Postprint: Time Series Reconstruction Methods for Massive Star Catalog Data Based on Spark Distributed Framework

Authors: Zhao Qing, Quan Wenli, Chen Yarui, Cui Chenzhou, Fan Dongwei

Date: 2024-03-22T00:00:00+00:00

Abstract

Time series reconstruction is an important data processing step in time-domain astronomy, and also forms the foundation for fitting light curves and conducting time-domain analysis research. For MapReduce distributed models such as Hadoop and Spark, the tasks {between nodes} in the distributed cluster are relatively independent during execution, requiring minimal cross-node data transmission. A non-blocking asynchronous execution flow is proposed, where each distributed process continuously processes data exclusively for independent sky regions, while identification tasks for newly added celestial objects at block edges that affect other nodes are appended in delayed batches. The appending method is determined based on the varying progress rates among processes, ensuring no omission in identification computations, thereby improving concurrency efficiency while guaranteeing algorithmic accuracy. Furthermore, different Join strategies between two tables are investigated from both theoretical and experimental perspectives, and a Join-free strategy is proposed. Finally, the aforementioned research is validated through the design of an efficient time series reconstruction system based on the Spark distributed framework. Experimental results demonstrate that, compared with previous studies, the efficiency of this time series reconstruction algorithm is significantly improved, establishing a solid foundation for conducting astronomical time series data analysis in time-domain astronomy.

Full Text

Preamble

Vol. 42, No. 1
Mar., 2024

Progress in Astronomy

Research on Time Series Reconstruction Method of Massive Astronomical Catalogues Based on Spark Distributed Framework

ZHAO Qing[1], QUAN Wen-li[1], CHEN Ya-rui[1;2], CUI Chen-zhou[3], FAN Dongwei[3]
(1. College of Artificial Intelligence, Tianjin University of Science and Technology, Tianjin 300457, China;
2. Engineering Research Center for Integration and Application of E-Learning Technology, Ministry of Education, Beijing 100039, China;
3. National Astronomical Observatories, Chinese Academy of Sciences, Beijing 100101, China)

Abstract

Time series reconstruction is a crucial data processing step in time-domain astronomy and serves as the foundation for fitting light curves and conducting time-domain analysis. MapReduce distributed models such as Hadoop and Spark feature relatively independent tasks among distributed cluster nodes during execution, requiring minimal cross-node data transmission. This paper proposes a non-blocking asynchronous execution flow where each distributed process performs continuous processing exclusively on data from independent sky regions. Additional identification tasks triggered by newly added celestial bodies at block edges are appended in delayed batches to other nodes, with the appending method determined based on the varying progress among processes to ensure no omission in identification calculations. This approach significantly improves concurrent efficiency while guaranteeing algorithmic accuracy. Furthermore, different Join strategies between two tables are investigated from both theoretical and experimental perspectives, and a Join-free strategy is proposed. Finally, the aforementioned research is validated through the design of an efficient time series reconstruction system based on the Spark distributed framework. Experimental results demonstrate a notable improvement in the efficiency of the proposed time series reconstruction algorithm compared to previous studies, establishing a solid foundation for astronomical time series data analysis in time-domain astronomy.

Keywords: time-domain astronomy; cross-match calculation; time series reconstruction; distributed computation; Spark

1 Introduction

In time-domain astronomy, a fundamental task is to perform cross-matching on continuous multiple observations from the same telescope to establish correspondences between celestial bodies across different observation records, thereby generating time series data. This process often involves a contradiction between massive data volumes and high-speed processing requirements. Taking the Ground-based Wide Angle Cameras (GWAC) as an example, this data processing must be completed within a single exposure cycle. Therefore, designing

and developing efficient time series data generation methods for large-field time-domain surveys is of significant importance.

In recent years, advances in time-domain observation technologies have provided extremely rich data resources for time-domain astronomy research. For instance, Gaia DR3 released astrophysical parameters for 1.59 billion sources and 470 million celestial bodies collected over 34 months. GWAC generates 1.1 GB of data every 15 seconds, with star catalog data alone reaching up to 2 TB per observation night. The Large Synoptic Survey Telescope (LSST) produces 15 TB of data per night, with a projected total of 60 PB over its 10-year operation. Faced with such vast accumulated data and rapidly growing new archival data, traditional cross-matching algorithms struggle to simultaneously meet accuracy and efficiency requirements.

The first challenge is the boundary source leakage problem that arises after data partitioning, which impacts both accuracy and efficiency. Current main solutions include: Zhao et al. adding extra edge data to each star catalog, which is complex to implement and inefficient; Du et al. and Yu et al. adopting hybrid identification calculation methods based on HTM and HEALPix indexing, which reduce cross-boundary data transmission but increase storage and computational overhead while still exhibiting a certain proportion of source leakage and accuracy loss. Therefore, the edge source leakage problem requires further optimization.

Secondly, time series identification requires cross-matching calculations across multiple continuous observation catalogs, resulting in higher computational demands than traditional multi-band cross-matching, necessitating further research on parallel computing frameworks. Du et al. proposed a multi-threaded identification algorithm to accelerate computation, while Zhao et al. introduced a multi-core parallel cross-matching algorithm capable of processing catalogs with over 100 million and 470 million records, respectively. However, these methods are not executed in large-scale distributed environments, imposing certain performance limitations for time series reconstruction across multiple catalogs. In studies of time series for homologous catalogs, Yu et al. implemented time series reconstruction in an MPI parallel environment with up to 6 processes, improving identification efficiency but using a parallel environment that is not easily scalable to handle further data volume increases. Xu et al. proposed a rapid time series reconstruction algorithm based on MongoDB, which offers reference value for non-relational data selection in time series reconstruction but lacks sufficient integration with large-scale distributed frameworks.

Therefore, for time series reconstruction of massive homologous catalogs, new parallel frameworks must be introduced to further improve efficiency while maintaining accuracy. Cloud computing based on the MapReduce ecosystem, represented by Hadoop and Spark, has gradually become an important solution for big data processing challenges. Spark, as a general-purpose distributed framework based on in-memory computing, offers greater efficiency advantages and has recently found applications in astronomical information processing. For ex-

ample, Brahem et al. proposed an astronomical data query processing method leveraging Spark' s efficient data processing characteristics, Zečević et al. introduced a scalable astronomical analysis framework based on Spark, and Liu et al. combined MapReduce/Spark with an unsupervised clustering approach for pulsar candidate screening. These developments demonstrate that adopting distributed computing technology has become a trend for astronomical research under massive data volumes.

However, direct application of Spark is unsuitable for this problem. MapReduce distributed models like Hadoop and Spark require relatively independent tasks among multiple processes during execution; otherwise, frequent inter-node communication becomes an efficiency bottleneck. Yet due to the classic boundary source leakage problem in cross-matching, edge data transmission is unavoidable, severely reducing the acceleration ratio in practical applications. Therefore, we designed a non-blocking asynchronous execution flow where each distributed process performs continuous processing exclusively on independent sky region data. Additional identification tasks triggered by newly added celestial bodies at block edges are appended in delayed batches, significantly improving concurrent efficiency while ensuring no omission in identification calculations.

Moreover, when transforming cross-matching calculations into Join operations between two tables, the choice of optimization strategy is also critical to efficiency. Considering these issues comprehensively, this paper designs a distributed time series reconstruction algorithm suitable for massive star catalogs, focusing on three key optimizations: (1) addressing the boundary source leakage problem by designing a non-blocking asynchronous execution flow suitable for Spark architecture, which reduces data transmission while ensuring accuracy and effectively improves algorithmic efficiency; (2) transforming time series identification into Join operations between large and small tables, comparing two different Join implementation strategies and a Join-free operation strategy, and providing applicable scenarios for different algorithms; and (3) validating the research content through the design and implementation of a time series identification system based on the Spark distributed framework.

2 Design of Asynchronous Non-blocking Multi-process Execution Mode and Distributed Update Synchronization

After distributed partitioning of star catalogs, the boundary source leakage problem emerges: due to positional errors, the counterpart of a celestial body located at a partition boundary in another observation may cross the boundary and appear in a different block. Therefore, edge data requires additional processing to improve matching accuracy. In time series reconstruction, where cross-matching occurs between continuous observation catalogs, newly added celestial bodies at block edges necessitate cross-node synchronous updates. If multi-nodes in a distributed architecture operate in blocking mode, global updates for newly added celestial bodies can be guaranteed, but this means that calculations for all celestial bodies in each observation record must be completed before pro-

cessing the next observation data, constraining the efficiency advantages of the distributed framework and causing computational resource waste.

To address these issues, this paper proposes an asynchronous non-blocking multi-process execution mode that significantly improves efficiency and concurrent processing capability. The non-blocking mode means that when newly added celestial bodies at edges are updated to adjacent blocks, the adjacent blocks do not halt their current batch processing to handle the new data. Instead, they continue with batch identification calculations for the star catalog, with each process operating independently in terms of processing progress, allowing each to efficiently process subsequent data within the block sequentially. Identification omissions caused by asynchronous progress among processes are addressed through centralized appending after batch identification calculations conclude. This non-blocking multi-process mode maximizes computational resource utilization across nodes, achieving high-efficiency parallel processing.

This paper uses the earliest star catalog on the identification timeline as the reference catalog. Each celestial body in subsequent observation records undergoes pairwise spherical distance calculation with celestial bodies in the same partition of the reference catalog. The calculation formula is:

$$d = \arccos[\sin\phi_1 \sin\phi_2 + \cos\phi_1 \cos\phi_2 \cos(\alpha_1 - \alpha_2)]$$

where (α_1, ϕ_1) represents the coordinates of celestial body A and (α_2, ϕ_2) represents the coordinates of celestial body B. When the angular distance between two celestial bodies is very small, the angular distance formula can be approximated as:

$$\sqrt{[(\alpha_1 - \alpha_2)\cos\phi]^2 + (\phi_1 - \phi_2)^2}$$

where $\phi = (\phi_1 + \phi_2)/2$.

Given certain error radii, assuming the error radii of the two star catalogs are r_1 and r_2, two celestial bodies can be identified as the same if they satisfy:

$$d \leq |r_1| + |r_2|$$

Since different observation catalogs are matched with the reference catalog, only the reference catalog requires edge replica redundancy storage, reducing replica storage data volume. A small number of newly added celestial bodies during identification are appended to the reference catalog, and if located at block edges, require simultaneous updates to the reference catalogs across multiple computing nodes.

Due to the designed non-blocking multi-process computation mode, each distributed process performs continuous processing exclusively on independent sky

region data. When newly added celestial bodies at block edges are synchronized across nodes, any identification omissions due to progress advancement are compensated in later batches. This results in stronger system concurrency, higher computational resource utilization, and greater suitability for time series reconstruction of massive star catalogs.

The specific update flow diagram is shown in Figure 1 [Figure 1: see original paper]. Since identification calculation tasks are executed asynchronously, the progress of identification in the current block and adjacent blocks is not synchronized. The solution to the specific update synchronization problem for newly added celestial bodies is as follows:

Figure 1. Flowchart of cross-node update for edge celestial bodies in the reference catalog

(1) **Adjacent block identification progress is ahead.** In this case, one possibility must be considered: if edge celestial body X is a supernova, its first appearance might occur in this adjacent block. Before updating X to the adjacent block' s reference catalog, it may already exist. As shown in Figure 2 [Figure 2: see original paper], a new celestial body X appears in block A on day 15. Because X is at the edge between blocks A and B, it is added to block A' s reference catalog while being sent to the node where block B resides. If X first appears in block B on day 10 and has already been sent to A, then block A would not have new celestial body X appearing on day 15. Due to transmission delays, X was not updated to block A' s reference catalog in time, leading to its rediscovery, while block B already had celestial body X before day 15. Therefore, when block B' s node receives celestial body X, it only needs to perform identification with newly added celestial body data in block B' s reference catalog from day 11 to day 25: if identification succeeds, discard X because it was previously discovered in B; if identification fails, add X to block B' s reference catalog and perform identification with all celestial bodies in block B from day 11 to day 25.

Figure 2. Schematic diagram of edge update when adjacent block identification progress is ahead

(2) **Adjacent block identification progress is behind.** Edge celestial bodies are added to adjacent blocks' reference catalogs and identified with subsequent celestial bodies based on observation time. If a subsequent celestial body' s observation time is earlier than that of the edge celestial body, identification is not performed.

3 Comparison of Different Join Strategies Between Two Tables

To calculate pairwise distances between celestial bodies under the same block number and perform threshold judgment, two approaches are considered: one

utilizes Join operations in databases, while the other directly employs Spark HDFS file data and operator operations without using Join. This chapter details the Join-based method; the Join-free method is described in Section 4.2.

To implement distance calculation and threshold filtering, non-equijoin is adopted, with the join condition being that the spherical distance between a record in the reference catalog and a record in the catalog to be identified is less than the threshold, as specified in formulas (1)-(5). Successfully joined results represent matched record pairs. To improve processing performance, Hash Join is employed, mapping join field values to buckets in a hash table to reduce reshuffle or reordering, demonstrating good adaptability for large-scale data.

Considering that reference catalog data needs to be matched one-to-one with other catalogs along the continuous time axis, the reference catalog is treated as the small table while other catalogs along the continuous time axis are integrated as the large table. To further enhance performance, two hash join strategies suitable for small and large tables are experimentally compared: Shuffle Hash Join (SHJ) and Broadcast Hash Join (BHJ).

SHJ implementation involves repartitioning both large and small tables according to the join key using the HashPartitioner method, i.e., the Shuffle operation. This aims to place data with the same key from both tables into the same partition, then perform join operations on data within the same partition.

BHJ implementation broadcasts small table data to other nodes in Spark, then independently executes join operations on each node, maximizing CPU resource utilization for join operations. This join strategy sacrifices node space to some extent to avoid time-consuming Shuffle operations, but broadcasting large tables may impose significant pressure on driver nodes, as the cost of broadcasting large tables is expensive.

This paper implements identification calculations between two tables based on these two join strategies and conducts comparative studies. Experimental results are presented in Section 5.3, demonstrating that when computational resources are sufficient, BHJ is more advantageous for identification calculations using Join operations between two tables, while SHJ should be selected when resources are insufficient.

Figure 3 [Figure 3: see original paper] illustrates the Join flow between two tables. First, the `textFile` operator reads HDFS metadata to create RDDs (Resilient Distributed Datasets) [13]. Then, the `map` operator and `toDF` function convert RDDs to DataFrames. The two tables are joined based on the Healpix index to return a new DataFrame. The `withColumn` function adds a new column (angular distance between two celestial bodies), and the `filter` function selects data where the angular distance is less than the threshold. Finally, qualified data is output to HDFS (Hadoop Distributed File System) [14], completing the time series reconstruction process for star catalog data.

Figure 3. Flowchart of Join between two tables

4.1 Design Architecture of Massive Star Catalog Data Time Series Reconstruction Method Based on Spark

This paper proposes a massive star catalog time series reconstruction method under Spark' s asynchronous non-blocking mode, with the overall framework shown in Figure 4 [Figure 4: see original paper]. In the preprocessing stage, raw star catalog data is effectively filtered to reduce storage and identification costs and generate metadata files. HEALPix indices are then calculated, and data is formatted. Notably, adjacent block indices for celestial bodies must also be recorded to enable redundant storage of edge data from adjacent blocks in the reference catalog. Finally, star catalog data is partitioned into different HEALPix block files through Spark' s custom partition function, awaiting subsequent identification calculations. Next, preprocessed star catalogs undergo distributed identification calculations. In the final stage, time series data for celestial bodies is generated.

Figure 4. Overall framework of massive star catalog data time series reconstruction method based on Spark

4.2 Join-Free Identification Calculation Method

Chapter 3 introduced the method of integrating distance calculation and threshold filtering into database join operations. While simple to program and implement, frequent Join operations still impact efficiency. To further optimize efficiency, a Join-free method is considered here, which directly applies distance calculation formulas and threshold filtering to HDFS data files. Data from different HEALPix partitions execute identification calculation tasks in parallel, with the reference catalog requiring frequent read/write operations stored in memory. Star catalogs along the continuous time axis are sequentially matched with celestial bodies in the reference catalog. The identification calculation flow is shown in Figure 5 [Figure 5: see original paper]. This method will be compared with the Join-based method and previous excellent methods.

(1) After reading star catalog data from HDFS, the reference star table for the current partition is created.

(2) If there are celestial bodies to be identified in the current partition and edge celestial bodies for the current partition exist in the Redis database, edge celestial bodies are updated to the reference star table according to the solution for reference star table update synchronization problems.

(3) If no edge celestial bodies for the current partition exist in the Redis database or updates are completed, the next time period' s star table is matched with the reference star table: if matching fails, the celestial body is considered new and added to the reference star table; if matching succeeds, data for both celestial bodies is recorded.

(4) If a new celestial body is an edge body, indicating potential edge source leakage in adjacent partitions, it is first added to Redis data. Through step (2), this edge celestial body will be added to adjacent partitions' reference star tables. Redundant storage of edge celestial bodies in reference star tables prevents edge source leakage.

(5) Repeat step (2) until identification for all time period star tables in the current partition is completed.

Figure 5. Flowchart of time series reconstruction calculation based on Spark

5.2 Star Catalog Preprocessing Performance Evaluation

To analyze preprocessing performance of star catalog files, Figure 6 [Figure 6: see original paper] shows file sizes before and after preprocessing and preprocessing time for different datasets. Preprocessed star catalog files for AST3-I datasets are approximately 28% of original files, while AST3-II datasets are about 83%. Preprocessing time is within acceptable ranges. The processed file sizes demonstrate that preprocessing raw star catalog data can significantly reduce data storage volume, save storage time, and avoid unnecessary cross-node data transmission time, thereby improving time series reconstruction efficiency.

Figure 6. Preprocessing performance (HEALPix level = 10)

5.3 Comparison of Different Join Algorithms Between Two Tables

To verify the performance of BHJ and SHJ algorithms for identification calculations between two tables, experiments were conducted using the Tess4 dataset across different nodes and data volumes.

Figure 7a [Figure 7: see original paper] shows that with fixed star catalog data volume, identification calculation time using BHJ is less than SHJ across different node counts. With 16 computing nodes, BHJ reduces identification calculation time by 56.6% compared to SHJ. Figure 7b shows that with 32 computing nodes, BHJ also outperforms SHJ across different star catalog data volumes because SHJ' s Shuffle operation is time-consuming, causing star catalog data transmission across different nodes. Notably, when computational node memory is configured at 1 GB and two-table data sizes reach 150 MB and 600 MB respectively, BHJ encounters insufficient memory, causing identification calculation failure. Therefore, BHJ is advantageous when computational resources are sufficient, while SHJ should be selected when resources are insufficient.

Note: (a) Comparison of two-table Join across different computing node counts; (b) Comparison of two-table Join across different data volumes.

Figure 7. Comparison of two-table Join

5.4.1 Performance of Join-Free Identification Method at Different HEALPix Levels

Different HEALPix levels produce different star catalog partitioning granularities. Higher levels yield finer granularity (more blocks), increasing edge data and requiring additional time to process edge data and update cross-node edge data. Lower levels increase the number of celestial bodies per partition, raising computational load. For homologous star catalogs, target sky regions typically have non-uniform celestial body distributions. In this paper' s star catalog partitioning, some partitions contain only a few celestial bodies, causing nodes with small counts to remain idle most of the time, wasting computational resources and failing to fully exploit distributed computing performance. Therefore, HEALPix level selection significantly impacts time series reconstruction performance, requiring optimal level selection to reduce computational complexity.

To select an appropriate HEALPix level, experiments were conducted using 64 computing nodes to perform time series reconstruction on multiple datasets with similar data volumes at different HEALPix levels. As shown in Figure 8 [Figure 8: see original paper], for all datasets in the experiments, computation time is minimized at HEALPix level 10, increasing when level decreases to 8 or 9 or increases to 11 or 12. Since HEALPix is a quadtree structure, each level increase multiplies the number of star catalog blocks by approximately 4, increasing identification calculation task counts. Excessive tasks increase task startup and switching overhead, degrading performance. Each level decrease reduces block count to 1/4 but increases data per partition by about 4 times, reducing task count and failing to fully exploit cluster parallel computing capabilities, slowing execution and potentially causing out-of-memory exceptions. For this paper' s experimental environment, HEALPix level 10 is optimal, and all subsequent experiments use level 10. This value may differ in different test environments.

Figure 8. Join-free identification calculation time at different HEALPix levels

5.4.2 Performance Evaluation of Join-Free Method with Different Nodes

To meet efficiency and accuracy requirements in time series reconstruction, this paper proposes a Join-free identification method under Spark' s asynchronous non-blocking mode, which breaks through traditional Join algorithms to further improve time series reconstruction efficiency.

Figure 9 [Figure 9: see original paper] shows that identification calculation time for each dataset decreases as computing node count increases. With fixed computing node count, identification calculation time increases with dataset file size (except for Tess5-13). Although Tess5-13' s total file size is similar to HD117688, its identification calculation time is about 56% longer due to more edge data requiring additional processing time.

Figure 9. Join-free identification calculation time at different HEALPix levels

5.4.3 Comprehensive Performance Evaluation of Join-Free Identification Method

To verify the overall performance of the Join-free method, experiments were conducted using the AST3-I dataset across different computing node counts with a dataset size of 100 GB. As shown in Figure 10 [Figure 10: see original paper], identification calculation time decreases as node count increases, requiring only 6.18 minutes with 64 computing nodes.

Figure 10. Comprehensive experimental performance across different node counts

Figure 11 [Figure 11: see original paper] shows the speedup ratio across different computing node counts. Acceleration becomes apparent as node count increases, partly due to excellent performance in reference star table replica strategy and reduced cross-node data transmission. As computing nodes increase, identification calculation time decreases, showing essentially linear growth trend.

Figure 11. Speedup ratio variation with computing node count

5.4.4 Comparison with Previous Work

Currently, time series reconstruction across continuous multiple star catalogs remains relatively rare. The work by Yu et al. [7] is most similar to this paper. They implemented parallel computation through MPI programming in an experimental environment with Ubuntu OS, Intel i7-4790 CPU (8 cores @ 3.6 GHz), and 16 GB memory. This paper compares results using the same dataset with their 6-process experiments. Although Yu et al.' s method offers a visualization interface with usability advantages, this paper' s method surpasses their performance using only 5 computing nodes. As shown in Figure 12 [Figure 12: see original paper], when computing node count continues to increase, this paper' s method achieves sustained performance improvements. With 64 nodes, identification calculation times for HD88500, HD117688, and HD136488 datasets are 30%, 18%, and 19% of Yu et al.' s method, respectively, representing substantial performance improvements. Moreover, as node count increases in the distributed environment, this paper' s method achieves near-linear speedup because it only replicates the reference star table, processing other catalogs in blocks according to HEALPix levels. The primary parallelization is data-parallel, with independent computation tasks across nodes and asynchronous multi-node synchronous updates and appended identification for newly added celestial bodies, resulting in minimal cross-node data transmission during identification and near-linear acceleration with increasing nodes. In summary, this paper' s large-scale distributed architecture based on Spark and excellent speedup provide good scalability and adaptability for time series reconstruction

tasks from increasingly large time-domain telescopes like GWAC. Additionally, this time series reconstruction method requires only one indexing approach, effectively saving data storage space. In terms of computational accuracy, this paper' s theoretical source leakage rate is 0, while Yu et al.' s hybrid HTM and HEALPix indexing method has a source leakage rate of approximately 4.5%. The introduction of new parallel frameworks and algorithmic design plays a decisive role in performance improvement.

Figure 12. Comparison with previous method

5.4.5 Comparison of Different Identification Methods

To compare the performance of Join-free identification with the two Join algorithms, experiments were conducted using the HD88500 dataset on 64 computing nodes. Table 3 shows that the Join-free method is significantly more efficient than traditional Join-based identification methods. Join-based methods suffer from cross-node data transmission and data skew issues, while the Join-free method avoids substantial cross-node data transmission through preprocessing, greatly reducing identification calculation time and improving efficiency for faster results.

Table 3. Comparison of different identification methods

6 Conclusion

This paper addresses challenges in time series reconstruction of massive astronomical star catalog data, combining latest theoretical methods for star catalog indexing, algorithm design, and data updates to solve efficiency and accuracy problems, providing data support for subsequent research. The paper first proposes a time series reconstruction method under asynchronous non-blocking mode, introducing a reference star table replica strategy that reduces data transmission while ensuring accuracy. For cross-node data update and synchronization issues in distributed environments, corresponding solutions are provided for different scenarios. Additionally, theoretical and experimental comparisons of Broadcast Hash Join and Shuffle Hash Join algorithms between two tables are presented, with applicable scenarios identified for each. Finally, a Join-free time series reconstruction method based on the Spark distributed framework is proposed, achieving more efficient time series reconstruction for massive astronomical star catalogs. Experiments demonstrate that the designed time series reconstruction algorithm is efficient, feasible, and practically valuable, promoting development in time-domain astronomy. The large-scale distributed computing architecture based on Spark and near-linear speedup make this method applicable to time series reconstruction tasks for larger-scale time-domain telescopes like GWAC.

References

[1] Cordier B, Wei J, Atteia J L, et al. Proceedings of Science, DOI: https://doi.org/10.22323/1.233.0005
[2] Vallenari A, Brown A G A, Prusti T, et al. A&A, 2023, 674: A1(2023)
[3] Wan M, Wu C, Ying Zhang, et al. Astronomical Research & Technology, 2016, 13(3): 373
[4] Ivezic Z, Tyson J A, Acosta E, et al. AJ, 2019, 873(2): 111
[5] Zhao Q, Sun J, Yu C, et al. Transactions of Tianjin University, 2011, 17(1): 62
[6] Du P, Ren J J, Pan J C, et al. Science China: Physics, Mechanics and Astronomy, 2014, 57(3): 577
[7] Yu C, Li K, Tang S, et al. MNRAS, 2020, 496(1): 629
[8] Xu D Y, Zhao Q, Quan W L, et al. Progress in Astronomy, 2022, 40(2): 298
[9] Brahem M, Yeh L, Zeitouni K. Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Seattle, WA: ACM, 2018: 229
[10] Zečević P, Slater C T, Jurić M, et al. AJ, 2019, 158(1): 37
[11] Liu Y, Ma Z, You Z Y, et al. Acta Astronomica Sinica. 2022, 63(3): 10
[12] Armbrust M, Xin R S, Lian C, et al. Proceedings of the 2015 ACM SIGMOD international conference on management of data, New York: ACM, 2015: 1383
[13] Zaharia M, Chowdhury M, Das T, et al. Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, Berkeley: USENIX Association, 2012: 15
[14] Shvachko K, Kuang H, Radia S, et al. Proceedings of the IEEE 26th symposium on mass storage systems and technologies (MSST), USA: IEEE, 2010: 10

Note: Figure translations are in progress. See original paper for figures.