


# An Efficient Workload-balancing Algorithm for a Parallel Environment Using Hybrid Spatio-temporal Indexes


**Claudio Gutiérrez-Soto**

(Universidad del Bío-Bío, Concepción, Chile)

 <https://orcid.org/0000-0002-7704-6141>, [cogutier@ubiobio.cl](mailto:cogutier@ubiobio.cl)


**Marco A. Palomino\***

(School of Natural and Computing Sciences, University of Aberdeen, Aberdeen, UK)

 <https://orcid.org/0000-0001-7850-416X>, [marco.palomino@abdn.ac.uk](mailto:marco.palomino@abdn.ac.uk)

**Patricio Galdames**

(Universidad San Sebastian, Concepción, Chile)

 <https://orcid.org/0000-0003-3051-2413>, [patricio.galdames@uss.cl](mailto:patricio.galdames@uss.cl)

**Abstract:** In recent years, we have witnessed the proliferation of applications that generate thousands of terabytes of data per day, due to the explosive increase in storage capacity across various devices. As a consequence, a new concept called *Data Deluge* has emerged. Data deluge refers to the situation where the quantity of data generated exceeds the processing power available, and spatio-temporal data is no exception to this phenomenon. In this context, the efficient processing of spatio-temporal queries becomes crucial to address this challenge, as slow query processing can result in obsolete answers, which may lead to errors. Considering this dynamic context of storage and processing, we explore a new online workload algorithm in a distributed parallel environment using hybrid spatio-temporal indexes. This algorithm is able to update the indexes with the most appropriate data, aiming to achieve more efficient query processing. To measure the efficiency of this algorithm, we present its time complexity along with an empirical evaluation of its performance, considering processing time, number of accessed nodes, and communication costs. The empirical results show a significant reduction in processing time, communication costs, and number of accessed nodes.

**Keywords:** Parallel Algorithms, Spatio-temporal Data, Hybrid Indexes, Query Efficiency

**Categories:** H.2.4, H.3.4, I.1.2, I.3.1

**DOI:** 10.3897/jucs.164671

## 1 Introduction

Big data is a generic term to refer to the storage and processing of large amounts of data. The term is closely related to the phenomenon known as *data deluge*, which highlights the need for high availability of processing power to exploit data within reasonable time—for example, when decisions must meet a deadline and such decisions depend on previously processed data. Query processing to make decisions is further complicated when the databases have additional features, such as spatio-temporal features. *Spatio-temporal databases* (STDBs) consist of spatial objects whose positions or shapes change over time and help us represent the dynamic nature of real-world applications.

---

\* Correspondance: [marco.palomino@abdn.ac.uk](mailto:marco.palomino@abdn.ac.uk)

A relevant aspect to consider is that these databases tend to be extremely large when the modeled objects are in highly dynamic contexts (i.e., objects change their locations within short time periods). For instance, STDBs based on images, graphs, and remote sensing data can provide several terabytes or even petabytes of data per day [Xiong et al., 2025]. Thus, global positioning systems [Yang et al., 2024] and geographic information systems [Li et al., 2024] can be considered as STDBs.

Existing studies have demonstrated that the exploitation of STDBs can provide valuable knowledge, such as human sociological behavior in location-based social networks [Cats, 2024], weather analysis [Zhang et al., 2024], and road traffic [Jin et al., 2024]. Also, many applications for mobile devices now use spatio-temporal data. An example of this kind of software is Uber, an application provided by the popular transportation company of the same name. With the Uber app, users can request rides by querying in real-time if there are any available drivers nearby, knowing that both the former and the latter can change their location at a moment's notice. During hours of high demand, these spatio-temporal queries have to be processed as efficiently as possible, so as to ensure that the results will continue to be valid under these changing conditions. As with Uber, other location-based applications may need to process different kinds of complex spatio-temporal queries in real-time including k-nearest neighbours [Qian and Tian, 2024], time-interval [Almaslukh et al., 2021], time-slice [Wang et al., 2024] or range queries [Xie et al., 2024], to name a few. Depending on the type, different strategies may be used to improve efficiency.

To solve this problem, a wide range of ad-hoc indexes can be found in the literature [Ayeelyan et al., 2016], each tailored to efficiently answer specific types of queries. Nevertheless, querying a centralized index in a high demand application can generate bottlenecks in the index servers. One way to avoid this problem is by using a parallel approach to serve incoming queries without blocking. In this manner, the main contributions of this paper are as follows:

**A new online algorithm for workload balancing:** To the best of our knowledge, the use of multiple indexes that adapt to different query types in a parallel environment has not been explored before to improve efficiency. In this paper we propose a new online algorithm for the workload balancing problem in a parallel environment with distributed memory. The algorithm works on the most common queries in spatio-temporal databases, which are time-interval and time-slice [Hernández et al., 2008]. This algorithm updates the index (MVR-Tree or HR-tree) in each processor, according to the most frequent type of query (i.e., evaluating a cost function), which provides a reduction in processing time in a distributed parallel environment.

**Extensive experimentation:** Aiming to simulate a dynamic context where the queries change type (Time-interval queries are efficiently processed by MVR-tree, while HR-tree provides better results for Time-slice queries), different probability distributions have been applied (i.e., Exponential, Gamma and Zipf). Additionally, synchronous and asynchronous communications were evaluated. The performance metrics studied included the running time and the saving in nodes accessed.

**The cost function and the time complexities** Broadly speaking, the cost function has the purpose of providing a value by which it is possible to decide what index is the most convenient to answer the efficient way to the type of most frequent query. To illustrate, let us assume the following scenario, in which the index that is working corresponds to HR-tree, however at any particular moment in time, if the most frequent type of queries corresponds to time-interval, then the current index should

be updated to the MVR-tree according to the value provided by the cost function. It should be noted that the cost function considers the cost of the updating in terms of accessed nodes, assuming that this updating is not optimal since the decision about the updating is taken online. Besides, the time complexities of updating one index to another are provided in detail.

The remainder of this paper is organized as follows: In section The Workload Balancing Algorithm, HR-tree and MVR-tree are presented. In section Overview of the Distributed Algorithm, a description of how the algorithm works is presented. In section Empirical results experiment environments and final results are discussed. Finally, in section Conclusion and Future Work, final remarks are provided.

## 2 Related Work

The spatio-temporal literature is crowded with diverse research that deals with approaches such as infrastructure, programming languages, and the databases used for storage, as well as proposals for categorizing technologies used to process *Geographical Information Systems* (GIS) data, as mentioned in [Liang et al., 2024]. Few studies have explored the use of hybrid spatio-temporal indexes to improve query performance. For instance, [Ke et al., 2014] proposed a spatio-temporal indexing method denominated HBSTR-tree. This method is a hybrid index structure, which is formed by B\*-tree, R-tree and Hash table structures. The method involves gathering consecutive trajectory points as nodes and considering their spatio-temporal semantics before inserting them into an R-tree as leaf nodes. A hash table is employed to minimize the frequency of insertions. Additionally, a B\*-tree sub-index of leaf nodes is used to efficiently answer queries related to object trajectories. To facilitate data storage in the cloud, a database storage scheme based on a NoSQL DBMS is proposed. Results show that HBSTR-tree performs better than TB\*-tree in aspects such as generation efficiency, query performance, and query type. Another approach is presented by [Wang et al., 2017], where a segmentation hybrid index, called the SHB+-tree, is proposed. This new index is based on the B+-tree. Temporal data is split into fragments, which are stored in a temporal table according to chronological order. In each fragment, the hybrid index is built considering the temporal index, the object index, and temporal metadata. Better performance in terms of construction and maintenance is achieved using a segmented storage strategy, as well as approaches based on bottom-up index construction. Experimental results demonstrate the proposed method's improved effectiveness and efficiency.

Notably, few previous approaches have explored the use of parallel algorithms to enhance query processing in spatio-temporal databases [Alharthi et al., 2015], [Ding et al., 2015], [Liao et al., 2015], [Gutiérrez-Soto et al., 2008], [Deng et al., 2017]. Indeed, to the best of our knowledge, the use of hybrid spatio-temporal indexes has not been sufficiently studied. In [Cheng et al., 2020], the authors propose a middleware called UniIndex. The underlying idea is to provide efficient data location services with minimal user effort. To achieve this goal, UniIndex uses annotation extraction along with an in-memory cache layer, in-situ indexing, and parallel query processing. Final results demonstrate that files can be located within a dataset containing millions of files in microseconds while scanning the entire dataset, achieving a speed-up of two orders of magnitude. Further, the most similar work to this paper was proposed by Gutiérrez-Soto et al. [Gutiérrez-Soto et al., 2008], where an online algorithm on the CREW RAM parallel model using MVR-Tree is presented.

As opposed to [Gutiérrez-Soto et al., 2008], we propose a new online algorithm for the workload balancing problem in a parallel environment with distributed memory. The algorithm works with the most common queries in spatio-temporal databases, which are time-interval and time-slice [Tao et al., 2002], [Tao and Papadias, 2001], [Hernández et al., 2008]. It updates the index—MVR-Tree or HR-tree—in each processor, according to the most frequent type of query, which provides a reduction in processing time in a distributed parallel environment. Time-interval queries are efficiently processed by MVR-tree, while HR-tree provides better results for time-slice queries. Different probability distributions have been applied, and several experiments performed. Our results show that our algorithm is scalable when the number of processors is increased, considering both, synchronous and asynchronous communication.

### 3 Workload-balancing Algorithm

Sometimes, when an algorithm has all the information—an off-line algorithm—it is feasible to design a good strategy to have an efficient algorithm [Boyar et al., 2016]. An example of this is a sorting algorithm. A sorting algorithm knows the quantity ( $N$ ) and the numbers that it must sort. Using the decision-tree model [Grunwald and Vítanyi, 2008]), it is possible to determine the time complexity for an optimal sorting algorithm, where the worst-case running time is  $O(n \log_2 n)$ . Thus, for this example, Merge Sort and Heapsort algorithms have always this performance. Contrary to Merge Sort and Heapsort algorithms, the Quicksort worst-case is  $O(n^2)$ , meanwhile its best-case is  $\Omega(n \log_2 n)$ . In simple words, in an optimal algorithm the worst-case is equal to the best-case running time. In contrast, when the algorithm does not have all the information or receives that information partially, it is not possible to obtain an optimal algorithm—that is, the problem that the algorithm solves is more complex. A measure to evaluate the efficiency of an online algorithm is by competitiveness. Broadly speaking, competitiveness can be seen as integer number  $C$ , which multiplies the cost function of an optimal algorithm. The cost function is evaluated with the goal of making a decision. This function is associated to the parameter that defines the behaviour of an online algorithm. An optimal algorithm has competitiveness 1, due to that algorithm knowing all information and therefore can provide the best result. Unlike the optimal algorithm, an online algorithm cannot provide the best result and as a consequence its competitiveness is always greater than 1.

Algorithmic efficiency can be given in terms of processing time, space—for instance, in compression data algorithms—or distance—for example, algorithms in metric spaces—to mention a few. In our case, the efficiency of the algorithms for spatio-temporal indexes is given by the number of nodes accessed—that is, the fewer nodes accessed by the algorithm the more efficient it is.

In this paper, we are concerned with the response times, mainly with the processing time for each processor and the communications time in a parallel environment with distributed memory. However, there is not a one-to-one correspondence between nodes accessed, processing and communications time, because it depends on the hardware and software characteristics of a particular parallel environment.

To clarify our contributions, we provide the following definitions:

**Definition 1:** An MVB-tree corresponds to a Multi-Version B-tree, which allows storing different versions of the same data at different timestamps. This spatio-temporal data structure efficiently answers queries and relies on historical state when the queries refer to time intervals—a time-interval query considers an interval of timestamps; for example,  $[t_1, t_4]$ .

**Definition 2:** An MVR-tree is a multiple R-tree, which corresponds to an extension of MVB-Tree. Its main characteristic is the efficiency for the queries that are time-interval. The difference between an MVB-tree and an MVR-tree underlying the attribute changing through time is the spatial component.

**Definition 3:** An HR-tree is a historical R-tree. The idea behind an HR-tree is to store an R-tree for each timestamp. This approach saves space since branches that remain unchanged across timestamps are kept intact. This structure efficiently answers queries when they are time-slice queries—a time-slice query considers a unique timestamp; for instance,  $t_0$ .

In this manner, efficiency of our online algorithm must be given in terms of time—that is, it involves the processing time alongside communications time—and not of the nodes accessed. As a consequence, the cost function used by our algorithm is based on time complexities. Thus, this cost function takes into account the time complexities of the algorithms used with the MVB-tree and HR-tree indexes, specifically, the insertion and search algorithms. In addition, the cost function evaluates the communications time to send a query and receive the answer as well as the type and number of queries.

### 3.1 Overview of the Distributed Algorithm

Our approach deals with any parallel computer such as PC clusters or distributed memory multiprocessors. It is formed by a set of  $P$  processor-local-memory components which communicate with each other through messages. The computation involves both the local data processing in each processor and communication, which is given by sending messages to other processors. We assume the master/slave communication-model, our online algorithm operates inside of the master processor aiming to solve the workload balancing problem. The master processor spreads out each query among processors and receives the results from the slave processors. The communication among processors can be synchronous and asynchronous. When the communication is asynchronous, each slave processor immediately sends its results to the master processor, once it has completed its work (i.e., the communication is made in  $O(P)$  steps). carried out in the shape of a tree by which the master processor receives the results in  $O(\log P)$  communication steps.

The online algorithm works as follows. At the beginning, the master processor scatters the  $N$  objects among the  $P$  processors, and only one index is built on all processors (e.g., the MVR-tree is the initial index used on each processor; see Definitions 2 and 3). In this case, the communications among the processors is asynchronous. Subsequently, the master processor immediately distributes each received query among the processors, and stores the query type to which  $q$  belongs as well as the counter of processed queries (with this information it is possible to update the index). Finally, it gathers the partial results from slave processors to give the whole result. When a specific number of queries have been processed and the master processor receives a new query, the master processor evaluates the cost function to make a decision about the index change. In simple words, if the most processed queries are time-slice and the indexes in processors are MVR-tree then the indexes in each processor should be updated to HR-tree with the aim to efficiently process the queries. It should be noted that only one index takes place among processors at any point in time.

Every time a new index update takes place, the Round Robin strategy is applied. In this way, it is possible to prevent the most consulted objects (which belong to the query results) from remaining in a few processors. Thus, a better load balance is achieved.

A key observation, concerns the representativeness of the cost function, which is given in terms of the algorithm time complexities involved in updating the indexes. Thus, we assume that if the order of an algorithm is  $O(n)$ , it is expressed as  $(n)$  in the cost function. We argue, and empirically show, that time complexities of algorithms involved in online algorithm's cost function are good parameters to evaluate the update of indexes. Table 1 lists the main symbols frequently used in time complexities. Some symbols are briefly described in the following subsections to provide a better understanding.

Symbols	Definitions
$N$	Number of objects in an MVR-tree or HR-tree
$P$	Number of processors
$l$	The cost of synchronizing the processors in a parallel environment
$g$	Unit of running time in a parallel environment
$n'$	Maximum number of incoming/outgoing messages per processor
$n''$	The number of retrieved nodes as answer to query $q$
$Q_s$	A set of queries time-slice
$Q_i$	A set of queries time-interval
$K$	Number of versions retrieved for all nodes in an MVB-Tree, given $q$
$b$	Number of entries a node contains when it splits in an MVB-tree
$T$	Number of timestamps for searching of several R-tree with $m$ branches

Table 1: List of Symbols Used in Time Complexities

Overall, the parallel running time of our approach is the sum of three quantities:  $j$ ,  $ng$  and  $l$ , where  $j$  is the maximum computations performed by each processor,  $n$  is the maximum messages sent/received by each processor with each word costing  $g$  units of running time, and  $l$  is the cost of synchronising the processors. The effect of the computer architecture is taken into account by the parameters  $g$  and  $l$ , which are increasing functions of  $P$ . The aforementioned values along with the processor speeds can be empirically obtained through benchmark programs. Aiming to shed light, a simple example is provided, letting us consider a broadcast operation. At the beginning, the master processor sends a word  $w$  (including itself), whose size is  $w'$  to all processors, with a cost of  $O(P(l + gw' + w'))$ . Afterwards, the master processor receives the answer from other processors. In this case, the answer for  $w$  is  $w''$ . Then, the final cost should be given by  $O(P(l + gw' + w')) + O(P(l + gw'' + w''))$ . Note that in this example, the local processing power  $j$  is not considered.

The cost function considers the following. Let  $O(P(l + gn'))$  be the time complexity for sending a query  $q$ , such as  $q$  can be time-slice or time-interval,  $n'$  is the maximum number of incoming/outgoing messages by processor for a network with  $P$  processors, where  $g$  corresponds to the total number of local operations performed by all processors and  $l$  minimal number of steps for synchronization operations. Additionally, the cost function considers  $O(P(l + gn''))$ , where  $n''$  corresponds to the retrieved nodes as answer for the query  $q$ , which are sent by  $P$  processors. It is important to mention that  $O(P(l + gn'))$  and  $O(P(l + gn''))$ , corresponds with asynchronous communication.

Synchronous communication is carried out over a number of logarithmic steps. The main idea is that the master processor gathers the answers for the query  $q_s$  from each processor. To illustrate this, let us consider a system with four processors.

In the first step, processor two sends its answer to processor one, while processor four sends its answer to processor three (i.e., both answer transmissions are carried out simultaneously; strictly speaking, the cost is  $O(l + gn'')$ ). In the second and final communication step, both processors one and three send the collected answers to the master processor. In this manner, the total response is communicated in  $\log_2 P$  steps. Following this rationale, synchronous communication is represented as  $O((l + gn'')\log P)$ .

Roughly, both indexes should have the same number of spatio-temporal objects. In a parallel environment, all processors will build an MVR-tree or HR-tree with  $N'$  objects, such as  $N' = N/P$ . On the other hand, it should be noted that the basic operations of the online algorithm rely on insertion and search in a tree structure (i.e., both indexes are structured as trees). Overall, the time complexities of both operations—insertion and search—are similar in spatio-temporal indexes [Tao et al., 2002]. Thus, according to [Tao et al., 2001], the time complexities of the operation in a MVB-Tree (see *Definition 1*) are similar to those in a MVR-tree (i.e., insertion and search algorithms). Therefore, for the rest of this paper, we will consider that the complexity values for a MVB-tree and a MVR-tree are the same. The time complexity to visit each node in an MVB-tree is  $O((\log(\frac{N'b}{P}) + \frac{K'}{b}))$ . Therefore,  $O(\log(\frac{N'b}{P}) + \frac{K'}{b})$  corresponds to the visit each node in an MVB-trees for each processor, where  $K'$  is  $K$  in a parallel environment. Similarly, the time complexity to traverse an HR-tree is  $O(\log_M(N))$  (According to [Guttman, 1984], this time complexity is equal to a B-tree). Thus, we have  $O(\log_m(\frac{N'}{TP}))$ , such as  $m$  is  $M$  in a parallel environment.

It is important to mention that the cost function is formed by two evaluations. The first evaluation is performed when the indexes in processors are MVR-trees. The second evaluation takes place when the indexes in processors are HR-trees.

**Definition 4:** Let  $M_s$  be part of the cost function, which applies when all processors are MVR-trees, communication is synchronous, and queries are time-slice, considering the cost of visiting each node in the index. Then

$$M_s = [(P(l + gn')) + \log(\frac{N'b}{P} + \frac{K'}{b}) + P(l + gn'')]$$

**Definition 5:** Let  $H_s$  be part of the cost function for changing the indexes of the HR-tree. It should evaluate the cost of visiting the  $N'$  objects from the master processor (i.e.,  $O(N')$  considering the cost of sending the  $N'$  objects  $O(P(l + gN''))$ ), as well as the cost of distributing these objects to each processor. Then,

$$H_s = (N') + (P(l + gN'')) + (\frac{N'}{P} \log \frac{N'}{P})$$

**Definition 6:** Let  $C_H$  be the cost of processing a time-slice query in an HR-tree in a parallel environment. Then,

$$C_H = (\log_m(\frac{N'}{TP})) + (P(l + gn')) + (P(l + gn''))$$

Thus, the first part of the cost function is given by the following inequality, such that  $|Q_s|$  corresponds to the number of queries for the set  $Q_s$ , thus

$$f_1(Q_s, M_s, H_s, C_H) = \begin{cases} true & |Q_s| M_s > H_s + |Q_s| C_H \\ false & |Q_s| M_s \leq H_s + |Q_s| C_H \end{cases}$$

In this way, if the inequality holds true, HR-trees are built in the  $P$  processors. It is important to point out that  $|Q_s|$  appears by multiplying both  $M_s$  and  $C_H$ . It depends on the amount of time-slice queries processed at the moment to evaluate the cost function.

**Definition 7:** Let  $\mathcal{H}_f$  be part of the cost function, used when all processors are HR-trees, communication is asynchronous, and queries are time-interval, considering the cost of visiting each node in the index. Then,

$$\mathcal{H}_f = ((l + n'g)\log P) + (\log_m(\frac{N'}{TP})) + (P(l + gn'))$$

**Definition 8:** Let  $\mathcal{M}_f$  be part of the cost function for changing the indexes of the MVR-tree. It must calculate the cost of visiting the  $N'$  objects from the master processor, taking into account the cost of sending the  $N'$  objects and the cost of distributing these objects to each processor. Then,

$$\mathcal{M}_f = (N') + ((l + N'g)\log P) + (\log(\frac{N'b}{P}) + \frac{K'}{b})$$

**Definition 9:** Let  $\mathcal{C}_M$  be the cost of processing a time-interval query in an MVR-tree in a parallel environment. Then,

$$\mathcal{C}_M = ((l + n'g)\log P) + (\frac{N'}{P}(\log(\frac{N'b}{P}) + \frac{K'}{b})) + ((l + n''g)\log P)$$

The second part of the cost function is given by:

$$f_2(Q_i, \mathcal{M}_f, \mathcal{H}_f, \mathcal{C}_M) = \begin{cases} true & |Q_i|\mathcal{H}_f > \mathcal{M}_f + |Q_i|\mathcal{C}_M \\ false & |Q_i|\mathcal{H}_f \leq \mathcal{M}_f + |Q_i|\mathcal{C}_M \end{cases}$$

Thus, if  $f_2$  returns true, each processor should build a MVR-tree. The values  $n'$  and  $n''$  were obtained empirically, and these values depend on each network's architecture. The cost function is formed by  $f_1$  and  $f_2$ . Equation  $f_1$  takes place when the indexes in each processor are MVR-trees and the communication among processors is asynchronous. By contrast,  $f_2$  makes sense when the indexes in all processors are HR-trees and the communication among processors is synchronous. Both  $f_1$  and  $f_2$  can be extended to consider asynchronous and synchronous communication.

The algorithm takes several parameters into account—see Algorithm 1. *Cont* counts the number of queries; *CostF* corresponds to the evaluation of the cost function; and *Evaluate* indicates the number of queries where the cost function should be evaluated. *CIndex* points out the current index; if the result of the cost function is 1, then processors are alerted to the index change—see line 10, Algorithm 1. Afterwards, the most consulted objects are distributed among the processors. Round-robin scheduling is applied to distribute the rest of objects among processors—see lines 10 to 20, Algorithm 1. The function *Send* and *Receive* have the parameter *Asyn*, which indicates that the communication between processors is Asynchronous. The parameter *Syn* indicates that the communication between processors is synchronous and carried out in a logarithmic way—in other words, the communication takes  $O(\log P)$ .

As mentioned earlier, an optimal algorithm always knows all the parameters used in the cost function. Hence, when the indexes in the processors are MVR-trees and the communication is asynchronous, the optimal algorithm should at least know the number of time-slice queries ( $|Q_s|$ ), time-interval queries ( $|Q_i|$ ), and retrieved nodes as answers

for each query ( $n''$ ). Readers should note that knowing these parameters is impossible, as the algorithm operates on demand. Hence, the cost of the HR-tree in each processor is given by the components  $f_1$ , ( $H_s$ ), and  $C_H$ . Therefore, if

$$|Q_s|M_s = H_s + |Q_s|C_H \quad (1)$$

It is not necessary to update the indexes in all the processors.

By contrast, the online algorithm knows all parameters used in the cost function except  $|Q_s|$  (Note that  $|Q_i|$  is also unknown). Thereby  $|Q_s|[M_s - C_H]$  can be redefined as  $k[M_s - C_H] + H_s$  for any  $k$  value ( $k \in \mathbb{N}$ ), which represents the number of queries. This clearly is not optimal (i.e., it should be optimal if the result of one of the ratios in (1) was 1). Whereby, competitiveness is given for the ratio

$$\text{Max} \left( \frac{k[M_s - C_H + H_s]}{|Q_s|[M_s - C_H]}, \frac{k[M_s - C_H] + H_s}{H_s} \right) \quad (2)$$

If the difference is not equal to 0 between  $M_s$  and  $C_H$  ( $|(1) - C_H|$ ) and the value of  $k$  is at least 1, then competitiveness is at least 2 for  $\left(\frac{k[M_s - C_H + H_s]}{H_s}\right)$ . However, it is important to highlight that competitiveness depends on  $|Q_s|$ .  $f_2$  can be considered to calculate competitiveness for  $|Q_i|$ , replacing  $|Q_s|$  with  $|Q_i|$ .

The time complexity to update indexes from MVR-trees to HR-trees using asynchronous communication is given first, for the evaluation of the cost function, which takes  $O(1)$  (see line 8, Algorithm 1). The broadcast to report the index change (see line 9, Algorithm 1) takes  $O(P(l + gn'))$ . Reading all objects involves  $O(N')$  and sending to the processors takes  $O(P(l + gN'))$  (see line from 10 to 19, Algorithm 1). Updating the index in each processor takes  $O((\log_m(\frac{N'}{TP}))$ , (i.e., it is the cost to building and inserting objects in each processor using HR-trees). Thus the time complexity updating the index in each processor takes  $O(N') + O(P(l + gN')) + O((\log_m(\frac{N'}{TP}))$ .

Otherwise, the time complexity to update from HR-trees to MVR-trees takes  $O(N') + O(P(l + gN')) + O(\log(\frac{N'b}{P} + \frac{K'}{b}))$ . Note that the costs of answering a query—see lines 26 to 29, Algorithm 1—are not considered. To consider the cost of answering the query using synchronous communication, we should add  $O((l + n'')\log P)$ . To consider the cost of answering the query using asynchronous communication, we should add  $O(P(l + gn''))$ . To validate our algorithm, empirical results are reported below.

## 4 Empirical Results

Broadly speaking, the experimentation can be split into two general steps. The first step involves building the same tree—data insertion into an MVR-tree or HR-tree—with different data on each processor. To achieve this goal, four experimental scenarios were analyzed. In each experiment 23,264 objects were inserted with 10% mobility. Ten timestamps were considered; therefore, it was possible to obtain a total of 46,521 insertions. The second and most important step involves the submission of queries. Aiming to simulate repetitive queries, three probability distributions were used—each distribution in a single experiment.

**Algorithm 1**  $q, Cont, Evaluate, N', b, K, n', n''$ 


---

**Require:**  $q_j$  is a time-interval or time-slice  
**Ensure:**  $Q_s$  is the set of time-slice queries and  $Q_i$  is the set of time-interval queries

```

1:  $t \leftarrow Type(q_j)$ 
2: if  $t = 1$  then
3:    $Q_i \cup q_j$ 
4: else
5:    $Q_s \cup q_j$ 
6: end if
7: if  $Cont = Evaluate$  then
8:   if  $CostF(CIndex, N', K, b, n', n'', Cont, Q_s) = 1$  then
9:      $Broadcast(Change - Index)$ 
10:    for  $n \leftarrow 1, p \leftarrow 0, n \leq N'$  do
11:      if  $p \leq P$  then
12:         $Send(n, Asyn)$  to  $p$ 
13:         $p++$ 
14:      else
15:         $p \leftarrow 1$ 
16:         $Send(n, Asyn)$  to  $p$ 
17:         $p++$ 
18:      end if
19:    end for
20:     $Broadcast(q_j)$ 
21:    for  $p \leftarrow 1, p \leq P$  do
22:       $Receive(nodes, q_j, Asyn|Sync)$  from  $p$ 
23:    end for
24:  end if
25: else
26:    $Broadcast(q_j)$ 
27:   for  $p \leftarrow 1, p \leq P$  do
28:      $Receive(nodes, q_j, Asyn|Sync)$  from  $p$ 
29:   end for
30: end if
31:  $Cont++$ 

```

---

**4.1 Experimental Environments**

Exponential distributions ( $\alpha = 0.5$  and  $\alpha = 1.0$ ), Gamma ( $\lambda = 1.0$ ) and Zipf (1.0) over a single file, which contains fifty time-slice queries and fifty time-interval queries, which are interspersed. In each experiment 1,000 queries were selected and executed from the file mentioned previously. The idea behind using different probability distributions lies in how they affect the subset size of queries and the frequency with which some queries in the subset are repeated—the subset corresponds to the queries selected from the file according to the probability distribution, and the frequency indicates how many times each query was chosen.

Suppose that under an exponential distribution, the query subset consists of ten elements. Among these, four are repeated more frequently, while the others are chosen only once. Thus, the subset size, as well as the number of frequently repeated queries, would likely differ under a different probability distribution.

Several queries were repeated in each experiment. The points of analysis used to check the performance of our algorithm comprised 300, 500 and 700 queries executed, where the algorithm spread the most consulted objects and updated the index according to the most frequent type of queries among processors. This update and distribution took place only in a single point of analysis (300, 500 or 700) when the 1,000 were processed. It is noted that the three points of analysis were chosen considering the thousand queries that were processed. However, these points of analysis can be executed in different instants of time. Besides, it is important to point out that the saving percentages of nodes accessed do not include the communications time—there is not a direct correspondence between communications time and nodes accessed, which should be obtained empirically—and the percentages are calculated with respect to a processor.

We evaluated the performance of our algorithm by using two metrics. The first metric considered the number of nodes accessed in each processor without communications time between processors. The results are presented as saving (in nodes accessed) with respect to one processor. The second metric taken into consideration was the parallel runtime incorporating communications time between processors. The two types of communication considered were synchronous and asynchronous.

The experimental evaluation was carried out on a cluster with 8 nodes (for this work, 1 node is equivalent to 1 processor). Each node/processor with Intel® core™ i7-2600 CPU @ 3.40 GHz, 16 GB RAM DDR3 1333MHz, Linux Debian Squeeze 7.8 – 64 Bits. An implementation of MVR-tree in C++ provided by Yufei Tao was used in these experiments. HR-tree was implemented by us. The programming language was MPI, in particular MPICH-1 version. The cluster configuration is based on HPCC (High Performance Computing Cluster) - Linux Debian. Each node has a 10/100 Ethernet network card, which is connected to a 24-Port 10/100 Unmanaged Rackmount Switch (D-Link) using twisted-pair cable 100-ohm, category 5e UTP. The network topology is Star, where each node has a static IP address.

Regarding the variables used to visit each node in a MVB-tree ( $O(\log(\frac{N'b}{P}) + \frac{K'}{b})$ ), and according to the implementation provided by Yufei Tao,  $b$  was instantiated with the value 6, whereas  $K'$  was internally calculated in that implementation. On the other hand, in the costs to visit each node in an HR-tree ( $O(\log_m(\frac{N'}{TP}))$ ),  $T$  was instantiated with the value 10 in each experiment.  $m$  was instantiated with values between 30 and 100 to calculate the function cost. Note that  $m$  represents the approximate of an HR-tree, which can change in each experiment and therefore it is not accurate.

## 4.2 Experimental Results

Four experiments were carried out, yielding the parallel runtime along with communications time and nodes accessed. The first and second experiments, we used exponential distributions to simulate the repetitive queries. In the third and fourth experiment, gamma and Zipf distributions were applied to simulate repetitive queries.

In Figure 1, 3, 5 and 6, when the number of processors was one, neither synchronous nor asynchronous communications were considered. Essentially, when there was only one processor, we did not evaluate communications times. Times in every Figure are expressed in seconds, where the number of processors is expressed as  $\mu\rho$ .

Every time, synchronous (*S*) and asynchronous (*A*) were considered in the total runtime with the aim of providing the answers for every query.

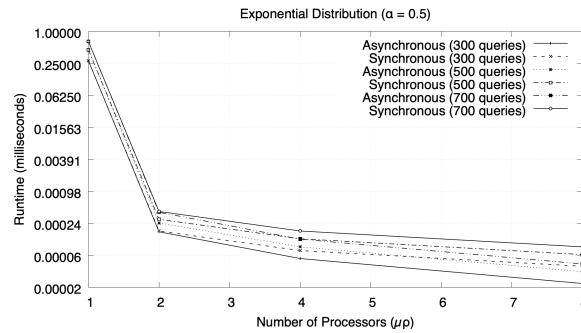


Figure 1: Run times (in seconds) considering synchronous and asynchronous communications using Exponential Distribution with  $\alpha = 0.5$ .

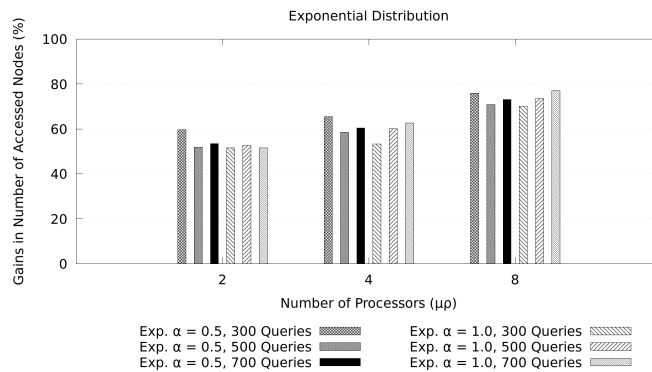


Figure 2: Savings percentages of nodes accessed using Exponential Distribution with  $\alpha = 0.5$ .

Empirical results for the Experiment 1—Exponential distribution  $\alpha = 1.0$ —are displayed in Figure 1 and 2. In Figure 1, asynchronous and synchronous communication time are displayed. From Fig. 1, we can see that when the number of processors was increased, times decreased. The best results were provided when the point of analysis was 300 for both types of communications. Furthermore, it is possible to see that asynchronous communications were always better than synchronous. On the other hand, in Fig. 2 the saving percentages of nodes accessed were presented. Similar to Figure 1, the best savings were given when the point of analysis was 300.

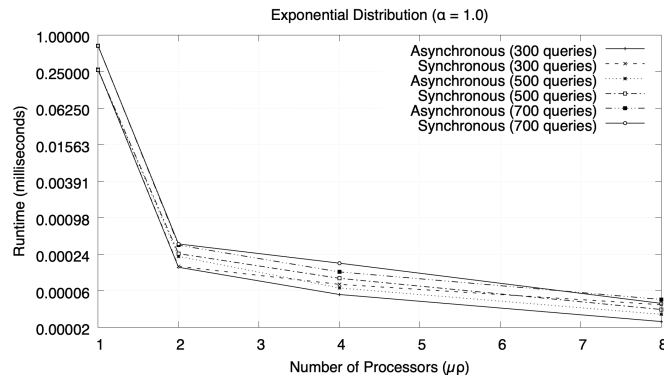


Figure 3: Run times (in seconds) considering synchronous and asynchronous communications using Exponential Distribution with  $\alpha = 1.0$ .

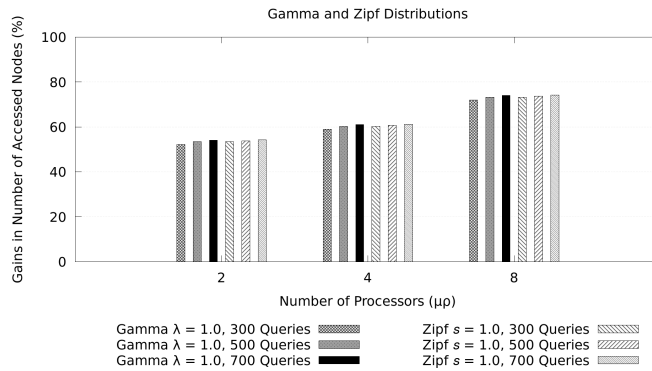


Figure 4: Savings percentages of nodes accessed using Gamma and Zipf Distribution.

Fig. 2 and Fig. 3 display the results of the Experiment 2, where an exponential distribution with parameter  $\alpha = 1.0$  was used to simulate repetitive queries. As in Fig. 1, we can observe in Fig. 3 that when the number of processors increases the times decrease. In Fig. 2, the saving percentages of nodes accessed are displayed. The results of Experiment 3—Gamma distribution  $\lambda = 1.0$ —are shown in Fig. 4 and Fig. 5. As in Fig. 1 and Fig. 3, Fig. 5 displays reductions of times. The same applies for Fig. 4 and Fig. 6 which show the results of Experiment 4—Zipf distribution 1.0. The speed-ups for asynchronous and synchronous communications are displayed in Fig. 7 and Fig. 8.

Even though Fig. 7 and Fig. 8 rely on the highest times, these are scalable. Fig. 1, Fig. 3, Fig. 5 and Fig. 6 show that the best times were provided by asynchronous communications. When the point of analysis was 300, the best saving percentages of nodes accessed were provided with an exponential distribution  $\alpha = 1.0$ , where 260 time-interval queries—from 1,000 queries—took place, in contrast to 347 time-interval queries in the points of analysis 500 and 700.

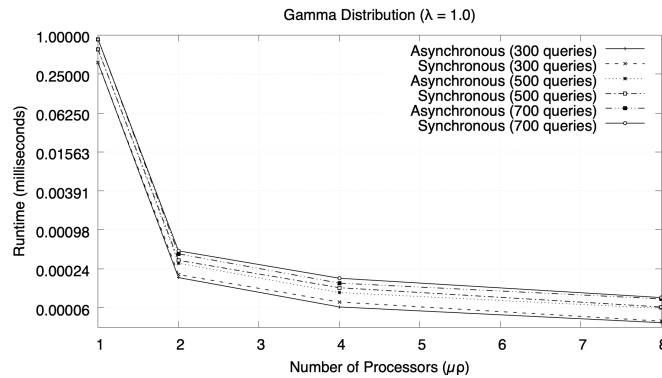


Figure 5: Run times (in seconds) considering synchronous and asynchronous communications using Gamma Distribution with  $\lambda = 1.0$ .

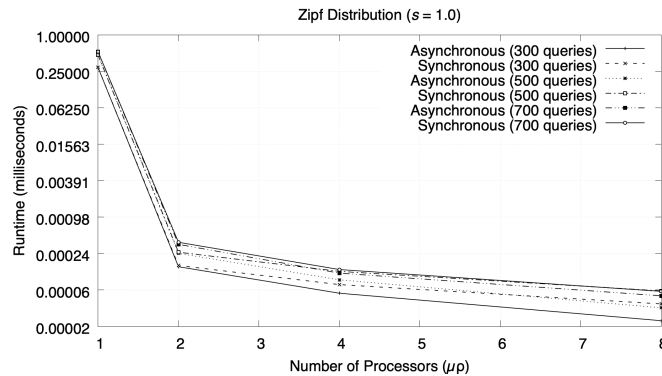


Figure 6: Run times (in seconds) considering synchronous and asynchronous communications using Zipf 1.0.

For the analysis points 500 and 700, the best saving percentages of nodes accessed were given by Zipf distribution where 886 time-interval queries were selected (115 queries were time-slice). (i.e., it implies also that all processors used mainly MVR-tree indexes). This is in line with some results for synchronous communications for the points of analysis 500 and 700. The best results for the synchronous communications in the point of analysis 700 were given for Zipf distribution. For the points of analysis 300 and 500 considering synchronous communications, the difference among them is minimal. For the asynchronous communications in all points of analysis, there is not a big difference. In conclusion, for both cases, synchronous and asynchronous communications, total runtime decreases when the number of processors was increased. Therefore, from Fig. 1, 3, 5, 6, 7 and 8, is feasible to conclude that our online algorithm provides a scalable solution. In addition, from Fig. 2 and Fig.4, every time the number of processors was increased, major savings of nodes accessed were observed.

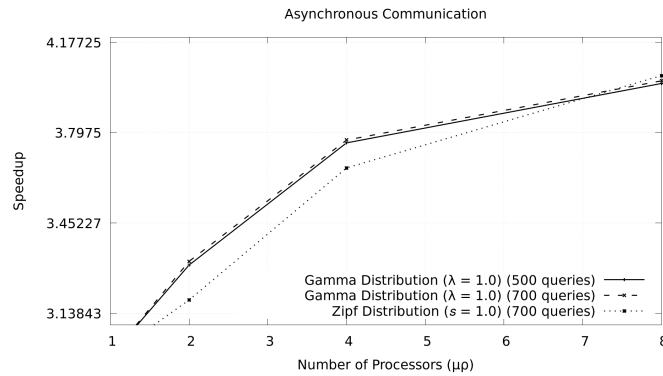


Figure 7: Asynchronous speed up for the highest processing times.

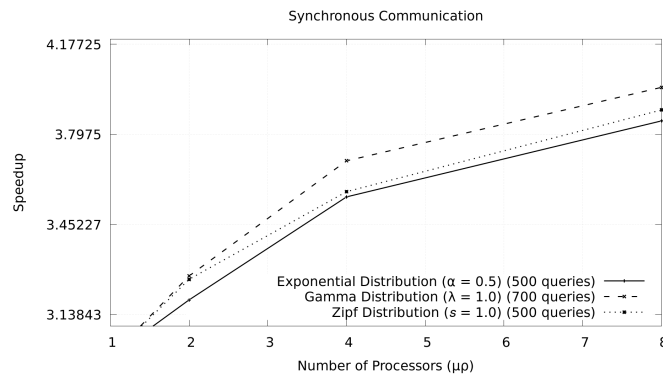


Figure 8: Synchronous speed up for the highest processing times.

## 5 Discussion

Even though the set of queries was small, we tackled this challenge by selecting a thousand queries from the hundred queries from the file using several distribution probabilities. Thus, we simulated repeated queries. The online algorithm takes advantage by using the most frequent type of query. Another interesting matter to analyze is the exact moment when the index should be changed. It can be obtained empirically, due to each architecture having different physical characteristics (i.e., the points in which it is possible to obtain the best results. However, it leads to an optimization problem where not only the architecture of the distributed system should be considered but also the amount and type of queries to be processed). Additionally, measuring only communication times does not make sense in this paper. When the metric used in the experiments was accessed nodes, results were presented without considering the costs of communications because the relation between communications time and nodes accessed depends on the architecture interconnection in a parallel environment with distributed memory.

## 6 Conclusions and Future Work

We have presented a new workload balancing algorithm for a parallel environment with distributed memory on spatio-temporal databases. The algorithm considers the most frequent type of query between time-interval and time-slice queries to construct the most efficient index. Aiming to show the efficiency of our algorithm, four experiments were carried out using different probability distributions to simulate the dynamic context. Empirical results showed significant savings of accessed nodes when answering queries, as well as the parallel runtime and communications time among processors. These results showed that our algorithm is scalable when the number of processors is increased.

Ideas for future research directions in this area are the following: first, we could study an improvement for our online algorithm having both indexes at the same time, where each index has different objects according to the query type. We should take advantage of the multi-cores in each node, by which both the cost function of algorithm and the experimental environment should be considered. Furthermore, different strategies not only to split the objects but also the workload balancing problem should be tackled.

### Acknowledgements

This research was supported by Universidad del Bío-Bío, Chile, under Grants No. DIUBB GI 195212/EF, and DIUBB 2130253 IF/R.

### References

- [Alharthi et al., 2015] Alharthi, S., Eldawy, A., Mokbel, M., Tarek, K., Alzaidy, A., and Ghani, S. (2015). SHAHED: A MapReduce-based System for Querying and Visualizing Spatio-temporal Satellite Data. volume 2015.
- [Almaslukh et al., 2021] Almaslukh, A., Liu, Y., and Magdy, A. (2021). Scalable spatio-temporal top-k community interactions query. In Proceedings of the 29th International Conference on Advances in Geographic Information Systems, SIGSPATIAL '21, page 293–296, New York, NY, USA. Association for Computing Machinery.
- [Ayeelyan et al., 2016] Ayeelyan, J., Muthukumarasamy, S., and S, R. (2016). Indexing and query processing techniques in spatio-temporal data. *ICTACT Journal on Soft Computing*, 06:1198–1217.
- [Boyar et al., 2016] Boyar, J., Favrholt, L. M., Kudahl, C., Larsen, K. S., and Mikkelsen, J. W. (2016). Online algorithms with advice: A survey. *SIGACT News*, 47(3):93–129.
- [Cats, 2024] Cats, O. (2024). Identifying human mobility patterns using smart card data. *Transport Reviews*, 44(1):213–243.
- [Cheng et al., 2020] Cheng, P., Wang, Y., Lu, Y., Du, Y., and Chen, Z. (2020). Uniindex: An index and query middleware for parallel file systems. *Concurrency and Computation: Practice and Experience*, 32(9):e5609. e5609 cpe.5609.
- [Deng et al., 2017] Deng, Z., Wang, L., Han, W., Ranjan, R., and Zomaya, A. (2017). G-ml-octree: An update-efficient index structure for simulating 3d moving objects across gpus. *IEEE Transactions on Parallel and Distributed Systems*, 29:1–1.
- [Ding et al., 2015] Ding, Z., Yang, B., Chi, Y., and Guo, L. (2015). Enabling smart transportation systems: A parallel spatio-temporal database approach. *IEEE Transactions on Computers*, 65:1–1.
- [Grunwald and Vitanyi, 2008] Grunwald, P. D. and Vitanyi, P. M. B. (2008). Algorithmic information theory.

- [Gutiérrez-Soto et al., 2008] Gutiérrez-Soto, C., Gutiérrez, G. A., Rodríguez, P., and Campos, P. (2008). Paralelización de consultas del tipo time-interval en base de espacio-temporales. In Villavicencio, M., Monsalve, C., Mendoza, M. V. M., Pizarro, G., Lema, L., and Flores, S., editors, VII Jornadas Iberoamericanas de Ingeniería de Software e Ingeniería del Conocimiento 2008, Guayaquil, Ecuador, January 30 - February 1, 2008. Proceedings, pages 41–48. Área de Ingeniería de Software VLIR-ESPOL Componente 8, Facultad de Ingeniería Eléctrica y Computación, Escuela Superior Politécnica del Litoral.
- [Guttman, 1984] Guttman, A. (1984). R-trees: a dynamic index structure for spatial searching. In Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, SIGMOD '84, page 47–57, New York, NY, USA. Association for Computing Machinery.
- [Hernández et al., 2008] Hernández, C., Rodríguez, M. A., and Marin, M. (2008). Complex queries for moving object databases in dht-based systems. In Luque, E., Margalef, T., and Benitez, D., editors, Euro-Par 2008 – Parallel Processing, pages 424–433, Berlin, Heidelberg. Springer Berlin Heidelberg.
- [Jin et al., 2024] Jin, J., Liu, P., Huang, H., and Dong, Y. (2024). Analyzing urban traffic crash patterns through spatio-temporal data: A city-level study using a sparse non-negative matrix factorization model with spatial constraints approach. *Applied Geography*, 172:103402.
- [Ke et al., 2014] Ke, S., Gong, J., Li, S., Zhu, Q., Liu, X., and Zhang, Y. (2014). A hybrid spatio-temporal data indexing method for trajectory databases. *Sensors*, 14(7):12990–13005.
- [Li et al., 2024] Li, B., Lu, Y., Li, Y., Zuo, H., and Ding, Z. (2024). Research on the spatiotemporal distribution characteristics and accessibility of traditional villages based on geographic information systems—a case study of shandong province, china. *Land*, 13(7).
- [Liang et al., 2024] Liang, H., Zhang, Z., Hu, C., Gong, Y., and Cheng, D. (2024). A survey on spatio-temporal big data analytics ecosystem: Resource management, processing platform, and applications. *IEEE Transactions on Big Data*, 10(2):174–193.
- [Liao et al., 2015] Liao, S., Chen, L., Li, J., Xiong, W., and Wu, Q. (2015). A spatio-temporal aggregation query method using multi-thread parallel technique based on regional division. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, II-4/W2:1–5.
- [Qian and Tian, 2024] Qian, S. and Tian, Z. (2024). A nearest neighbor query method for searching objects with time and location informations based on spatiotemporal similarity. *Evolutionary Intelligence*, 17:1–11.
- [Tao and Papadias, 2001] Tao, Y. and Papadias, D. (2001). Mv3r-tree: A spatio-temporal access method for timestamp and interval queries. In Proceedings of the 27th International Conference on Very Large Data Bases, VLDB '01, page 431–440, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [Tao et al., 2002] Tao, Y., Papadias, D., and Zhang, J. (2002). Cost models for overlapping and multiversion structures. *ACM Trans. Database Syst.*, 27(3):299–342.
- [Wang et al., 2024] Wang, H., Xia, J., Yang, Y., Wang, S., and Cao, J. (2024). Sts2anet: Spatio-temporal synchronized sliding attention network for accurate cross-day origin-destination prediction. In Onizuka, M., Lee, J.-G., Tong, Y., Xiao, C., Ishikawa, Y., Amer-Yahia, S., Jagadish, H. V., and Lu, K., editors, Database Systems for Advanced Applications, pages 186–202, Singapore. Springer Nature Singapore.
- [Wang et al., 2017] Wang, M., Xiao, M., Peng, S., and Liu, G. (2017). A hybrid index for temporal big data. *Future Generation Computer Systems*, 72:264–272.
- [Xie et al., 2024] Xie, J., Zhong, B., Mo, Z., Zhang, S., Shi, L., Song, S., and Ji, R. (2024). Autoregressive queries for adaptive tracking with spatio-temporal transformers. In 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 19300–19309.
- [Xiong et al., 2025] Xiong, R., Chen, Z., Pan, H., Liu, D., Sun, A., and Chen, N. (2025). Stspam: Software for intelligently analyzing and mining spatiotemporal processes based on multi-source

big data. *ISPRS International Journal of Geo-Information*, 14(2).

[Yang et al., 2024] Yang, C., Yin, Y., Zhang, J., Ding, P., and Liu, J. (2024). A graph deep learning method for landslide displacement prediction based on global navigation satellite system positioning. *Geoscience Frontiers*, 15(1):101690.

[Zhang et al., 2024] Zhang, W., Pan, W., Zhu, X., Yang, C., Du, J., and Yin, J. (2024). Identification of traffic flow spatio-temporal patterns and their associated weather factors: A case study in the terminal airspace of hong kong. *Aerospace*, 11(7).