International Journal of u- and e- Service, Science and Technology
Vol.8, No.5 (2015), pp.263-274
http://dx.doi.org/10.14257/ijunesst.2015.8.5.24

Research on Long Transaction Support for HBase

Bo Shen*, Xing Miao, Cong Zhang and Ningwei Wang

Department of Electronic and Information Engineering, Key Laboratory of Communication and Information Systems,
Beijing Municipal Commission of Education
Beijing Jiaotong University
Beijing, China
bshen@bjtu.edu.cn

Abstract

Supporting long transaction in HBase has always been a hot issue since NoSQL is widely approved. In this paper, we first divide the existing methods about long transaction of Key-Value databases into two categories: distributed and centralized, and analyze the distributed methods named Percolator and Haeinsa and the centralized methods named HBaseSI and Omid. Then we propose a non-intrusive long transaction approach for HBase, which bases on the idea of pre-conflict checking and timestamp comparison. We design three algorithms for the proposed approach, namely pre-submit, submit-process and snapshot obtain. We also introduce a timeout mechanism based on collision detection. Experiment results indicate that the proposed method has better performance than other long transaction schemes in the system with more-frequent transaction conflict.

Keywords: *HBase; long transaction; pre-conflict checking; snapshot*

1. Introduction

Nowadays, the data expansion in the Internet has make it difficult to manage these heterogeneous data by relational databases [1], because of their limitation in data reading, writing, extention and capacity supporting [2]. To meet the demand of storing complex heterogeneous data, NoSQL database is developed, which uses Key-Value structure to store data [3]. NoSQL database has many advantages, such as easy to extend, high performance, large storage capacity, flexible data model and high availability [4]. It well makes up for the weakness of relational database management system [5]. HBase is one of NoSQL databases, and has many excellent characteristics, such as distributed and column-oriented storage structure [6]. It is a good choice for lot of Internet applications with the pursuit of high-performance.

On the other hand, HBase, like other NoSQL databases, also has some faillings. First, HBase doesn't provide the function of multi-column index, which stops many applications that need multi-condition query to use it. So, it is a meaningful issue how to bring multi-column index mechanism into HBase[7]. Second, for keeping higher performance, HBase only provide weak transaction on the level of data row, instead of strong transaction of ACID level [8]. That leads to the absence of reliable data operation in systems with a lot of high concurrency and long transaction. Therefore, it is also a hot topic that how to enable long transaction service with minimal lost of performance in the current research on HBase and all other Key-Value storage systems [9].

In this paper, we compare the advantages and disadvantages of two main kinds of snapshot methods used in long transaction first. And then, we propose a pre-conflict checking based non-intrusive long transaction approach for HBase, which aims to realize reliable long transaction in ACID level for HBase [10] and meanwhile impact on the

ISSN: 2005-4246 IJUNESST

original performance as little as possible. Next, we present the experiment results from five aspects, i.e. the related metadata size received by client, snapshot for processing delay, the number of transactions per second, the amount of data per second rollback and the fault recovery time. Finally, data analysis results and comparison with the other transaction methods are also given.

2. Related Work

Distributed snapshot method [11] employs the idea that transactional mechanism is distributed into each client, which maintains its own transactional copy of the metadata [12]. Percolator and Haeinsa are the two popular HBase transaction framework implementing distributed snapshot method.

Percolator uses the snapshot isolation (SI) level of transaction support and utilizes a lightweight lock to achieve two stage distributed protocol algorithm [11]. Because it needs to handle huge amounts of data of incremental updating problem, its efficiency on transactional request is not high. At present, an implement of ACID data system has achieved, but only is used in closed system of Google.

Haeinsa is a lightweight transaction framework of HBase to provide strong ACID support across row and table. To use Haeinsa in real system, a lightweight client jar is needed, and then a lock is needed to add to HBase table. Although without complex clustering support, it also increases data overhead. Further, Haeinsa does not provide a global Timestamp to make it only applies to good time synchronization and short transaction environment cluster machines.

Distributed transaction methods mainly employ the algorithms of snapshot isolation [13] level to solve the writing-writing conflict of transaction. In the phase of transaction commit, it is very convenient to design and use. However, these methods need to lock data and add overhead to the data at the data level, which increase the additional overload and bring the characteristics of intrusion data. Moreover, the lock can only be released after the transaction commit failures, which leads to the incensement of delay probability of transaction commitment easily. So, it's not suitable for Key-Value data storage, which required the high performance in most instances.

The main idea of centralized snapshot method is to use a transactional state server to store transactional metadata, receive the request of multi-line affairs, maintain information of committed transactions, judge the time-overlapping and spatial overlap of the current transaction, and ensure transactional data submission. Because transactional state server maintains the whole transaction metadata, the method doesn't need data note to keep the data lock. It then also avoids the overhead caused by the data lock and blocking [14]. HBaseSI and Omid are the typical centralized snapshot methods.

HBaseSI uses two global queues to ensure transaction submission, and to ensure that the former submitter successfully completed the transaction in the case of writing conflict [15]. This approach simplifies the design and implementation of transactional process, and avoids the death caused by longtime data locks and more complex submit agreement negotiations between the nodes. But writing conflict can only be detected by traversing the whole table, and the concurrent in a lot of affairs to submit will bring very big spending. Moreover, there is no rollback strategy when it failed, and two global queues also make obvious performance bottlenecks. So, there is still lots of work to do to achieve a more practical HBaseSI.

Omid designs a lock-free submit algorithm to avoid the damage of distributed lock, and synchronize a small number of transaction metadata to the client. A large part of the query can be realized only on the client side. So, network overhead is reduced [16]. Although the method has a certain improvement in real-time performance, there is no real detection to the competition between submitted data. So, the way of writing data before submitting should be employed. This usually leads to a large amount of data rollback operation,

which increases the overhead of the original distributed system of HBase.

Obviously, the centralized unlocked snapshot transactional method will make little damage to the system performance of HBase. It is more suitable for the system with existing data. But the characteristics of centralized will also lead to larger memory consumption to store the metadata of transaction. Lots of communication loss and memory footprint may be the main problems when a concentrated snapshot method is implemented.

3. A Non-intrusive Long Transaction

We introduce a non-intrusive long transaction approach with pre-conflict checking for HBase in this section. The approach employs the idea of multiple version ordering by timestamp [17]. First, we design the long transaction reading, writing and submit algorithm in accordance with the complete submit order. Then, we design a timeout mechanism based on collision detection.

In our design, we use the idea of forward validation to detect the waiting transaction in the process of submission. When it conflicts, we employ the sacrifice priority strategy which only the first submitted successfully. The idea is not only able to diminish the frequency of the rollback data, but also prevent the correct data from being covered by accident [18]. Therefore long transaction commit algorithm is divided into two parts, namely pre-submit and submit-process, and snapshot algorithm is the reliable guarantee to avoid dirty data being read.

3.1 The Long Transaction Pre-Commit Algorithm

The algorithm is used to detect writing-writing conflict [19] of the transaction, and determine the transaction to submit correctly or break rolled back. It requires the algorithm should be able to decide the conflict by detecting the transaction overlaps on each row changes, so that we can judge the rationality of the long transaction submit, namely judge the correctness of long transaction writing operation.

After the TO server receives the request of txn_i, adopting the tactics of time overlaping and fixed space. Judging from the relationship between the transaction start time and the last time when the row is changed. If the transaction begins after the last changes, it can continue to submit. Otherwise the transaction processing in data processing has been re-written by other transaction, and then the transaction is terminated. If the above situation didn't happen, adopting the tactics of detecting space overlaping and fixed time, and judging writing-writing conflict from committed transaction. If it happened, we ensure that only submit the first request. If all changes are in different rows, it can be submitted concurrently. The design fully avoids the possible conflict of writing-writing problems, and provids a reliable guarantee for the long transaction concurrency. On the basis of no conflict in the validation of all the data writing operation, writing into toBeCommitted() list after a single validation conflict-free. Thus reducing unnecessary terminated transaction, and enhancing the success rate of transaction. Get the new timestamp $T_c\left(txn_i\right)$ from service TO, and write in the waiting list unCommitted. Finally, submit to the client to send pre confirmation, the long transaction pre-submit was ending.

The algorithm implementation process as shown in figure3-1.

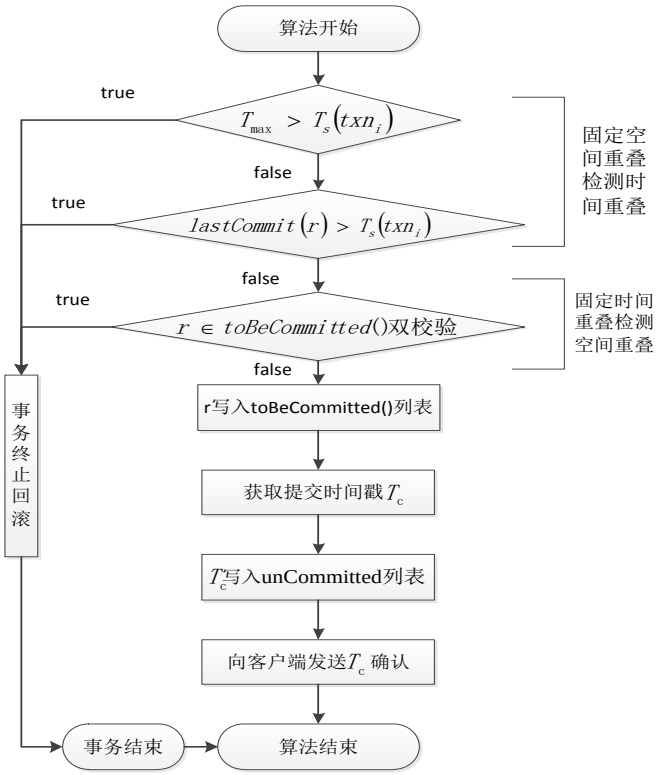

Figure 3-1. The Long Transaction PreCommit Algorithm Flow Chart

The pseudocode of preCommit algorithm as follow, txn_i is this transaction, W is a collection of all write operations of txn_i.

```
preCommit(txn_i ,W){precommit,abort}:
if(T_max > T_s(txn_i))
        return abort;
for each r ∈ W do
if(lastCommit(r) > T_s(txn_i))
        return abort;
else if(r ∈ toBeCommitted())
        return abort;
end if
end for
for each r ∈ W do
        if(r ∉ toBeCommitted()){
            toBeCommitted() ← r
        else
            return abort;
end if
end for
T_c(txn_i) ← TO.time().next()
unCommitted() ← T_c(txn_i)
return precommit;
```


3.2 The Long Transaction CommitEnd Algorithm

After the preCommit algorithm completed, the client received the reply by the server TO, then we can start writing data into the data node. After the completion of the data writing, the client send submit processing requests to the server TO, TO enter the processing transaction after received a request, this phase will commit processing algorithms. The main purpose of submiting processing algorithm is to modify the metadata after the completion of the transaction commit, to provide data support for the subsequent transaction, therefore， we design the following algorithm.

At first, updating the list lastCommit to judge the relationship between subsequent transaction writing operation the corresponding column and the start time T_s .Then, deleting toBeCommitted and unCommitted list relevant information, it has no impact on writing conflicts judgment of subsequent transactions any longer. Finally, we join the transaction log TC to record all information of successful transactions.

The algorithm implementation process as shown in figure3-2.

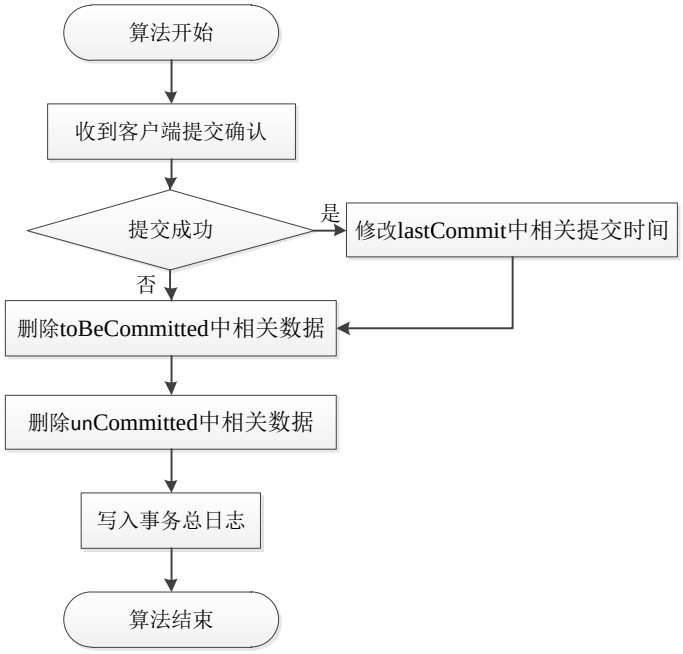

Figure 3-2. The Long Transaction Commit End Algorithm Flow Chart

The pseudocode of commitEnd algorithm as follow, txn_i is this transaction, W is a collection of all write operations of txn_i.

commitEnd(txn_i ,W):{commit}:
    for each $r \in W$ do
        $lastCommit(r) \leftarrow T_c(txn_i)$
        $delete(toBeCommitted(r))$
    end for
    $delete(unCommitted(T_c(txn_i)))$
    $TC \leftarrow txn_i$
return commit;

3.3 Long Transaction isSnapshot Algorithm

For reading operations, ensuring that data received by long transaction is the last submitted snapshot before the transaction started, it is also a long transaction snapshot for the core of the algorithm [20]. According to the idea of forward validation when commit the transaction, we need to use unfinished submitted transaction list unCommitted information in the snapshot algorithm, to judge whether the read data were visual data for this transaction. Therefore, we designed a simple and effective long transaction snapshot algorithm, described as follow.

First, continuously looking for the timestamp of the transaction $T_c(txn_j)$ upward,get the first data which doesn't belong to the unCommitted list timestamp data. then, we judge whether the timestamp Ts later than the start time, if so, constantly reading timestamp version until earlier than the start time of Ts. At this point, the data of this version snapshot timestamp is correct and can be used in this transaction. Algorithm implementation process as shown in figure3-3.

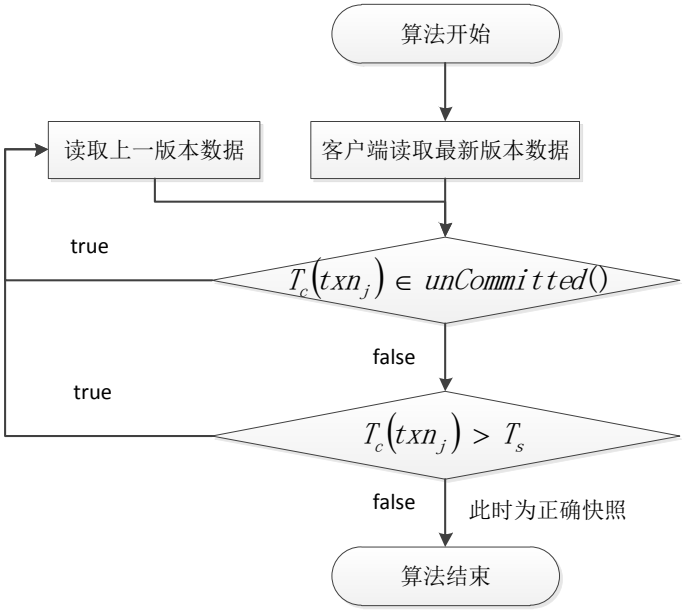

Figure 3-3. The Long Transaction isSnapshot Algorithm Flow Chart

The pseudocode of isSnapshot algorithm as follow, txn_i is this transaction, W is a collection of all write operations of txn_i.

```
    isSnapshot(txn_i, txn_j){true,false}:
if(T_c(txn_j) ∈ unCommitted())
        return false;
else if(T_c(txn_j) > T_s(txn_i))
        return false;
else
        return true;
end if
```


3.4 The Timeout Mechanism Design based on Collision Detection

The timeout mechanism design is mainly divided into timeout judgment mechanism design and timeout waiting time mechanism design.

(1)Timeout judgment mechanism design

During the execution of long transaction pre-submit algorithm on the server, in the toBeCommitted() list test phase, we find out writing-writing conflict, and calculate the difference value between the submission time of the column in the toBeCommitted list and the current time first, if the value is greater than the server timeout waiting time, it is regarded that this transaction which on behalf of the submit time in the toBeCommitted list is timed out. First of all, we need to undo rollback operation, then execute the current transaction, otherwise, it means that there exists writing conflict, then we cancel the current transaction, and rollback the data of the current transaction that has been written to toBeCommitted () list.

The benefits of this timeout judgment mechanism is that there's no need to start process for timeout transaction judge timeout separately, if there is no conflict in the transaction which is due to reasons such as network latency and using of a long time to submit can be completed smoothly, improve the transaction success rate. If there exists conflict, the data can be rolled back in time, and clear the conflict, make the follow-up related transactions can be submitted.

(2)Timeout waiting time mechanism design

Due to the beginning of the transaction was assigned to a submit timestamp, so, the server waiting time T_{wait_server} can be defined as the start from the time which is assigned to a submit timestamp to long transaction commit processing algorithms unCommitted to delete the corresponding data, from the definition we can know:

$$T_{wait\_server} = 2T_{communi} + T_{write} + T_{commit\_process} \quad \ldots(1)$$

$T_{precommit\_confirm}$ represents the time of the data writing between client to the data node.

$T_{commit\_process}$ represents the time that server-side executive transaction commit processing algorithm to success, $T_{communi}$ represents the communications time between the client and server. Without considering the time delaying difference between two-way communication , we can think that:

$$2T_{communi} = T_{precommit\_confirm} + T_{commit\_confirm} \quad \ldots(2)$$

$T_{precommit\_confirm}$ represents the communication time from the server sends pre-submit successfully to the client receives it, $T_{commit\_confirm}$ represents the communication time from the submission of the client confirmation requests to the server received, so

It's ok that the server timeout waiting time $T_{overtime\_server}$ meets: $T_{overtime\_server} > T_{wait\_server}$

In practical applications, according to business requirements and actual transaction size in actual network conditions, we obtain $T_{communi}$, T_{write} and $T_{commit\_process}$,then use these values to design timeout waiting time, to ensure correct operating in long transaction method**.**

4. Performance Analysis

The performance analysis of long transaction method mainly takes five technical indicators into consideration. Such as the size of relevant data which are received by the client, the snapshot for processing delays, the number of transactions by reading and writing per second, the amount of data rollback per second and the fault recovery time .

4.1 The Size of Relevant Data which are Received by the Client

Using three computers to simulate the multi-user concurrent in the form of multithreading, make single value update operations to a table which contain10000 data, obtain the size of the transaction metadata before the TO server sends it. Calculating the size of the metadata received per second, the calculation of method is shown in the following type:

$$N_{meta} = \frac{\sum_{n} N_{meta\_sigle}}{t} \quad \ldots(3)$$

N_{met_asig} is the size of metadata which single access to the transaction. Compared with Omid method the result just like the figure 4-1.

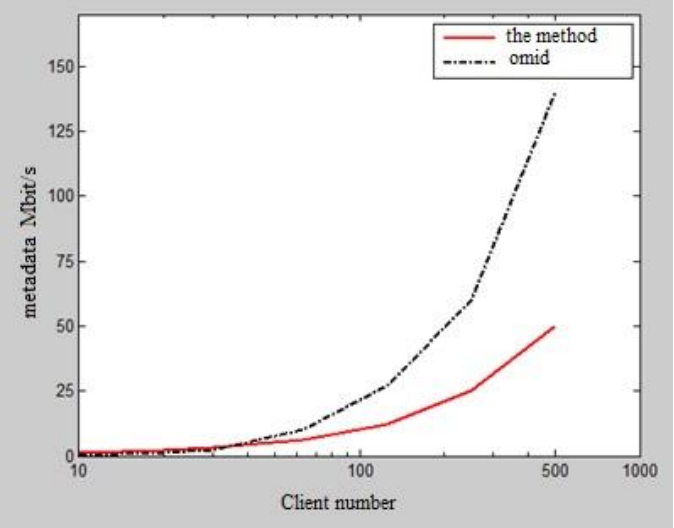

Figure 4-1. Amount of Transactions Metadata Sending by Server

4.2 The Snapshot for Processing Delays

Conducting reading-writing and read-only conflict transactions operations, each transaction reads eight rows of data. The conflict refers to the concurrent updates for the same data. In the reading-writing operations, the difference between the start time stamps of client obtain the transaction T_{getTs} and the time of submit a request T_{cmtrq} is the processing delays. Namely:

$$T_{snapDelay} = \frac{\sum_{n} (T_{cmtrq} - T_{getTs})}{n} \quad \ldots(4)$$

Compared with the original HBase system and Omid，the contrast of data acquisition

process delay is shown in figure 4-2.

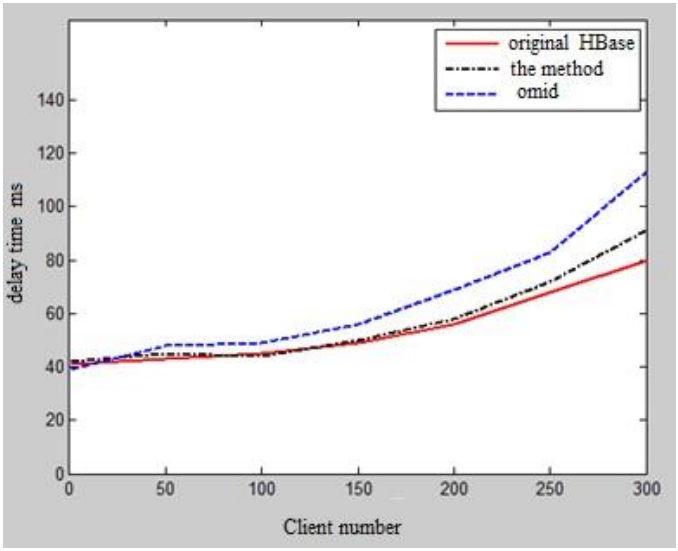

Figure 4-2 Delay of Getting the Right Snapshot

4.3 The Number of Transactions Reading and Writing per Second

Simulating multi-user using at the same time by using multi-threaded concurrent mode, using implementation method based on the pre-conflict checking noninvasive HBase long transaction to update conflicting data operation and record the processed transaction in the TC table. So according to the total number of records in this table for a period of time N_{total},calculating the transaction amount of reading and writing per second, namely:

$$tps = N_{total} / t \quad \ldots(5)$$

Compared with Omid method, HBaseSI method the result like the figure 4.3.

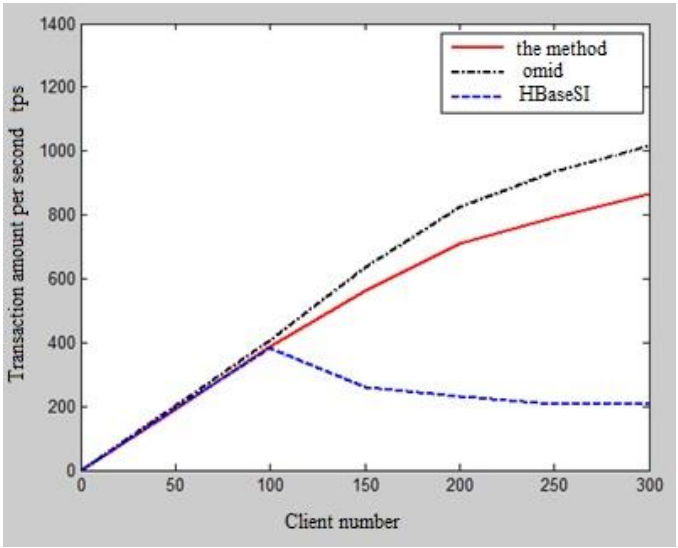

Figure 4-3. TNumber of Transactions per Second

From the comparison results, we can find that with the increment of concurrent transactions, the long transaction implementation method based on the pre-conflict

checking noninvasive HBase, the number of transactions reading and writing per second is less than Omid method, but much larger than HBaseSI method. This is because based on the Pre-conflict checking noninvasive HBase long transaction implementation method write data is based on the pre-conflict checking with a more step for conflict certification in communication, slowing down the speed transaction processing, but can reduce a large number of data rollback after conflict.

4.4 The Amount of Data Rollback per Second

Simulating multi-user using at the same time by using multi-threaded concurrent mode, using implementation method based on the pre-conflict checking noninvasive HBase long transaction to update conflicting data operation. Recording the amount of rollback data that caused by transaction failure after the client receives submit in advance successful validation $N_{rollback}$,namely recording the amount of data that in a period of time the rollback method rollbacked, amount of data per second rollback is calculated according to the type:

$$N_{t\_rollbask} = \frac{\sum_{n} N_{rollback}}{t} \quad \ldots(6)$$

Viewing the change of time that server recovery from failure, and compared with Omid method server, as shown in figure 4-4.

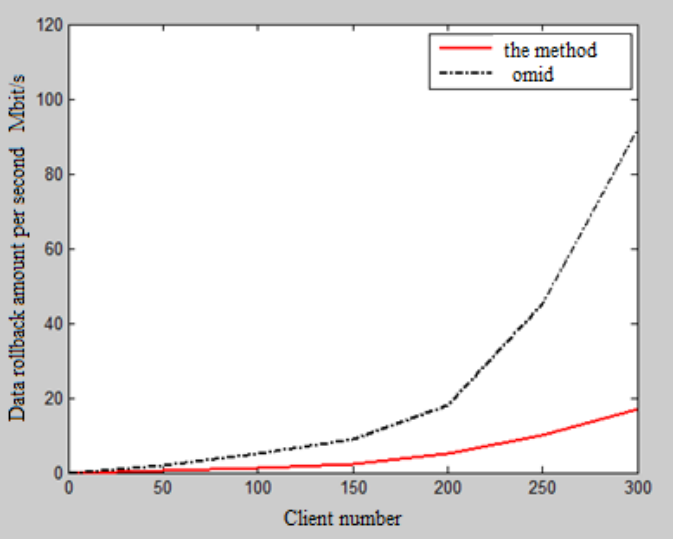

Figure 4-4. Number of Rollbacks per Second

From the comparison results we can conclude that with the increase of transaction volume concurrency conflict, the long transaction implementation method based on the pre-conflict checking noninvasive HBase has less amount of data rollback per second, it has a big advantage on the system with a lot of writing conflict.

4.5 TO Average Server Recovery Time

During the execution of a transaction at a random time, restarting the metadata server process. The difference between record restart time and a long time to re-offer transaction capabilities is the TO server failure recovery time. The formula of repeatedly record and calculate the average is as follows.

$$T_{rc} = \frac{\sum_{n} (T_{funcpr} - T_{reStart})}{n} \quad \ldots(7)$$

To observe system server fail over time, and compared with Omid method server failure recovery time, shown in figure 4-5.

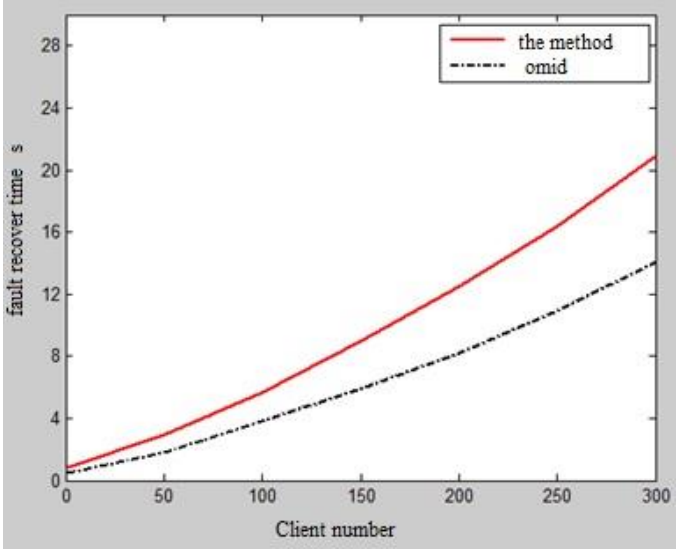

Figure 4-5. Recovery Time Comparison Chart

The results can be seen from the diagram, it may be due to differences in the implementation, the failure recovery time of the method is slower than Omid method, but still in acceptable within the range.

In comprehensive comparison of the five performances, compared with other transaction method, the method has better performance in the system of more frequently transaction conflict, also to avoid the Omid methods to increase the amount of data due to be rolled back and the possiblity of lossing useful information on defects. Therefore, as long transaction high concurrency [11], conflict characterized in a scene with a large amount of e-commerce, baesd on pre-conflict checking non-invasive method to achieve long transaction HBase has better performance.

5. Conclusion

In this paper, we design a kind of noninvasive based on front conflict checking HBase long transaction implementation method, which based on the timestamp comparison, respectively designed a long transaction and submitted by transaction commit processing and long transaction snapshot for three algorithms, and according to the characteristics of the method to design a kind of based on collision detection timeout mechanism. Although its performance exhibits a certain advantages on many aspects, it's necessary for some place to study and design in further.

First, it's not permanent and stable in real applications such as network latency and other applications, so it results the inaccuration of calculation of overtime waiting time. Therefore, it can be considered from the perspective of design prediction and adaptive timeout period.

Second, further reduce the long transaction commit processing algorithm in addition to the completed transaction time impact on subsequent transaction process, which will be able to enhance the transaction success rate.

Third, if we can further reduce each writing transaction amount of data storage, it will

obviously increases the method of massive and the bearing capacity of the write transaction.

Acknowledgement

This work has been supported by the National Natural Science Foundation of China under Grant 61172072, 61271308, and Beijing Natural Science Foundation under Grant 4112045, and the Research Fund for the Doctor-al Program of Higher Education of China under Grant W11C100030, the Beijing Science and Technology Pro-gram under Grant Z121100000312024.

References

[1] Y. Wang, W. Lu, B. Wei, "Transactional multi-row access guarantee in the key-value store. Cluster Computing (CLUSTER)", 2012 IEEE international Conference on **(2012)**, pp. 572-575.
[2] M. Xiaofeng, Z. Longxiang and W. Shan, "Development trend of Database technology", Journal of Software, vol. 12, **(2004)**, pp. 1822-1836.
[3] C. R. Scalable SQL and NoSQL data stores. ACM SIHMOD Record, vol. 39, no. 4, **(2011)**, pp. 12-27.
[4] W. Shan, W. Huiju, Q. Xiongpai, *et al.*, "Architecture of big data: Challenge, The present situation and Prospect", JCST, **(2011)**, vol. 10, pp. 1741-1752.
[5] R. Hecht, S. Jablonski, "NoSQL Evaluation", International Conference on Cloud and Service Computing, **(2011)**.
[6] L. Xing, "HBase performance in-depth analysis", Programmer, (2011), vol. 7, pp. 102-104.
[7] P. Cai, L. Ni, "An approach of multi-row transaction management on HBase with serializable snapshot isolation", Computer Science and Network Technology, 2nd International Conference, **(2012)**, pp. 1741-1744.
[8] F. Jianyong, L. Ming, X. Wei, "The research of vector space data distributed storage based on the HBase", Geography and Geo-information Science, vol. 28, no. 5, , **(2012)**, pp. 39-42.
[9] L. George, "HBase: the definitive guide", O'Reilly Media, Inc., **(2011)**.
[10] C. Xie, C. Su, M. Kapritsos, "Salt:combining ACID and BASE in a distributed database", USENIX Operating Systems Design and Implementation (OSDI), **(2014)**.
[11] Yu W, Lixin L, Shaoyue Z, "Research and implementation on a distributed database synchronization protocol based on the snapshot isolation", Application Research Of Computers, vol. 8, **(2012)**, pp. 3012-3017.
[12] D. Peng, F. Dabek, "Large-scale Incremental Processing Using Distributed Transactions and Notifications", OSDI, vol. 10, **(2010)**, pp. 1-15.
[13] S. Elnikety, F. Pedone, W. Zwaenepoel, "Database replication using generalized snapshot isolation", Reliable Distributed Systems, SRDS 2005, **(2005)**, pp. 73-84.
[14] F. Junqueira, B. Reed, M. Yabandeh, "Lock-free transactional support for large-scale storage systems", Dependable Systems and Networks Workshops (DSN-W), IEEE/IFIP 41st International Conference on. IEEE, **(2011)**.
[15] C. Zhang, H. De Sterck, "Supporting multi-row distributed transactions with global snapshot isolation using bare-bones HBase", Grid Computing, IEEE/ACM International Conference on. IEEE, **(2010)**.
[16] D. Gomez Ferro, F. Junqueira, I. Kelly, "Omid: Lock-free transactional support for distributed data stores", Data Engineering(ICDE), 2014 IEEE 30th International Conference on. IEEE, **(2014)**, pp. 676-687.
[17] L. Junke, W. Yujun, W. Ting, "The solution to the roll back problem in Multi-version concurrency control Timestamp protocol", Computer Science and Network Technology(ICCSNT), 2011 International Conference on. IEEE, **(2011)**, vol. 4, pp. 2803-2806.
[18] A. Fekete, D. Liarokapis, E. O'Neil, "Making snapshot isolation serializable", ACM Transactions on Database Systems (TODS), vol. 30, no. 2, **(2005)**, pp. 492-528.
[19] T. Changcheng, Y. Feng, D. Dong, "Research on the HBase data persistence and usability", Computer Systems & Applications, vol. 10, , **(2013)**, pp. 175-180.
[20] S. Revilak, P. O'Neil, E. O'Neil, "Precisely serializable snapshot isolation", Data Engineering (ICDE), 27th International Conference on. IEEE, **(2011)**.

