a use case as messages, grouping the data on the disk of messages per user ID as the primary dimension and time or other features such as the ID of the object or word in the message (for purposes of search index) as the secondary dimension it is a reasonable option scheme as we want to be able to post messages of a particular user to take effective.

Traditional RDBMS uses a single mutated B-tree like data structure on disk for each index. when the seatback by a traditional RDBMS, messages workload can be very random write caused the B-tree index the index is grouped by user ID and several users receive messages at any given time. Progressive data sets, this approach brings a substantial fine and, over time, every write requires updating a potentially different blocks of the B-tree. In addition, bring a change in the location of the tree B a fine read each write because the block that first must be read updated.

Combine tree structured approach to work recording HBase (LSM) your disk data makes it very effective in handling random writes. HBase gather first all the changes in a data structure in memory and then periodically dumps the structure in memory to disk by creating a data file organized immutable index (or HFILE). Over time a few unchangeable indexes are created on the disk, and merged in the background. In this approach, all written in the drive is writing large followings; and disk IOPS can, in turn, will be saved to serve read!

2.2 Low latency random reads

Although planned storage system with a layer of cache application-level messages on Facebook,

storage system that is necessary to an area serving with a relatively low latencies. HBase offered ok to start, and we knew that even the application was not as effective out of the box performance, it can optimize read many after especially for the type of reading recent workload would be using techniques such as bloom filters, tips slot and looking lazy.

2.3 Elasticity

We expect that Facebook data messages is set to steadily grow over time. It can add incremental

ability to the system easily and without downtime was important. HBase and HDF are both building systems with an elasticity as a fundamental design principle

2.4 Cheap and fault tolerant

We expect the size of the data for a few petabytes reached rather quickly. The use of inexpensive disks and basic hardware That would be the most cost-effective way to be the scale. But the failures are the norm with commodity hardware. A solution it must be resistant to failures and troubles do not require operative intervention. HBase on HDFS offer very cheap, robust and fault-tolerant solution store.

2.5 Strong consistency within a data center

strong guarantee, such as being able to read his own writings, or reading a value consistent regardless of consistency Replication read, it was a requirement for the product messages. In fact, to a wide class programs, this model is much easier to work with them. solid consistency is an attractive property offers HBase.

2.6 Experience with HDFS

HDFS the underlying distributed file system used by HBase, offers several interesting features such as replication, checksums from start to finish, and automated data rebalancing. In addition, our technical teams have already many development and operational experience in HDF data processing with Hadoop.

3 Pre-flight preparations

HBase and was a fairly functional and a rich feature set when we began experimenting with it in 2009. For example, features such as automatic load balancing and and mistakenly call, press and map reduce integration. There was also a fair amount of accuracy, reliability and performance at work we had to do before they can implement to scale. In this article, we will have some of the early work we did describe in HBase and HDF Facebook to get used HBase ready for production.

accuracy

3.1 Correction Fixes

HBase keep your transaction log HDF HDFS- but at the time did not have a way to ensure that the attached to a file contents are scanned / synchronized with multiple nodes. It was one of the reasons why HBase. It was not popular as a storage solution for users online versus workload stringent durability requirements data. Add synchronization support for HDF, in order to ensure the durability of committed transactions, it was one of the first qualities that we work with.

HBase promised to guarantee transactions at the row level. A clip which should be many generations of column atomic fact provide all records belong to the same row must either continue or any changes or not.

But the initial implementation of registration confirmation allowed this property is violated. We have improved the commit registration application to support the activities of the family of a few atomic columns. We also corrected several errors in the recovery (commit log replay) code as an error that could cause eliminated elements raised after a node failure, or cause transactions registration confirmation to be deleted as a result of Mishandling of the order of transaction IDs.

3.2 New/Pluggable HDFS Block Placement Policy

HDFS, the underlying file system used by HBase, offers several interesting features such as replication, end to the other checksums, failover, storage and elasticity. But the default mapping rule HDF block, while shelf aware minimum limit. The non-local replicas of the different blocks of a file can almost finished until it covers the entire cluster. To reduce the possibility of data loss in the event of multiple nodes were to fail around At the same time, a new policy of placing blocks in HDF, which allowed us to have been implemented restrict the location replicas of the blocks of a file in groups of nodes configurable smaller size. This meant that there was much less combinations of multiple node failures that can lead to data loss compared to before. the use of the new block placement policy, we could probability of loss of theoretical data reduced about two orders of range.

3.3 Availability

HBase handle node failures too. In addition to the HDF master (also known as NameNode) fails, the system automatically management and repair of failures of other components (such as teachers or regionservers HBase or HDF DataNodes). Given this, we expect the most common reasons for a standstill for maintenance. since HBase software was still relatively new at that time, there was expected that we will probably have to push critical error improvements and changes regularly production groups, and therefore applied subsequent updates, whereby we You can download the software on the cluster in a way rolling work without the whole database. we also implemented the ability to have a schedule of the table (for example, the addition of new families change column or change the configuration of an existing column) family in a way online. We have also to be recharged fragile HBase compaction To complete a server or the entire cluster does not have to wait for the compaction.

3.4 Performance

Combine tree structured approach to recording HBase (LSM) to keep data on disk makes it very effective in write. On the reading side, but since organized several data files,, unchanging indices uncombined aka HFiles must be consulted before a read request can be served in a naive implementation, overload Lee is likely to be greater in the general case compared to a traditional RDBMS.

We have implemented some performance characteristics to help reduce the overall reading.

• Bloom filters Keys: This maintained at a level HFILE, and help reduce the number HFiles that need to be searched in a lecture.

• Time Range hints- We have added Time Range HFiles tracks that supplies timestamp or time search key range queries only HFiles with data for consulting a time interval of overlap.

• Search optimizations- also fixed some inefficiency in the previous application that causes looking for multiple columns in a row for scans entire row, where instead of an index could have been made used for efficient reseek the desired columns.

3.5 Shadow Testing

Instead of speculating about the nature of the problems we will encounter when we production system, or restriction run ourselves to the synthetic load test against groups, invest effort in the shade / duplication traffic for users of the older generation of the product messages and chat while the new still developing. Shadow cluster configuration we have very good ideas about what to expect in goods traffic, and also serve as a useful platform to failure scenario testing, and alternative assessment Scheme designs.

After the release of the new version of the product has reached a crucial part of our user base has changed our one a new Essay shadow the new product version. This configuration keeps the Location and helps us to quickly and safely iterate over software modifications. All changes by the shadow tests before they are pushed into production.

3.6 Backups V1

HBase has not previously been used, according to scale, for critical applications that the user sees. So during the first phase, We were worried about unknown / errors in HBase, especially those that can cause data corruption. So we invested implemented in a pipeline backup and recovery outside the HBase. Writer use Facebook distributed framework for combined records of thousands of servers to all actions against postbox's login user (As the add / remove message changed to read / unread, etc.) to a HDF cluster. But Scribe ago buffered expedition, and is not intended for guaranteed delivery. So to reduce the likelihood data largely loss on this step, double Scribe both the application server and HBase levels. The two log streams are then merged / -by trick and stored in HDF groups in a remote data center locally as well.

4 Mid-air refueling

Once in production, we had to go fast in several performance and stability improvements diagnosability to HBase. Problems in production test and helped the cluster shadow prioritize this work. In this article, We will describe some of these improvements.

4.1 HFile V2

A few months after production, such as datasets continue to grow, increasing the size of the index began a touch problem. While the data portion in a HFILE is fragmented into blocks, lazy loaded in the cache block, and Free at the age of cache block as the cold, the index part, however, was a one-level monolithic entity by HFILE. The index, cold or otherwise, it is loaded into memory and nailed before any application can be serve. For families column rarely viewed or old HFiles, it was a non-optimal use of memory resources. And it has also resulted in restarting operations as cluster and region (fragment) load balancing to take much longer needed. Bloom filters suffered the same problem.

As a temporary measure, we have doubled the size of the data blocks; This halves the size of the indexes. In parallel, We started working on a new HFILE format HFILE P2 one in which the index is a multi-level technique would be structure (as in a traditional tree b) It is made up of fixed size blocks relative. P2 divided HFILE Bloom filter monolithic hFile in smaller flowers, each matching a series of keys on a hFile. The flower and index blocks interspersed with blocks of data in the HFILE loaded on demand and closet same LRU cache block that is used for data blocks. An individual can now read Scrolling in several index blocks, but in practice index blocks frequented almost always the case.

In traditional databases, such changes are almost impossible to do without a halt. Thanks to compaction, in HBase, changes in the data format can be pushed into the production and data gradually new format without any downtime, provided that the new code continues to support the old format. But it was not HFILE for P2 many had time to soak in the shadow cluster configuration and compatibility testing significant that we feel comfortable pushing the code for the production

4.2 Compactions

Compaction is the process by which two or more HFiles sorted-merge arranged in a single HFILE. Help colocation all data for a given key drive, and help drop the deleted data. If compaction takes all existing disk HFiles and combine them into a single HFILE, and compaction is known as an important compaction. The rest of the densification "small" compaction or referred to as compaction just done. We usually choose 3 files similar size and compact them. If the files are different sizes, we do consider it viable candidates for less compaction. The first problem we encountered was the case of insufficient lower compaction waterfall. Consider the case when we have 3 files to find a similar size and compact for a A single row of three times the size (3X). It may be that there are 2 existing files in the shop of a similar size (3X), and it would cause a new compaction. Now the result that it could trigger a new compaction.

This cascade system compact is very inefficient, because you have to read the same data and re-write and much more. Hence Adapted compaction algorithm to take into account the amount of data. The next question that we made was the parameter that allows a file to unconditionally compaction If its size is less than a given threshold. This parameter is initially set to 128 MB. What this has led for rinsing several generations of the columns with small files to disk, compress our same files over and over again until the new files led to a 128 MB file. Therefore, in order 128MB range, we would be a few GB write data, make inefficient use of disk. So we changed to be the parameter of a very small size.

Originally compact was some strings. Therefore, if the wire is carrying a large compaction that It takes a while for a large data set, complete all minor compaction and it would have delayed reduce system performance. And to make matters worse, the main compaction compact every family in the area of the column before moving to a different compaction. In order to improve efficiency Here, we have a few things. We multithreading compaction, as one of the wires will work in a "big" densification and the other will work on a "small" compaction. It improves things a bit. And we have the request queue compaction take a family granularity by region, column (instead of Simply by region) column for a family as HFiles get first compacted.

4.3 Backups V2

The backup solution P1 written records of action for the user mailboxes as they at Scribe. For a recovery user data records read all user action was outside the author and replayed in HBase cluster. Then secondary indexes for mailbox had recreated the user.

As data sizes began to grow, the solution P1 copies of existing security is becoming less effective because of the size records of action will have to be read and recorded in the recovery he continued to increase. It has a solution P2 needed.

The backup solution V2 is a pure backup solution HBase. Backup is divided into two parts - the HFILE Backup (HFiles is HBase data files) and backup Congrupo (HLogs for HBase write records). Both of This backup, and knowing that the beginning and end of the backup HFILE, it is possible to stage recovery. So we started back up HFILE and increased retention in the production HLogs cluster. This backup, stay on the production cluster is our backup of Phase 1. In order cluster survives failures, but also increased 2 backups where data continues in another group in the same data center. Walking 3 backup, data also copied to a remote data center.

One problem we found pretty quickly as in step 1 of cluster backup that be if the files bigger, we need to do more backup disk IOPS in the areas of productive activities and were more time to do what Also back up. To solve this problem, we adjusted the HDF to hard links to create an existing file blocks, instead of reading and writing data to back.

4.4 Metrics

We have started to be a basic set of indicators have with, for example - HBase, sit and function ops removed per second and latency of each of the resulting latencies DFS reading, writing and synchronization calls, and related compaction Metric. We also keep statistics system level, such as CPU, memory and network byte input and output. Over time, the need for a better overview to correct errors or improve the need to add and track number other statistics.

In order to understand how our block cache do at the family level by column (so that we can decide Do not cache some families column at all, or look at possible application errors), we started tracking by column Family cache hit ratios. It should also be noted that for a few calls, the average time several operations was low, but there was a big outliers - that while most users have a good experience, some users or some operations They were very slow. Hence we started tracking the N worst times for surgery based on the machine name so we can records can deepen and make an analysis. Similarly, we have a per-get size result added, so that we can keep the size of the application servers request answers.

Once we HBase backup P2 production, the total consumption disk has an important measure, so

tracking start using DFS. In addition, it was found that there are some good shelf switch failures and start tracking those.

It's just a handful of examples of how we have more statistics on the fly added.

4.5 Schema Changes

The chat program complex business logic. In the days before his release, even for a short then, the product stop. Therefore, took a quick decision to invest in a complex scheme to store data in HBase. But the burden of all messages in memory of the application server from HBase in a closet when a user access was prohibitively your mailbox. So the initial solution It was all user messages in a family column store, and maintain an index / snapshot memory position structures needed to display pages most viewed in a different family column. This means we are well In the days of schedule evolution as the snapshot is versions, and the application server might interpret changes direct formatting correctly.

But, as messages, chats and e-mail storage, it was found that the family of the column "snapshot" a few megabytes in size for active users - reading and writing was great, and it is becoming a major concern. Therefore we have to invest in making the scheme finer grain. We went to a scheme of "hybrid" where the different structures that are stored in the column families more closely. It helped only the most work variable data rate and the number of changes to data reduction with little change frequency. It helped us make smaller Scripture, although the readings were as big as before.

Finally, we look at the usage patterns of the application and we came up with a scheme very fine grain where reading and writing are made in small units that all calls. It improves efficiency things

not enough. But as we by changing all the disk layout of the data, to implement this feature on our scale without remarkable time of inactivity was a great challenge. In addition, since the card reduces jobs being implemented to transform the data was very intense and it was very risky to carry out the work of the production groups, we decided to run the transformation work in a separate group machines in a Throttled way.

4.6 Distributed Log Splitting

Briefly describe what it is, look at the situation where all or most of the regionservers die (for example, when NameNode DFS is unreachable because of a network sharing). Restarting groups, all regions have a few entries in records light (HLogs), which would have before the repeat regions can be placed online. To do so, all HLogs that ought to all regionservers split and regrouped in expenses of each region, and should play the opening of new regions regionservers these issues can be opened for regions to serve read and write take traffic.

Implementation of achieving this was the main list HBase all round logs and split them by a few cloths. Demand during the execution of the larger groups - for example, with 100 regionservers - was that it would more than take 2 hours for the process to just master records of dividing all servers. Since it is unacceptable enter production in this state, the first and simplest version of sawing logs distribute a script this parallelize would all regionservers before the start of the cluster. It reached the target effect of making the time log split in the range of 5-10 minutes.

The problem with the script-based approach is that it is transparent to the master HBase. it requires cluster be completely until the process is completed sawing logs, so it was not suitable for the cases where a subset of regionservers (such as those in a rack) divided because DFS. So we have a function implemented automatic sawing logs on the master distributed HBase itself - where Zookeeper teacher uses HBase as a working rope and spread the work of sawing logs all regionservers. Once the registration section of the region open as usual.

4.7 Data locality

Rack available despite being failures, they do not want to be all replicas of a file on the same stretch. For one many applications of data-intensive, such as messaging, which caused a lot of network traffic between racks. If the data size It became larger, which increases network traffic until it causes became the highest shelf around the bottleneck. A) Yes, reducing the bandwidth of the network when read, location data is paramount. In order to maintain Location data, the regions around servers region most of the data locally drive. We solved this by calculating the machine's location data for each region at the beginning of time and regions assigned therefore HBase start time.

4.8 **Operational aspects and issues**

If not the norm when groups scale runs. Therefore drive failures, failures of the machine, network node walls, rack network failures and restart the switch shelf were all recurring events. Automatically we wanted detect and treat the most common of these. At first, we decided that we should automatically handle most of the problems of a single node. Hence we shell script is disabled and repair will send the machines that were not accessible network for a long period of time, those with the root disk failures, and so on. In fact, retired the process of the machines, claiming the recovery of the pool is a constant an approximately 2-3% of the machines always line.

We have a lot of warnings that scroll send people involved in a variety of failure scenario. For example, if the number of errors detected exceed the application server a certain threshold steep then triggered warnings.

Also, if the error rate was not strong enough to trigger warnings, but remain persistently over a window of time, then it would also be a warning trigger (prolonged errors, although smaller in volume). Similarly, the master nodes (Nodes "controller" as we call it) to warn of problems, as they require higher availability than others nodes. In addition, a tester health HBase run constantly and will warn you if health fails checks.

In light of the problems we had, operator error is a non-trivial some of the problems we face cause. There was an incident of this nature, where accidental disconnect the power cable caused about 60% of the nodes in the group to fall. The grouping was restored in a few minutes and no data loss.

5 New use cases

As an adult HBase use within Facebook, both to serve and work load, as well as real-time for the user workload and batch processing, grew. In this article, we will describe some of these new use cases.

5.1 Puma: Real-time Analytics

Puma is a stream processing system designed for real-time analysis. Using Puma, content publishers / advertisers You can, for example, real-time information on the number of impressions, holding and unique visitors per domain / URL / ad, and a breakdown of these statistics by demographics. Such analytical support previously A periodic basis using batch-oriented and Hadoop hive, and therefore available for use only a few hours later. Puma, the data collected in real time by writing, reading ptail and processed / aggregated by Puma buttons and enter HBase. Puma buttons also act as a cache to serve the analytical queries memory postponement any cache misses to HBase.

5.2 Operational Data Store (ODS)

ODS is the internal control system of Facebook, which has a variety of systems and applications gathered level statistics each server for real-time monitoring, anomaly detection, alerts and analysis. Workload, it's out A few tens of millions of time series data is very intensive writing. 100B on individual data points They are written per day. The lectures issued for both the user and work short time, mostly recent data.

SAO done before on top of MySQL, but has recently moved to a back-end application HBase for reasons of workability and efficiency. The constant need to re Sharding and existing load balancing fragments, such as datasets grow unpredictably, causing a lot of inconvenience in the operation of MySQL prepare. Furthermore, from the point of view of efficiency, HBase is a very good choice for this intensive writing, recent lecture type workload. HBase offer a much higher performance than that updates records randomly sequential write to the disk subsystem level. And, of course, because HBase keep the old and new data separately HFiles, for applications such as ODS, most recent reading data, data can be cached more efficiently.

SAO make heavy use of close integration with HBase MapReduce to run the rollup function of time work from time to time. SAA also uses HBase TTL function (time to live) to automatically clean the raw data A number of days. And there is no reason to sit and manage additional cron jobs this purpose.

5.3 Multi-tenant key/value store

Various applications within Facebook has a need for a large-scale, persistent store key / value. One of each application keystore / HBase now backed by a value is offered as a service in Facebook and a few programs Use this solution for your storage needs. A Shared HBase provides controlled multi-level rent this programs and relieves application developers from the usual overhead associated with capacity planning, provision and configuration of infrastructure, and scale as their data sets grow.

5.4 Site integrity

integrity systems collect Facebook site data and perform real-time analysis to detect and prevent spam and other attacks. One such system now uses a backup infrastructure for a track HBase registration of new users and surfaces several false accounts every day. HBase was chosen because it offers a high performance writing, elasticity, a data model that effective implementation of analysis on the keys of lexicographical order to allow driving and primitives that support automatic aging out old data.

5.5 Search Indexing

HBase is used to find a generic framework indexes on many building implement vertical markets. multiple document sources are introduced into HBase continuously and asynchronously. HBase save the document, and is used to link information from different sources for each document. In the court case, search indexes for each fragment is built using MapReduce to read the relevant data from HBase. Index servers and then upload these fragments.

6 **Future work**

HBase is still in its early stages, and there is much room and need for improvement. On the front of reliability and availability, HDF NameNode HA (by HDF tolerate failures NameNode), better resistance to all network data location fail deliberately node and load balancing regions, rolling restart for HBase / HDF for software updates and rapid detection of a single node failures are some of the areas of current interest. We are also engaged in cross-sectional data replication center and restore backups a point in time so that each application does not have to implement your own custom solution. Native support for physicists implemented in HDF for super fast data snapshotting allow in HBase. On the side of efficiency, we are seeking to verify checksums reduced maintaining checksums inline data to overall hard disk, improving the ability to HBase make optimal use of the machines large amount of memory, reducing dependency on HBase on page OS cache, and adds support for compressed RPC, analyze performance, compression compromise registration and so on. As new applications emerge HBase new requirements continue to emerge. Better support for C ++ clients, better integration Hive / HBase, support for secondary indexes and coprocessors and adaptation of HBase for use in storing mixtures (flash + disk) or pure environments flash stuff is also on the radar.

References

[1] Apache HBase. http://hbase.apache.org.

[2] Apache HDFS. http://hadoop.apache.org/hdfs.

[3] Kannan Muthukkaruppan. The Underlying Technology of Messages. http://www.facebook.com/- note.php?note id=454991608919, Nov. 2010.

[4] Dhruba Borthakur et al. Apache Hadoop Goes Realtime at Facebook. In Proc. SIGMOD, 2011. [5] Sanjay Ghemawat, Howard Gobioff and Shun-Tak Leung. The Google File System. In Proc. SOSP, 2003.

[6] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In Proc. OSDI, 2004.

[7] Fay Chang et al. BigTable: A Distributed Storage System for Structured Data In Proc. OSDI, 2006.

[8] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira and Benjamin Reed. ZooKeeper: Wait-free coordination for Internet-scale systems. In Proc. USENIX, 2010

[9] Alex Himel. Building Realtime Insights. http://www.facebook.com/note.php?note id=10150103900258920.

[10] Joel Seligstein. Facebook Messages http://www.facebook.com/blog.php?post=452288242130, 2010.

[11] Patrick E. O'Neil, Edward Cheng, Dieter Gawlick and Elizabeth J. O'Neil. The Log-Structured MergeTree (LSM-Tree). In Acta Inf. 33(4): 351-385 (1996).

[12] Eugene Letuchy. Facebook Chat. https://www.facebook.com/note.php?note id=14218138919.

[13] Scribe. https://github.com/facebook/scribe/wiki.