Research on Cassandra Data Compaction Strategies for Time-Series Data

Bai Lu*, Yang Xiaohui
College of Computer and Information Technology, Beijing Jiaotong University, Haidian District, Beijing, China.

* Corresponding author. Tel.: +86 15210575497; email: bailu_bj@163.com
Manuscript submitted July 30, 2015; accepted December 16, 2015.
doi: 10.17706/jcp.11.6.504-512

Abstract: Storage and analysis of time-series data is a subject of intense interest in the current international database research field. Time series data, a sequence of collected data information points by fixing time interval, is an important basis to proceed business analysis and prediction in the future. As an excellent NoSQL database, Cassandra is often used to storage time-series data because of its characteristics of data model. In the scene of real application, time-series data used to proceed the management of data life cycle by setting up TTL; the real delete operation would not be executed immediately, while unnecessary data will be deleted during the compaction course. This paper focuses on the issue of the effect of different strategies for time-series data storage and the research on three Cassandra storage strategies: Size-Tiered Compaction Strategy, Leveled Compaction Strategy and Date-Tiered Compaction Strategy; and comparative test based on stable data storage, recording speed sorted string tables file numbers and so on. Finally, the compaction strategies suitable for time-series data application scenarios are obtained by carrying on experiments.

Key words: Cassandra, time-series data, distributed storage system, compaction strategy.

1. Introduction

With the rapid computer network development of nowadays, people have entered a period of high-speed information transmission. There is a large amount of data to be produced and consumed throughout the world every moment, among which, the time-series data is a special one. While a single instant data, lasting a very short period of time, will be ignored, the data lasting a period of time will draw people's much attention. A sequence of collected data information points by fixing time interval can be called time-series data. Time-series data is often used to proceed business analysis in a successive period of time; with the increasingly dense data, the way how to organize and store time-series data, collect and analyze a large amount of data and get HD pictures at an extremely fast speed and predict the evolution of the market, customer behavior, environmental conditions, the healthy energy consumption trends, etc. has drawn wide attention of researchers in domestic and abroad.

This paper has introduced a comparative study of the compaction strategy of the time-series data storage system based on Cassandra [1]. At first, the time-series data storage system based on Cassandra has the characteristics of higher performance, scalability, and so on [2], when compared with other time-series data storage systems. Any data can be recorded or read at any place or any time by using Cassandra; and it's easy to balance its consistency and availability through adjusting the record-write copy number without hardly any single point of failure. In Cassandra, the SSTable(sorted string table) is used as the final storage carrier; in

order to improve the system performance and reduce the time loss of disk reading and writing, it is only written into a "tombstone" tag during the process of data deletion which is completed when SS Table is being compacted. Cassandra provides a wide range of optional data compaction strategies according to different application scenes; then the difference between different strategies for the time-series data and the strategy selection to make is a problem to be solved urgently. This paper has made experimental comparison according to Cassandra compaction strategies, to explore its effect on distributed storage system from three aspects: the file number under stable state, the final file size, and reading and writing performance.

This paper structure arrangement follows: Section 2 introduces the related methods of time-series data storage; Section 3 introduces Cassandra data model and characteristics; and three compression strategies based on Cassandra are detailed introduced in Section 4. Section 5 introduces related experiment and analysis and Section 6 is the conclusions of this content and work prospect.

2. Related Works

The storage of time-series data is a problem which can be widely studied and applied. Literature [3] points out that the traditional databases are used to store relatively fixed data which is not related to time, unless the time attribute is specified. However, the emerging data applications require the database to support real-time data storage and analysis. The defects of traditional database for storing time-series data have been fully realized, and it has played an incentive role in the time-series data storage related research.

Literature [4] introduces the time-series data storage model and it points out that a time-series data storage model is a sequence of data series organized by data arrival time and may only be used once. However, considering the possible explosive arrival of data, a time-series system is more suitable to be modeled as a sequence of data elements.

Literature [5] introduces characteristics and challenges of time-series data and it points out that the research of analysis and management of stream data is a subject of intense interest in the current international database research field. In the past 30 years, although the traditional database technology has developed rapidly and has been widely used, it is not able to deal with a new kind of data, stream data, generated by the application of network routing, sensor networks, stock analysis, etc. The characteristics of the data stream data is the data's successive arrival, the fast speed and the huge scale.

For the application of Cassandra database in time-series data, the document [6] points out that the use of time-series data for business analysis is not a new type of application, while the way how to collect and analyze massive amounts of data at a terrific speed faces huge challenges. It is of great significance to use a clear temporal image to predict the market's changes and to analyze user's behavior, environmental change, resource consumption, health conditions and so on. In view of the above challenges, Apache Cassandra is a very remarkable NoSQL [7] database and Cassandra data model is very suitable for dealing with the sequence data, without regard to the type and size of the data. When data is written to the Cassandra, the data is written to the disk in sequence. Cassandra will read the data at the cost of the minimum disk drive by looking up through row key and the subsequently range search. So the time-series applies well to this storage mode, the commercial application of Cassandra can quickly find meaningful features from the time-series, and make clear predictions about the future.

3. Cassandra and Data Model

The current time-series data system generally use the processing method based on the schedule of arrival time while dealing with real-time data. With the data files firstly received to a server, analytical process is triggered at a specific time according to the statistical condition of all kinds of data arrival time during a day, frequently polling the original file server, getting data file, parsing and writing.

The Apache Cassandra is a set of high performance, scalable distributed database management system, which was originally developed by Facebook to store the out-sized data in particular, open source in 2008. Cassandra is absorbed and integrated with the Bigtable [8] from Google which is based on some ideas of the column store and distributed key/value ideas of Dynamo [9].

The Cassandra [10] is P2P distributed storage system, a completely P2P framework based on consistent hashing in fact. The Gossip [11] protocol is used to communicate between nodes, in order to carry out the failure node test and node state transfer. The disaster recovery capability of the system can be guaranteed by the copy of data, ensuring the content of the copy of data on each node for the latest by the inverse entropy and read repair.

The consistency of the Cassandra is flexibly adjustable: its strong consistency can be ensured by specifying consistency level for "ALL" and if you just want the best performance but not consistency, you can specify the consistency level for "ONE". In addition, the deployment and configuration of Cassandra is very simple, and it can be easily applied to large data caching system. In the Cassandra process, Thrift API can be used for read and write of the data operations [12].

During the write operation in Cassandra, data is the first to be written in the CommitLog, and CommitLog is an error recovery mechanism that Cassandra could achieve its persistence. After written to the CommitLog, the data would be written into the memory data structure called memtable, and memtable will eventually be recorded into the disk and placed in a file called SSTable, which cannot be changed. In Cassandra, all of the write operations are proceeded in order, which is also the reason for the outstanding performance of the Cassandra.

In Cassandra, the data will not be deleted immediately when a delete operation is performed. Instead, it will be regarded as an update operation, with a tombstone (a deletion mark) put on the corresponding value. Only when the compaction performed can the data be cleared.

The Bloom filter is used to improve the performance to read of the data from the SSTable in Cassandra, which is an ultra fast, but uncertain algorithm for determining whether a member is from a set [13]. The reason of the method of uncertainty is that Bloom filter may results in "false positive", but not "false negative", that is to say, the judgment to belong to something is not necessarily true and the judgment not to belong to something is certainly not. The Bloom filter maps the values of data sets to a single digit group and concise a large data set to an abstract string. According to the definition, the string will occupy memory space far less than the original data. With Bloom filter located in the memory, the disk access can be reduced when searching for keys, so as to improve the performance.

4. Cassandra Data Compaction Strategies

The tombstone strategy is used for the write order of Cassandra and deletion which can lead to existence of repeat, obsolete data or data remains to be deleted in SSTable; so there is something to deal with SSTable, re-organizing data, to obtain a better read performance. Such a performance process is called compaction in Cassandra.

With the compaction operation acts on merging the SSTable, data in SSTable will be combined during the process of compaction: keys merged, columns grouped together, tombstone discarded and a new index created. The compaction operation is the process of releasing the space by combining the large accumulation of documents.

In the compaction operation, the combined data is in order, and a new index file will be created for these ordered data. At the same time, these just merged, ordered, indexed data will be written into a separate new SSTable. After the completion of the merger, the old SSTable will be deleted. Another important feature of compaction is to improve the performance by reducing the number of times to locate. For a given key value,

the upper limit of the number to find a list of data is determined by the number of SSTable. If a key value was often changed, each changes may be written in the SSTable. With the compaction operation to the SSTable, it is possible to avoid data searching in each SSTable.

There are many kinds of strategies for compaction operation of SSTable, SizeTieredCompactionStrategy is used by default in Cassandra and LeveledCompactionStrategy is added in Cassandra version 1.0, just like LevelDB. The DateTieredCompactionStrategy is introduced into Cassandra version 2.0.11, first proposed by Björn Hegerfors [14] in Spotify, aiming to solve compaction problem for time-series data.

4.1. Size-Tiered Compaction Strategy

Size-Tiered Compaction Strategy in Cassandra parallels to the compaction method described in papers from Google Bigtable: when similar-sized SSTables reaches a level (the default is 4), they will be merged in Cassandra. As shown in Fig. 1, each of the green blocks represents a SSTable and the arrow indicates the compaction. At the very beginning, the new SSTable is created without doing anything, until the number of SSTable reached 4 and they will be merged together; and this process will continue. Fig. 2 shows the state of the SSTable in this system under the Size-Tiered merged strategy: the longer the green block, the greater the SSTable after a period of time.

Fig. 1. Size-Tiered strategy merger diagram.

Fig. 2. Under the Size-Tiered strategy, the system state after a period of time.

The SSTables to be merged together from the Size-Tiered strategy will be combined into the same bucket, with the default number of SSTables cannot be less than 4 and more than 32. And the size of the SSTable cannot be less than the value specified by the min_sstable_size parameter (the default is 50MB), otherwise it will be added to a particular bucket.

For the Size-Tiered strategy, it is not suitable for updating or deleting operating frequently works, because it will lead to the existence of a record in the presence of multiple SSTable, with reducing the performance to read; and the old data may still exist in the old SSTable until the SSTable involved in the merger. With the passage of time, some big SSTable will appear in the system, and it takes a lot of space when they combined.

4.2. Leveled Compaction Strategy

The introduction of leveled compaction into the Cassandra version 2.0.11 refers to the merger mechanism of leveledDB from google. In the leveled compaction, the generated SSTable has a qualitative size: the default is 5MB and it is raised to 160MB in the Cassandra version 2.0.11. The SSTable is distributed in different levels that the next layer is 10 times the size of the upper layer and each layer will not have duplicated data.

As is shown in Fig. 3, the new SSTable is added to the L0 layer (pale green) immediately and then it is merged with the L1 layer (blue). When the layer L1 is full, the redundant SSTables on it will be placed on the layer L2. Subsequently, the new SStable generated on the layer L1 will merge with the SSTables on L2, and the SSTables in the final system are expressed like that in Fig. 4.

With leveled compaction, the problem that insufficient capacity of the Size-Tiered strategy towards frequent update and delete operations can be solved, which ensures that 90% of reading can be done in a SSTable with the space wasted up to 10%. And the free space needed is 10*sstable_size_in_mb during the compaction process.

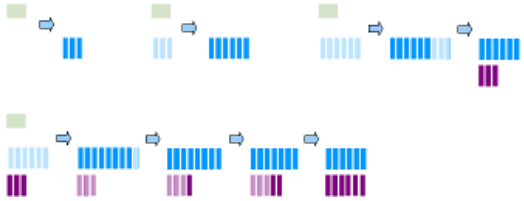

Fig. 3. Leveled strategy with schematic diagram.

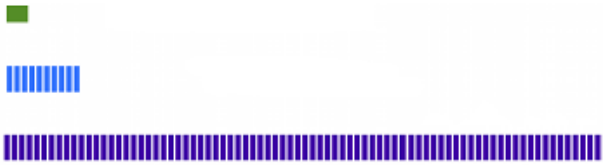

Fig. 4. Under the Leveled strategy, the system state after a period of time.

The I/O operation, the compaction process requiring for more, is the main drawback of the leveled compaction; and the use of leveled compaction is not a good choice if the system I/O has a great pressure. In addition, leveled compaction may continue to generate on the layer L0 and a large number of small files will keep in stock under the scene of more write and less read, which leads to a decline in system performance.

4.3. Date-Tiered Compaction Strategy

Date-Tiered strategy is the latest addition to the Cassandra in the compaction strategies , and the basic idea is that the data of the SSTable will be classified into different time windows along with the compaction operations, which guarantees that the new data and old data will not be mixed together. The compaction process is shown in Fig. 5 below.

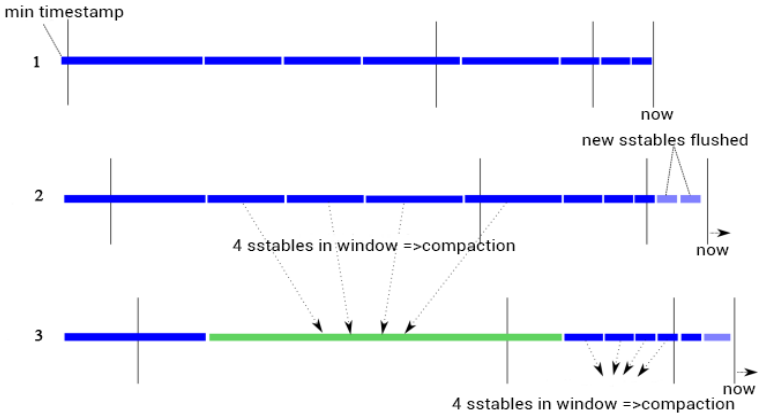

Fig. 5. Date-Tiered strategy merger diagram.

It can be seen in this case that the time window moves along with time. The SSTable are ranged in sequence by the age of its earliest data (calculated by min timestamp) , with the latest data on the right. As the phase 1shown above, no time window contains 4 SSTable (the number how many SSTable will trigger compaction to configure a time window by min_threshold at least), so as to there will not be the compaction process. In phase 2, there is a time window containing four SSTable, in which the SSTable will be merged into one. In phase 3, there will be a window containing four SSTable with the time of moving forward and the SSTable will be merged again.

5. Comparison of Different Compaction Strategies

5.1. Description of Experimental Problems

In the practical application of time series data system, the data is time-series data and its scale is very large. The new data is frequently inserted, and the inserted part is rarely updated. In addition, consumers use the data from only a sort of time window, other than out-of-date data. Consequently, there will be an expiration time TTL when the data is inserted during the time-series data caching system. However, according to the characteristics of Cassandra itself, it is not easy to delete the out-of-date data, and the delete operation will not be executed immediately, but to delete the unwanted data in the Compaction process.

Through the researching process of compaction and the background of the time-series data from the data system, the following problems are found:

During the compaction process of Cassandra, the SSTable having all expired data is also added to the merger process, but the fact is that it will be deleted directly without participating in the merger to bring the consumption of resources.

The Size-Tiered strategy and Leveled strategy are merged by default in SSTable, and there is no need to take account of time during the compaction process. For the time-series data, it has a relationship with the time sequence, which is a very important factor. If the time parameter is used as a kind of parameters to search data, such as query of certain recent physical quantities within an hour (It is very common in the practical application), the Size-Tiered and Leveled strategy will not have a good performance.

Without considering the time factors, the Size-Tiered and Leveled strategy will lead to a mix of the old and new data in SSTable, which means that there will be new data inserted as well as the data for a long time. And this will result in the deletion difficulty of out-of-date data.

A new kind of compaction algorithm is going to be achieved for time series data to improve the system performance, but to find by referring to relevant material that the DateTieredCompactionStrategy algorithm [15] for this kind data in the new version of Cassandra 2.0.11. After the research, it is found that the DateTieredCompactionStrategy theory is very suitable for time-series data system, and it will improve the system performance. But the theory needs to be proved by practice. Therefore experiments are designed to explore the effects of different compaction strategies on the performance of the system under real conditions; it will come to the real and general conclusions according to the experimental facts. Then we need to analyze the causes of the problem and explore the way how to improve the design.

5.2. Experiment Schemes

5.2.1. The Experimental Environment

The experiments will be tested on three real machines and the RedHat operating system for each of the memory of the machine is 32G. The Cassandra cluster is deployed on this three machines, with setting the copy factor of keyspace to 3. The service data is imported into the system by a desktop machine running Windows and written into the Cassandra cluster by network connection.

5.2.2. Specific Experimental Methods

Simulating the application environment of this system, the same type of time series data will be written into the Cassandra ceaselessly in the next few hours (keep the data in the same column family), with the data inserted into the same line but not the column name; and then the performance to read the data can be measured by the number of SSTable in the system. Therefore, it needs to look up in all the SSTable to read a single data under this situation, and the more SSTable, the more the read performance affected. In this experiment, a total of five different tests was carried out, with different compaction strategy or parameters used each time, and the system logs and other relevant data generated by Cassandra from each test were collected. The total amount of the non-decoded data for each test is about 171G, and the expiration time is setting to 2 hours. The five experimental setups are shown in Table 1:

Table 1. Experimental Setup

Test names	Application strategies	Parameter settings
DT-1	DateTieredCompactionStrategy	base_time_seconds=3600
DT-2	DateTieredCompactionStrategy	base_time_seconds=7200
DT-4	DateTieredCompactionStrategy	base_time_seconds=14400
LC	LeveledCompactionStrategy	the default
ST	SizeTieredCompactionStrategy	the default

During the 5 tests exploring the time-series data, the performance status from the effect of different compaction strategies on the system performance and the compaction strategy of DateTieredCompactionStrategy designed for time-series data will be settled when the base_time_seconds parameter values are less than and greater than or equal to the expiration time of data (TTL). In the specific experiment, the strategy described above will not be implemented in the system table and other related tables.

5.3. Experimental Results and Analysis

With the experiment completed and the experimental data collected in the process of each test, analysis of coding for Cassandra system log, the results are obtained as following:

As is shown in Fig. 6, the figure shows that there is no longer a new SSTable produced and a compaction operation triggered when the write of data to the Cassandra is completed and the comparison of the three physical quantity, 5 times experimental stability data, write time and the number of stable SSTable, when the system reach a stable state. Among them, the stable data indicated that the total size of the SSTable data file on the disk (The more the number, the more the redundant data) and write time represents the time spent of the 171G encoded data to write on the Cassandra, and along with that the number of the stable SSTable indicating the number of SSTable under a steady state.

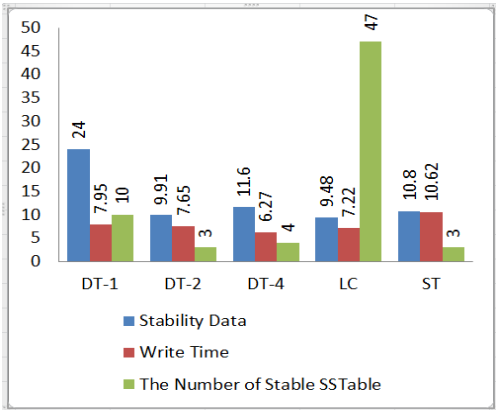

Fig. 6. Comparison of macroscopic physical quantities.

a) The experimental data show that the write of the same amount of data, the time of the Size-Tiered strategy is significantly longer than the rest when compared with the other experiments. Therefore, with the application of Size-Tiered strategy, the write performance of Cassandra is not very good.

b) To set the base_time_seconds of Date-Tiered to 3600 seconds (an hour) in the test DT-1 indicating that the total size of the SSTable files on disk is about 2 times the number of other tests. Through the analysis of the causes of the situation, it is found that the number of the SSTable is necessary to achieve four by default to trigger compaction. So the first and the second time window can quickly meet the compaction conditions to merge, and the window behind will be harder to perform the compaction. With the detailed information of the SSTable files on the disk in the final steady state of the test DT-1, in which two SSTables reached the third window but did not merge owing to the compaction dissatisfaction, and these data are in fact already expired and should be deleted. The size of these expired data is about 9G and the existence of these redundant data directly led to the results of the tests shown in Fig. 6.

c) In the five tests, the number of stable SSTable is more than the other four tests during the application of the Leveled strategy. According to the description of several strategies above, it is easy to know that the sizes of the SSTable generated by the Size-Tiered and Date-Tiered strategies are larger than the size of

the SSTable generated by the Leveled strategy. Therefore, the number of SSTable will be larger naturally in the case of the same total size of the data. So for this kind of continuous insertion and less updating application background in the test, the Leveled strategy is not suitable and it will lead to the decline of reading performance.

d) According to several indicators, the test DT-2, applying Date-Tiered strategy and setting base_time_seconds to 7200 seconds (the same with expiration time), has the optimal system performance.

6. Conclusion

In this chapter, we explore the effect of different compaction strategies on the system performance with the time-series data and analyze the experimental results. According to the experiment results, we can draw the conclusion: the Date-Tiered strategy could be better applied to the environment of time-series data and it is conducive to the data query based on the time condition. It is more convenient to delete the expiration time settled data, with better performance. Therefore, the large data-caching system can migrate the Cassandra to high level versions, in order to improve the system performance by applying the Date-Tiered strategy.

References

[1] Lakshman, A., & Malik, P. (2010). Cassandra: A decentralized structured storage system. *Acm Sigops Operating Systems Review, 44(2)*, 35-40.

[2] Guster, D. C., Schmidt, M. B., & Rice, E. P. (2014). An empirical assessment of performance and scalability of decentralized disk storage for real-time business applications. *Journal of International Business Disciplines, 9(1),* 49-60.

[3] Golab, L., & Zsu, M. T. (2003). Issues in data stream management. *Acm Sigmod Record, 32(2),* 5-14.

[4] Tucker, P., Maier, D., & Sheard, L. T. (2002). Fegaras: Enhancing relational operators for querying over punctuated data streams*. Proceedings of the 28th Very Large Data Bases Conference.*

[5] Jin, C. Q., Qian, W. N., & Zhou, A. Y. (2004). Analysis and management of streaming data: A survey. *Journal of Software, 15(8),* 1172-1181.

[6] Brady Gentile. (2015, June). Getting Started with Time Series Data Modeling. Retrieved June, 2015, from http://academy.datastax.com/demos/getting-started-time-series-data-modeling/

[7] Cattell, R. (2010). Scalable sql and nosql data stores. *Acm Sigmod Record, 39(4),* 12-27.

[8] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., & Burrows, M., *et al.* (2006). Bigtable: A distributed storage system for structured data. *Proceedings of the 7th Conference on Usenix Symposium on Operating Systems Design and Implementation: Vol. 7* (pp. 205-218).

[9] Decandia, G. (2007). Dynamo: amazon's highly available key-value store. *Proceedings of SOSP*: *Vol. 41* (pp. 205-220).

[10] Lakshman, A., & Malik, P. (2009). Cassandra: structured storage system on a P2P network. *Symposium on Principles of Distributed Computing* (p. 5).

[11] Boyd, S., Ghosh, A., Prabhakar, B., & Shah, D. (2006). Randomized gossip algorithms. *IEEE Transactions on Information Theory, 52(6),* 2508-2530.

[12] Slee, M., Agarwal, A., & Kwiatkowski, M. (2007). Thrift: Scalable cross-language services implementation. *Facebook White Paper*.

[13] Lu, G., Debnath, B., & Du, D. H. C. (2011). A forest-structured bloom filter with flash memory. *Mass Storage Systems & Technologies,* 1-6.

[14] Björn, H. (2014, December). Date-Tiered compaction in Apache Cassandra. Retrieved December, 2014,

from https://labs.spotify.com/2014/12/18/date-tiered-compaction/
[15] Marcus, E. (2014, November). DateTieredCompactionStrategy: Compaction for time series data. Retrieved November, 2014, from http://www.datastax.com/dev/blog/datetieredcompactionstrategy/

Bai Lu was born in Zunyi City, Guizhou Province in 1992. She received her B.S degree in computer science and technology from Beijing Jiaotong University in 2013. Now she is a master at Beijing Jiaotong University. Her research interests include distributed computing and cluster computing, etc.

Yang xiaohui was born in 1962. She received her B.S degree in wireless communication from Beijing Jiaotong University in 1983 and received her M.S degree in automatic control from Beijing Jiaotong University in 1986. She is an associate professor at school of computer and information technology in Beijing Jiaotong University. Her research interests include computer application, distributed computing, etc.