International Journal for Housing Science and Its Applications
Publish August 3, 2025. Volume 46, Issue 3 Pages 1294-1303

https://doi.org/10.70517/ijhsa463100

Research on Big Data Streaming Storage and Fast Indexing Method Based on APSO Algorithm Enablement

Zhaozhen Zeng[1], Kezhi Zhen[1,*], Lin Feng[1], Guosong Fan[1], Ming Wu[1] and Chaoyun Luo[1]
[1] China Tobacco Guizhou Industrial Co., LTD., Guiyang, Guizhou, 550001, China
Corresponding authors: (e-mail: 17708541846@163.com).

Abstract The article constructs TSM Tree storage model for big data stream, optimizes the fast query method by using APSO algorithm thinking, and constructs fast query model for big data stream by using discrete wavelet transform. The storage and retrieval performance of the big data stream storage model and fast query model are verified respectively.The TSM Tree storage model has high storage efficiency. The storage rate of the TSM Tree storage model increases rapidly when the file size is larger than 20MB, and the TSM Tree storage model is suitable for storing big data files over 20MB. In this paper, the fast indexing method has the highest Put execution efficiency and becomes the secondary indexing method for single condition query. The fast indexing method in this paper is optimized on multi-conditional query with the fastest response time of 0.954, 0.898, and 0.907s.

Index Terms apso algorithm, TSM Tree, data storage model, fast querying

I. Introduction

With the continuous development of information technology, big data streams have become an important information resource in today's society [1], [2]. In order to effectively manage and utilize big data streams, various storage methods have emerged [3]. The common big data stream storage methods are distributed file system, columnar storage database, NoSQL database, in-memory database and distributed storage system [4], [5]. The choice of big data stream storage methods should be evaluated based on actual requirements and scenarios [6]. Distributed file systems are suitable for scenarios requiring high fault tolerance and scalability, columnar storage databases are suitable for scenarios requiring high-speed querying and analyzing large amounts of data, NoSQL databases are suitable for scenarios requiring high concurrent read/write and storing semi-structured data, in-memory databases are suitable for scenarios requiring real-time processing and analyzing large-scale data, and distributed storage systems are suitable for high reliability and high availability and distributed storage systems are suitable for scenarios requiring high reliability and high availability [7]-[10]. Choosing the appropriate storage method for big data streams according to specific needs can improve the efficiency of data processing and analysis, and thus realize better business value [11], [12].

In database systems, the main role of indexing is to make the search or reading easier and faster, and to find the needed part quickly and accurately in a large pile of information [13], [14]. In general, indexes require a lot of time and space to build, but they can improve the efficiency when reading [15]. Different index structures and query methods are suitable for different scenarios. Common types of indexes include B-tree indexes, hash indexes, bitmap indexes, full-text indexes, etc [16]-[18]. Indexing methods, on the other hand, are specific algorithms and strategies implemented in a specific database system according to the indexing type [19], [20]. Common indexing methods include sequential scanning, binary lookup, hash lookup, and bitwise mapping [21], [22].

The article adopts methods such as multi-computer distributed storage and compressed storage for data storage for large data streams, and proposes an improved TSM Tree storage model based on the LSM Tree. With the optimization thinking of Adaptive Particle Swarm Optimization (APSO) algorithm based on adaptive inertia weights, the data stream detection method is optimized, and the discrete wavelet transform summary and indexed distributed data stream are introduced to construct the fast query model of data stream. The storage performance of the data stream storage model constructed in this paper is analyzed to explore the storage efficiency of TSM Tree storage model. Finally, the insertion performance, single-conditional query performance and multi-conditional query performance of the proposed data stream fast query model are verified.

II. Large data stream storage and fast indexing methods

II. A. Large data stream storage

II. A. 1) Data storage

Due to the massive nature of the data, if it is stored directly, then hundreds of millions of data will require a large disk storage space, which is not only costly, but also very inconvenient to manage, so this paper uses the appropriate compression algorithms as well as adopts the multi-machine distributed storage, this subsection will be specifically introduced in this paper to design the storage method.

(1) Multi-computer distributed storage

In this paper, multi-machine distributed storage adopts a decentralized design as shown in Figure 1, all nodes store data individually, communicate with each other through the intermediate cache to achieve, due to the timed collection of data, so every day and every month to produce the data is quite, so the realization of load balancing is to ensure that each machine to store the data for the same number of days can be. In this paper, the first preprocessing, the data where the server information is stored in the table, so that in the index, only need to find the data can be found to the corresponding server, and then the use of each storage node storage index for the second lookup.

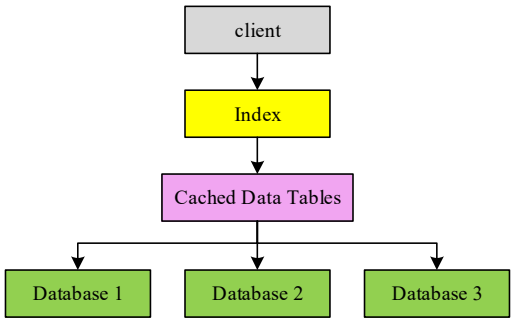


Figure 1: Overall storage design

(2) Compressed storage

The data contains time as well as attribute information, so the data types are mainly time type, integer, floating point, Boolean and string. In order to solve the problem of large amount of time series data, this paper adopts different compression algorithms for different data types, and then the compressed data is merged into a byte stream for storage, the specific process is shown in Figure 2.

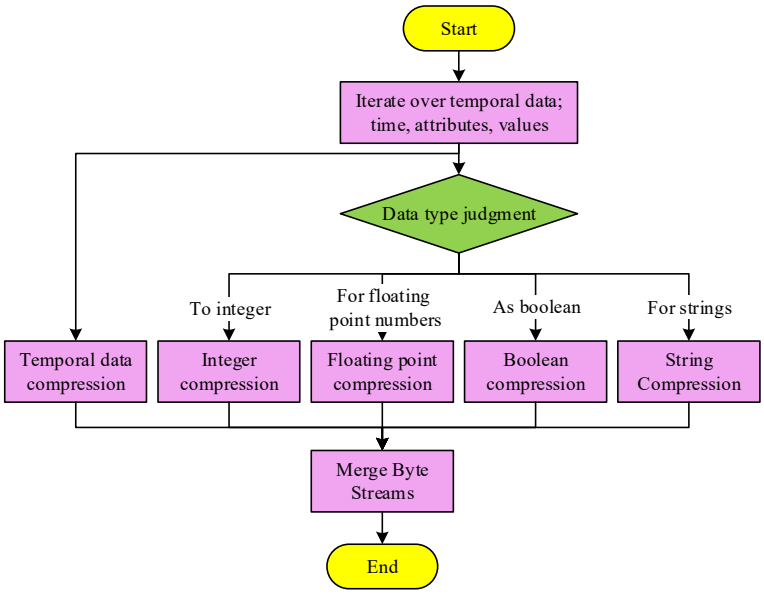


Figure 2: Data compression process

Compression of integer data: When storing, if the data is the first integer data stored, then it is stored directly without compression. If it is not the first data, then calculate the difference between the data and the previous integer data, and then Zig-Zag encode the difference result, so that the difference can be mapped to a positive integer, and then data compression is carried out in different cases: (1) due to the continuity of time data, so the difference will appear a lot of repetitions of the data, and if the difference occurs to be the same, in order to reduce the storage space, it is only necessary to store the difference value and the number of times the difference appears, and then the difference appears again on the difference storage can be added one. (2) If the difference exceeds a certain size, then it will be stored directly and will not be compressed. (3) If the above two cases are not satisfied, then the encoded data will be stored directly with simple8b algorithm.

Time type data compression: in the time series data, each data have time, the time data compression method is similar to the integer type, the first to determine the time data is not stored in the first time type data, if so, then do not compress the direct storage, if not, then the previous time for the difference. Difference and then compressed, determine the difference is the first difference stored, if so, then do not compress, otherwise calculate the difference with the previous one. If the difference is zero, which means that the data is stored at the same interval, then only the zero value and the number of times it occurs are stored. Otherwise the simple8b algorithm is used to store the new compression.

Boolean type compression is relatively simple, Boolean type generally accounts for a byte, compressed storage can be Boolean data with a bit of storage, a byte of 8 bits can be stored 8 Boolean values. Strings are sequentially added to the byte stream after compression with the snappy algorithm.

Floating-point data compression: float type for 8 bytes, the main idea is that the byte stream in accordance with a certain format for continuous appending. First of all, similar to the integer data compression method, the first float is not compressed, the difference is that the second after the data need to be different from the previous float operation delta, delta is a 64-bit unsigned integer. If delta is 0, it means that the difference is 0, that is, the floating-point number and the last data duplication, only need to store a 0 can be. If it is not 0, first store a 1, then use 5 bits to store the number of leading 0s, 6 bits to store the number of trailing 0s, and finally write the valid bits.

II. A. 2) Storage model

(1) LSM Tree

The traditional b-tree and hash storage structure although the query speed is very fast, but the hash storage does not support traversal operations, while the b-tree can not meet the requirements of fast data writing, so in this paper, we study the LSM tree to improve the write performance.

LSM tree is a log-structured merge tree [23], the main idea is to change every time to write the disk to a one-time batch write to reduce the number of random seeks on the disk. Each time the data is updated, the idea of caching is utilized to store the latest data temporarily in memory, and then the cached data is directly merged into the disk data when it accumulates to the pre-set threshold of the cache.The specific structure of the LSM tree is shown in Fig. 3.

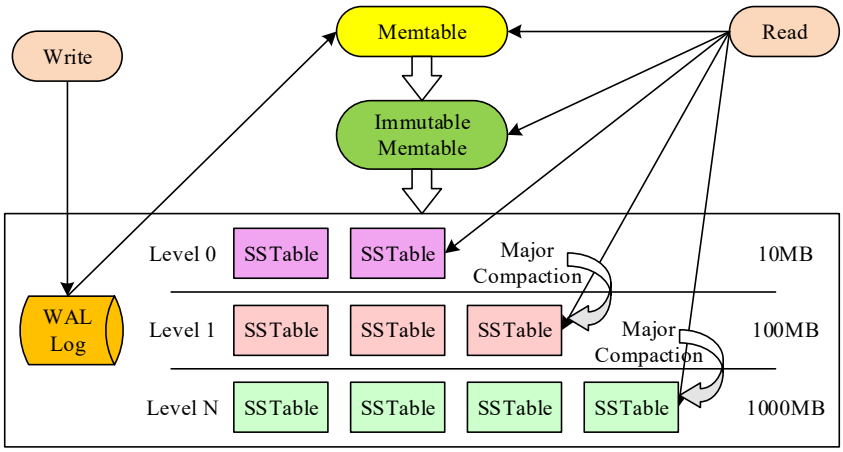


Figure 3: LSM Tree internal structure

(2) TSM Tree

TSM tree inherits the advantages of LSM tree and also has good insertion efficiency for data. The difference is that the LSM tree stores the data directly, resulting in the need for a large storage space, while TSM can use

compressed data.TSM greatly improves the speed of reading data, because when looking for data, generally more interested in the most recent data, so most of the time the query does not go to query the TSM File, but the query results are returned directly in the Cache.

II. B.Fast indexing method based on APSO algorithm optimization

II. B. 1) Particle swarm optimization algorithm based on adaptive inertia weights

Particle swarm algorithm is a stochastic search heuristic algorithm based on group intelligence, which is able to efficiently search for problems with a large solution space and find candidate solutions under the premise of knowing less information. In this paper, particle swarm algorithm is introduced to solve the hyperparameter optimization problem of long and short-term memory neural networks. The particle swarm algorithm will quickly locate the particles near the optimal candidate solution in the early iteration period, and in the later iteration period, the particles will carry out finer-grained optimization in the vicinity of the optimal candidate solution in order to ensure the accuracy of the optimization results. However, at the same time, due to the premature convergence problem inherent in the particle swarm algorithm, an adaptive inertia weighted particle swarm algorithm (APSO) is proposed on the basis of the basic particle swarm algorithm for balancing the global and local searching behaviors of the particles [24], in order to improve the ability of the particle swarm algorithm to jump out of the local optimum.

Inertia weight is a very important parameter in particle swarm algorithms, Shi et al. first introduced the concept of inertia weight into particle swarm algorithms, which indicates how much the current state of a particle remembers its historical state, taking values between [0, 1]. A larger inertia weight is beneficial to the global search ability of the particle swarm algorithm, while a smaller inertia weight is beneficial to the local search ability of the particle swarm algorithm [98, 120], which can be adjusted to achieve the purpose of balancing the global and local search behaviors of the particle swarm algorithm. After the introduction of inertia weights, the convergence speed of the particle swarm algorithm is improved, which is analyzed for the following two reasons: (1) In the early iteration period, the inertia weights are large, the global search ability is strong, and the local search ability is weak, at this time, the particles can be coarsely and rapidly localized to the vicinity of the optimal candidate solution. (2) In the late iteration, the inertia weight is small, the global search ability of the particle is weakened, and the local search ability is strengthened, at which time the particle is able to perform a finer-grained search in the range of the optimal candidate solution to optimize to the solution with higher accuracy. This search avoids a lot of fearless repetitive calculations, so the convergence speed is enhanced.

In the particle swarm algorithm, inertia weights are regulated by static constants, random numbers, time-varying and adaptive changes. Linear decreasing inertia weight is a commonly used regulation, compared with the complex calculation of the particle swarm algorithm, linear decreasing inertia weight seems relatively single, which can not control and balance the searching behavior of the particles well. In the late iteration of the particle swarm algorithm, the inertia weights are small, when the particles of $pbest_{i,d}^{t}$ and $gbest_{d}$ are closer, then all the particles will converge to the same point quickly, then $pbest_{i,d}^{t} = gbest_{d} = x_{i,d}^{t}$ the equation is established, at which point the particle's velocity update can be simplified to Eq. (1). Since the inertia weight is a real number between [0, 1], with the increasing number of iterations, according to Eq. (1), it can be seen that the particle velocity will be closer and closer to zero, until finally equal to 0. At this time, the position of the particles will be kept unchanged, i.e., all the particles will converge at a certain point of local optimum until the end of the iteration of the algorithm and exit. To address the above problems, this paper proposes a dynamic adaptive inertia weights, whose update formula is shown in (2). In the pre iteration period of the APSO algorithm, the inertia weights decrease faster, while the decrease slows down in the late iteration period. Since the APSO algorithm in the late iteration of the inertia weight decreases slower, the inertia weight can still be maintained in a larger value interval, which reduces the risk of the particles being in a stationary state, and improves the possibility of the APSO algorithm to get out of the local optimal solution:

$$v_{i,d}^{t+1} = \omega \cdot v_{i,d}^{t} \tag{1}$$

$$\omega_{iter} = \omega_{\max} - (\omega_{\max} - \omega_{\min}) \cdot \tan(\frac{\pi}{4} \cdot \frac{iter}{iter_{\max}}) \tag{2}$$

where $iter$ and $iter_{\max}$ denote the current iteration number and the maximum iteration number, respectively, and $\omega_{\max}$ and $\omega_{\min}$ denote the maximum and minimum inertia weights.

II. B. 2) Data flow fast query modeling

The APSO algorithmic thinking from Section 2.2.1 is introduced into the query indexing domain to optimize fast queries on data streams.

The number of distributed data streams processed is growing rapidly in large-scale applications of the Web, e.g., measurements for network monitoring, stock trading, transactions in retail chain stores, ATM operations in banks, logging for web services, sensor data, etc.

StatStream is currently the latest relevant detection method in data streams. It mainly considers the problem of monitoring a large number of data streams in real time by subdividing the stream's historical data into a certain number of basic windows and maintaining the Discrete Fourier Transform (DFT) coefficients for each basic window [25], which allows the DFT coefficients to be updated in chunks throughout the historical period. The literature also adds an orthogonal regular grid applied to the feature space, which decomposes this space into cells of length ε, the associated threshold. Each stream is mapped onto a certain number of cells of the feature space based on a subset of its DFT coefficients (the number depends on the delay time), and the proximity of this feature space is exploited to report correlations. The application of all of the above techniques suffers from the fact that the linear constants grow exponentially with increasing feature space. Cells in the k-dimensional space have 3^k-1 neighboring cells, resulting in a search space of $(3^k-1)\varepsilon^k$, and if the same grid structure is used to detect thresholds greater than ε (e.g. $2\varepsilon,3\varepsilon,...,n\varepsilon$) correlation, then the cell spacing also needs to be tested. This ratio is further increased for correlation detection when the radius exceeds the threshold $n\varepsilon$ because a cell in the k-dimensional space will have $(2n+1)^k$ neighboring cells, which means that StatStream performs poorly in the high-dimensional and high-threshold cases.

In this paper, we propose a novel scheme to summarize and index distributed data streams, using the Discrete Wavelet Transform (DWT) [26], to reduce the overhead required to maintain the indexing structure by calculating the transform coefficients in real-time online and inserting them into a sequence of high-dimensional indexing structures.

A data stream consists of a sequence of data points $...x_i...$, and the data range of each data point is $[R_{\min}, R_{\max}]$. In this paper, we discuss systems with M data streams and are only interested in the first N most recent data items in each stream. We use K sliding windows of size N to store the data streams, with the most recently arrived N streams data stored in the latest basic window, and so on, with the sliding window constantly updated for each new arrival, and processing mainly two main types of queries on the data streams, i.e., inner-product queries and similar queries.

(1) Inner product query

Inner product queries are very important in statistical computation and conditional specification. An inner product query can be represented by a ternary function (I,V,δ), where I denotes the index vector (the data item of interest), V denotes the weight vector (the weights corresponding to each data item), and the result of the inner product query of I and V $I\square V$ has an accuracy of δ, usually both point and range queries can be expressed as inner product queries.

(2) Similar queries

Supporting similar queries is an important part of most data mining applications [27], identifying companies with similar growth patterns and finding stocks with similar stock price movements are typical types of similar queries on sequential databases. The distance measure used in this section is the Euclidean distance (L_2), which is stored in the orthogonal transform [28]. Since the Euclidean distance between two sequences can range from zero to infinity, only the distance corresponding to the distance between normalized sequences is considered. And mainly two types of similar queries are discussed, namely correlation queries and subsequence queries.

Correlation queries have an important role in correlation trend analysis. A correlation query (all pairwise queries) can be defined by the formula as follows: given a threshold ε, find all pairs of data streams that are correlated within ε. The correlation between two sequences x and y can be reduced to the Euclidean distance between their normalized sequences. We normalize the streams as follows: $\hat{x}_i=(x_i-\mu_x)/\sigma_x, i=1,2,...,N$, where μ_x has a mathematical meaning and σ_x denotes the standard deviation.

A subsequence query is an integral part of an analytic model. A subsequence query can be defined by the formula as follows: given a query sequence Q and a threshold ε, find all subsequences within the given sequence stream whose distance is within ε. We take a stream $x_1,x_2,x_3,...,x_N$ are normalized as follows: $\hat{x}_i = x_i/\sqrt{N} * R_{\max}, i=1,2,...,N$ and map it to the unit sphere.

III. Storage and indexing effect analysis

III. A. Storage performance

III. A. 1) Construction of document classification rules

HDFS file sizes are usually at the GB to TB level, so HDFS is designed to be adapted to support the storage of large files, and when storing data smaller than the HDFS chunk size (128M by default), HDFS will also store it according to the chunk size. Increased storage space also affects access efficiency. The experiment designs the

optimal storage model by analyzing the comparison of the efficiency of HDFS and the TSM Tree storage model of this paper for storing large data streams. The validation program is as follows:

(1) Select the dataset smaller than HDFS chunk size (128M) from the test dataset.

(2) 5 test data sets of 100, 500, 1000, 5000 and 10000 different number sets are selected from the re-selected test set.

(3) Record the storage time under different test data sets.

The results of comparison of HDFS and TSM Tree storage characteristics are shown in Fig. 4. From the results in Fig. 4, it can be seen that TSM Tree is more efficient in storing small files with the same amount of data, and in different orders of magnitude of small files, the efficiency improved by using the TSM Tree storage model over HDFS appears to increase first and then decreases, and the improved efficiency reaches 88.7% at 1000 files. Overall, TSM Tree is more suitable for small file storage than HDFS.

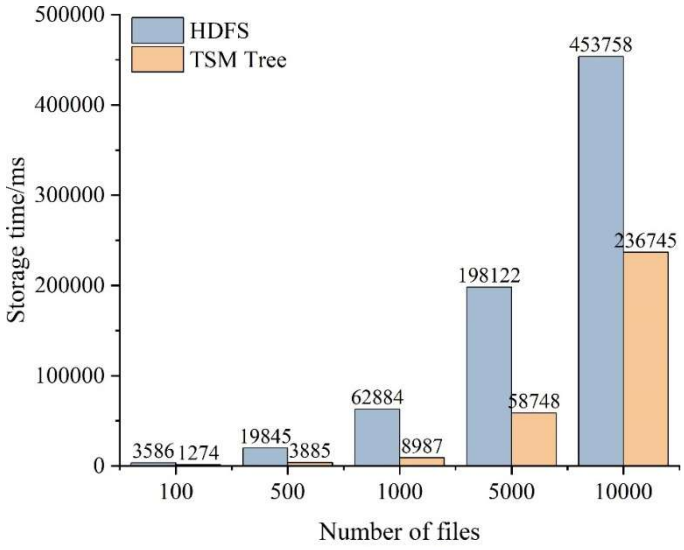


Figure 4: Comparison of Storage Characteristics between HDFS and TSM Tree

The TSM Tree storage model is more efficient for storing small files, but it is not possible to determine the specific size range of small files. In this experiment, the storage efficiency of the TSM Tree storage model is analyzed by setting up data files with different data volume sizes to derive the optimal amount of data for storing small files. The validation program is as follows:

(1) 500 remote sensing data files each of 200KB, 500KB, 1MB, 20MB, 50MB and 100MB are selected from the test set.

(2) Store the files with different data volume sizes in batches in the TSM Tree database.

(3) Record the storage time and calculate the storage rate.

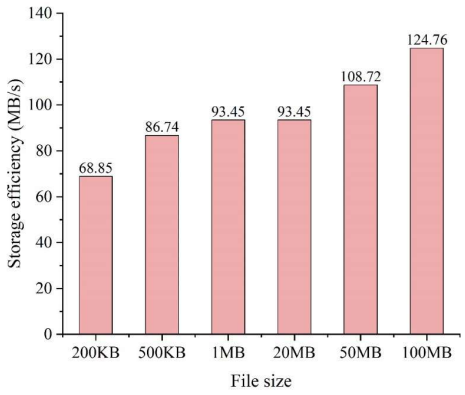


Figure 5: Storage rate of TSM Tree under different data volumes

The storage rate of TSM Tree storage model with different data size is shown in Fig. 5. It can be seen from Fig. 5 that the size of the data volume is gradually increasing. When the file data size reaches 1MB~20MB, the storage efficiency of TSM Tree storage model is relatively smooth (93.45MB/s). When the file size is larger than 20MB, the storage rate of the TSM Tree storage model increases rapidly, so the TSM Tree storage model is suitable for storing large data files with data volume larger than 20MB.

III. A. 2) Model storage efficiency

Data storage query efficiency is an important indicator of the storage system, in the test data randomly selected data sets of different data volume sizes, by verifying the native distributed storage solution in the same hardware configuration environment and the storage solution used in this platform for comparison testing.

(1) Randomly select data sets of about 10GB, 50GB, 100GB, 500GB, and 1TB from the test data.

(2) Store different test datasets in the storage schemes of HDFS, MongoDB and TSM Tree in this paper, respectively.

(3) 10 storage operations are performed on each test set respectively, and the 10 average value is taken as the final consumption time.

The storage performance comparison results are shown in Figure 6. It can be seen that when the number of files is small, the number of large and small files is relatively balanced, and both storage models need to send requests to the main storage node, although the number of requests of this study's scheme is much less than that of the original storage technology scheme, the storage time of the three is similar, and with the gradual increase in the number of files, the advantage of the storage management model is gradually embodied, and with the increasing size of the data, this study's scheme's The storage efficiency gradually increases and remains stable, and the improvement of storage efficiency is stabilized at about 16.7% compared with the original distributed storage method.

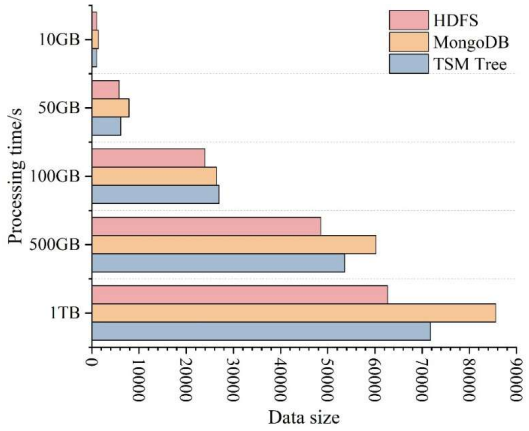


Figure 6: Storage performance comparison

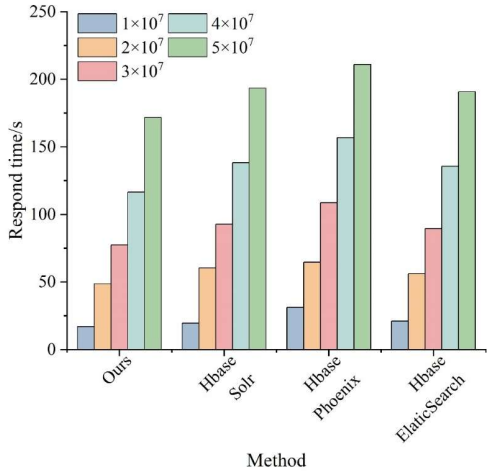


Figure 7: Comparison of response time of inserted data

III. B. Performance Validation of Data Insertion and Retrieval Scheme

III. B. 1) Insertion Performance Comparison

The experiment inserts data into the HBase table through 4 clients at the same time, and counts the Put time of every 10^7 data on 4 clients, after repeating the experiment for 10 times, the average value is taken, and test the Put time of the fast index, ElasticSearch-based index, Solr's index and Phoenix's index in the same conditions in this paper, respectively, and the result is shown in Fig. 7 The results are shown in Fig. 7.

As can be seen from Fig. 7, the highest efficiency of Put execution is the fast indexing method of this paper, and the response times of the indexing scheme of this paper are 16.91, 48.67, 77.32, 116.51, and 171.69 s when the data are 1×10^7, 2×10^7, 3×10^7, 4×10^7, and 5×10^7 entries, respectively. This is because there is no need to reallocate the resources during Put operation for index construction, whereas all other approaches require additional construction of secondary indexes. It can be seen that the insertion time is getting longer and longer when the same 10^7 pieces of data are added, this is because as the amount of data increases, the indexed data is also getting more and more, which makes it difficult to perform index insertion. At the same time, because Phoenix underlying need to build the appropriate storage index structure in the coprocessor, additional consumption of computing resources. And based on ElasticSearch and Solr only need to build indexes in their own clusters, do not need to program additional computational resources, so Phoenix secondary indexes on the insertion of the largest performance loss.

III. B. 2) Comparative analysis of single-condition query performance

Retrieve the data is still using the above data, and its retrieval performance comparison is shown in Figure 8. As can be seen in Figure 8, ElasticSearch response speed is reduced more significantly, when the amount of data reaches 1×10^8, the response time is greater than 5s. For the comparison of the secondary index, when the amount of data reaches 1×10^8 entries, the efficiency of this paper's fast indexing method is 1.75 times that of Solr. Phoenix's retrieval efficiency is close to the speed of this paper's fast indexing method, but the coupling of Phoenix's coupling is stronger, so finally choose the fast indexing method of this paper as the secondary index.

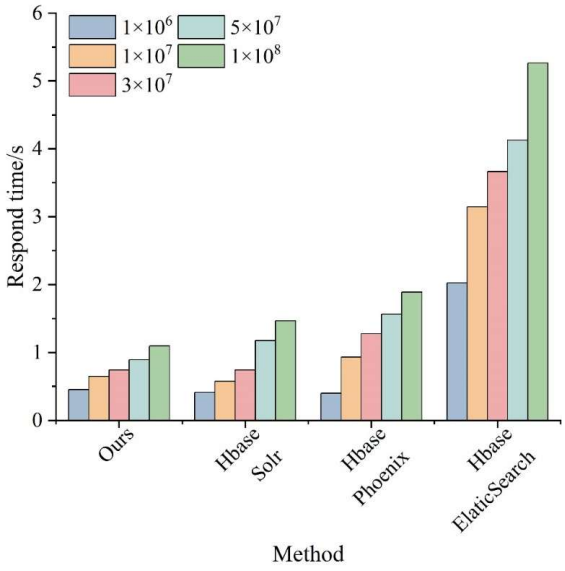
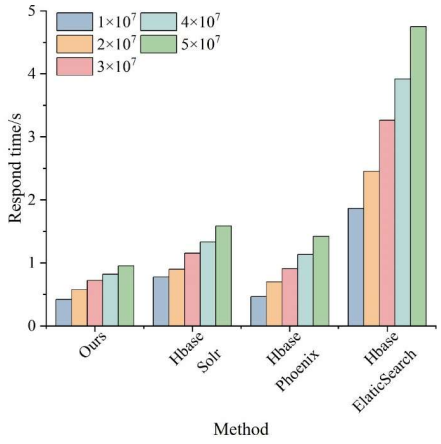


Figure 8: Comparison of retrieval performance

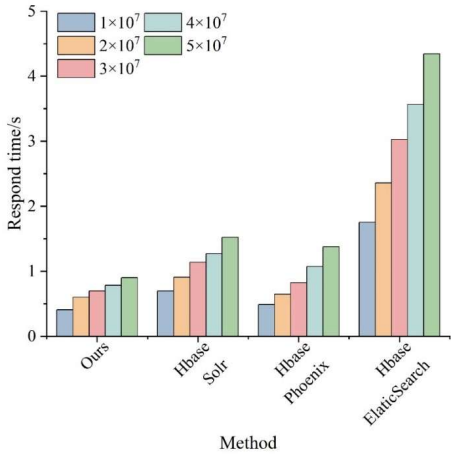
III. B. 3) Multi-conditional queries, performance analysis and their optimization

Retrieval data is still used as above, the number of retrieval conditions were 3, 6, 9, take the average value after 10 experiments, its average retrieval performance comparison shown in Figure 9.

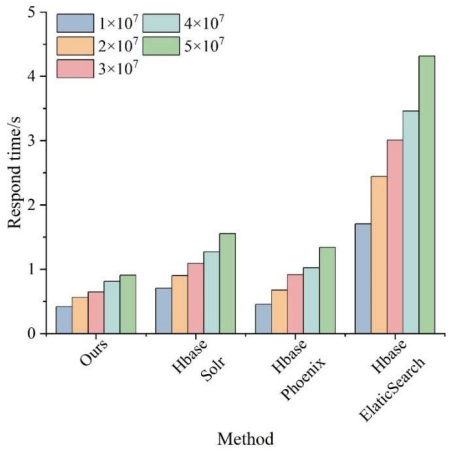
Compare the data horizontally and find that the retrieval time is negatively correlated with the retrieval conditions, and the more retrieval conditions the shorter the retrieval time, for example, when the data is 5×10^7, the number of retrieval conditions is taken to be 3, 6, 9, and the retrieval time of ElasticSearch is 4.752, 4.343, and 4.315s, and that of this paper's fast indexing optimization is 0.954, 0.898, and 0.907 s, which is due to the increase of retrieval conditions in this paper's fast indexing method, the amount of data retrieval that meets the conditions is reduced, which is equivalent to reducing the amount of user concurrency, and therefore, the retrieval time has been reduced.



(a) Retrieval condition is 3



(b) Retrieval condition is 6



(c) Retrieval condition is 9

Figure 9: Comparison of multi-condition retrieval performance

IV. Conclusion

The article adopts multi-computer distributed storage and compressed storage for the storage work of large data streams, and constructs TSM Tree data stream storage model. Using discrete wavelet variation to deal with indexed distributed data streams, APSO algorithm thinking is used for optimization to construct data stream fast query model.

In this paper, the storage efficiency of TSM Tree storage model is higher, and the improved efficiency reaches 88.7% at 1000 files. When the file data size reaches 1MB~20MB, the storage efficiency of TSM Tree storage model is 93.45MB/s. When the file size is larger than 20MB, the storage rate of TSM Tree storage model rises rapidly, and TSM Tree storage model is suitable for storing large data files.

The Put execution efficiency of this paper's fast indexing method is the highest, and the response time of this paper's indexing scheme is 16.91, 48.67, 77.32, 116.51, and 171.69s when the data are 1×10^7, 2×10^7, 3×10^7, 4×10^7, and 5×10^7 entries, respectively.The fast indexing method of this paper is selected as the second level index on single-condition query. On multi-conditional query, the fast indexing optimization of this paper has the fastest response time of 0.954, 0.898, 0.907s.

References

[1] Kolajo, T., Daramola, O., & Adebiyi, A. (2019). Big data stream analysis: a systematic literature review. Journal of Big Data, 6(1), 47.

[2] Mehmood, E., & Anees, T. (2020). Challenges and solutions for processing real-time big data stream: a systematic literature review. IEEE Access, 8, 119123-119143.

[3] Swami, D., Sahoo, S., & Sahoo, B. (2018). Storing and analyzing streaming data: A big data challenge. Big Data Analytics: Tools and Technology for Effective Planning, 229-246.

[4] Siddiqa, A., Karim, A., & Gani, A. (2017). Big data storage technologies: a survey. Frontiers of Information Technology & Electronic Engineering, 18, 1040-1070.
[5] Tantalaki, N., Souravlas, S., & Roumeliotis, M. (2020). A review on big data real-time stream processing and its scheduling techniques. International Journal of Parallel, Emergent and Distributed Systems, 35(5), 571-601.
[6] Alshamrani, S., Waseem, Q., Alharbi, A., Alosaimi, W., Turabieh, H., & Alyami, H. (2020). An efficient approach for storage of big data streams in distributed stream processing systems. International Journal of Advanced Computer Science and Applications, 11(5).
[7] Blomer, J. (2015, April). A survey on distributed file system technology. In Journal of Physics: Conference Series (Vol. 608, No. 1, p. 012039). IOP Publishing.
[8] Zeng, X., Hui, Y., Shen, J., Pavlo, A., McKinney, W., & Zhang, H. (2023). An empirical evaluation of columnar storage formats. Proceedings of the VLDB Endowment, 17(2), 148-161.
[9] Khan, W., Kumar, T., Zhang, C., Raj, K., Roy, A. M., & Luo, B. (2023). SQL and NoSQL database software architecture performance analysis and assessments—a systematic literature review. Big Data and Cognitive Computing, 7(2), 97.
[10] Giannopoulos, I., Singh, A., Le Gallo, M., Jonnalagadda, V. P., Hamdioui, S., & Sebastian, A. (2020). In-memory database query. Advanced Intelligent Systems, 2(12), 2000141.
[11] Baljak, V., Ljubovic, A., Michel, J., Montgomery, M., & Salaway, R. (2018). A scalable realtime analytics pipeline and storage architecture for physiological monitoring big data. Smart Health, 9, 275-286.
[12] Lawal, Z. K., Zakari, R. Y., Shuaibu, M. Z., & Bala, A. (2016). A review: Issues and Challenges in Big Data from Analytic and Storage perspectives. International Journal of Engineering and Computer Science, 5(3), 15947-15961.
[13] Maesaroh, S., Gunawan, H., Lestari, A., Tsaurie, M. S. A., & Fauji, M. (2022). Query optimization in mysql database using index. International Journal of Cyber and IT Service Management, 2(2), 104-110.
[14] Lee, L., Xie, S., Ma, Y., & Chen, S. (2022). Index checkpoints for instant recovery in in-memory database systems. Proceedings of the VLDB Endowment, 15(8), 1671-1683.
[15] Neuhaus, P., Couto, J., Wehrmann, J., Ruiz, D. D. A., & Meneguzzi, F. R. (2019). GADIS: A genetic algorithm for database index selection. In The 31st International Conference on Software Engineering & Knowledge Engineering, 2019, Portugal.
[16] Mostafa, S. A. (2020). A case study on B-Tree database indexing technique. Journal of Soft Computing and Data Mining, 1(1), 27-35.
[17] Hu, D., Chen, Z., Wu, J., Sun, J., & Chen, H. (2021). Persistent memory hash indexes: An experimental evaluation. Proceedings of the VLDB Endowment, 14(5), 785-798.
[18] Dai, S., You, R., Lu, Z., Huang, X., Mamitsuka, H., & Zhu, S. (2020). FullMeSH: improving large-scale MeSH indexing with full text. Bioinformatics, 36(5), 1533-1541.
[19] Gani, A., Siddiqa, A., Shamshirband, S., & Hanum, F. (2016). A survey on indexing techniques for big data: taxonomy and performance evaluation. Knowledge and information systems, 46, 241-284.
[20] Saidu, I. C., Yusuf, M., Nemariyi, F. C., & George, A. C. (2024). Indexing techniques and structured queries for relational databases management systems. Journal of the Nigerian Society of Physical Sciences, 2155-2155.
[21] Sadineni, P. K. (2020, December). Comparative study on query processing and indexing techniques in big data. In 2020 3rd International Conference on Intelligent Sustainable Systems (ICISS) (pp. 933-939). IEEE.
[22] Sharma, S., Gupta, V., & Juneja, M. (2019). A survey of image data indexing techniques. Artificial Intelligence Review, 52, 1189-1266.
[23] Xinwei Lin,Yubiao Pan,Wenjuan Feng,Huizhen Zhang & Mingwei Lin. (2024). RIOKV: reducing iterator overhead for efficient short-range query in LSM-tree-based key-value stores. The Journal of Supercomputing,81(1),343-343.
[24] Khalil Abbal,Mohammed El Amrani,Oussama Aoun & Youssef Benadada. (2025). Adaptive Particle Swarm Optimization with Landscape Learning for Global Optimization and Feature Selection. Modelling,6(1),9-9.
[25] Carlo Ciulla. (2025). Two-dimensional image noise removal and reconstruction using discrete Fourier transform, k-space filtering and Z-space filtering. Progress in Engineering Science,2(1),100056-100056.
[26] Yuting Yan,Ruidong Lu,Jian Sun,Jianxin Zhang & Qiang Zhang. (2025). Breast cancer histopathology image classification using transformer with discrete wavelet transform. Medical Engineering and Physics,138,104317-104317.
[27] Hideki Mutai,Fuyuki Miya,Kiyomitsu Nara,Nobuko Yamamoto,Satomi Inoue,Haruka Murakami... & Tatsuo Matsunaga. (2025). Genetic landscape in undiagnosed patients with syndromic hearing loss revealed by whole exome sequencing and phenotype similarity search. Human Genetics,144(1),1-20.
[28] Qiang Zhang,Xiongwei Zhang,Jibin Yang,Meng Sun & Tieyong Cao. (2025). Introducing Euclidean distance optimization into Softmax loss under neural collapse. Pattern Recognition,162,111400-111400.