Research on Concurrency Control in Database Systems

Pengxu Shen[1] and Hanwei Qian[1,2,3](✉)

[1] Department of Computer Information and Cyber Security, Jiangsu Police Institute, Nanjing 210031, China
qianhanwei@jspi.edu.cn
[2] Engineering Research Center of Electronic Data Forensics Analysis, Nanjing, Jiangsu, China
[3] Key Laboratory of Digital Forensics, Department of Public Security of Jiangsu Province, Nanjing, China

Abstract. In a database system, if the degree of concurrency is high, using a concurrency control algorithm alone will reduce the performance of the system, and the process of selecting a concurrency control algorithm will virtually improve the knowledge threshold of users. In order to overcome this limitation, this paper proposes a hybrid concurrency control algorithm, which is called cluster based concurrency control algorithm. It creatively puts forward the concept of transaction working set, uses the minimum hash algorithm to calculate the Jaccard similarity between different transaction working sets to measure the conflict rate between different transactions, and uses this as a standard to place transactions in different clusters, Transactions in the same cluster adopt pessimistic concurrency control algorithm, and transactions in different clusters adopt optimistic concurrency control algorithm. The clustering based concurrency control algorithm combines the traditional pessimistic concurrency control algorithm with the optimistic concurrency control algorithm to obtain the advantages of the two algorithms and alleviate the performance bottleneck of the two algorithms. Finally, through simple experiments, it is proved that the concurrency control algorithm based on clustering is indeed better than the traditional pessimistic and optimistic concurrency control algorithms.

Keywords: Database · Concurrency control · Clustering · MinHash

1 Introduction

In a database, a transaction is a set of operations that constitute a single logical unit of work. Take MySQL as an example. In MySQL, a transaction consists of one or more SQL statements of a single unit, that is, in a database transaction, there can be only one operation or multiple operations, but these operations form a logical whole, so these operations can only be executed successfully or not.

Q. Liu et al. (Eds.): CENet 2022, LNEE 961, pp. 1223–1232, 2022.
https://doi.org/10.1007/978-981-19-6901-0_128

In any database, transactions have four characteristics of acid: atomicity, consistency, isolation and durability. Atomicity, that is, all operations in a transaction are inseparable and consistent with the nature of atoms. Either all operations are successfully executed or all operations are not executed; Consistency means that when a transaction is completed, the database must be transferred from one consistency state to another consistency state; Isolation means that when multiple transactions are executed concurrently, different transactions will not affect each other; Persistence means that the update operation of the transaction to the database is permanent and will not cause data loss under any circumstances.

Among the four characteristics of acid in transactions, consistency is the most important pursuit of transactions. Therefore, ensuring data consistency is the primary goal of concurrency control. In order to improve the access speed of different clients to the data in the database, multiple transactions generally do not use serial execution, but use concurrent execution. In this operation mode, the idle time of each part of the computer can be fully utilized. However, because different transactions in the database share the data in the database with each other, if the operation of these transactions is not controlled, multiple transactions operate on the same data at the same time, which may lead to data inconsistency, which may cause some disastrous consequences.

2 Overview of Database Concurrency Control

2.1 Introduction to Database Concurrency Control

Concurrency control refers to various technologies used to protect database integrity when multiple users update and run at the same time. In computer science, especially in the fields of program design, operating system, multiprocessing and database, concurrency control is a mechanism to ensure the timely correction of errors caused by concurrent operations. The purpose of concurrency control is to ensure that the work of one user will not have an unreasonable impact on the work of another user. In some cases, these measures ensure that when the user operates with other users, the result is the same as when she operates alone. In other cases, this means that the user's work is affected by other users in a predetermined manner [1].

Concurrency control is not only to ensure the consistency of transactions, but also to maximize the performance of the database. In order to ensure the consistency of transactions, the simplest way is to let transactions execute one by one, that is, serial execution. In such a running environment, transactions are perfectly isolated from each other, and data inconsistency cannot occur. However, this running mode will make other transactions wait for a long time, and make computer components produce a lot of idle performance, resulting in a great loss of database system performance. Therefore, how to improve the performance of database system as much as possible under the condition of ensuring transaction isolation and data consistency is a permanent proposition. However, the reality is that although the research of concurrency control has a long history, there is still no concurrency control algorithm that can ensure its performance better than all other algorithms in any workload scenario.

In order to make up for the above defects and reduce the use threshold of the database system on the basis of ensuring the data consistency and performance of the database,

this paper mainly studies whether different concurrency control algorithms can be integrated, designs a new concurrency control algorithm, and allows the algorithm to choose different concurrency control methods, so as to give full play to the advantages of the original concurrency control algorithm and overcome the shortcomings of the original concurrency control algorithm, So that each concurrency control algorithm can work under the workload that can play its own performance. In this way, users do not need to assume the workload before using the database system or understand the internal mechanism of the database system in advance. Instead, the database system can detect the different characteristics of the actually processed transactions and intelligently select the isolation mode from other transactions, On the premise of ensuring isolation and data consistency, it obtains better performance than the traditional concurrency control algorithm.

2.2 Concurrency Control Algorithms

From the way of conflict handling, the existing concurrency control algorithms can be divided into pessimistic concurrency control and optimistic concurrency control.

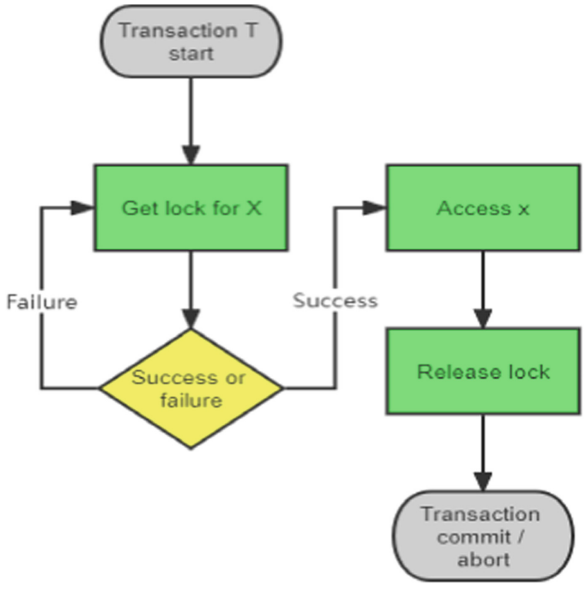

Fig. 1. Transaction workflow of the simplest pessimistic concurrency control algorithm

As shown in Fig. 1. Transaction workflow of the simplest pessimistic concurrency control algorithm, the pessimistic concurrency control method adopts the "pre prevention" method, that is, lock acquisition is required before accessing each data object, so as to ensure that there will be no conflict between transactions. For example, when a transaction enables the pessimistic concurrency control algorithm and updates a data such as K1, and another transaction wants to update K1, the update fails and the lock timeout

occurs. The representative of this kind of method is the two-phase locking algorithm (2PL), which requires the transaction to try to obtain the lock corresponding to the data object before accessing the data object. If the transaction successfully obtains the lock, it can run normally. On the contrary, it must wait or abort, so as to avoid conflict.

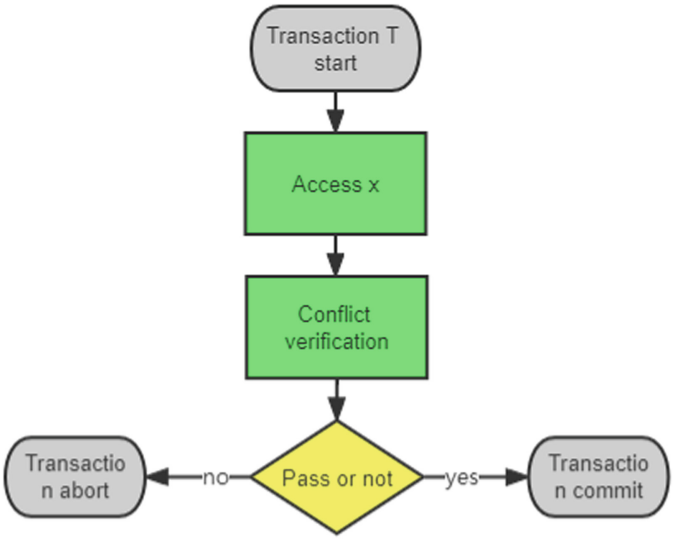

Fig. 2. The basic workflow of transactions under the simplest optimistic concurrency control algorithm

In contrast, as shown in Fig. 2. The basic workflow of transactions under the simplest optimistic concurrency control algorithm, the optimistic concurrency control (OCC) algorithm adopts the method of "making up for the lost sheep". Each transaction first operates on the data record. After the operation is completed, the transaction will not be submitted immediately, but will go through a conflict detection stage, that is, to detect whether the transaction conflicts with other running transactions. If a conflict is detected, Then the operation of the current transaction must be rolled back and the transaction must be aborted. For example, if an optimistic transaction is enabled, when a transaction wants to update K1, and then another transaction wants to update K1, the operation will succeed, but it will go through a conflict detection process in the commit phase, and then commit fails.

There are great differences in the design concepts between the two algorithms, resulting in their different limitations. Firstly, for pessimistic concurrency control algorithms, the performance loss is mainly the time loss of transactions in the process of competing for locks. This is because in common pessimistic concurrency control algorithms, a transaction must hold the lock corresponding to the data record during the whole process of work, that is, from the beginning of accessing the data record to the end of the operation submission, so as to avoid data inconsistency. In contrast, the optimistic concurrency control algorithm only needs the transaction to hold the lock of the corresponding data record in the conflict verification stage after the data access and before the transaction

submission, but it does not need to hold any lock in the data access stage. Therefore, the transaction with pessimistic concurrency control algorithm holds the lock for much longer than the transaction with optimistic concurrency control algorithm. From the definition of lock, it can be concluded that when a transaction holds the lock of a data record, any other transaction can no longer obtain the lock of the data record, and then can no longer access the data record. The lock can be obtained only after the transaction holding the lock completes all operations and commits. Unless both transactions have to read the data record, it obviously greatly limits the performance of the database system.

3 Concurrency Control Principle Based on Clustering

3.1 Basic Idea of Algorithm

The basic idea of clustering based concurrency control algorithm is similar to that of other hybrid concurrency control algorithms. They all try to combine pessimistic concurrency control algorithm and optimistic concurrency control algorithm, hoping to give full play to their respective advantages and overcome their respective defects. Generally speaking, when there is a high possibility of conflict between two transactions, the pessimistic concurrency control algorithm should be used to control them by letting them compete for locks in advance, which can effectively avoid the invalid operation caused by the failure of conflict verification by using the optimistic concurrency control algorithm. When the possibility of conflict between the two transactions is small, the optimistic concurrency control algorithm should be adopted to let the transactions directly operate the data records and maximize the performance of the database system.

According to the above idea, the concurrency control algorithm based on clustering uses the minimum hash algorithm to cluster the transactions in the system, that is, grouping, and divide the transactions with high conflict possibility into the same group, while the transactions with low conflict possibility into different groups. Such a group is called a cluster. After the grouping is determined, based on the above principle, transactions in the same group should be controlled by pessimistic concurrency control algorithm because they are likely to conflict with each other, that is, lock based isolation. The transactions in different groups are isolated by optimistic concurrency control algorithm, that is, verification based method, because the possibility of conflict is small.

The method of transaction clustering is described in detail below. The concurrency control algorithm based on clustering clusters transactions according to the similarity of the data set to be accessed between the transactions that will access the data records. This set is called the working set of transactions. The reason for clustering in this way is that in the database system, conflicts may occur only when two transactions operate on the same data record at the same time. Therefore, the higher the similarity of the working sets of two transactions, the higher the conflict rate of the two transactions. Therefore, the coincidence degree of the working sets of transactions can be taken as an approximation of the conflict possibility between transactions. The following describes the clustering based concurrency control algorithm, which combines pessimistic concurrency control algorithm and optimistic concurrency control algorithm.

3.2 MinHash Algorithm

Although the implementation details of traditional clustering algorithms such as k-means and hierarchical clustering algorithm are different, they generally need to traverse the whole data set for many times before calculating the similarity between different data point pairs, so their algorithm performance is poor. The concurrency control algorithm based on clustering introduced in this paper does not need to traverse the whole data set repeatedly. It only needs to adjust some parameters to adjust the time complexity of the algorithm. Both flexibility and performance are better than the above traditional clustering algorithm.

Transaction clustering algorithm is based on a similarity algorithm, which is called local sensitive hashing (LSH) [5]. This is a special hash function, which can map two data with high similarity to the same hash value with high probability, and two data with low similarity to the same hash value with very low probability [6].

Different algorithms use different similarity measures, and for different similarity measures, the specific local sensitive hash algorithm is also different. The algorithm design used in the standard Jaccard similarity of transaction clustering proposed in this paper is called the minimum hash algorithm [8]. The principle of minimum hash algorithm is introduced in detail below.

4 Implementation

4.1 Implementation of Transactions Based on Clustering

This section mainly discusses the implementation of the cluster based concurrency control algorithm in the specific database system, because this algorithm needs to be implemented on the basis of the traditional pessimistic concurrency control algorithm and optimistic concurrency control algorithm. Therefore, the implementation of the algorithm selects the Redis open source database that provides these two algorithms, which can maximize the code reuse and reduce the implementation difficulty.

In the cluster based concurrency control algorithm, according to the working principle of the above minimum hash function, first, all transactions in the database system need to be hashed, and whether different transactions will fall into the same hash bucket is taken as the standard to indicate whether the tags of the hash bucket that should belong to are the same, and different transactions are put into a hash bucket. Such a hash bucket is called a cluster. As described in the previous chapters, transactions in the same cluster should use pessimistic concurrency control algorithm, while transactions in different clusters should use optimistic concurrency control algorithm.

For transactions in the same cluster, a lock contention process should be carried out just like the traditional pessimistic concurrency control algorithm before running. In order to truly realize this lock contention process, a global lock table should be maintained at the system level of the database. The lock table contains the data record of the currently held lock, the holder's ID, the lock mode and the minimum hash key of the holder's ID.

The following details the process of obtaining locks by transactions.

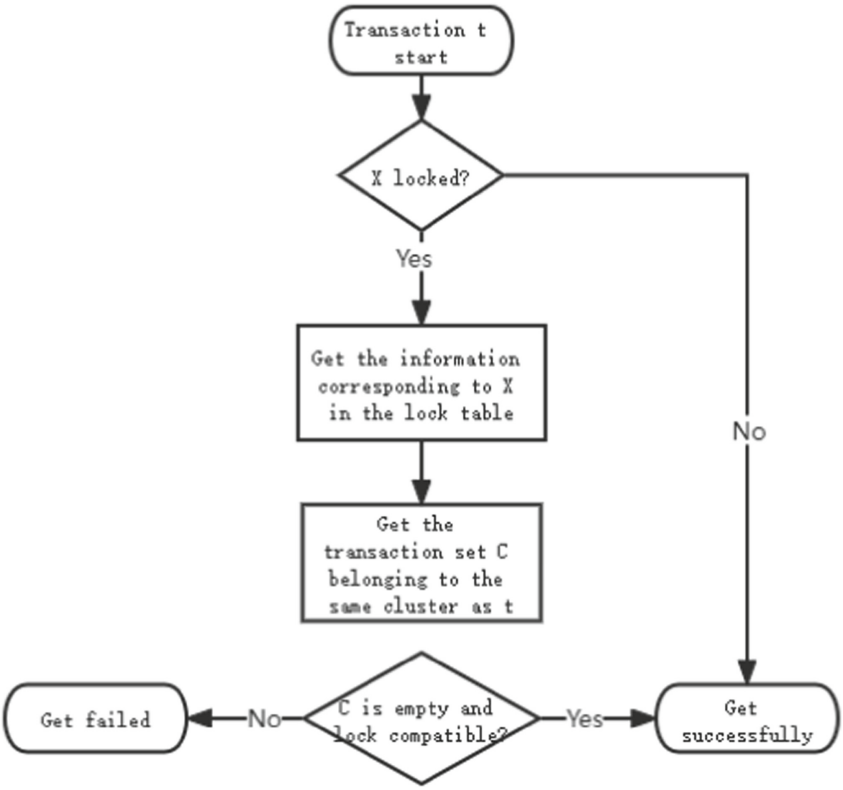

Fig. 3. The whole process of lock acquisition in cluster based concurrency control algorithm

As shown in Fig. 3, when a transaction t starts to operate, it first needs to determine whether the data object it needs to access is locked. If the data object is not locked, the transaction can directly obtain the lock corresponding to the data object; If the data object is locked, it is necessary to obtain the corresponding information of the data object in the lock table, put the transactions that hold the lock and belong to the same cluster as transaction t into the same set C [9], and judge whether the set is empty. If it is empty, and the lock mode that transaction t wants to acquire is compatible with the lock mode held by the already held transaction, transaction t can successfully acquire the lock. Otherwise, transaction t fails to acquire the lock [10–12].

The above is an implementation of the pessimistic concurrency control method in the cluster based concurrency control algorithm, which is slightly different from the traditional pessimistic concurrency control algorithm, and the optimistic concurrency control method in the algorithm is not much different from the traditional optimistic concurrency control. Therefore, any optimistic concurrency control algorithm can be applied to the algorithm proposed in this paper. Thus, the concurrency control algorithm based on clustering can be realized.

4.2 Experimental Evaluation

Next, we will evaluate the performance of clustering based concurrency control algorithm through experiments, and verify the superiority of the algorithm by comparing the experimental results with the performance of pessimistic concurrency control algorithm and optimistic concurrency control algorithm. This experiment simply defines four transactions, which run concurrently under the single pessimistic concurrency control algorithm, the single optimistic concurrency control algorithm and the cluster based concurrency control algorithm respectively. The performance of the three concurrency control algorithms is measured by recording and comparing the running time of transactions.

The experiment first needs to write a total of 100 key value pairs from item 0 to 99 in the database, and then define four different transactions a, B, C and D. among them, transaction a needs to modify the values from item 0 to 99, transaction B also needs to modify the 100 values, transaction C needs to modify the value values from item 0 to item 50, and transaction D needs to modify the value values from item 51 to item 99. From the clustering standard proposed in this paper, it can be seen that transaction a and transaction B have a high conflict rate, so pessimistic concurrency control algorithm should be adopted, while transaction C and transaction D have a low conflict rate, so optimistic concurrency control algorithm should be adopted. By recording the running time of transactions under three different concurrency control algorithms, the results shown in the figure below are obtained.

Through the Table 1, it is obvious that the running time of the four transactions under the cluster based concurrency control algorithm is significantly less than that using the traditional pessimistic and optimistic concurrency control algorithm alone. Therefore, the hybrid algorithm proposed in this paper does have advantages.

Table 1. Transaction time cost.

Transaction name	Befor optimize (seconds)	After optimize (seconds)
A	0.13	0.07
B	0.13	0.11
C	0.09	0.11
D	0.09	0.09

5 Conclusion

The clustering based transaction concurrency control method proposed in this paper is mainly inspired by the local sensitive hash algorithm. There are many transaction clustering methods, but using the local sensitive hash algorithm to group according to the similarity between transactions can skillfully equate the similarity to the conflict rate between transactions. Transactions with high conflict rate will be divided into the

same group, and transactions with low conflict rate will be divided into different groups, In this way, we can conveniently and effectively select the most suitable concurrency control algorithm for different transactions.

This algorithm can be regarded as a pragmatic concurrency control algorithm, which can judge the conflict possibility between transactions according to whether the transactions are divided into the same group. If the transactions are in the same group, it indicates that the conflict possibility between transactions is high, so pessimistic concurrency control algorithm should be used for control; If the transactions are not in the same group, it indicates that the possibility of conflict between transactions is small, and the optimistic concurrency control algorithm should be used for control. Through such grouping, the advantages of the traditional pessimistic concurrency control algorithm and the optimistic concurrency control algorithm can be effectively brought into play, and the disadvantages of the two algorithms can be effectively avoided, and the effect of developing the advantages and avoiding the disadvantages between different isolation mechanisms can be achieved. Therefore, in the high workload conflict environment, the performance of this hybrid algorithm is much better than the traditional pessimistic concurrency control algorithm and the optimistic concurrency control algorithm. Experiments show that this algorithm is better than the traditional pessimistic and optimistic concurrency control algorithm in high load working environment. Even in low load working environment, this algorithm does not cause additional performance loss, and its performance is basically similar to that of the traditional algorithm.

Supported by the Fund: the “Cyberspace Security” construction project of key disciplines in Jiangsu Province during the “14th Five-Year Plan”.

Philosophy and Social Foundation of the Jiangsu Higher Education Institutions of China under Grant No. 2019SJA0443.

References

1. Wu, W., Li, B., Chen, L., Gao, J., Zhang, C.: A review for weighted minhash algorithms. IEEE Trans. Knowl. Data Eng. **34**(6), 2553–2573 (2020)
2. Sun, L.: An improved apriori algorithm based on support weight matrix for data mining in transaction database. J. Ambient. Intell. Humaniz. Comput. **11**(2), 495–501 (2019). https://doi.org/10.1007/s12652-019-01222-4
3. Msb, P., Cv, G. R., Vangipuram, R., Cheruvu, A.: Similarity association pattern mining in transaction databases. In International Conference on Data Science, E-learning and Information Systems 2021, pp. 180–184. ACM, Petra (2021)
4. Stit, O., Riffi, J., Yahyaouy, A., Tairi, H.: Comparative study of different association rule methods. In: 5th International Congress on Information Science and Technology (CiSt), pp. 323–327. IEEE, Marrakesh (2018)
5. Priya, N., Punithavathy, E.: A review on database and transaction models in different cloud application architectures. In: Proceedings of Second International Conference on Sustainable Expert Systems, pp. 809–822. Springer, Singapore (2022). https://doi.org/10.1007/978-981-16-7657-4_65
6. Yan, X., et al.: Carousel: low-latency transaction processing for globally-distributed data. In: Proceedings of the 2018 International Conference on Management of Data, pp. 231–243. ACM, Houston (2018)

7. Hu, H., Zhou, X., Zhu, T., Qian, W., Zhou, A.: In-memory transaction processing: efficiency and scalability considerations. Knowl. Inf. Syst. **61**(3), 1209–1240 (2019). https://doi.org/10.1007/s10115-019-01340-7
8. Zhang, C., Li, Y., Zhang, R., Qian, W., Zhou, A.: Benchmarking for transaction processing database systems in big data era. In: Zheng, C., Zhan, J. (eds.) Bench 2018. LNCS, vol. 11459, pp. 147–158. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32813-9_13
9. Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-26253-2
10. Singh, A.A., Nawab, F.: WedgeDB: transaction processing for edge databases. In: Proceedings of the ACM Symposium on Cloud Computing, p. 482. ACM, California (2019)
11. Sadoghi, M., Blanas, S.: Transaction concepts. In: Transaction Processing on Modern Hardware. Springer, Cham (2019). https://doi.org/10.1007/978-3-031-01870-1_2
12. Redis Homepage. https://redis.io/. Accessed 1 July 2022