Received: 23 June 2021 | Revised: 15 October 2021 | Accepted: 26 December 2021

DOI: 10.1002/cpe.6996

RESEARCH ARTICLE

WILEY

Toward fast theta-join: A prefiltering and amalgamated partitioning approach

Jiashu Wu[1,2] | **Yang Wang**[1] | **Xiaopeng Fan**[1] | **Kejiang Ye**[1] | **Chengzhong Xu**[3]

[1]Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China

[2]University of Chinese Academy of Sciences, Beijing, China

[3]University of Macau, Macau, China

Correspondence
Yang Wang, Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China.
Email: yang.wang1@siat.ac.cn

Funding information
Key-Area Research and Development Program of Guangdong Province, Grant/Award Number: 2020B010164002; Shenzhen Basic Research Program, Grant/Award Numbers: JCYJ20170818153016513, JCYJ20200109115418592

Abstract

As one of the most useful online processing techniques, the theta-join operation has been utilized by many applications to fully excavate the relationships between data streams in various scenarios. As such, constant research efforts have been put to optimize its performance in the distributed environment, which is typically characterized by reducing the number of Cartesian products as much as possible. In this article, we design and implement a novel fast theta-join algorithm, called *Prefap*, by developing two distinct techniques—*prefiltering* and *amalgamated partitioning*—based on the state-of-the-art FastThetaJoin algorithm to optimize the efficiency of the theta-join operation. Firstly, we develop a prefiltering strategy before data streams are partitioned to reduce the amount of data to be involved and benefit a more fine-grained partitioning. Secondly, to avoid the data streams being partitioned in a coarse-grained isolated manner and improve the quality of the partition-level filtering, we introduce an amalgamated partitioning mechanism that can amalgamate the partitioning boundaries of two data streams to assist a fine-grained partitioning. With the integration of these two techniques into the existing FastThetaJoin algorithm, we design and implement a new framework to achieve a decreased number of Cartesian products and a higher theta-join efficiency. By comparing with existing algorithms, FastThetaJoin in particular, we evaluate the performance of *Prefap* on both synthetic and real data streams from two-way to multiway theta-join to demonstrate its superiority.

KEYWORDS

amalgamated data stream partitioning, cartesian product reduction, online data stream, prefiltering, theta-join (θ-join)

1 | INTRODUCTION

As the big data technology becomes more prevalent[1,2] and widely deployed,[3-6] tremendous amount of online data streams have been generated.[7-9] In the financial market,[10,11] the price of stocks keeps fluctuating, the currency conversion rates change every few seconds in an online manner. For the meteorological monitoring services,[12] thousands of monitoring stations constantly monitor the meteorological data in real-time,[13] such as wind speed and temperature and so forth. Therefore, how to process these online data streams efficiently and fully excavate the knowledge[14,15] behind them become crucial to explore.

To relate data streams together, the join operation[16,17] is one of the vital operations that is capable to detect scenarios that satisfy certain conditions. To find out data elements in two data streams that are equal, equi-join[18-20] should be used. As for nonequal relationships between data

streams, the theta-join[21,22] comes to help. Theta-join, denoted as follows

$$\mathcal{R} \bowtie \mathcal{S}(\mathrm{A}\ \theta\ \mathrm{B}), \tag{1}$$

serves as a special kind of join operation that relates the attribute A of the data stream $\mathcal{R}$ and the attribute B of the data stream $\mathcal{S}$ using a nonequal theta condition among one of the following $\{<, \leq, >, \geq\}$. Cartesian products[23,24] between two data streams will be generated and data element pairs that satisfy the theta condition are picked out to form the theta-join results. For instance, as illustrated in Figure 1, the inputs of this two-way theta-join operation are data stream phone (Stream $\mathcal{R}$) and data stream laptop (Stream $\mathcal{S}$). Under the join predicate $\mathcal{R} \bowtie \mathcal{S}$ ($\mathcal{R}$.B > $\mathcal{S}$.B), all phone-laptop combinations that satisfy the price of the phone is higher than the price of the laptop will be returned as results. Therefore, four satisfied combinations are returned as output as shown in Figure 1.

Moreover, the theta-join can also be generalized to work on multiple data streams. For instance, the 3-way theta-join predicate is expressed as follows

$$\mathcal{R} \bowtie \mathcal{S} \bowtie \mathcal{T}(\mathrm{A}\ \theta_1\ \mathrm{B}\ \theta_2\ \mathrm{C}) \tag{2}$$

which exploits all A, B, and C attribute combinations that satisfy the A θ_1 B θ_2 C condition from three data streams $\mathcal{R}$, $\mathcal{S}$, and $\mathcal{T}$, respectively. When dealing with data streams that come in an online manner, the theta-join algorithm will receive the data stream in the form of a window, with the window size *w* denotes the amount of data consisted in the window. Then, the theta-join operation will be performed between windows of data streams. As a powerful data analysis and processing tool,[25] theta-join can be widely utilized in broad applications, such as discerning on which day stock $\mathcal{R}$ performs worse than stock $\mathcal{S}$ in 2019, or whether the wind speed in July is faster than June 2020 in most of the days. The theta-join operation has also been implemented in popular database management systems such as Oracle database,[26] PostgreSql database[27] and so forth.

However, the theta-join's efficiency is heavily affected by the number of involved time-consuming Cartesian products.[17,28] Handling the theta-join in a careless way can result in the number of Cartesian products grows drastically, and will even lead to an exponential surge if multiway theta-join is conducted. The unacceptably large number of Cartesian products also become a curse, especially in distributed environments[29-31] where the incurred I/O overhead and the communication cost[25] make it an influential factor that dramatically compromises the efficiency of the theta-join operation. Therefore, they should be reduced as much as possible. As a result, numerous efforts have been made by both industry and academia from different aspects to optimize the efficiency of the theta-join operation.[32-35]

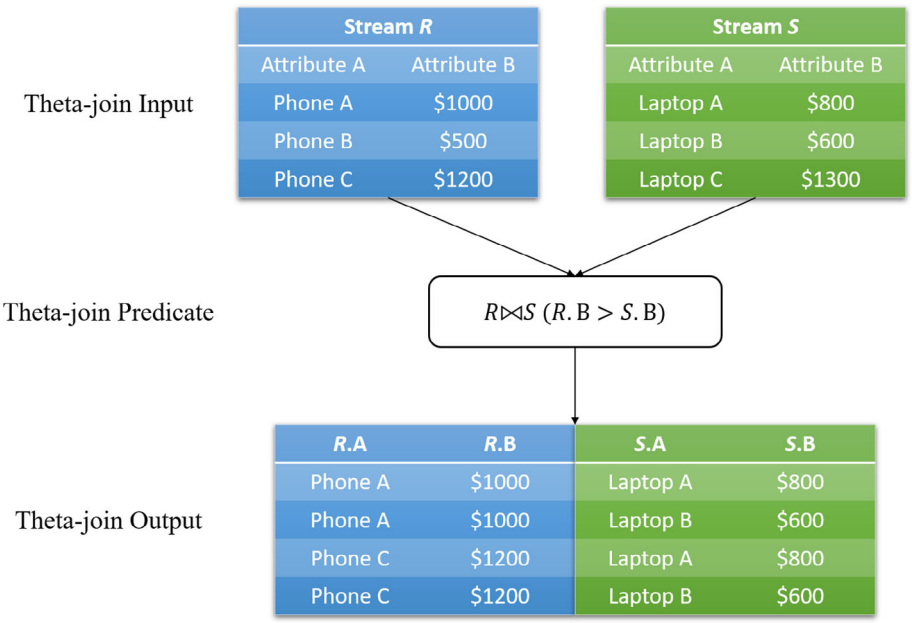

FIGURE 1 An illustrative example of a theta-join predicate $\mathcal{R} \bowtie \mathcal{S}$ ($\mathcal{R}$.B > $\mathcal{S}$.B), its input and its corresponding output. In this example, there are only four phone-laptop combinations that satisfy the price of the phone is more expensive than laptop

Although the existing algorithms in the literature exhibited different merits in making the theta-join operation more efficient, they still suffered from some deficiencies in effective Cartesian product reduction and data stream partitioning, which more or less compromised the theta-join operation efficacy. Therefore, we still have rooms to further improve them in various ways. In this paper, we propose a novel fast online theta-join algorithm, called *Prefap*. Instead of performing filtering only after partitioning as in References 32-35, it makes sense to perform a *prefiltering* based on the theta condition before the partitioning takes place as it can not only reduce the amount of data involved in the partitioning but also make the partitioning more fine-grained. Since our partitioning is based on the range boundaries calculated using the minimum and the maximum values of the data streams, the prefiltering of the data streams is possible to condense the range between the minimum and the maximum value, hence making the partition more fine-grained. Moreover, this prefiltering mechanism does not suffer from the severe overhead caused by the time-consuming sorting operation.[36,37] As opposed to References 32-34, the proposed algorithm requires neither interpartition nor intrapartition ordering.

Furthermore, for all the aforementioned algorithms, the partitioning of two data streams are conducted in a coarse-grained isolated manner, that is, the partitioning of data stream $\mathcal{R}$ has no impact on the partitioning of data stream $\mathcal{S}$, which may impair the effectiveness of the filtering mechanism after the partitioning. To overcome the drawback incurred by the isolated coarse-grained partitioning, the proposed algorithm introduces an *amalgamated partitioning* mechanism. With this mechanism, the data stream will be partitioned based on the amalgamated partitioning boundary that fuses the partitioning information of data streams. Hence, worthless Cartesian products between partitions that are deemed impossible to possess valid theta-join results will be avoided, and hence making the algorithm more efficient, which is an improvement to References 32-35.

To validate the effectiveness of the proposed method and to make it applicable, we integrate the prefiltering strategy and the amalgamated partitioning mechanism to form a new theta-join processing framework. The framework uses FastThetaJoin[35] as a basis. However, the proposed *Prefap* framework makes contributions by performing prefiltering and avoiding the isolated partitioning. With the *Prefap* framework, we can substantially boost the efficiency of the theta-join operation. The proposed *Prefap* framework is comprehensively evaluated on both synthetic and real data streams in distributed environments and compared with other algorithms, demonstrating its superior performance.

Therefore, in summary, the article makes the following contributions:

- We propose a prefiltering strategy that can not only reduce the amount of data involved in the partitioning but also make the partitioning more fine-grained.
- We introduce the amalgamated partitioning mechanism that avoids the coarse-grained isolated partitioning and hence benefits the reduction of Cartesian products and makes the algorithm more efficient.
- We unify the proposed prefiltering strategy and the amalgamated partitioning mechanism to form a holistic framework, called *Prefap*, that can boost the effectiveness of theta-join operations while avoiding the time-consuming burdens such as sorting.
- The *Prefap* framework is implemented and comprehensive empirical evaluations on both synthetic and real data streams are conducted to testify the superiority.

The remainders of the article are organized as follows. Section 2 overviews some related works to show the distinct features of the proposed algorithm. The *Prefap* algorithm is introduced in Section 3. Section 4 presents the experimental results, which are also comprehensively analyzed and discussed. The last section concludes the article.

2 | RELATED WORK AND RESEARCH OPPORTUNITY

As the scale of data streams keeps growing rapidly,[7-9] how to efficiently utilize the theta-join to process data streams becomes a crucial problem, and thereby attracting numerous attentions from both industry and academic communities. In this section, we first overview some past research efforts on improving the efficiency of the theta-join operation, and then identify their deficiencies to show our research opportunities.

2.1 | Range-based method

Dewitt et al.,[32] proposed a range-based method (RBM) to carry out the theta-join operation in distributed environments. The joint attributes in data streams were firstly sorted, then two data streams were partitioned where the partitioning boundaries were characterized by sampled ranges, and Cartesian products were performed without any filtering being performed.

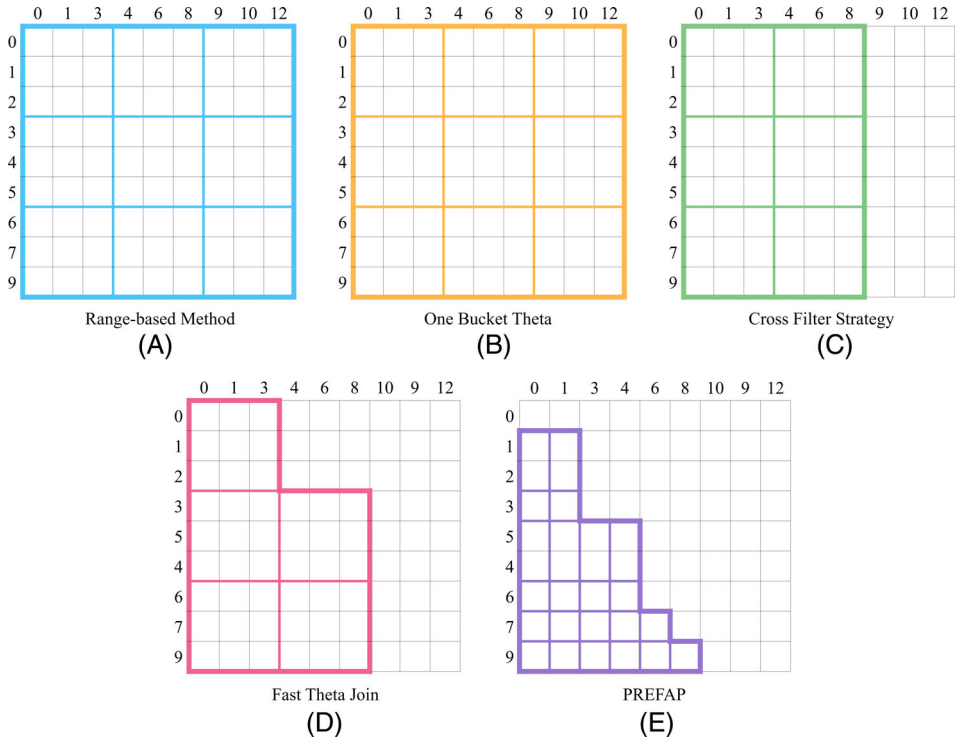

FIGURE 2 Illustration of the number of Cartesian products involved in each algorithm. The theta condition in this example is $\mathcal{R} > \mathcal{S}$. The data stream $\mathcal{R}$ is placed vertically and the data stream $\mathcal{S}$ is placed horizontally. The thick colored border indicates the boundary of Cartesian products that need to be performed, while the thin colored border indicates the partitioning boundary

For example, in Figure 2A, if we set the number of partitions to three, the range-based method will sample the fourth and the seventh value from the sorted data stream $\mathcal{R}$, which are 3 and 6, respectively. Then the data stream $\mathcal{R}$ is partitioned based on the ranges as follows: $(-\infty, 3)$, $[3, 6)$, and $[6, +\infty)$. The same partitioning is applied for data stream $\mathcal{S}$ as well.

However, the range-based method suffered from several disadvantages, which made it extremely time-consuming. Firstly, sorting data streams is very inefficient,[36,37] especially when the size of the data stream is large. Secondly, the range-based method does not apply any kind of filtering, hence, entire data streams will participate in the Cartesian products, resulting in a huge number of Cartesian products being performed. Finally, the range-based method suffers a lot from data skewness. For instance, `[1,1,1,1,1,2,2,3]` will be partitioned into $(-\infty, 1)$, $[1, 2)$, and $[2, +\infty)$, which could lead to severely imbalanced workloads between partitions, and is especially hurtful in distributed environments.

2.2 | Randomized method

To address the issues faced by the Range-based method, Okcan et al.,[33] put forward an algorithm, called one-bucket-theta (OBT), which partitioned the data stream as evenly as possible, and then randomly distributed the partitioned blocks in the distributed environment.

For instance, in Figure 2B, the one-bucket-theta algorithm first sorts the data streams. Then, take the number of partitions to be three as an example, instead of partitioning the data stream based on ranges, the one-bucket-theta partitions the first → third, fourth → sixth, and seventh → ninth data elements into three partitions, respectively, resulting in a relatively more even partitioning scheme in practice.

Furthermore, the randomized distribution of partitions to processes in the distributed environment further balances the workloads among processes and avoids the severe load imbalance. However, the one-bucket-theta still suffers from the time complexity brought by its sorting operation. Also, the lack of filtering strategy makes entire data streams being involved when performing the Cartesian products, which significantly impairs its efficiency.

2.3 | Filtering method

In order to improve the efficiency of the theta-join by reducing the number of Cartesian products, the cross filter strategy (CFS) proposed by Liu et al.,[34] performed stream-level filtering after the data stream partitioning to eliminate data elements that are unlikely to form valid theta-join results based on the theta condition.

As shown in Figure 2C, under the ">" theta condition, the Cartesian products will not be performed between the entire data stream $\mathcal{R}$ and partition [9,12] of data stream $\mathcal{S}$. Since the maximum value 9 of data stream $\mathcal{R}$ equals to the minimum value 9 in partition [9,12] of data stream $\mathcal{S}$, hence there is no way for the Cartesian products between data stream $\mathcal{R}$ and range [9,12] of data stream $\mathcal{S}$ to possess valid ">" theta-join results, and hence partition [9,12] of data stream $\mathcal{S}$ is eliminated by the stream-level filtering strategy. Following the same principle, the stream-level filtering can also be applied to other similar theta conditions as in $\{\geq, <, \leq\}$.

However, this stream-level filtering is coarse-grained and fails to remove unnecessary Cartesian products as much as possible. The stream-level filtering finds out that partition [0,3) of data stream $\mathcal{R}$ can form valid theta-join results with partition [0,4) of data stream $\mathcal{S}$ under the ">" theta condition, and hence partition [0,3) of data stream $\mathcal{R}$ could not be eliminated and the Cartesian products between it and the entire filtered data stream $\mathcal{S}$ will be performed, even though the Cartesian products between [0,3) and [4,9) are totally redundant.

To further improve the efficiency of the theta-join operation in terms of reducing the number of Cartesian products, Hu et al.,[35] presented the FastThetaJoin (FTJ) algorithm. In contrast to performing filtering at the stream-level, the FastThetaJoin algorithm utilized the partition-level filtering strategy, which compared all the partition pairs from both data streams and avoided performing the Cartesian products between the partitions that are unable to form any valid theta-join results based on the theta condition as shown in Figure 2D. However, the lack of the prefiltering mechanism before partitioning not only incurred all data elements in two data streams to be involved in partitioning, irrespective of whether they are able to form valid theta-join results or not, but also made the partitioning more coarse-grained, and hence compromised the overall performance.

Furthermore, in the FastThetaJoin algorithm, the partitioning of data streams was conducted in an isolated manner. Failing to partition the data streams collaboratively made the FastThetaJoin highly laborious as the coarse-grained isolated data stream partitioning is incapable to remove some worthless data elements in some partitions. In terms of auxiliary procedures, the FastThetaJoin algorithm adopted the repartitioning on oversized partitions so that the workloads would be well-balanced and the method will be more distributed-environment-friendly. The Cartesian products were performed between the remaining partitions of two data streams.

2.4 | Research opportunity

Despite constant optimization efforts being made, there is still room to extend and improve the efficiency of the theta-join operation, hence it brings us the motivation of the proposed framework. As shown in Figure 2D, performing the filtering strategy after partitioning ends up with performing the Cartesian products between 0 in data stream $\mathcal{R}$ and the partition [0,4) of data stream $\mathcal{S}$, which is unnecessary under the ">" theta condition. Hence, a prefiltering strategy that filters the data streams before the partitioning will be beneficial to reduce the amount of data that needs to be partitioned, and it can also make the partitioning become more fine-grained as the prefiltering shortens the ranges of the data streams.

Moreover, the isolated partitioning of data streams can be substituted with the amalgamated partitioning mechanism. Under the isolated manner with the theta condition as ">", the Cartesian products between partition [3,6) and [4,8) will be conducted simply because of a single valid theta-join result which is $5 > 4$, but the Cartesian products between [3,4] and [4,8) are completely unnecessary. To avoid these pointless Cartesian products incurred by the isolated partitioning, the partitioning information of two data streams will be amalgamated to form an amalgamated partitioning scheme so that the partition [3,6) will be split to filter more unnecessary Cartesian products that possess no valid theta-join result.

Hence, by introducing the prefiltering strategy and the amalgamated partitioning mechanism, we can integrate them into a unified framework to effectively reduce the amount of unnecessary Cartesian products while balancing the workload in the distributed environment and avoiding the time-consuming sorting operation. Therefore, we can substantially boost the efficiency of the theta-join operation.

As illustrated in Figure 3, the bar chart presents the number of Cartesian products that are performed by each algorithm in the example in Figure 2. The rightmost red bar, as well as the red dotted line, indicate the number of theta-join results, that is, no matter how the theta-join algorithm is optimized, it is the lower bound of the number of Cartesian products that needs to be performed. The proposed *Prefap* algorithm achieves the lowest number of Cartesian products in this example and is very close to the optimal lower bound, which indicates that the research direction of the *Prefap* algorithm is promising.

3 | PREFAP FRAMEWORK

In this section, we will introduce the proposed *Prefap* approach in terms of its framework and workflow, followed by a detailed explanation of each constituting component, that is, the prefiltering strategy, the amalgamated partitioning scheme, as well as some auxiliary steps to complete the theta-join operation. Note that in this section and the corresponding pseudocode, we explain the theta-join operation on two data streams for the

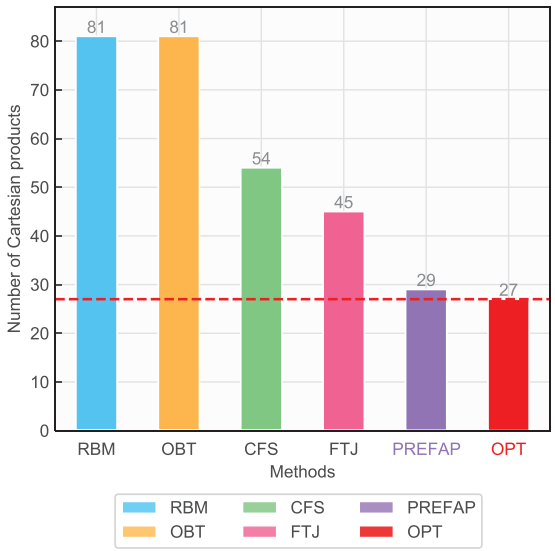

FIGURE 3 The bar chart presents the number of Cartesian products performed by each algorithm in the example in Figure 2. The rightmost red bar, as well as the red dotted line, indicate the number of theta-join results, that is, no matter how the theta-join algorithm is optimized, this is the lower bound of the number of Cartesian products that the algorithm needs to perform and the algorithm cannot perform better than this

TABLE 1 Interpretation of symbols and acronyms

Symbol / acronym	Interpretation
$\theta\,(\star)$	The θ operator, $\theta \in \{<, \leq, >, \geq\}$
$\mathcal{R}, \mathcal{S}, \mathcal{T}\,(\star)$	Data stream R, S, T
D	A variable which stands for data stream, $D \in \{\mathcal{R}, \mathcal{S}, \mathcal{T}, \ldots\}$
D_{min}	The minimum value of entire data stream D
sp_D	The span of each partition of data stream D
$p\,(\star)$	Number of partitions
PB_D	Partitioning boundary of data stream D before being amalgamated
APB	Amalgamated partitioning boundary
P_D	Partitions of data stream D
P_D^i	The ith partition of data stream D
rn_D^i	The repartitioning number of P_D^i
AS_D	The average partition size of data stream D
$sp_D^{i'}$	The repartitioning span of the ith partition of data stream D
$w\,(\star)$	The window size

Note: Input parameters are marked with $(\star)$.

sake of simplicity. However, it is not difficult to extend our approach to multiway theta-joins. The symbols and acronyms used and their corresponding interpretations are given in Table 1.

3.1 | Prefap workflow

The framework of the proposed *Prefap* is illustrated in Figure 4, together with its workflow as follows:

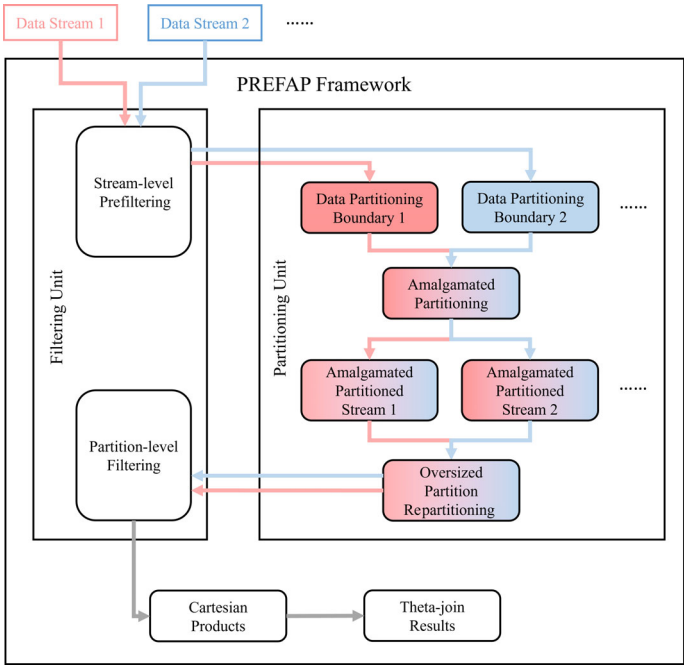

FIGURE 4 The *Prefap* framework. In the filtering unit of the *Prefap* framework, the data streams will firstly be prefiltered based on the theta condition to get rid of unnecessary data elements that are deemed not possible to form any valid theta-join results. Then, their corresponding partitioning boundaries will be calculated by the partitioning unit, followed by the amalgamated partitioning procedure, in which the partitioning boundaries are amalgamated. The resulted data streams are partitioned based on the amalgamated partitioning boundaries, and those oversized partitions are repartitioned to achieve load balancing in the distributed environment. After that, the partitions will be handled by the filtering Unit and the partition-level filtering is performed. Finally, Cartesian products are conducted and the theta-join results are retrieved based on the theta condition. Note that the diagram shows how two data streams are processed by the *Prefap* framework for the purpose of illustration only, the framework can be extended to work on multiway data stream theta-join as well. For three-way theta-join, two data streams will firstly be processed by the *Prefap* framework, the result produced will then be joined with the third data stream to yield the final result

Step 1–prefiltering strategy: Firstly, the proposed prefiltering strategy will be employed to filter out data elements in two data streams based on the theta condition, so that the workloads in later steps can be lowered and the partitioning will be more fine-grained (as in Section 3.2 and Algorithm 1). This step will be completed by the Filtering Unit of the *Prefap* framework.

Step 2–amalgamated partitioning mechanism: Then, it comes to the amalgamated partitioning mechanism. The partitioning boundaries of data streams are calculated, amalgamated, and the amalgamated partitioning boundaries will be used to produce fine-grained partitions to benefit the partition-level filtering (as in Section 3.3 and Algorithm 2). This step will be completed by the partitioning unit of the *Prefap* framework.

Step 3–auxiliary procedures: Finally, after completing the developed prefiltering strategy and the amalgamated partitioning mechanism, some auxiliary procedures adopted in FastThetaJoin are followed to complete the theta-join operation for the final output results:

1. *Step 3.1–oversized partition repartitioning*: To balance the workload among processes under the distributed environment, oversized partitions will be repartitioned to balance the workload as much as possible. This step will be completed by the Partitioning Unit of the *Prefap* framework.
2. *Step 3.2–partition-level filtering*: The partitions will then be filtered again based on the theta condition to avoid unnecessary Cartesian products. This step will be completed by the filtering unit of the *Prefap* framework.
3. *Step 3.3–cartesian products and theta-join results*: Finally, the Cartesian products are performed and the theta-join results are retrieved based on the theta condition. All auxiliary procedures will be presented in Section 3.4 and Algorithm 3.

3.2 | Prefiltering strategy

As opposed to all the aforementioned algorithms that directly perform the partitioning without any prefiltering, a prefiltering strategy is applied in the *Prefap* framework to eliminate certain amounts of unnecessary data involved in the operation. This is performed in the Partitioning Unit and the pseudocode of this step is given in Algorithm 1. Specifically, the prefiltering strategy scans and filters the two data

Algorithm 1. The *Prefap* algorithm—stream-level prefiltering strategy (Step 1)

Input:

θ operator, $\theta \in \{<, \leq, >, \geq\}$,

Data streams $\mathcal{R}$ and $\mathcal{S}$, attribute $\mathcal{R}$.A and $\mathcal{S}$.B

Output: Pre-filtered data streams at the stream-level.

1: **if** θ is "$>$" **then**
2: $\mathcal{R} \leftarrow \mathcal{R}$.remove($\leq \mathcal{S}_{min}$)
3: $\mathcal{S} \leftarrow \mathcal{S}$.remove($\geq \mathcal{R}_{max}$)
4: **else if** θ is "$\geq$" **then**
5: $\mathcal{R} \leftarrow \mathcal{R}$.remove($<\mathcal{S}_{min}$)
6: $\mathcal{S} \leftarrow \mathcal{S}$.remove($>\mathcal{R}_{max}$)
7: **else if** θ is "$<$" **then**
8: $\mathcal{R} \leftarrow \mathcal{R}$.remove($\geq \mathcal{S}_{max}$)
9: $\mathcal{S} \leftarrow \mathcal{S}$.remove($\leq \mathcal{R}_{min}$)
10: **else**
11: $\mathcal{R} \leftarrow \mathcal{R}$.remove($>\mathcal{S}_{max}$)
12: $\mathcal{S} \leftarrow \mathcal{S}$.remove($<\mathcal{R}_{min}$)
13: **end if**
14: **return** $\mathcal{R}, \mathcal{S}$

streams according to the theta condition to eliminate the data elements that are deemed impossible to produce valid theta-join results before the partitioning is performed. Also, unlike some previous methods, the proposed prefiltering strategy does not require data stream sorting, which severely hurts the efficiency of the theta-join operation. For example, as illustrated in Figure 2E, given the theta condition is "$>$", any value in data stream $\mathcal{R}$ that is less than or equal to the minimum value of data stream $\mathcal{S}$ is safe to be removed as it is not possible to form valid theta-join results with any value in data stream $\mathcal{S}$. The similar mechanism applies for data stream $\mathcal{S}$ as well. Therefore, we can safely eliminate any value in data stream $\mathcal{S}$ that is greater than or equal to the maximum value of data stream $\mathcal{R}$, as is shown in line 2–3 in Algorithm 1. The prefiltering mechanism works similarly for other theta conditions as presented in Algorithm 1. As indicated in Figure 2E, upon applying the prefiltering strategy, the first row and the last three columns will be directly filtered out and not involved in later processing steps. Hence the prefiltering strategy can greatly reduce the amount of data that needs to be processed and improve the efficiency of the theta-join operation.

Meanwhile, since the partitioning boundaries being used in the subsequent processing are calculated based on the minimum and maximum values of data streams, the use of the prefiltering strategy to filter useless data is likely to reduce the span between the maximum and the minimum values, and therefore making the partitioning more fine-grained and benefiting the partition-level filtering performed later.

3.3 | Amalgamated partitioning mechanism

The data streams are partitioned in the Partitioning Unit based on the range defined by the partitioning boundaries in the course of the theta-join operation as shown in line 3 in Algorithm 2. The partitioning boundaries of data streams will be calculated after the prefiltering strategy is executed with each partition having a span, which is denoted as *sp* and is calculated as follows:

$$sp_D = \frac{D_{max} - D_{min}}{p}, D \in \{\mathcal{R}, \mathcal{S}\}, \tag{3}$$

where p denotes the number of partitions.

As such, in the example as shown in Figure 2E, given that data streams are partitioned into three partitions, data stream $\mathcal{R}$ has a partition span equal to $\frac{8}{3}$ and will be partitioned into the following three partitions: `[1,3.67)`, `[3.67,6.33)` and `[6.33,9]`, and the same partitioning boundary calculations can also be applied to data stream $\mathcal{S}$ as indicated in lines 2–3 in Algorithm 2.

For all aforementioned algorithms, two data streams are partitioned separately based on their respective partitioning boundaries after they are obtained. As such there is no interference between the partitions of each stream, which implies the partitioning is accomplished in an isolated manner. Clearly, the isolated partitioning lacks the notion of collaborative partitioning information of the data streams, and thus damages the efficacy

Algorithm 2. The *Prefap* algorithm—amalgamated partitioning mechanism (Step 2)

Input:

θ operator, $\theta \in \{<, \leq, >, \geq\}$,
number of partitions p,
Data streams D after stream-level pre-filtering, $D \in \{\mathcal{R}, \mathcal{S}\}$

Output: Amalgamated partitioned data streams

1: // Amalgamated Partitioning Mechanism
2: Calculate sp_D based on Equation (3)
3: $PB_D \leftarrow [\, D_{min}, D_{min} + 1 \times sp_D\,)$,
$[\, D_{min} + 1 \times sp_D, D_{min} + 2 \times sp_D\,)$,
$\vdots$
$[\, D_{min} + (p-1) \times sp_D, D_{max}\,]$
4: // Generate amalgamated partitioning boundary
5: $APB \leftarrow PB_{\mathcal{R}}$.append($PB_{\mathcal{S}}$).sort()
6: // Partition data streams using the amalgamated partitioning boundary
7: $P_D \leftarrow$ Partition D based on APB
8: $P_D \leftarrow P_D$.filter(P_D^i.size() != 0)
9: **return** $P_D, D \in \{\mathcal{R}, \mathcal{S}\}$

of the partition-level filtering. Take Figure 2E as an example, for the coarse-grained partitioning, partition `[1,3.67)` of data stream $\mathcal{R}$ and partition `[2.67, 5.33)` of data stream $\mathcal{S}$ will be produced separately and the Cartesian products are performed between them under the ">" theta condition since valid theta-join results can be available between these two partitions. However, not all Cartesian products between these two partitions are necessary, say, the Cartesian products between `[1,2.67]` and `[3.67,5.33)` are completely useless as they are judged to be impossible to possess any valid theta-join results based on the ">" theta condition. Therefore, the coarse partitions produced by the coarse-grained isolated partitioning strategy will impair the efficacy of the partition-level filtering, which could result in more Cartesian products than necessary, and thus seriously hinder the efficiency of the theta-join operation.

Given the drawback caused by this isolated coarse-grained partitioning strategy, we consider an amalgamated partitioning mechanism as shown in Algorithm 2, which is useful to address this issue. To make the partitioning more fine-grained, after calculating the partitioning boundaries, we amalgamate the partitioning boundaries of data streams to form the amalgamated partitioning boundaries as presented in line 5 in Algorithm 2. By fusing the partitioning information of data streams, the aforementioned drawback can be circumvented. Therefore, the effectiveness of the partition-level filtering is improved, which would lead to the reduction of the number of Cartesian products to be conducted.

In the example of Figure 2E, given the partitioning boundaries of data stream $\mathcal{R}$ are `[1,3.67)`, `[3.67,6.33)` and `[6.33,9]`, and the partitioning boundaries of data stream $\mathcal{S}$ are `[0,2.67)`, `[2.67,5.33)` and `[5.33,8]`, lines 4–5 of Algorithm 2 will amalgamate these two partitioning boundaries together and the amalgamated partitioning boundaries produced in this case would be `[0,1)`, `[1,2.67)`, `[2.67,3.67)`, `[3.67,5.33)`, `[5.33,6.33)`, `[6.33,8)`, and `[8,9]`. After applying the amalgamated partitioning scheme to both data streams, the Cartesian products in the above case between `[2.67,3.67)` from data stream $\mathcal{R}$ and `[2.67,3.67)` from data stream $\mathcal{S}$ will be conducted as usual, while the unnecessary Cartesian products between `[1,2.67]` from data stream $\mathcal{R}$ and `[3.67,5.33)` from data stream $\mathcal{S}$ will be avoided by the partition-level filtering thanks to the fine-grained amalgamated partitioning strategy. This significantly decreases the number of useless Cartesian products, and hence benefits the efficiency of the algorithm.

3.4 | Auxiliary procedures

To obtain the theta-join results, some auxiliary procedures as shown in Algorithm 3 that are adopted in the FastThetaJoin algorithm are exploited to follow up the proposed approaches in the *Prefap* framework, which are detailed as follows:

(1) Oversized partition repartitioning: By following the common design scheme in Reference 35, we design the framework that can repartition any oversized partitions to balance the workload as much as possible in the distributed environment as shown in lines 2–9 in Algorithm 3. Specifically, any partition with its size larger than the average partition size is regarded as an oversized partition. Once a partition is judged as an oversized partition, it will be repartitioned into a number of subpartitions defined as:

$$rn_D^i = \left\lceil \frac{P_D^i.size()}{AS_D} \right\rceil = \left\lceil \frac{P_D^i.size()}{\frac{w}{P_D.size()}} \right\rceil, D \in \{\mathcal{R}, \mathcal{S}\} \tag{4}$$

Algorithm 3. The *Prefap* algorithm–auxiliary procedures (Step 3)

Input:

θ operator, $\theta \in \{<, \leq, >, \geq\}$,
window size w,
Amalgamated partitioned data stream P_D, $D \in \{\mathcal{R}, \mathcal{S}\}$

Output: θ-join results of two input data streams.

1: // **Step 3.1: Oversized Partition Re-partitioning**
2: $AS_D \leftarrow \frac{w}{P_D.size()}$
3: **for** each P_D^i of D **do**
4: **if** P_D^i.size() > AS_D **then**
5: $sp_D^{i'} \leftarrow \lceil \frac{P_{D}{}^{i}_{max} - P_{D}{}^{i}_{min}}{rn_D^i} \rceil$
6: Repartition P_D^i as follows
$[\, P^i_{Dmin}, P^i_{Dmin} + 1 \times sp_D^{i'})$,
$[\, P^i_{Dmin} + 1 \times sp_D^{i'}, P^i_{Dmin} + 2 \times sp_D^{i'})$,
$\vdots$
$[\, P^i_{Dmin} + (rn_D^i - 1) \times sp_D^{i'}, P^i_{Dmax} \,]$
7: **end if**
8: **end for**
9: $P_D \leftarrow P_D$.filter(P_D^i.size() != 0)
10: // **Step 3.2: Partition-level Filtering**
11: **for** each $P^i_{\mathcal{R}}$ of $P_{\mathcal{R}}$ **do**
12: **for** each $P^j_{\mathcal{S}}$ of $P_{\mathcal{S}}$ **do**
13: **if** θ is ">" and $P^i_{\mathcal{R}max} > P^j_{\mathcal{S}min}$ **then**
14: Distribute Cartesian_product($P^i_{\mathcal{R}}$, $P^j_{\mathcal{S}}$) to processors
15: **else if** θ is "≥" and $P^i_{\mathcal{R}max} \geq P^j_{\mathcal{S}min}$ **then**
16: Distribute Cartesian_product($P^i_{\mathcal{R}}$, $P^j_{\mathcal{S}}$) to processors
17: **else if** θ is "<" and $P^i_{\mathcal{R}min} < P^j_{\mathcal{S}max}$ **then**
18: Distribute Cartesian_product($P^i_{\mathcal{R}}$, $P^j_{\mathcal{S}}$) to processors
19: **else if** θ is "≤" and $P^i_{\mathcal{R}min} \leq P^j_{\mathcal{S}max}$ **then**
20: Distribute Cartesian_product($P^i_{\mathcal{R}}$, $P^j_{\mathcal{S}}$) to processors
21: **end if**
22: **end for**
23: **end for**
24: // **Step 3.3: Cartesian Products and Theta-join Results**
25: **for** all Cartesian products generated **do**
26: Only keep those that satisfy the θ condition
27: **end for**
28: **return** θ-join results

and the pseudocode of this process is shown in lines 3–8 in Algorithm 3. The oversized partition repartitioning is completed by the partitioning unit of the *Prefap* framework. Together with the fine-grained amalgamated partitioning, the load balancing of the framework can be improved.

(2) Partition-level filtering: Once both data streams are partitioned, the partitioning unit of the *Prefap* framework will filter the partitions based on the theta condition as is presented in lines 11–23 in Algorithm 3, so that the Cartesian products between the partitions that possess no valid theta-join results will not be performed.

In the example illustrated in Figure 2E, even though partition `[1,2.67)` from data stream $\mathcal{R}$ and partition `[3.67,5.33)` from data stream $\mathcal{S}$ are both produced, the Cartesian products between these two partitions will not be performed under the ">" theta condition, because even the maximum value in the former partition is smaller than the minimum value in the latter partition, which is exactly lines 13–14 in Algorithm 3. Hence, by applying the partition-level filtering based on the theta condition, unnecessary Cartesian products are eliminated, leading to a more efficient algorithm.

(3) Cartesian products and theta-join results: After completing the prefiltering, amalgamated partitioning, oversized partition repartitioning, and the partition-level filtering, the remaining Cartesian products are highly refined, and then the Cartesian products will be distributed to different processes to output the final theta-join results based on the theta condition as shown in line 24–28 in Algorithm 3.

4 | EMPIRICAL STUDIES

To validate the effectiveness of the *Prefap* algorithm, comprehensive evaluations are performed on both synthetic and real data streams from two-way to multiway theta-join operations. We compare our algorithm against several theta-join algorithms, including the state-of-the-art algorithm FastThetaJoin (FTJ)[35] and several well-known and widely used algorithms, such as range-based method (RBM),[32] one-bucket theta (OBT),[33] and cross filter strategy (CFS).[34]

4.1 | Experimental setup and data streams

We follow the *Prefap* workflow described in Section 3.1 to conduct the experiments where the number of partitions p is set to be 10 and the window size w to be 1000. The evaluation metrics we use to evaluate the performance of the algorithms are listed in Table 2 with their corresponding interpretations. To fully testify the effectiveness of the proposed algorithm, the theta-join on two-way data streams and multiway data streams are performed to validate that the proposed algorithm scales well. The hypothesis testings are then conducted to demonstrate that the *Prefap* algorithm achieves a statistically significant performance improvement compared with FTJ. The ablation studies are also conducted to show the efficacy of the algorithm and reveal the importance and necessity of each component introduced in the *Prefap* framework. Finally, the number of partitions and the window size are adjusted to show the superior performance of the *Prefap* algorithm in various settings.

We use both synthetic and real data streams to testify the effectiveness of the proposed algorithm. For the synthetic datasets, randomly generated *uniform* and *normal* data streams are exploited to represent some typical nonskewed data streams, while the randomly generated *zipf* data stream acts as a representative of a typical skewed data stream. As for the real datasets, by following the method in Reference 34, we use the Clouds dataset provided by the U.S. Department of Energy in our experiments where the real-time wind speed measured in meter per second (m/s) of different months in 2000 are leveraged to form different data streams. Additionally, the stock market price data streams provided by Yahoo! Finance[38] are also utilized, in which the real-time high price of stocks of different companies between 2010 and 2020 are used to form different data streams. The detailed parameter settings of different synthetic data streams will be given when they are used in the following subsections.

In terms of the experimental infrastructure, a server equipped with Intel Core i7 7600U CPU and 32GB of memory is utilized. The CPU has two cores with four processors, so that the Cartesian products involved in the *Prefap* framework can be computed in a distributed manner.

4.2 | 2-way data stream theta-join

The performances of theta-join algorithms for two-way uniform data streams, zipf data streams, clouds data streams, and stock price data streams are illustrated in Figures 5–8, respectively. Note that the detailed data stream and distribution configurations (including stream size, distribution parameters, join attribute used, θ condition) have been given in the corresponding image captions.

The results show that, in all cases, even for the highly skewed data streams such as the zipf data streams, the *Prefap* algorithm performs better than the state-of-the-art method FTJ in terms of efficiency, and significantly outperforms all other algorithms. More specifically, the *Prefap* algorithm

TABLE 2 Theta-join performance evaluation metrics and their corresponding interpretation

Metric	Interpretation
Number of Cartesian products	The number of Cartesian products that the algorithm performs
Elapsed time	The total time elapsed, measured in milliseconds (ms)
Load balancing ratio (In)	The maximum input load among processes divided by the average input load among processes
Load balancing ratio (Out)	The maximum number of theta-join results among processes divided by the average number of theta-join results among processes

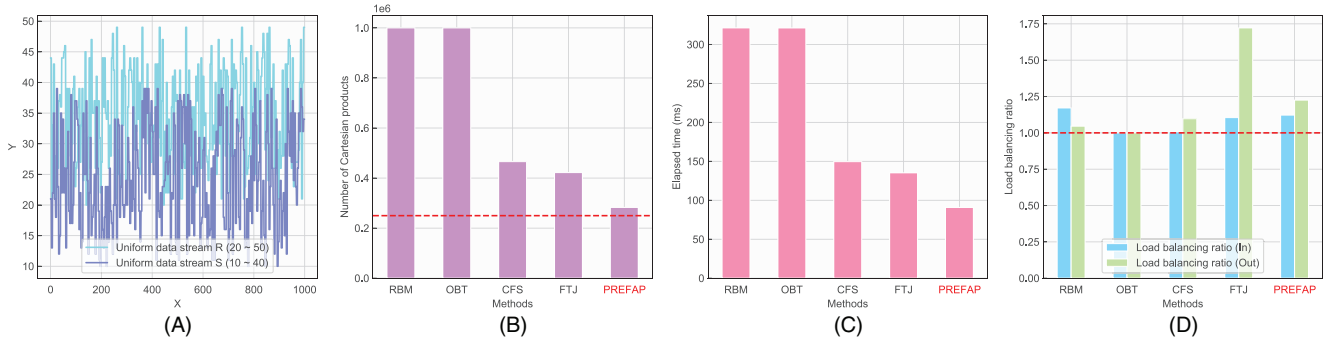

FIGURE 5 The performance comparisons between algorithms when performing theta-join on synthetic 2-way Uniform data streams. The theta condition is $\mathcal{R} \leq \mathcal{S}$. The stream size of both data streams are 1000. The uniform data stream $\mathcal{R}$ fluctuates in range [20, 50], while the Uniform data stream $\mathcal{S}$ is in range [10, 40]. (A) depicts the 2-way data streams, (B–D) present the number of Cartesian products, elapsed time and the in/out load balancing ratio, respectively. The red dotted line in (B) indicates the number of theta-join results, that is, the minimum number of Cartesian products that need to be performed. The red dotted line in (D) marks 1.0, which indicates perfect load balancing

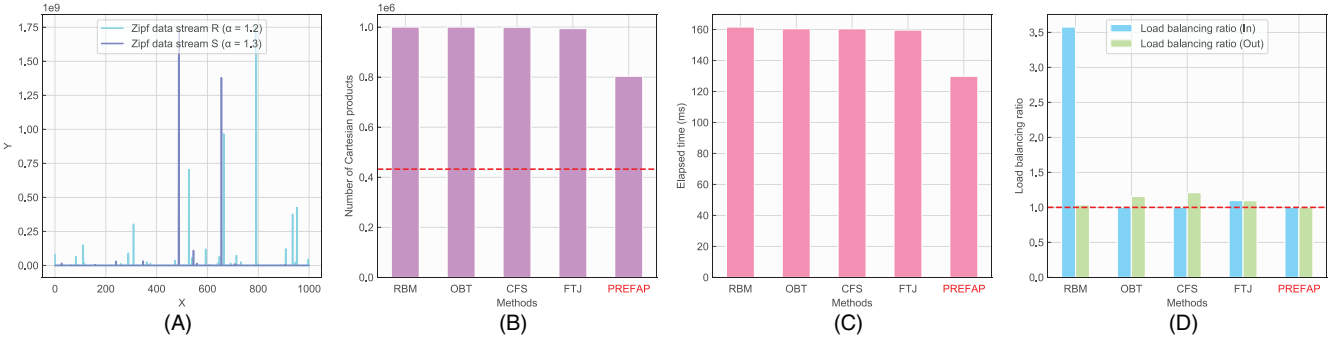

FIGURE 6 The performance comparisons between algorithms when performing theta-join on synthetic 2-way Zipf data streams. The theta condition is $\mathcal{R} \leq \mathcal{S}$. The stream size of both data streams are 1000. The shape parameter α of Zipf data stream $\mathcal{R}$ and $\mathcal{S}$ are set to be 1.2 and 1.3, respectively. (A) depicts the 2-way data streams, (B–D) present the number of Cartesian products, elapsed time and the in/out load balancing ratio, respectively. The red dotted line in (B) indicates the number of theta-join results, that is, the minimum number of Cartesian products that need to be performed. The red dotted line in (D) marks 1.0, which indicates perfect load balancing

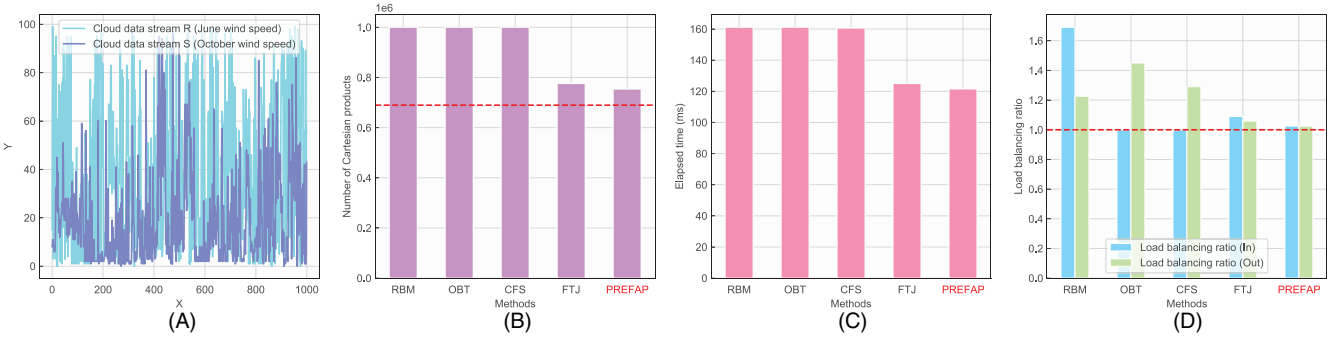

FIGURE 7 The performance comparisons between algorithms when performing theta-join on real 2-way Clouds data streams. The theta condition is $\mathcal{R} \geq \mathcal{S}$. The stream size of both data streams are 1000. The Clouds data stream $\mathcal{R}$ and $\mathcal{S}$ are the real-time wind speed captured in every 5 s in June 2000 and October 2000, respectively. Note that these two months are randomly selected as representatives. (A) depicts the 2-way data streams, (B–D) present the number of Cartesian products, elapsed time and the in/out load balancing ratio, respectively. The red dotted line in (B) indicates the number of theta-join results, that is, the minimum number of Cartesian products that need to be performed. The red dotted line in (D) marks 1.0, which indicates perfect load balancing

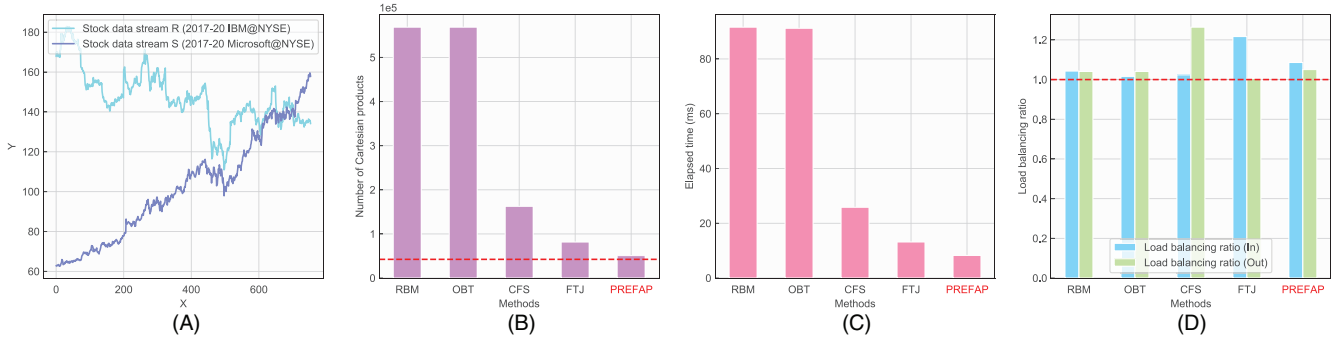

FIGURE 8 The performance comparisons between algorithms when performing theta-join on real 2-way stock data streams. The theta condition is $\mathcal{R} \leq \mathcal{S}$. The stream size of both data streams are 755. The Stock data stream $\mathcal{R}$ and $\mathcal{S}$ are the New York stock exchange (NYSE) stock high price recorded daily between Jan 3, 2017 and Jan 2, 2020 of company IBM and Microsoft, respectively. Note that there are only 755 stock exchange open days in this period, and these two companies are randomly selected as representatives. (A) depicts the 2-way data streams, (B–D) present the number of Cartesian products, elapsed time and the in/out load balancing ratio, respectively. The red dotted line in (B) indicates the number of theta-join results, that is, the minimum number of Cartesian products that need to be performed. The red dotted line in (D) marks 1.0, which indicates perfect load balancing

achieves 34.7%, 19.4%, 3.0%, and 37.7% reductions on the number of performed Cartesian products compared with FTJ with respect to four different kinds of data streams, respectively. The significant reduction of the number of Cartesian products yields better efficiency, as indicated by the shortest elapsed time performance in all cases.

The red dotted lines in Figures 5B–8B indicate the number of theta-join results, that is, no matter how the algorithm is optimized, it must perform at least this number of Cartesian products to yield the complete result. As shown in the results, the *Prefap* algorithm significantly minimizes the gap between the number of Cartesian products it performs and the optimal case. Compared with FTJ, it clearly indicates an effective performance boost and thus demonstrates that the cooperation between the prefiltering strategy and the amalgamated partitioning mechanism contributes positively toward reducing the redundancy in Cartesian products, which in turn significantly boosts the performance.

In terms of the load balancing as illustrated in Figures 5–8, compared with the FTJ, the *Prefap* algorithm performs better for both input and output load balancing ratios, indicated by the red dotted line in subfigure (D). Hence, it justifies that with the collaboration between the fine-grained amalgamated partitioning scheme and the oversized partition repartitioning mechanism, the *Prefap* algorithm performs well in the distributed environment by distributing workloads in a relatively even way.

4.3 | Multiway data stream theta-join

To further verify that the *Prefap* algorithm scales well when processing multiway data streams, comprehensive experiments are conducted on both synthetic and real multiway data streams. As shown in Figure 9A, the uniform multiway data streams represent a typical example of the nonskewed

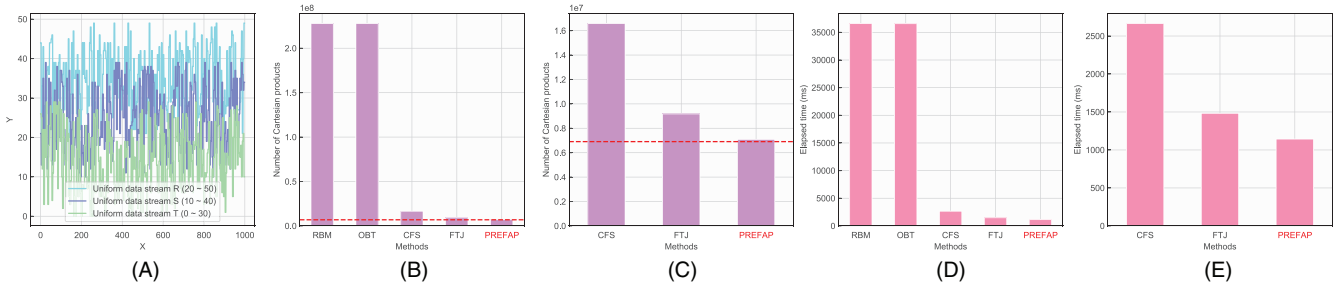

FIGURE 9 The performance comparisons between algorithms when performing theta-join on synthetic 3-way Uniform data streams. The theta condition is $\mathcal{R} < \mathcal{S} \leq \mathcal{T}$. The stream size of all three data streams are 1000. The Uniform data stream $\mathcal{R}$, $\mathcal{S}$ and $\mathcal{T}$ fluctuate in range [20, 50], [10, 40] and [0, 30], respectively. (A) depicts the 3-way data streams, (B,D) present the number of Cartesian products and the elapsed time, respectively. To clearly show the performance gain, (C,E) are the zoom-in version of (B,D), respectively. Both RBM and OBT are omitted due to their worst performances. The red dotted lines in (B,C) indicate the number of theta-join results, that is, the minimum number of Cartesian products that need to be performed

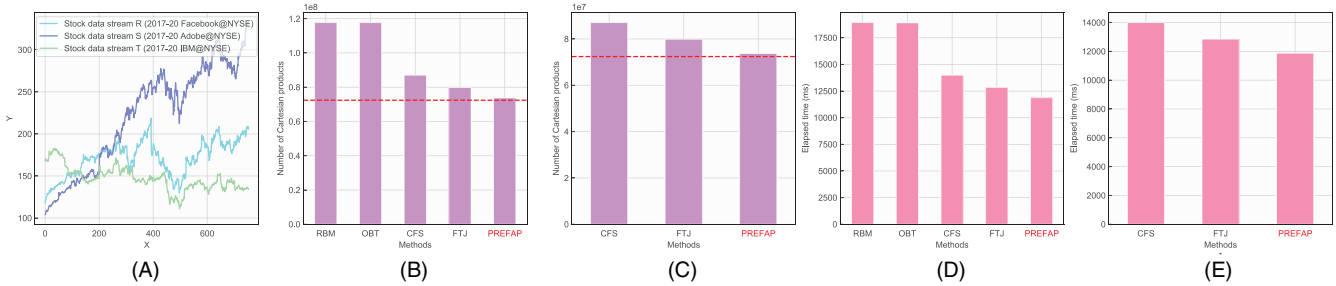

FIGURE 10 The performance comparisons between algorithms when performing theta-join on real 3-way stock data streams. The theta condition is $\mathcal{R} \geq \mathcal{S} < \mathcal{T}$. The stream size of all three data streams are 755. The stock data stream $\mathcal{R}$, $\mathcal{S}$ and $\mathcal{T}$ are the New York Stock Exchange (NYSE) stock high price recorded daily between Jan 3, 2017 and Jan 2, 2020 of company Facebook, Adobe, and IBM, respectively. Note that there are only 755 stock exchange open days in this period, and these three companies are randomly selected as representatives. (A) depicts the 3-way data streams, (B,D) present the number of Cartesian products and the elapsed time, respectively. To clearly show the performance gain, (C,E) are the zoom-in version of (B,D), respectively. Both RBM and OBT are omitted due to their worst performances. The red dotted lines in (B,C) indicate the number of theta-join results, that is, the minimum number of Cartesian products that need to be performed

data streams. While as illustrated in Figure 10A, the New York Stock Exchange (NYSE) stock prices of three randomly selected companies between 2017 and 2020 are skewed and are therefore capable of fully testifying the effectiveness of the proposed algorithm. Note that the detailed data stream configurations have been given in the corresponding image captions.

In Figures 9 and 10, subfigure (B) and (D) present the evaluation results of the number of Cartesian products and the elapsed time, respectively. As the gaps between the *Prefap* algorithms and other algorithms are so significant, the subfigure (B) and (D) have been zoomed in as shown in subfigure (C) and (E), respectively. As we can see, under both synthetic and real multiway data streams, the *Prefap* algorithm outperforms the FTJ algorithm by a large margin, achieving a 24.1% and a 7.7% decrease in the number of Cartesian products in two cases, respectively. Hence, it is also natural to observe that the elapsed time in both cases are reduced accordingly, hence the algorithm becomes more efficient. Furthermore, the *Prefap* algorithm attains a near-optimal performance. Given the number of theta-join results, that is, the minimum number of Cartesian products required to be performed that is indicated by the red dotted line in subfigure (B) and (C) in Figures 9 and 10, the *Prefap* algorithm only performs 0.026% and 1.7% more Cartesian products compared with the optimal scenario, indicating excellent theta-join performance.

Therefore, the superior performance of the *Prefap* algorithm in various cases strongly testifies that the collaboration of the prefiltering strategy and the amalgamated partitioning mechanism is effective in improving the theta-join performance.

4.4 | Significance Tests

To further validate that the improvement achieved by *Prefap* over the state-of-the-art FTJ algorithm is statistically significant, three representative data stream settings, that is, theta-join between 2-way zipf data streams, 2-way cloud data streams, and multiway stock price data streams are performed 30 times and the *T-tests* are employed to verify the significance of the performance boost using 0.05 as the significance threshold. Note that the detailed data stream configuration has been given in the image caption.

As shown in Figure 11, four bars I, II, III, IV in subfigure (A) and (B) indicate the $-\log(p\_value)$ of the one-sided T-test on four different evaluation metrics as presented in Table 2 for two 2-way theta-join settings. The $-\log(p\_value)$ results of the multiway theta-join are presented in subfigure (C), in which the bars correspond to the $-\log(p\_value)$ results of the differences between *Prefap* and FTJ on the number of Cartesian products and the elapsed time, respectively. In Figure 11, the red dotted lines indicate $-\log(0.05)$, which is the significance threshold of the one-sided *T*-test. If the bar is higher than the red dotted line, it indicates the acceptance of the alternate hypothesis, which is $\text{metric}_{Prefap} < \text{metric}_{FTJ}$. And the higher the bar is, the more significant the performance improvement produced by the *Prefap* algorithm is.

From Figure 11, we can clearly observe that all bars in all cases are significantly higher than the red dotted line, which indicates that in all data stream settings, the *Prefap* algorithm achieves a lower evaluation metric value than the FTJ algorithm, that is, less number of Cartesian products are performed, less amount of elapsed time is required, and the load balancing ratio of both input and output is closer to one. The heights of the bars are much higher than the red dotted line, which indicate superior performances with high statistical confidence. Hence, statistically, the *T*-test results verify the superiority of the *Prefap* algorithm, and the performance enhancement is statistically significant. The statistically testifiable superior performance also demonstrates that the *Prefap* framework can collaboratively contribute toward better theta-join efficiency and excellent load balancing in the distributed environment, the benefit brings by the *Prefap* framework is statistically significant.

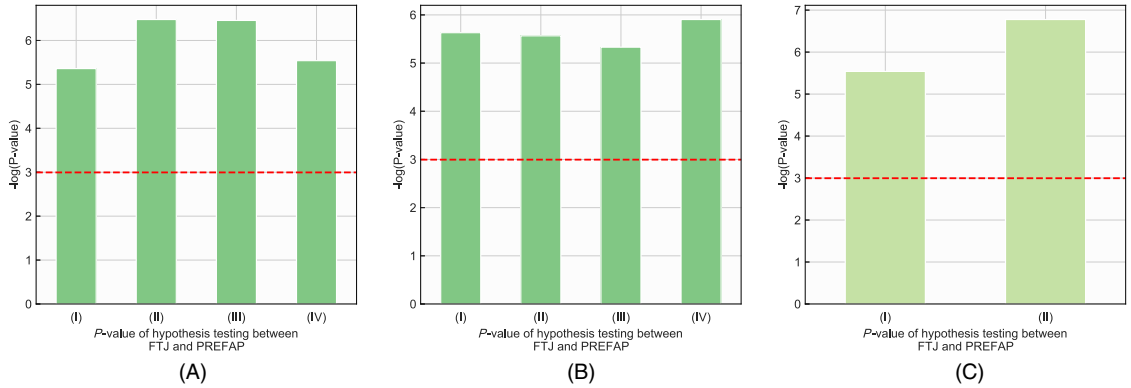

FIGURE 11 The one-sided significance T-tests with 0.05 as the significance level are conducted under three tasks to testify the performance gains of the *Prefap* algorithm compared with the best compared method FastThetaJoin (FTJ). The y-axis denotes the $-log(p\_value)$, and the red dotted lines in all three subplots mark the significance threshold, that is, $-log(0.05)$. (a), (b), and (c) are for theta-join on synthetic 2-way Zipf data streams (stream size = 1000, $\alpha = 1.2, 1.3, \theta : \mathcal{R} \leq \mathcal{S}$), real 2-way clouds data streams (stream size = 1000, $\theta : \mathcal{R} \geq \mathcal{S}$, data stream $\mathcal{R}$ and $\mathcal{S}$ are the real-time wind speed captured in every 5 s in June 2000 and October 2000, respectively.), and real 3-way Stock data streams (stream size = 755, $\theta : \mathcal{R} \geq \mathcal{S} < \mathcal{T}$, data stream $\mathcal{R}$, $\mathcal{S}$ and $\mathcal{T}$ are the New York Stock Exchange (NYSE) stock high price recorded daily between Jan 3, 2017 and Jan 2, 2020 of company Facebook, Adobe and IBM, respectively). (I), (II), (III), and (IV) indicates the significance of the number of Cartesian products, elapsed time, load balancing ratio (in) and load balancing ratio (out). If the bar is higher than the red dotted line, it indicates that the alternate hypothesis is accepted, that is, the *Prefap* algorithm outperforms the FastThetaJoin algorithm on that metric. The higher the bar is, the more significant the performance improvement produced by the *Prefap* algorithm is

4.5 | Ablation study

The ablation study is conducted by evaluating several variants of the *Prefap* framework: (1) *Prefap* with the prefiltering being removed; (2) *Prefap* with the amalgamated partitioning being turned off; and (3) *Prefap* with both the prefiltering and the amalgamated partitioning being ablated.

Two-way and multiway stock price data streams of randomly selected companies are used as the representative tasks on which the theta-join is performed by using both the full *Prefap* algorithm and its ablated variants. The results are presented in Table 3 and 4 for two-way and multiway theta-join tasks, respectively. Note that the detailed data stream configuration is also provided in table captions. We can observe that the full *Prefap* algorithm outperforms all its variants by a large margin, which indicates that any one of these components in the *Prefap* framework plays an indispensable role and brings benefits to reduce the number of Cartesian products being performed and enhance the overall efficiency. Among all these components, the amalgamated partitioning scheme brings the highest amount of Cartesian product reductions in two tasks, which is 16.8% and 6.3%, respectively. Correspondingly, the elapsed time drops by 17.5% and 6.2%, respectively. This further validates the importance of amalgamating the partitioning scheme and the benefits of avoiding coarse-grained isolated partitioning. As indicated in Tables 3 and 4, although the prefiltering strategy attains relatively less amount of performance gain, which is 1.0% and 0.2% in the two-way and multiway task, respectively, however, the collaboration of both the prefiltering and the amalgamated partitioning yields a tremendous decrease of Cartesian products of 29.1% and 12.9%, respectively, for these two tasks. Therefore, by collaborating the prefiltering strategy with the amalgamated partitioning mechanism in the *Prefap* framework, it is highly effective in improving the efficiency of the theta-join operation. As such, the promising results successfully verify the effectiveness of the *Prefap* algorithm.

TABLE 3 Ablation study of the *Prefap* algorithm performed on real 2-way stock data streams ($\theta : \mathcal{R} \leq \mathcal{S}$, $\mathcal{R}$: 2017–2020 Johnson&Johnson@NYSE, $\mathcal{S}$: 2017–2020 Microsoft@NYSE, stream size = 755, joined attribute: stock high price, symbol ✗ represents "remove")

Setting	Number of Cartesian products	Elapsed time (ms)
Full version	104,138	33.68
✗ Prefiltering	105,132 (1.0% ↑)	33.89 (0.6% ↑)
✗ Amalgamated partitioning	121,647 (16.8% ↑)	39.58 (17.5% ↑)
✗ Prefiltering & ✗ Amalgamated partitioning	134,449 (29.1% ↑)	43.03 (27.8% ↑)
# of θ-join results (best performance possible)	92,240	––

TABLE 4 Ablation study of the *Prefap* algorithm performed on real 3-way Stock data streams ($\theta : \mathcal{R} \geq \mathcal{S} < \mathcal{T}$, $\mathcal{R}$: 2017–2020 Facebook@NYSE, $\mathcal{S}$: 2017–2020 Adobe@NYSE, $\mathcal{T}$: 2017–2020 IBM@NYSE, stream size = 755, joined attribute: stock high price, symbol ✗ represents "remove")

Setting	Number of Cartesian products	Elapsed time (ms)
Full version	74,923,241	12,087.37
✗ Prefiltering	75,093,479 (0.2% ↑)	12110.81 (0.2% ↑)
✗ Amalgamated partitioning	79,619,026 (6.3% ↑)	12838.10 (6.2% ↑)
✗ Prefiltering & ✗ Amalgamated partitioning	84,558,245 (12.9% ↑)	13,628.14 (12.7% ↑)
# of θ-join results (best performance possible)	72,510,414	––

4.6 | Algorithm efficiency

Number of partitions: To verify the effectiveness of the *Prefap* algorithm when working with different number of partitions, the performance of the *Prefap* algorithm with different number of partitions when processing two randomly selected tasks are presented in Figures 12 and 13. The detailed data stream configuration has been given in the corresponding image caption. For both tasks, the number of Cartesian products decreases with the increase of the number of partitions, thanks to the more fine-grained partitioning effect brought by a larger number of partitions, as it benefits the filtering strategy and makes it more effective. In all partition settings, the *Prefap* algorithm outperforms the FTJ algorithm by significantly reducing the number of Cartesian products being performed, and the elapsed time is reduced accordingly. This further demonstrates the superiority of the *Prefap* algorithm with different number of partitions.

Also, when the number of partitions is raised, the input and output load balancing ratio are improved marginally and are relatively close to 1, which indicates a relatively even workload distribution in the distributed environment achieved by the *Prefap* algorithm.

Window sizes: The performance of the *Prefap* is also evaluated when the window size is varied. The performance on two randomly selected tasks, that is, 2-way uniform data streams and 2-way stock price data streams, are shown in Figures 14 and 15, respectively. The detailed data stream configuration has been given in the corresponding image caption.

According to the experimental results, the number of Cartesian products being conducted and the elapsed time grow relatively proportionally with the increase of the window size, which demonstrates that the *Prefap* algorithm scales well in terms of window size. In all window size settings, the *Prefap* algorithm attains the lowest number of Cartesian products compared with its counterparts, and is relatively close to the optimal case marked by the red dotted line in subfigure (A). The excellent performance of Cartesian product reduction benefits the efficiency of the algorithm, as indicated by the lowest elapsed time achieved by the *Prefap* algorithm.

Meanwhile, from subfigure (C) and (D) of Figures 14 and 15, the *Prefap* algorithm attains a superior performance, compared with the FTJ algorithm, on both the input and the output load balancing ratio in all window size settings. Hence, it further validates the excellent scalability of the *Prefap* algorithm in terms of window size.

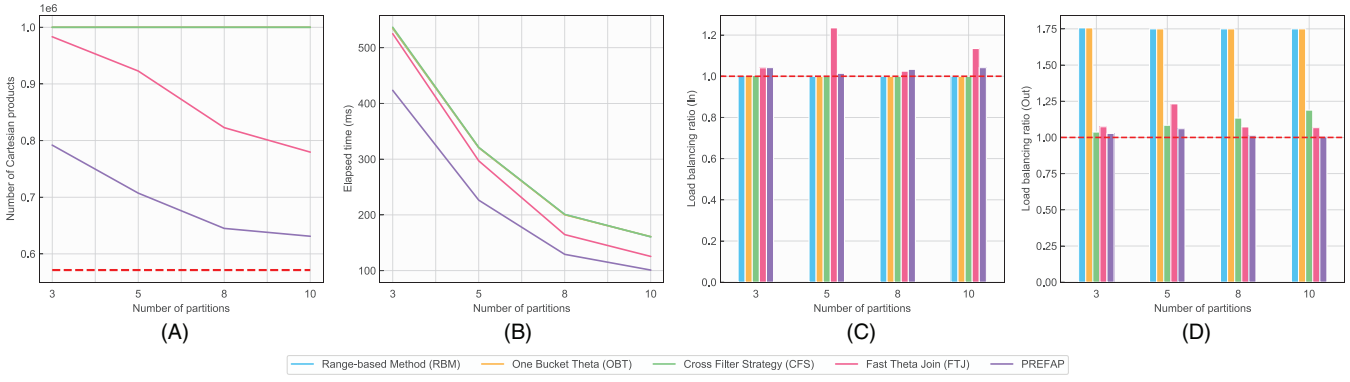

FIGURE 12 The performance comparisons between algorithms when performing theta-join on synthetic 2-way normal data streams ($\mu = 1.2, 1, \sigma = 1, 1, \theta : \mathcal{R} > \mathcal{S}$, stream size = 1000) under different number of partitions (3, 5, 8, and 10). (A–D) present the number of Cartesian products, elapsed time, load balancing ratio (in) and load balancing ratio (out), respectively. The red dotted line in (A) indicates the number of theta-join results, that is, the minimum number of Cartesian products that need to be performed. The red dotted lines in (C,D) mark 1.0, which indicates perfect load balancing

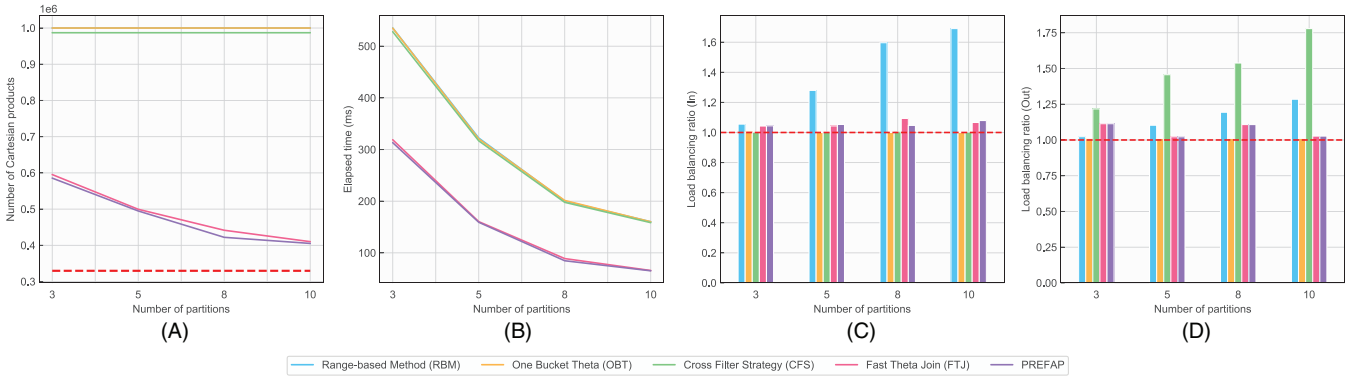

FIGURE 13 The performance comparisons between algorithms when performing theta-join on real 2-way clouds data streams ($\theta : \mathcal{R} \leq \mathcal{S}$, stream size = 1000. The Clouds data stream $\mathcal{R}$ and $\mathcal{S}$ are the real-time wind speed captured in every 5 s in June 2000 and October 2000, respectively.) under different number of partitions (3, 5, 8, and 10). (A–D) present the number of Cartesian products, elapsed time, load balancing ratio (in) and load balancing ratio (out), respectively. The red dotted line in (A) indicates the number of theta-join results, that is, the minimum number of Cartesian products that need to be performed. The red dotted lines in (C,D) mark 1.0, which indicates perfect load balancing

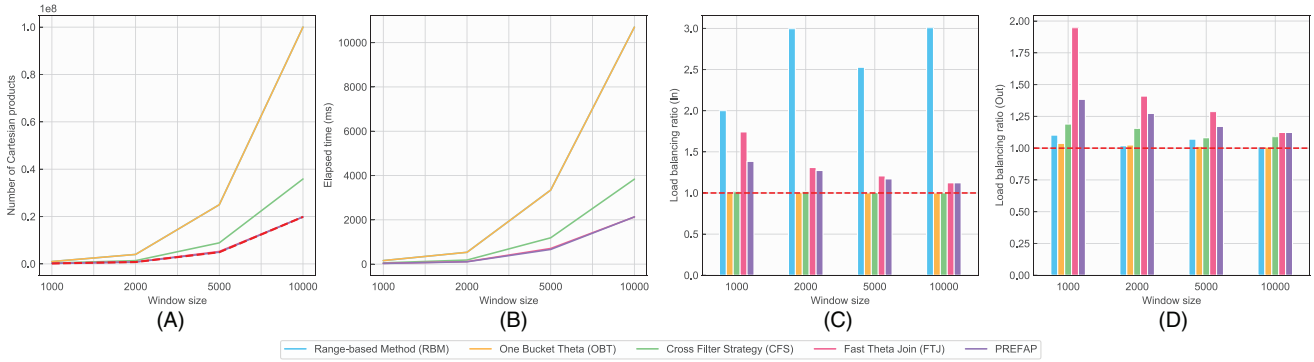

FIGURE 14 The performance comparisons between algorithms when performing theta-join on synthetic 2-way uniform data streams (*range* $\in [0, 15], [5, 20], \theta : \mathcal{R} > \mathcal{S}$) in different window sizes (1000, 2000, 5000, and 10,000). (A–D) present the number of Cartesian products, elapsed time, load balancing ratio (in) and load balancing ratio (out), respectively. The red dotted line in (A) indicates the number of theta-join results, that is, the minimum number of Cartesian products that need to be performed. The red dotted lines in (C,D) mark 1.0, which indicates perfect load balancing

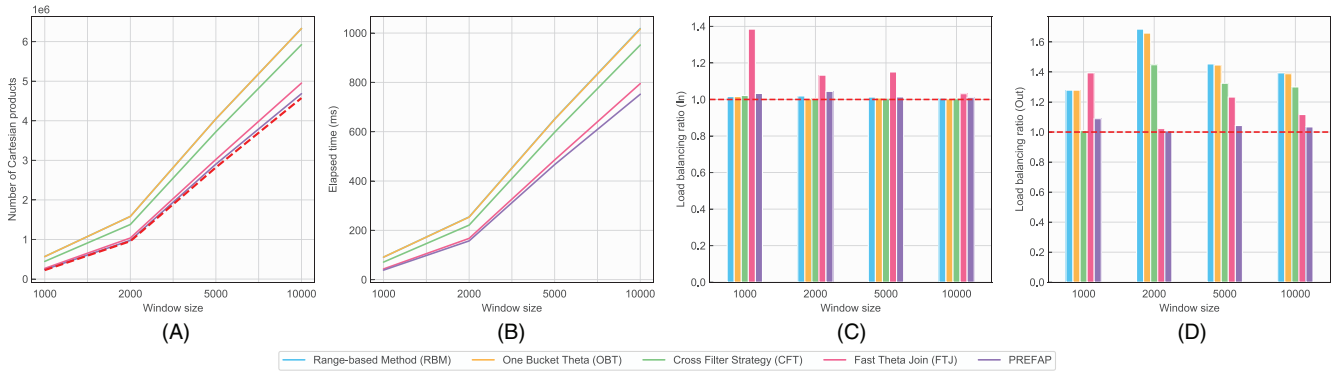

FIGURE 15 The performance comparisons between algorithms when performing theta-join on real 2-way stock data streams ($\theta : \mathcal{R} > \mathcal{S}$, $\mathcal{R}$: 2010–2020 FedExpress@NYSE, $\mathcal{S}$: 2010–2020 Adobe@NYSE) in different window sizes (1000, 2000, 5000, and 10,000). (A–D) present the number of Cartesian products, elapsed time, load balancing ratio (in) and load balancing ratio (out), respectively. The red dotted line in (A) indicates the number of theta-join results, that is, the minimum number of Cartesian products that need to be performed. The red dotted lines in (C,D) mark 1.0, which indicates perfect load balancing

5 | CONCLUSION

In this article, we propose the *Prefap* algorithm to enhance the performance of the theta-join operation. Compared with the FastThetaJoin algorithm, the prefiltering strategy is applied to filter data elements in data streams that are deemed not possible to produce any valid theta-join results. The prefiltering not only reduces the amount of data and hence lessens the workload, but also makes the partitioning more fine-grained to benefit further filtering. Then, during partitioning, the amalgamated partitioning scheme is employed to amalgamate the partitioning of two data streams, so that the performance degradation of the partition-level filtering caused by the coarse-grained isolated partitioning is avoided. By collaborating these mechanisms with the oversized partition repartitioning strategy, as well as the partition-level filtering mechanism based on the theta condition, it forms our proposed *Prefap* framework and it becomes more efficient when performing the theta-join operation. Comprehensive experiments and analyses are conducted to demonstrate the superiority of the performance against the recently published FastThetaJoin and several well-known theta-join algorithms.

ACKNOWLEDGMENT

This work is supported in part by Key-Area Research and Development Program of Guangdong Province (2020B010164002) and Shenzhen Basic Research Program (Nos. JCYJ20170818153016513 and JCYJ20200109115418592).

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

ORCID

Yang Wang https://orcid.org/0000-0001-9438-6060

REFERENCES

1. Buckee C. Improving epidemic surveillance and response: big data is dead, long live big data. *Lancet Dig Health*. 2020;2(5):e218-e220.
2. Tsai CW, Lai CF, Chao HC, Vasilakos AV. Big data analytics: a survey. *J Big Data*. 2015;2(1):1-32.
3. Deng F, Lv Z, Qi L, Wang X, Shi M, Liu H. A big data approach to improving the vehicle emission inventory in China. *Nat Commun*. 2020;11(1):1-12.
4. Wang J, Yang Y, Wang T, Sherratt RS, Zhang J. Big data service architecture: a survey. *J Internet Technol*. 2020;21(2):393-405.
5. Marjani M, Nasaruddin F, Gani A, et al. Big IoT data analytics: architecture, opportunities, and open research challenges. *IEEE Access*. 2017;5:5247-5261.
6. Luo J, Wu M, Gopukumar D, Zhao Y. Big data application in biomedical research and health care: a literature review. *Biomed Inform Insights*. 2016;8:BII–S31559.
7. Din SU, Shao J, Kumar J, Ali W, Liu J, Ye Y. Online reliable semi-supervised learning on evolving data streams. *Inf Sci*. 2020;525:153-171.
8. Dong E, Du H, Gardner L. An interactive web-based dashboard to track COVID-19 in real time. *Lancet Infect Dis*. 2020;20(5):533-534.
9. Bifet A, de Francisci Morales G, Read J, Holmes G, Pfahringer B. Efficient online evaluation of big data stream classifiers. Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; August 10, 2015:59-68.
10. Rezaei H, Faaljou H, Mansourfar G. Stock price prediction using deep learning and frequency decomposition. *Expert Syst Appl*. 2021;169:114332.
11. Silva JA, Faria ER, Barros RC, Hruschka ER, Carvalho AC, Gama J. Data stream clustering: a survey. *ACM Comput Surv (CSUR)*. 2013;46(1):1-31.
12. Hahn C, Warren S, Eastman R. Extended edited synoptic cloud reports from ships and land stations over the globe, 1952-2009 (NDP-026C). Technical report, Environmental System Science Data Infrastructure for a Virtual Ecosystem; 1999.
13. Väänänen O, Hämäläinen T. Sensor data stream on-line compression with linearity-based methods. Proceedings of the 2020 IEEE International Conference on Smart Computing (SMARTCOMP); September 14, 2020:220-225; IEEE.
14. Gaber MM, Zaslavsky A, Krishnaswamy S. Mining data streams: a review. *ACM SIGMOD Rec*. 2005;34(2):18-26.
15. Gama J. *Knowledge Discovery from Data Streams*. CRC Press; 2010.
16. Garcia-Molina H. *Database Systems: The Complete Book*. Pearson Education India; 2008.
17. Mishra P, Eich MH. Join processing in relational databases. *ACM Comput Surv (CSUR)*. 1992;24(1):63-113.
18. Lee T, Kim K, Kim HJ. Join processing using bloom filter in mapreduce. Proceedings of the 2012 ACM Research in Applied Computation Symposium; October 23, 2012:100-105.
19. Rui R, Tu YC. Fast equi-join algorithms on gpus: design and implementation. Proceedings of the 29th International Conference on Scientific and Statistical Database Management; June 27, 2017:1-12.
20. Chen R, Prasanna VK. Accelerating equi-join on a CPU-FPGA heterogeneous platform. Proceedings of the 2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM); May 1, 2016:212-219; IEEE.
21. Wilschut AN, Apers PM. Pipelining in query execution. PARBASE-90: International Conference on Databases, Parallel Architectures, and Their Applications; March 7, 1990:562; IEEE.
22. Penar M, Wilczek A. The evaluation of map-reduce join algorithms. *Beyond Databases, Architectures and Structures. Advanced Technologies for Data Mining and Knowledge Discovery*. Springer; 2015:192-203. doi:10.1007/978-3-319-34099-9_14
23. Koumarelas IK, Naskos A, Gounaris A. Binary theta-joins using mapreduce: efficiency analysis and improvements. EDBT/ICDT Workshops; 2014:6-9.
24. Bellas C, Gounaris A. GPU processing of theta-joins. *Concurr Comput Pract Exp*. 2017;29(18):e4194.
25. Cao S, Haihong E, Song M, Zhang K. Optimization of data distribution strategy in theta-join process based on spark. Proceedings of the 2018 2nd International Conference on Algorithms, Computing and Systems; July 27, 2018:71-75.
26. Oracle O. Oracle Sql documentation: creating and maintaining joins; 2021 [Online]. Accessed May 20, 2021. https://docs.oracle.com/cd/B25016_08/doc/dl/bi/B13916_04/joins.htm

27. PostgreSql P. PostgreSql documentation 8.3: joins between tables; 2021. Accessed May 20, 2021. https://www.postgresql.org/docs/8.3/tutorial-join.html
28. Sviridov A, Grishchenko A, Belousova S. Performance estimation of selecting and inserting procedures in the structure-independent database. Proceedings of the 2014 IEEE 8th International Conference on Application of Information and Communication Technologies (AICT); October 15, 2014:1-5; IEEE.
29. Del Monte B, Zeuch S, Rabl T, Markl V. Rhino: efficient management of very large distributed state for stream processing engines. Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data; June 11, 2020:2471-2486.
30. Rodrigues ER, Madruga FL, Navaux PO, Panetta J. Multi-core aware process mapping and its impact on communication overhead of parallel applications. Proceedings of the 2009 IEEE Symposium on Computers and Communications; 2009:811-817; IEEE.
31. Liew CS, Abbas A, Jayaraman PP, Wah TY, Khan SU. Big data reduction methods: a survey. *Data Sci Eng*. 2016;1(4):265-284.
32. DeWitt DJ, Naughton JF, Schneider DA, Seshadri S. Practical skew handling in parallel joins. Technical report, University of Wisconsin-Madison Department of Computer Sciences; 1992.
33. Okcan A, Riedewald M. Processing theta-joins using MapReduce. Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data; June 21, 2011:949-960; ACM, New York, NY.
34. Liu W, Li Z, Zhou Y. An efficient filter strategy for theta-join query in distributed environment. Proceedings of the 2017 46th International Conference on Parallel Processing Workshops (ICPPW); August 14, 2017:77-84; IEEE.
35. Hu Z, Fan X, Wang Y, Xu C. FastThetaJoin: an optimization on multi-way data stream theta-join with range constraints. Proceedings of the International Conference on Algorithms and Architectures for Parallel Processing; October 2, 2020:174-189; Springer, Cham.
36. Aliyu AM, Zirra P. A comparative analysis of sorting algorithms on integer and character arrays. *Int J Eng Sci*. 2013;25-30. doi:10.1145/1989323.1989423
37. Al-Kharabsheh KS, AlTurani IM, AlTurani AMI, Zanoon NI. Review on sorting algorithms a comparative study. *Int J Comput Sci Secur (IJCSS)*. 2013;7(3):120-126.
38. Yahoo F. Yahoo! finance; 2021 [Online]. Accessed May 20, 2021. https://finance.yahoo.com/

How to cite this article: Wu J, Wang Y, Fan X, Ye K, Xu C. Toward fast theta-join: A prefiltering and amalgamated partitioning approach. *Concurrency Computat Pract Exper*. 2022;e6996. doi: 10.1002/cpe.6996