Research and Implementation of Cache Optimized Query Privacy Protection for Spatio-Temporal Data Based on PGSRQ-PIR

Lin Qian\textsuperscript{1, a}, Jun Yu\textsuperscript{1, b}, Guangxin Zhu\textsuperscript{1, c}, Mingjie Xu\textsuperscript{1, d}, Hengmao Pang\textsuperscript{1, e}, Feng Mei\textsuperscript{2, f}, Wenda Lu\textsuperscript{2, g}, Lin Wang\textsuperscript{1, h}, Haiyang Chen\textsuperscript{1, i}, and Zhu Mei\textsuperscript{1, j}

\textsuperscript{1}State Grid Electric Power Research Institute, Nanjing, China
\textsuperscript{2}State Grid Zhejiang Electric Power Company Information & Telecommunication Branch, Hangzhou, China

Abstract. In view of the high redundancy of the nearest node of the Spark based user privacy protection query algorithm CRIP-V, the caching optimization algorithm PCPIR-V-CA is proposed. PCPIR-V has two parallel strategies, row strategy and bit strategy. The bit strategy divides the row into smaller pieces and improves the efficiency while the grid division is to small. The cache optimized algorithm of PCPIR-V first clusters the CPIR data transforms the data into tuples and caches them in each cluster. Finally it calculates the result based on the cached tuples and data. It has about 20 percents improvement than PGSRQ-PIR.

1. Introduction

In recent years, cloud computing technology and location-based service technology have been greatly developed. Cloud computing services can provide basic data processing protection for mobile data privacy protection processing [1] in big data environment. The PGSRQ-PIR algorithm proposed in the document "Query Privacy Protection for Spatio-temporal Data Based on Spark" is a group-wide query algorithm based on Spark and CPIR [2] parallel privacy retrieval [3] (Parallel grouping space range query algorithm based on private information retrieval, PGSRQ-PIR).This method realizes the optimization of the simple range query algorithm by dividing the spatial grid [4] in parallel computing. Since the space object spatial distribution of CPIR-V is adjacent [5], when the amount of calculation is too large, there will be many redundant calculations in the calculation of CPIR [6], so the preprocessing of data is very important and needs The strategy of streamlining and caching data [7] and other methods to improve the efficiency of data calculation. Because the PGSRQ-PIR algorithm is a method of CPIR based on Spark's parallel design, the same problem exists. Although the performance is solved by passing and exercising the problem, there is still such redundant calculation, and there is a waste of computing resources. Therefore, it is necessary to pre-process the calculated data and optimize the
Spark-based features to reduce redundant calculations. The main calculation method of CPIR is shown in Figure 1.

![Figure 1. The schematic diagram of CPIR](image)

Figure 1 shows the basic calculation method of CPIR. The calculation process is $Z_1 = y_1 \times y_2 \times y_3 \times y_4 \times y_5$, $Z_2 = y_2 \times y_3 \times y_6$, $Z_3 = y_2 \times y_6$, $Z_4 = y_1 \times y_2 \times y_3 \times y_4$, $Z_5 = y_2 \times y_3 \times y_6$, $Z_6 = y_2 \times y_6$. It can be seen that $y_2 \times y_3$ is calculated 4 times, $y_2 \times y_6$ is calculated 4 times, $y_1 \times y_2 \times y_3 \times y_4$ is calculated 2 times, $y_2 \times y_3 \times y_6$ is calculated 2 times, so there are many redundant calculations. If the calculation is repeated every time, because of the multiplication of large integers, the computing resources of the cluster will be consumed sharply. Therefore, the data needs to be processed and optimized by the cache to solve the problem of redundant computing.

2. Algorithm basic ideas

In order to solve the problem of computational redundancy proposed in the first section, this section proposes a cache optimization method based on clustering and binary intersection with reference to distributed Impact factor propagation behavior [8], and through the characteristics of Spark cluster nodes will cache distributed storage [9], improve the speed of computing.

This approach divides the privacy query into two phases: Data preprocessing stage, online cache query stage. The data preprocessing phase is divided into two small steps: It is generated by CPIR data clustering and buffering binary group [10] respectively. The specific process is as follows:

First, the binary array is used as the clustering entry. For example, the data in the first row in Figure 1 is a sequence of 111110. All the binary arrays in the database are clustered by the k means clustering algorithm. For this binary 0 or 1 The data uses an optimized distance calculation method, and the calculation method of the cluster center, from the case where it can be applied to the case where the values of the bits are mostly the same, in order to apply the clustering with the binary 0 or 1 form, The section calculates the calculation method of a new distance and cluster center. The new calculation method is shown in Equations 1, 2. Suppose that the class is divided into k classes by formula, where $a$ and $b$ represent binary arrays of any two rows of data in the data matrix. Round is a rounding function, $e$ is the binary value in a column in a class, and $i$ represents the number of columns. $n_{cluster}$ is the number of elements of this class, $c_i$ represents the average of each column, then $(c_{i_1}, c_{i_2}, \ldots, c_{i_{GV}})$ is the center of this class, where $GV$ is the number of columns of the CPIR matrix. Taking $a = 111110$, $b = 111100$ as an example, Equation 1 can be calculated as follows: first calculate $a \wedge b = 111100$, then $\text{count}(a \wedge b) = 4$, then $d_{a,b} = \frac{1}{4}$. If $a, b$ are the same class, and in this class Only $a, b$ two elements, then $c_i$ can be calculated as follows, $c_i = \text{round}(\frac{1+1}{2}) = 1$, other $c$ calculations are similar.
First, through the k-means clustering algorithm, and using the distance formula of 1, 2 and the cluster center calculation formula, all the data of the database are clustered into k classes, and the value of k is the same as the number of cluster nodes. This ensures that highly similar binary sequences are in a class and that each class is tagged.

Secondly, all the data in each class are judged by each other, and the items satisfying formula 3 are added to the map to be cached in this class, id is id, $\theta$ is a threshold value, indicating that the minimum number of digits where both $a$ and $b$ are 1 is $\theta$, where the value rule of $\theta$ is: The time to multiply the $\theta$ large integers is greater than the minimum value of the memory lookup time condition. If the following conditions are met, the part where $a \land b$ is 1 in $a$ and $b$ is set to 0 and changed to $a', b'$ respectively, and the two records are saved as a binary group. For example, $(a', i_d_a), (b', i_d_b)$ where $a', b'$ is called a shared difference, $i_d_a, i_d_b$ are called shared bases, and if each record produces multiple sets, then select the smallest $\text{count}(a')$ for preservation.

$$\text{count}(a \land b) > \theta$$  \hspace{1cm} (3)

Finally, Spark divides different categories into different nodes to run, and each kind of stay cached data priority calculation, and then calculates the binary group, wherein the values that $i_d_a, i_d_b$ needs to calculate can be from the local cache of the node. Obtained, thus achieving the sharing of calculations, and the operations of each node are executed in parallel, which greatly reduces the computational cost and improves the efficiency of the calculation.

Figure 2. The architecture of the optimized algorithm of PCPIR-V

Figure 2 shows the basic process of the optimization algorithm. First, cluster the data, the distance formula is 1, the calculation formula using the cluster center is 2. All data is aggregated into two categories, C1 and C2, and then the two types are respectively distributed to the same node and converted into two groups respectively. Each record in the class is judged against each other. If the condition of Equation 3 is satisfied, a new binary group is generated. The first element of the binary group is the
shared difference, and the second position is the shared base, and the shared base. For the two records to take the code after the combination, 2 in the figure represents the code of 111100, which represents the position of the two C1 classes with the repetition of 1 being the first four digits. If there is more than one repetition of one record, the maximum is taken. The common difference is the binary data obtained by sharing the position 0 of the base 1. In the figure, the first row of the C1 class data is 111110, and the position where 111100 is 1 is set to 000010 after 0. Finally, the shared base of the two groups is calculated and put into the cache of the node, and then the shared difference of the two groups is calculated. There is no need to perform large integer multiplication calculation, and the shared base id position has been stored in the cache, so the calculation can not only be allocated to each node but also can improve the calculation efficiency and effectively reduce the repeated calculation.

3. ALGORITHM implementation

This section mainly optimizes the CPIR algorithm for the problem of double counting based on the cache, which is mainly divided into three steps, the clustering of all the data is divided into k classes, and then calculate the intersection of all the records in each class to form a two-group. The specific method has been introduced in detail in the second section. Finally, each type of data is distributed to each node of Spark, and the calculation of the cached data is first performed at each node, and then the calculation of other data is performed. The cached ones are directly read from the cache and are not calculated. The two steps are the preprocessing steps, and the third step is the query algorithm. The first step clustering algorithm is as follows.

| Algorithm 1 The binary k-means clustering algorithm |
|-----------------------------------------------------|
| Output: Category label                              |
| Algorithm Description:                               |
| 1: Randomly select k data as the initial clustering center; |
| 2: Calculate the distance from each cluster center using Equation 3.7 for each piece of data and incorporate that piece of data into the nearest center; |
| 3: Recalculate the cluster center for each class according to Equation 3.8; |
| 4: End if the cluster center is not changing, otherwise skip to 2; |
| 5: End.                                              |

The first line of the algorithm first randomly selects k data from the data set as the initial cluster center, where the value of k is the number of executor nodes in the Spark cluster. Since each class has its own cached data, the reason for this is it is possible to group more similar records together and the cache hit rate will be higher. The second line of the algorithm calculates the distance between all the data and various cluster centers. The distance formula uses formula 1 and assigns each record to the nearest center, which is the center with the smallest value of formula 2. Line 3 of the algorithm calculates the cluster center separately for the categories that have been clustered, using Equation 3. Finally, the convergence condition is judged. The convergence condition is that the cluster center for re-clustering has not changed from the previous center. The second step algorithm is shown in Algorithm 2.
Algorithm 2 The data reducing and data generating algorithm

Input: One of the types of data, the number of data n, the threshold θ
Output: Simplify the tuple of binary groups of data and data to be cached

Algorithm Description:
1: Declare temporary array variables temp, element
2: for(int i=0;i<n;i++)
3:   for(int j=0;j<n;j++)
4:     if(i!=j)
5:       temp[j]=data[i]^data[j]
6:       if(count(temp[j])>θ)
7:         element[j]= reverse(temp[j])^data[i])
8:     end if
9:   end for
10: end for
11: Select count minimum element[min] from element, tuple[i] = (element[min], encode(temp[j]))
12: end for
13: End

The 1-2 lines of the algorithm are mutual judgments on the records in the class, wherein the fourth line is to exclude the comparison with itself, the fifth line is calculates the result of a repetition of 1 between the two records, and the sixth line is judged according to formula 1., record only when satisfied. After the first layer of the 11th row is completed, an element with the most repetition of 1 is selected in the element to be added to the binary group, and the second element of the binary group is the code of the repetition of 1 part. This algorithm corresponds to the second step in Figure 2. The data in the second row of the figure is 011001, which is the same as 010001 in the third row, the 011001 in the fifth row, and the 010001 in the six rows belong to the C2 class. After the data with 3, 5, 6 lines is taken, the result is 010001, 011001, 010001, the sharing difference is 001000, 000000, 001000, the minimum sharing difference is 000000, the shared base is 011001, the number is 25. The preprocessed data can be queried through the Spark cluster.

Algorithm 3 The PCPIR-V cache algorithm at the server

Input: Query q(y1 y2...yn), the number of meshes g_x, g_y
Output: CPIR calculation result Array(Z)

Algorithm Description:
1: Caching binary information to RDD;
2: Aggregate the binary groups in the RDD to obtain the form of (id, List(value));
3: By query q, CPIR calculation is performed on the data corresponding to id in (id, List(value)) to cache to RDD;
4: Calculate the CPIR result of List(value) using cached data and q and cache it to RDD;
5: Aggregate data is sent back to the client;
6: End.

The first line of the algorithm first reads the binary data generated in the first two steps into the RDD of Spark. Line 2 aggregates the two groups according to the cache id (groupByKey) to generate a new binary group (id, seq (element)), so that the id is guaranteed to be unique in the RDD. The third line is for the id. The code is understood and the result is calculated by CPIR with the query q. The fourth line of the algorithm multiplies the remaining elements to obtain a sequence of (seq (Z)). This algorithm shows the process of parallel computing in Spark in Figure 2. First, the shared base cache is calculated, and then the whole data is calculated. Finally, the data is merged and the result is returned to the client.
4. Performance analysis and experimental results

The PCPIR-V cache optimization algorithm basically solves the problem of excessive redundancy calculation in PCPIR-V due to spatial distribution characteristics. Through three steps, the first step is to cluster the data in order to gather similar data together, which is convenient for the second step. Only the intra-class comparison can find the record with the highest similarity. Finally, through Spark the cached data is prioritized and the cached data is reused.

The main performance optimization of the optimization algorithm is that the calculation of large integer multiplication in each record can be cached, thereby improving the efficiency of calculation. The cost model of the cache optimization algorithm can be expressed by formula 4.

\[
C = C_{\text{total}} - C_{\text{redundant}} + C_{\text{cache}}
\]

Where \( C \) represents the total cost of the cache optimization algorithm, \( C_{\text{total}} \) is the computational cost when there is no cache, \( C_{\text{redundant}} \) is the computational cost of the duplicate record, and \( C_{\text{cache}} \) is the cost of the cache read. The calculation times of \( C_{\text{cache}} \) and CB are consistent. However, the time of reading the cache is much smaller than the time of multiplication of large integers. Therefore, it can optimize the redundancy calculation, and the cache optimization effect is the relationship between data and the optimal solution of threshold \( \theta \) will be studied and discussed in future work. And through the Spark distributed features, this cache calculation can be paralleled, which further improves the computational efficiency.

This experiment mainly optimizes the algorithm of the server, reduces the redundant calculation of data, and realizes the sharing of calculations. Therefore, only the experimental analysis of the algorithm of the server is made, and the experiments are carried out for different data sets and different meshes.

Figure 3, Figure 4, and Figure 5 show the comparison of the time between the cache-optimized PCPIR-V and PCPIR-V under different meshing under the uniformly distributed data set, the Gaussian distribution data set and the real data set, respectively, where PCPIR-V-CA is a cache-optimized PCPIR-V algorithm.

![Figure 3. Server time of different grid divisions of uniform distributed data](image-url)
From the comparison of the three experimental results, the real data set and the Gauss data set are much higher than the average data set. This is due to the concentrated distribution of Gaussian data and real data. The more concentrated the data, the more duplicate calculations there may be, so the better the effect of cache optimization. Both of these algorithms will decrease after a large peak at the beginning of the grid, because at the beginning of the grid division, there are more potential nearest neighbors in the grid, so the data can not be very It is distributed to each node for distributed execution, which results in slow calculation speed and poor optimization of the cache. It is also because the main bottleneck at the beginning is not to calculate the large integer but to determine whether the current bit is 1, so the cache optimization. It is only after the number of grids is 100, which has started to increase significantly, especially the Gaussian distribution of data and real data, mainly because they have more repeated calculations.

5. Conclusion
This paper proposes a PGSRQ-PIR-based cache optimization privacy protection query algorithm. This algorithm has been greatly improved compared with the previous algorithm, but there are still some shortcomings. The Bit based parallel strategy of PCPIR-V has improved the performance problem in the case of too little grid division, but the problem still exists, and the division of Spark partition needs to be further optimized to achieve more balanced load. Moreover, the threshold of PCPIR-V has not been studied on the optimal solution. If the optimal value can be found, the cache optimization performance may be further improved. And, this paper does not study the compression optimization scheme of the binary matrix in the CPIR algorithm. Because there are more 0 bits in the CPIR binary matrix, this compression may reduce the number of comparisons and improve the computational efficiency.
Acknowledgments
This work was financially supported by the science and technology projects of headquarters of State Grid Corporation (Research on key technologies of secure network coding and data protection for secure renewable storage System).

References
[1] Wei W, Xu F, Li Q. Mobishare: Flexible privacy-preserving location sharing in mobile online social networks [C] // INFOCOM 2012 Proceedings IEEE. IEEE, 2012: 2616-2620.
[2] Settimi V. A Study of Computational Private Information Retrieval Schemes and Oblivious Transfer [J]. 2007.
[3] Zheng Huo, Xiaofeng Meng, Jianliang Xu. Privacy-Preserving Query Processing in Cloud Computing [J]. Journal of Frontiers of Computer Science and Technology, 2012, 6 (5): 385-396.
[4] Balaji Palanisamy, Ling Liu, Kisung Lee, Shicong Meng, Yuzhe Tang, Yang Zhou. Anonymizing continuous queries with delay-tolerant mix-zones over road networks [J]. Distributed and Parallel Databases. 2014 (1)
[5] Pan X, Xu J, Meng X. Protecting location privacy against location-dependent attacks in mobile services [J]. Knowledge and Data Engineering, IEEE Transactions on, 2012, 24 (8): 1506-1519.
[6] Jeffrey D, Sanjay G. MapReduce: simplified data processing on large clusters [J]. Communications of the Acm, 2004, 51 (1): 107-113.
[7] Shvachko K, Kuang H, Radia S, et al. The hadoop distributed file system [C] // Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on. IEEE, 2010: 1 - 10.
[8] Xie D, Li F, Yao B, et al. Simba:Efficient In-Memory Spatial Analytics. ACM SIGMOD International Conference on Management of Data. 2016
[9] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing [J]. In-Memory Cluster Computing. USENIX Symposium on Networked Systems Design and Implementation (NSDI, 2012, 70 (2): 141-146.
[10] Taylor R C. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. [J]. Bmc Bioinformatics, 2010, 11 suppl 12 (6): 3395-3407.