Journal of Engineering and Applied Sciences 14 (12): 4102-4108, 2019
ISSN: 1816-949X

HBase Based Multi-Row Transaction Management Techniques

Jeong-Joon Kim
Department of Computer Engineering, Korea Polytechnic University,
Gyeonggi-do, 15073 Siheung-si, South Korea

Abstract: Recently, NoSQL (Not Only SQL) is receiving attention as one of the technologies to deal with big data countless users are making around the world. NoSQL is a database presented as an alternative to the technical and monetary restrictions of conventional relational database and focusing on availability and expandability for swift treatment of atypical data. But NoSQL does not guarantee data integrity because it abandons transaction for availability and scalability. Thus, studies are underway to implement the multi-row transaction on NoSQL in particular, various studies on the multi-row transaction system based on HBase has been actively in progress. However, traditional studies are limited to performance improvement proportional to number of clients and low concurrency because there are too many information that column manages. Therefore, in this study, propose an efficient multi-row transaction system based on HBase has exceptional performance improvement proportional to number of clients and supports high concurrency. This system creates column for managing transaction information. In addition, designs and implements the transaction manager for efficiently controlling the state of transaction and communication manager for exchanging information it need for transaction by communicating with HBase.

Key words: Multi-row transaction, HBase, concurrency control, transaction recovery, proportional, conventional relational

INTRODUCTION

Concurrent input and various attributes of big data have limitations in terms of technical or cost to process in existing relational database. Therefore, various kinds of NoSQL such as a column-based database and a document-based database have appeared to handle this problem (George, 2011). Many NoSQL solutions for big data processing are aimed at most distributed environments. Typical NoSQL solutions include Bigtable, HBase, MongoDB and Cassandra (Abadi *et al*., 2009).

However, these NoSQL solutions do not support multiple row transactions for availability and scalability which is a disadvantage of data integrity. The integrity of data is crucial to real services that deal primarily with financial or personal information. Therefore, to use a NoSQL solution as a commercial service, there is a need to develop a multi-row transaction system for the NoSQL solution. Because of this necessity, the NoSQL-based multi-row transaction system has recently become an important research field and related researches are actively proceeding (Dean and Ghemawat, 2004).

In this study, we design and implement an HBase-based Multi-Row Transaction system (HMRT) through client library implementation on HBase of Hadoop (Ghemawat *et al*., 2003; Levandoski *et al*., 2011; Peng and Dabek, 2010) which is one of the most recently studied NoSQL. The system designed and implemented a transaction execution manager to control transaction execution and added a collision detection and recovery manager to detect and effectively recover from a conflict situation. In addition, we added a special column to the HBase table to design for high concurrency. In the case of a transaction conflict, the collision information manager can quickly recover the collision using the collision information.

Literature review

Multi low transactions: A multi-row transaction means that individual operations for two or more rows are grouped together into a single unit of work. Table 1 is an instance of a table called account which manages account balance information (Stonebraker, 2010).

There are currently three rows A-C in the account table in Table 1. Each row contains a balance of $ 100, 20 and 30. Assuming that there is a transaction “fetch $ 50 from A”, this transaction is a single row transaction because it only accesses one row of the table’s rows. However, assuming that there is a transaction of ‘transfer

Table 1: Account table

Row key	Values
A	100
B	20
C	30

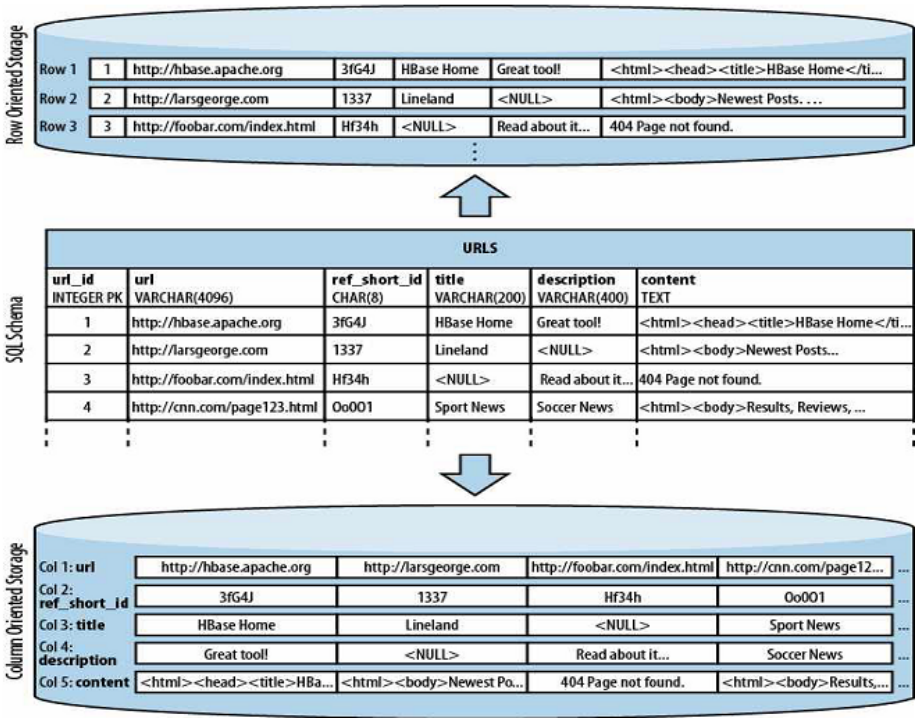

Fig. 1: Column-based and row-based storage

Table 2: In the case of the operation on row C fails

Row key	Values
A	40
B	50
C	30

$ 30 to B and C rows in row A,' this transaction is a multiple row transaction because it needs to access two or more rows.

To execute the above example, you need to subtract the balance from A and add the balance to B and C. However, if any of these operations fails, all operations must be considered as failed. Table 2 shows that the operation to insert the balance subtracted from A into B succeeded but the operation to insert the balance subtracted from A into C failed.

Table 2, A has gone out of $ 60 to send $ 30 to B and C, respectively. Row B receives $ 30 of the balance subtracted from A to be $ 50 but row C can confirm that the calculation failed and there is no change in the balance. In this case, all operations must be canceled to ensure the integrity of the data, so, a multi-row transaction is required.

However, existing NoSQL does not support multi-row transactions, so, data integrity cannot be guaranteed in the above situation. Therefore, in order to utilize NoSQL even in areas where data integrity is important, there is a steady progress to apply multirow transactions to NoSQL recently.

HBase: HBase is an open source clone project from Google Bigtable and is a column-based distributed database that hooks into Hadoop, a kind of NoSQL (Shin *et al*., 2016). HBase manages the data in memory and takes a write delay method that flushes to an aligned file when memory is full. Therefore, it is possible to reduce the disk I/O as much as possible and HDFS is used as a storage as a distributed database suitable for random reading and real time reading/writing of small data.

HBase is a column-based database that stores data of the same type in columns. Relational databases perform data reading and writing in row units and are more advantageous when using the entire row. However, a column-based database such as HBase has better storage efficiency because it stores data with the same data type by column and performs better than existing relational database when I/O is performed on a single column basis. Figure 1 shows the differences between column-oriented storage and row-oriented storage.

When the schema of the web page is called URLS in Fig. 1, URL Schema of URLS consists of url_id (INTEGER), url (VARCHAR (4096)), ref_short_id (CHAR (8)), title (VARCHAR Description (VARCHAR (400)) and content (TEXT) columns. In a relational database, various attributes of URLS are stored in a single tuple through a

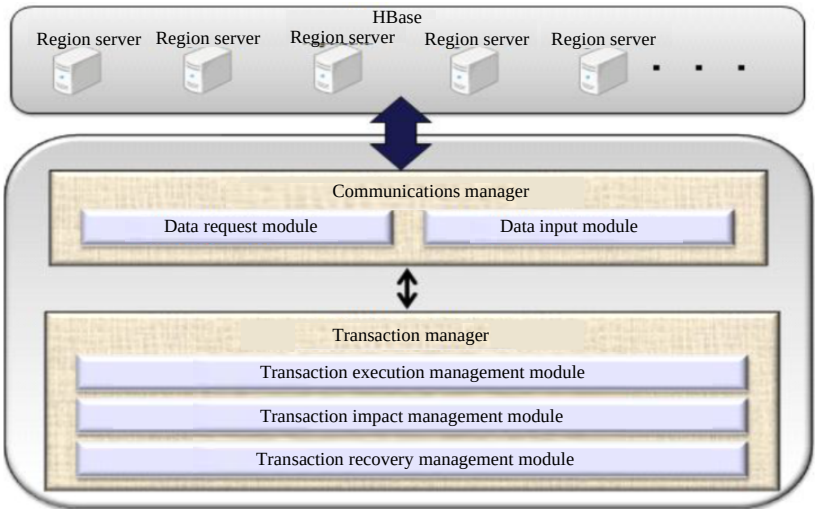

Fig. 2: Overall system architecture

row-based storage method as shown above in Fig. 1. This results in poor spatial efficiency because it requires storing various types of data and storing empty spaces such as NULL values. On the other hand in the column-based storage method, data having the same shape is displayed by storing data for each column having similar properties as shown in Fig. 1. This provides a basis for increasing space efficiency which can greatly reduce the load on the network in a distributed processing environment. Thus, a column-based database is an effective alternative to handling large data in which instances of rows are not uniformly typed.

MATERIALS AND METHODS

System design

Overall system architecture: The multirow transaction system proposed in this study is based on HBase, a column-based distributed database among Hadoop platforms.

Existing HBase supports transactions for a single row but does not support multiple row transactions that require concurrent access to multiple rows. Therefore, in this system, a transaction column that manages transaction information is added to HBase in order to support multi row transactions in HBase. And a transaction manager that manages transaction execution, conflict and recovery and a communication manager which is responsible for data request and data input are designed and implemented.

We have designed HMRT, a system that can efficiently process multiple row transactions in HBase by applying the functions described above. Figure 2 shows the structure of HMRT.

Table 3: Initial state of table A

Row key	Values	Transactions
1	100	Stable 1
2	5000	Stable 4
3	600	Stable 6
4	1300	Stable 4
5	1700	Stable 3
6	300	Stable 7
7	200	Stable 1
8	1000	Stable 2
…	…	…
100000	400	Stable 8

Communications manager: The communication manager is responsible for requesting HBase for transaction execution, collision and recovery between the transaction manager and HBase and for transferring them to the transaction manager. Transaction execution management module, data request module and data transfer module.

Transaction manager: The transaction manager is used to efficiently control the execution, collision and recovery of transactions. Transaction execution management module, transaction conflict management module and transaction recovery management module. Each module performs operations through the communication manager with transaction execution information, transaction conflict information and transaction recovery information.

RESULTS AND DISCUSSION

System implementation

Transaction execution: In this study, the HMRT developed in this study is shown. Table 3 shows that a total of 100,000 data are inserted in table A. Table 3 shows the initial states of table A in order to show the state change of transaction columns.

As shown in Table 1-4, table A has the row keys arranged in ascending order and the value column has

Fig. 3: The screen where the transaction is running

Table 4: Initial state of table B

Row key	Values	Transactions
1	900	Stable 3
2	700	Stable 6
3	1200	Stable 7
4	5000	Stable 4
5	1800	Stable 1
6	500	Stable 4
7	600	Stable 9
8	2000	Stable 5
….	…	…
100000	1200	Stable 5

arbitrary values. The lock state of the transaction column is all stable and each has its commit timestamp at the time of its final commit. A total of 100,000 of these data are input.

Table 4 shows the table B initial states in order to show the state change of the transaction columns. As shown in Table 4, the B table is also arranged in ascending order of row keys and arbitrary values are inputted in the value column. The transaction state of the transaction column is all stable and each has its commit timestamp at the time of its final commit. Similarly, 100,000 of these data are entered.

Now, we execute a total of 100 million multirow transactions by sending value of row A of table A to row 1 of table B and value of row 2 of table A to row 2 of table B. Figure 3 shows the screen where the multi-row transaction is executed.

Figure 3, push data success means that the transaction has been performed successfully and waring for lock released means that there is a conflict between transactions. Table 5 shows the moment when the transaction is executed in the row 1 of the table A during the execution of the transaction, the lock status of the transaction column and the commit timestamp are changed and the value 100 is exited with the transaction column.

Table 5: Situation where value is missing from table A

Row key	Values	Transactions
1	0	Prewritten 10
…	…	…
100000	400	Stable 8

Table 6: Situation where value is missing from table B

Row key	Values	Transactions
1	1000	Prewritten 10
…	…	…
100000	1200	Stable 5

Table 7: Table A with transaction execution completed

Row key	Values	Transactions
1	0	Stable 10
2	0	Stable 162
3	0	Stable 33
4	0	Stable 4523
5	0	Stable 8632
6	0	Stable 452
7	0	Stable 667
8	0	Stable 25609
…	…	…
100000	0	Stable 56283

Table 8: Table B with transaction execution completed

Row key	Values	Transactions
1	1000	Stable 10
2	5700	Stable 162
3	1800	Stable 33
4	6300	Stable 4523
5	3500	Stable 8632
6	800	Stable 452
7	800	Stable 667
8	3000	Stable 25609
…	…	…
100000	1600	Stable 56283

Table 5, a transaction with a commit timestamp of 10 is accessed to prevent other transactions from accessing by changing the lock state of row 1 of the row transaction to prewritten state. Table 6 shows the value 100 entered from table A in B. Similarly, the lock status and commit timestamp of the transaction column are changed and value 100 is added.

Table 7 shows the state of the table A in which the transaction execution is completed to show the state change of the transaction column together.

Table 7, all the values are changed to 0 and the transaction column status is returned to stable again. In addition, the commit timestamp has also changed. Table 8 shows the state of table B in which transaction execution is completed to show the state change of transaction column together. This shows how multiple row transactions are executed in the HMRT.

```
99985          column=cf:value, timestamp=1423056343331, value="300"
99986          column=cf:value, timestamp=1423056343331, value="700"
99987          column=cf:value, timestamp=1423056343331, value="1600"
99988          column=cf:value, timestamp=1423056343331, value="800"
99989          column=cf:value, timestamp=1423056343331, value="700"
99990          column=cf:value, timestamp=1423056343331, value="0"
99991          column=cf:value, timestamp=1423056343331, value="0"
99992          column=cf:value, timestamp=1423056343331, value="500"
99993          column=cf:value, timestamp=1423056343331, value="1200"
99994          column=cf:value, timestamp=1423056343332, value="300"
99995          column=cf:value, timestamp=1423056343332, value="1100"
99996          column=cf:value, timestamp=1423056343332, value="500"
99997          column=cf:value, timestamp=1423056343332, value="600"
99998          column=cf:value, timestamp=1423056343332, value="0"
99999          column=cf:value, timestamp=1423056343332, value="100"
100000         column=cf:value, timestamp=1423056343332, value="600"

100000 row(s) in 111.0500 seconds
hbase(main):021:0>
```

Fig. 4: Screen with data entered in table C

Table 9: Initial state of table C

Row key	Values	Transactions
1	1000	Stable 9
2	500	Stable 8
3	100	Stable 2
4	300	Stable 1
5	700	Stable 6
6	1300	Stable 4
7	1100	Stable 5
8	200	Stable 7
…	…	…
100000	600	Stable 7

Table 10: Situation where Value is missing from table A

Row key	Values	Transactions
1	0	Prewritten 10
…	…	…
100000	400	Stable 8

Table 11: State where a conflict occurred in table B

Row key	Values	Transactions
1	1900	Prewritten 7
…	…	…
100000	1200	Stable 5

Table 12: State of table B after T2 is complete

Row key	Values	Transactions
1	1900	Stable 7
…	…	…
100000	1200	Stable 5

Table 13: State of table A after the rollback is complete

Row key	Values	Transactions
1	100	Stable 1
…	…	…
100000	400	Stable 8

Transaction conflict: In this study, HMRT shows the collision of multiple row transactions and their solution. Figure 4 shows HBase's table C with 100,000 data entries.

Table 9 shows the initial state of table C as a table to show the state change of transaction column together. Table C has the same structure as table A and B and has 100,000 rows as well. You want to transfer the value from table A-B in the same way as the preceding transaction execution example. At this time, table C also executes 100,000 multiple row transactions that transfer the value to table B in the same manner as table A. Since, both the value of table A and the value of table C must be transmitted to Table B in B, a conflict occurs between the transaction accessed in table A and the transaction accessed in table C. A transaction with a conflict will be queued up as "waiting for lock released" as shown in Fig. 4.

Table 10 shows that Transaction T1 is executed in row 1 of row A and the lock status and commit timestamp of transaction column are changed and value 100 is exited.

Table 10, Transaction T1 with a commit timestamp of 10 is accessed to prevent another transaction from accessing by changing the lock state of row 1 of the row transaction to prewritten state. Table 11 shows that the Transaction T2 generated in the C table occupies the table before entering the value 100 that has exited from the table A in the B.

As in Table 11, you can see that transaction of B Table row 1 is already prewritten by Transaction T2 with commit timestamp 7. Therefore, transaction T1 waits until the transaction column lock status of B table row becomes stable. Table 12 shows that Transaction T2 completes the operation and changes the transaction column lock status of B table row 1 to stable and the transaction is completed.

Table 12, since, the lock status of the transaction column is stable, the Transaction T1 that is waiting is accessible. At this time, however, the commit timestamp 1 which was initially known to T1 is different from the commit timestamp 7 stored in the current transaction column, so, the transaction must be rolled back and re-executed. As described above, the transaction conflict management module transmits the rollback data to the data input module and the data input module performs the put operation using the rollback data, thereby returning the table A to its original state as shown in Table 13.

Table 14: State of table A after re-execution

Row key	Values	Transactions
1	0	Prewritten 10
…	…	…
100000	400	Stable 8

Table 15: State of table B after re-execution

Row key	Values	Transactions
1	2000	Prewritten 10
…	…	…
100000	1200	Stable 5

Table 16: Table A that has been re-executed after the rollback

Row key	Values	Transactions
1	0	Stable 10
2	0	Stable 162
3	0	Stable 33
4	0	Stable 4523
5	0	Stable 8632
6	0	Stable 452
7	0	Stable 667
8	0	Stable 25609
…	…	…
100000	0	Stable 56283

Table 17: Table B that has been re-executed after the rollback

Row key	Values	Transactions
1	2000	Stable 10
2	6200	Stable 162
3	1900	Stable 222
4	6600	Stable 1966
5	4200	Stable 8632
6	2100	Stable 452
7	1900	Stable 667
8	3200	Stable 9902
…	…	…
100000	2200	Stable 56283

As shown in Table 13, after the rollback is completed, a multi-row transaction is performed to transfer the value to the table B using the transaction execution information again. Since, the value corresponding to row key 1 of table B is 900-1,900, insert 2,000 instead of 1,000 which was inserted before rollback. Table 14 shows that the transaction is re-executed after the rollback and the value is again missing from the table A.

Transaction T1 with commit timestamp 10 re-accessed in Table 14 re-approaches to change the lock state of row 1 row transaction to prewritten state. Therefore, other transactions can not be accessed. In this case, no collision occurred and table B was changed to prewritten state as shown in Table 15 and values were normally inserted.

Table 4-13, we can see that the value of T1 is updated normally by performing the operation on table B. Table 16 shows the final state of table A after T1 has successfully completed re-execution.

Table 16, we can see that the value of table A is exited and the status of the transaction column becomes stable. Table 17 shows the final state of table B after T1 and T2 have successfully completed re-execution.

Table 18: Table C with transaction execution completed

Row key	Values	Transactions
1	0	Stable 7
2	0	Stable 83
3	0	Stable 222
4	0	Stable 1966
5	0	Stable 8409
6	0	Stable 437
7	0	Stable 1797
8	0	Stable 9902
…	…	…
100000	0	Stable 7921

Table 19: Table A where the transaction was aborted due to a system

Row key	Values	Transactions
1	0	Prewritten 10
…	…	…
100000	400	Stable 8

Table 20: State of table B after recovery

Row key	Values	Transactions
1	100	Prewritten
…	…	…
100000	400	Stable 8

It can be seen that the existing value is added to the value entered in table A and the value entered in table C. Finally, Table 18 shows the appearance of table C after the successful completion of T1 and T2. In this way, HMRT shows the collision of multiple row transactions and the solution process.

Transaction recovery: This study shows the recovery process, when multiple row transactions in HMRT are executed abnormally and are restarted. Table 19 shows the system error before adding the value of table B after the value of table A becomes 0 in the preceding transaction execution example.

Table 19 shows that when a transaction to be added to the value of a row with a row key of 1 is executed in table B except for the value of a row whose row key is 1 in table A, the status of the transaction terminated abnormally. Table 20 shows that the transaction is restored and the value is successfully inserted into the B table.

As shown in Table 20, it can be confirmed that the value is normally input to the table B. Table 21 shows the table A after the transaction completes the recovery and all the operations are completed successfully in the above manner.

Table 22 shows table B after the transaction completes the recovery and all the operations are completed successfully.

As shown in Table 22, it can be confirmed that the value is normally inputted into the Table B. As a result, the HMRT was shown to have completed its normal operation after recovering from an abnormal system error.

Table 21: State of table A after recovery

Row key	Values	Transactions
1	0	Stable 10
…	…	…
100000	0	Stable 56283

Table 22: Table B with recovery completed

Row key	Values	Transactions
1	1000	Stable 10
…	…	…
100000	2200	Stable 56283

CONCLUSION

As interest in big data has increased recently, NoSQL, a solution for storing and processing big data is getting attention. NoSQL supports high speed, high availability and high scalability but it is limited in areas where data integrity is important because it does not support multiple row transactions. To overcome these drawbacks, many studies are underway to support multiple row transactions in NoSQL.

Therefore, this study proposes an HBase-based efficient multi-row transaction system that can add a column that manages transaction information to all user tables and separate multiple modules for managing recovery information to perform multi-row transactions with low load and high concurrency. This study creates column for managing transaction information. In addition, designs and implements the transaction manager for efficiently controlling the state of transaction and communication manager for exchanging information it need for transaction by communicating with Hbase.

ACKNOWLEDGEMENT

This research was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2017R1A2B4011243).

REFERENCES

Abadi, D.J., P.A. Boncz and S. Harizopoulos, 2009. Column-oriented database systems. Proc. VLDB. Endowment, 2: 1664-1665.

Dean, J. and S. Ghemawat, 2004. MapReduce: Simplified data processing on large clusters. Proceedings of the 6th Symposium on Operating Systems Design and Implementation, December 6-8, 2004, San Francisco, CA., USA., pp: 137-150.

George, L., 2011. HBase: The Definitive Guide. O'Reilly Media, Sebastopol, California, USA., ISBN:9781449315221, Pages: 556.

Ghemawat, S., H. Gobioff and S. Leung, 2003. The google file system. ACM SIGOPS Operat. Syst. Rev., 37: 29-43.

Levandoski, J.J., D.B. Lomet, M.F. Mokbel and K.K. Zhao, 2011. Deuteronomy: Transaction support for cloud data. Proceedings of the 5th Biennial Conference on Innovative Data Systems Research, January 9-12, 2011, Asilomar, CA., USA., pp: 123-133.

Peng, D. and F. Dabek, 2010. Large-scale incremental processing using distributed transactions and notifications. Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, October 4-6, 2010, Vancouver, Canada, pp: 1-15.

Shin, I.S., J.J. Kim, Y.S. Lee and J.Y. Moon, 2016. NoSQL-based spatial data processing systems in big data environments. Intl. Inf. Instit. Tokyo Inf., 19: 4219-4236.

Stonebraker, M., 2010. SQL databases v. NoSQL databases. Commun. ACM., 53: 10-11.