Distributed Time Series Analytics

THÈSE N^{O} 7395 (2017)

PRÉSENTÉE LE 17 MARS 2017

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
LABORATOIRE DE SYSTÈMES D'INFORMATION RÉPARTIS
PROGRAMME DOCTORAL EN INFORMATIQUE ET COMMUNICATIONS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Tian GUO

acceptée sur proposition du jury:

Prof. B. Faltings, président du jury
Prof. K. Aberer, directeur de thèse
Prof. A. Bifet, rapporteur
Dr T. Papaioannou, rapporteur
Dr M. Rajman, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
Suisse
2017

The only easy day was yesterday.
— *No Easy Day*

Dedicated to

My dear parents and grandparents
for their unconditional love and support

&

The Swiss Alps
for everything it teaches me

Acknowledgements

The first person I want to thank must be my Ph.D. advisor, Prof. Karl Aberer. He did an excellent job in leading me throughout my PhD and gave me sufficient freedom and opportunities to concentrate on the topics of my interests. The discussion with him is always inspiring as well as offering insights into my research. I especially appreciate his patience during some tough period when my research progressed slowly in the presence of continuous paper rejections. I feel very lucky to finish my Ph.D. under the supervision of Karl in LSIR.

I also must thank all my colleagues whom I collaborated with during the course of my Ph.D., especially Thanasis Papaioannou, Saket Sathe, Zhixian Yan, Jean-Paul Calbimonte, Sofiane Sarni, Nguyen Quoc Viet Hung, Hao Zhuang and many. I would not achieve this without your guidance and collaboration. I thank all the colleagues from LSIR. Also, a special thanks should go to Chantal. As I did an internship in Germany, Chantal helped me sort out so many administrative issues.

I am also very grateful for my mentors of the internship in NEC Laboratories Europe. They are Mohamed Ahmed, Konstantin Kutzkov, Mathias Niepert and Zhao Xu. Thank you for providing such a precious opportunity to work with you. I gained valuable professional experience from you and greatly expanded my expertise during the internship.

I thank all my friends from the doctoral school in IC and the Chinese student community in EPFL, for their support and for all the great moments we spent together, including all our travels, hiking, football sports, adventures and parties and many others.

I am very thankful to the members of my thesis committee: Prof. Boi Faltings, Prof. Albert Bifet, Dr. Thanasis Papaioannou, Dr. Martin Rajman, for their insightful comments to improve my dissertation.

Finally, I would like to thank my parents for their love and for so many sacrifices that they made to support me during this long journey.

Lausanne, September 2016 Tian Guo

Abstract

In recent years time series data has become ubiquitous thanks to affordable sensors and advances in embedded technology. Large amount of time-series data are continuously produced in a wide spectrum of applications, such as sensor networks, medical monitoring, finance, IoT applications, news feeds, social networks, data centre monitoring and so on. Availability of such large scale time series data highlights the importance of of scalable data management, efficient querying and analysis. Meanwhile, in the online setting time series carries invaluable information and knowledge about the real-time status of involved entities or monitored phenomena, which calls for online time series data mining for serving timely decision making or event detection. In addition, in many scenarios data generated from sensors (environmental, RFID, GPS, etc.) are noisy and non-stationary in nature. In this thesis we aim to address these important issues pertaining to scalable and distributed analytics techniques for massive time series data. Concretely, this thesis is centered around the following three topics:

Managing and Querying Model-View Time Series.
As the number of sensors that pervade our lives significantly increases (e.g., environmental sensors, mobile phone sensors, IoT applications, etc.), the efficient management of massive amount of time series from such sensors is becoming increasingly important. The infinite nature of sensor data poses a serious challenge for query processing even in a cloud infrastructure. Traditional raw sensor data management systems based on relational databases lack scalability to accommodate large scale sensor data efficiently. Thus, distributed key-value stores in the cloud are becoming a prime tool to manage sensor data. On the other hand, model-view time series management, which stores the data in the form of modeled segments, brings the additional advantages of data compression and value interpolation. However, currently there are no techniques for indexing and/or query optimization of the model-view sensor time series data in the cloud. In Chapter 2, we propose an innovative index for modeled segments in key-value stores, namely KVI-index. KVI-index consists of two interval indices on the time and sensor value dimensions respectively, each of which has an in-memory search tree and a secondary list materialized in the key-value store. We show that the proposed KVI-index enables to perform efficient query processing upon modeled segments.

Mining Correlations in Streaming Time Series.
The dramatic increase in the availability of data streams fuels the development of many

distributed real-time computation engines (e.g., Storm, Samza, Spark Streaming, S4 etc.). In Chapter 3, we focus on a fundamental time series mining task in such a new computation paradigm, namely continuously mining dynamic (lagged) correlations in time series via a distributed real-time computation engine. Correlations reveal the hidden and temporal interactions across time series and are widely used in scientific data analysis, data-driven event detection, finance markets and so on. Existing correlation mining approaches specific for the centralized environment fail to handle the data shuffling issue in the distributed setting. We propose the P2H framework consisting of a parallelism-partitioning based data shuffling and a hypercube structure based computation pruning method, so as to enhance both the communication and computation efficiency for mining correlations in the distributed context.

Learning from Noisy and Non-Stationary Data.
In numerous real-world applications large datasets collected from observations and measurements of physical entities are inevitably noisy and contain outliers. Meanwhile, recently distributed machine learning algorithms arise as powerful tools for learning massive volumes of data. However, the outliers in such large and noisy datasets can dramatically degrade the performance of standard distributed machine learning approaches such as regression trees. To this end, in Chapter 4 we present a novel distributed regression tree approach that utilizes robust regression statistics, statistics that are more robust to outliers, for handling large and noisy data. We propose to integrate robust statistics based error criteria into the regression tree. A data summarization method is developed and used to improve the efficiency of learning such a robust regression tree in the distributed setting.
On the other hand, in the online and dynamic environment, besides outliers time series is often complicated with change points due to the changing behaviour and distribution of the data. Such outliers and change points can lead the learned models to deviate from the underlying true patterns of data and to fail to provide reliable forecasting. In Chapter 5 we present an adaptive gradient learning method for recurrent neural networks (RNN) to forecast streaming time series in the presence of both outliers and change points. We explore the local features of time series to automatically weight the gradients of the loss of the newly available observations with distributional properties of the data in real time, such that our RNN model is able to be robust to outliers as well as adapting to change points.

Keywords: *time series data mining, distributed computing, time series data management, recurrent neural network, robust regression, decision tree, data summarization.*

Résumé

Ces dernières années, les séries temporelles sont devenues omniprésentes grâce à des capteurs au prix abordable et des avancés technologiques dans le domaine des systèmes embarqués. Une grande quantité de séries temporelles sont produits en continu dans un large éventail d'application, telles que les réseaux de capteurs, la supervision médicale, les finances, les applications de l'Internet des choses (IoT), les flux RSS, les réseaux sociaux, la supervision des centres de données, etc. La disponibilité de ces séries temporelles à grande échelle met en évidence l'importance de la gestion de données à grande échelle, et d'outils d'analyse efficaces. Par ailleurs, dans le contexte du traitement en continu, les séries temporelles contiennent des informations et des connaissances cruciales concernant les propriétés des entités concernées ou des phénomènes surveillés, et cela en temps réel. Ceci motive le traitement en temps réel des séries temporelles afin de supporter une prise de décision en temps adéquat ou la détection d'événements. De plus, dans de nombreux cas de figures, les données générées par les capteurs (environnement, RFID, GPS, etc.) contiennent du bruit qui est, par natures, non stationnaires. Dans cette thèse, nous nous efforçons d'apporter des solutions à ces problèmes importants concernant les techniques d'analyse distribuées à grande échelle des séries temporelles. Plus concrètement, cette thèse est centrée autour des trois thèmes suivants :

Gestion et requêtes sur des séries temporelles "Model-View".
Récemment, le nombre de capteurs qui nous entourent a augmenté de manière significative (e.g., capteurs environnementaux, capteurs de smartphones, les applications de l'IoT, etc). La gestion et le traitement efficace de ces séries temporelles massives deviennent primordiaux. La nature infinie des données collectées par ces capteurs pose un réel défi pour le traitement des requêtes, même dans une infrastructure telle que le cloud. Les systèmes de gestion de données brutes traditionnels, basés sur les bases de données relationnelles manquent d'évolutivité pour accueillir de telles quantités de données de manière efficace. Ainsi, les systèmes de stockages distribués, basés sur des paires de clés/valeurs, et situés dans le cloud, deviennent des outils essentiels afin de gérer les données de capteurs. D'autre part, la gestion des séries temporelles en model-view, qui stockent les données sous forme de segments modélisées, apporte des avantages supplémentaires comme la compression de données ou l'interpolation. Cependant, il n'existe actuellement pas de techniques d'indexation et / ou d'optimisation de requêtes des séries temporelles de capteurs en model-view dans le cloud. Dans le second

chapitre, nous proposons un index innovant pour les segments modélisés dans les systèmes de stockage basés sur les pairs clé/valeurs, à savoir KVI-index. KVI-index se compose de deux intervalles d'indices. Un portant sur la dimension temporelle et l'autre sur la dimension des valeurs du capteur. Chacun de ces indices est stocké en mémoire sous forme d'arbre, ainsi que sous forme d'une liste secondaire matérialisée dans le système de stockage clés/valeurs. Nous montrons que le KVI-index proposé permet d'effectuer un traitement efficace des requêtes sur les segments modélisés.

Exploration de Corrélations en Flux de Séries.
L'augmentation spectaculaire de la disponibilité des flux de données encourage le développement de nombreux systèmes de gestion de données distribués en temps réel (par exemple, Storm, Samza, Spark streaming, etc.). Dans le chapitre 3, nous nous concentrons sur une tâche fondamentale d'extraction de données de séries temporelles dans un nouveau paradigme de calcul, à savoir l'exploitation continue et dynamique de corrélations par un système distribué de traitement de données en temps réel. Ces corrélations révèlent les interactions cachées et temporelles entre séries temporelles, et sont largement utilisés dans l'analyse scientifique des données, la détection d'événements pilotés par les données, les marchés financiers, etc. Les approches existants de'exploration de données et data mining de corrélations, suivent an modèle centralisé et ne parviennent pas à gérer la question de brassage de données dans le cadre distribué. Nous proposons le framework P2H, consistant d'un brassage de données basé sur un mécanisme qui combine parallélisme et partitionnement, et un calcul basé sur une structure de hypercube, qui vise à améliorer à la fois la communication et de l'efficacité de calcul pour exploiter les corrélations dans le contexte distribué.

Apprendre des données bruitées et non-stationnaires.
Dans de nombreuses applications du monde réel, grands ensembles de données collectées à partir d'observations et de mesures des entités physiques, sont inévitablement bruyants et peuvent contenir des valeurs aberrantes. Au même temps, des algorithmes d'apprentissage distribués se posent comme des outils puissants pour l'apprentissage des volumes massifs de données. Cependant, les valeurs aberrantes dans ces grands et bruyants ensembles de données peuvent considérablement dégrader les performances des approches d'apprentissage machine standard distribués tels que les arbres de régression. A cet effet, dans le Chapitre 4, nous présentons une approche basé sur un arbre de régression distribué qui utilise la régression robuste et des statistiques qui sont plus robustes aux valeurs aberrantes. Nous proposons d'intégrer des critères d'erreur statistiques robustes basés sur l'arbre de régression. Une technique de résumé des données a été développée et utilisée pour améliorer l'efficacité de l'apprentissage d'un tel arbre de régression robuste dans le cadre distribué. Dans le chapitre 5, nous présentons une méthode adaptative d'apprentissage de gradient pour les réseaux de neurones récurrents (RNN) pour effectuer des prédiction de flux de séries temporelles, en présence de valeurs aberrantes. Nous explorons les caractéristiques locales des séries temporelles pour pondérer automatiquement les gradients de la perte des observations nouvellement

disponibles avec des propriétés de répartition des données en temps réel, de telle sorte que notre modèle RNN est capable d'être robuste aux valeurs aberrantes, ainsi que l'adaptation au changement des points.

Mots clefs : *temps l'extraction de données de la série, informatique distribuée, le temps de gestion des données de la série, le réseau neuronal récurrent, régression robuste, l'arbre de décision, récapitulation des données*

Contents

List of Figures

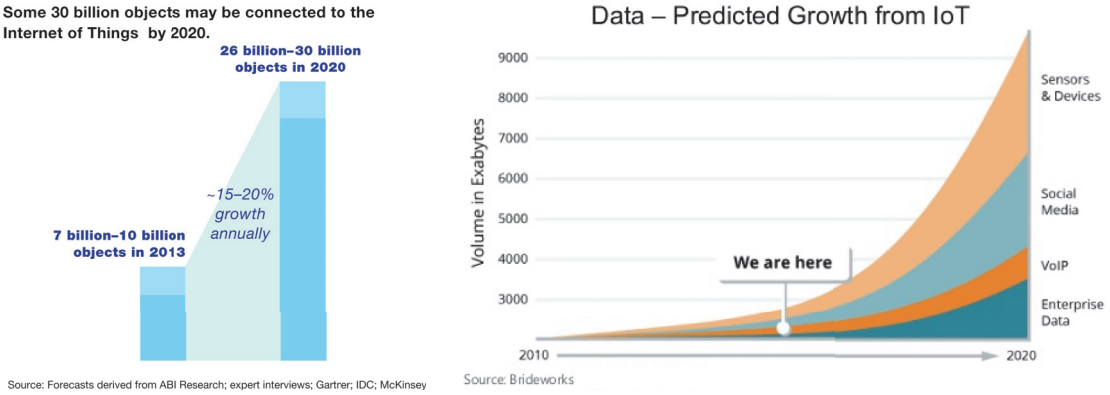
List of Tables

1 Introduction

There is probably no pleasure equal to the pleasure of climbing a dangerous Alp; but it is a pleasure which is confined strictly to people who can find pleasure in it.
— Mark Twain

Time series data occur naturally in countless domains including medical analysis, financial analysis, online text, sensor network monitoring and social activity mining and are widely used for capturing the status and dynamics of monitored entities or phenomenons. This trend is boosted by the fact that with the decreasing hardware cost and increasing demand for data acquisition in a variety of applications, our planet is undertaking the vast deployment of sensors embedded in different devices that monitor various phenomena, e.g., environmental indices, human activities, urban traffic, air pollution, and so on. For instance, the Internet of Things (IoT) produces an enormous amount of data every day: from smart shirts for athletes to smart meters in plants. As is shown in Figure 1.1, the amount of objects connected by IoT and the generated data are expected to experience considerable growth and consequentially

almost every object/entity in our daily life will have the ability to emit data. For instance, in a typical wind power farm[1], wind turbines equipped with hundreds of sensors can produce 200 terabytes of data that has to be processed in real time. Analyzing such massive data can enable energy providers to adjust the blades to maximize output as well as forecasting with greater accuracy and predictability. In addition, it is becoming increasingly common for companies and organizations to collect very large amount of data over time to improve the service and operations. For example, in the data centres of Internet companies, e.g., Google, Facebook, Yahoo, etc., many measurements on server performance are collected every hour for each of thousands of servers and used to identify servers that are behaving unusually. In one word, large scale time series data processing and analytics are becoming a big issue nowadays.



(a) (b)

Figure 1.1 – (a) Growth of objects connected by IoT; (b) Growth of data created in IoT.

In the meanwhile, data analytics techniques are getting more and more distributed in the sense that the distributed nature of diverse data sources as well as the quickly increasing amount have led to the emergence of many distributed, fault-tolerant computation systems for large scale data management and analysis as is shown in Figure 1.2. Conventional time series analytics approaches intended for centralized computing environment assumes that the volume of data needed for analysis can be fit into memory on a single machine and therefore is inadequate for contemporary high volume and high velocity time series data in terms of both communication and computation capacities. In addition, since most of large datasets are collected from observations and measurements of physical entities and events, such data is inevitably noisy and skewed in part due to equipment malfunctions or abnormal events [65, 67, 136]. Traditional learning approaches based on the assumption of small and clean training datasets are problematic.

[1] http://www.forbes.com/sites/mikekavis/2015/02/13/envision-energy-leverages-iot-technologies-to-optimize&-renewable-energy/#789aa167f9ea

Figure 1.2 – Typical distributed data processing systems

1.1 Motivation and Challenges

In face of the aforementioned observations in time series data analytics, a question arises naturally: how can we leverage distributed computing for large scale time series data mining and analytics? In this thesis, we aim to tackle this problem in three aspects: first, as large amount of time series data is continuously generated by massive devices, there is an eminent demand for scalable management and querying of time series for efficient analytics; second, in the online applications where decision-making is based on the real-time produced time series, discovering timely knowledge in data streams plays an important role, and thus an efficient online distributed mining approach over massive time series streams is indispensable; third, considering time series data in the real world is inevitably noisy and non-stationary, robust learning, which enables to model the true pattern behind data, is an essential for providing reliable forecasting.

In the following, we will elaborate on the challenges in above three aspects of distributed time series analytics.

- **Scalable Management and Querying of Model-View Time Series:** In traditional relational sensor data management systems, raw discrete observations of time series yielded by sensors are taken as the first citizen, which leads to a number of problems. On one hand, in order to perform analysis of raw time series data, users usually adopt other third-party modeling tools (e.g., Matlab, R and Mathematica), which involve of tedious and time-consuming data extract, transform and load (ETL) processes [118]. Moreover, such data analysis tools are usually used for static data sets and therefore cannot be applied for online processing of sensor data streams. On the other hand, unbounded data streams often have missing values and unknown errors, which also poses great challenges for traditional raw time series data management. To this end, model-view time series data management, which stores the data in the form of modeled segments, has attracted more and more attention [37, 118, 131]. It brings the additional advantages of data compression and value interpolation and these models can then be stored, retrieved, and queried, as required.

Traditional model-based sensor data management approaches mostly employ the relational data model and process queries based on materialized views [37, 131] on top of segment models of time series data. However, as the amount of data produced by a wide spectrum of sensing devices is exponentially increasing and the real-time production of such sensor data requires the data management system to be able to handle high-concurrent model-view data, this becomes difficult for traditional relational databases to realize. To this end, recent prevalent distributed store and computing techniques provide a promising way to manage model-view sensor data [3, 24, 35, 54, 55]. However, currently there are no techniques for indexing and/or query optimization of the model-view time series data in the contemporary distributed environment; full table scan is needed for query processing in the worst case.

- **Distributed Mining of Correlation in Massive Time Series Streams:** Recent advances in embedded sensor technology (e.g., smart meters, smart watches and mobile phones, the Internet of Things) are driving a massive growth of time series streams reflecting the real-time status of involved entities. These entities often evolve over time, influence each other and present correlated behaviours [7, 28, 82, 154]. For instance, one entity may cause a series of significant value or trend changes in the time series of other entities with some time lags. Dependencies of this kind are referred to as (lagged) correlation [22, 114, 143] and commonly observed in diverse scenarios including Internet of Things [149], intercorrelated financial markets [104], climate changes of neighbouring areas [143], social network interactions [112], and functional-dependent servers in data centres [88, 137]. It is a fundamental piece of knowledge for a variety of applications such as event and correlation anomaly detection, trend prediction, among others [7, 28, 82, 143].

  Mining such correlations in high volume and high velocity time series streams based on the distributed real-time computation paradigm (e.g., S4, Storm, Spark Streaming, Samza, etc.) requires a computation and communication efficient solution. Contrary to static time series mining on a standalone machine, in the highly dynamic environment as new observations of time series continuously arrive, computation for detecting correlations at a certain time instant should be finished before the subsequent new data arrives. Otherwise the system would lag further and further over time and cannot report timely correlation. On the other hand, the distributed environment complicates the problem more. The observations of time series are continuously distributed into different computing nodes of a distributed real-time computation engine and each node has no knowledge about the timely characteristics of time series other nodes receive. A brute-force way is that each node has to replicate and shuffle its local time series (sliding windows) among the nodes, so as to find qualified correlations. Such a method could generate quadratic computation and communication costs w.r.t. the number of time series under processing at worst (i.e., similar to the idea of cross-join using

MapReduce [107, 116]). High communication cost in the distributed context leads to prolonged processing latency and even bottlenecks, due to the increasing time spent on sending, receiving and parsing data [147].

- **Robust Learning over Noisy and Non-stationary Data:** As increasing amount of data is collected from devices monitoring physical entities in the real world, such large scale data is inevitably noisy and skewed in part due to equipment malfunctions and abnormal events [65, 67, 76, 136]. In particular, time series yielded in monitoring applications is usually contaminated by outliers that are abrupt observations deviating from the behaviour of the majority [76, 134]. A typical example might be cyber-attacks, which are often shown as anomalies in time series monitoring some measurements of network traffic. On the other hand, real time series data often bears change points as well, which indicate that the behaviour and hidden data distribution of the time-series significantly vary after change points.

  The effect of such noisy and non-stationary data on conventional learning (regression) approaches is often disastrous. Ordinary (distributed) regression learning minimizes the squared mean error objective function and outputs the conditional mean values as predictions, which is especially problematic and sensitive to noisy data [45, 65, 132]. As a consequence, learnt model can neither identify the true patterns in data, nor provide reliable predictions [64, 65, 67, 132, 136]. Furthermore, it is very common that outliers and change points mingle in real time series data, and it is non-trivial for the learning process to distinguish them, namely being robust against outliers as well as quickly adapting to the data after change points. With these challenges, robust learning, which aims to model the true pattern and distribution of data, is essential.

1.2 Contributions

In order to address the aforementioned challenges in distributed time series analytics, in this thesis we proposed several solutions summarized as follows.

For scalable management and efficient querying of model-view time series, our contribution includes:

- Innovative interval index: We propose an innovative interval index for model-view sensor data management in key-value stores, referred to as *KVI-index*. *KVI-index* is a two-tier structure consisting of one lightweight and memory-resident binary search tree and one index-model table materialized in the key-value store. This composite index structure can dynamically accommodate new sensor data segments very efficiently.
- Hybrid model-view query processing: After exploring the search operations in the in-

memory structure of the KVI-index for range and point queries, we introduce a hybrid query processing approach that integrates range scan and MapReduce to process the segments in parallel.

- Intersection search: We introduce an enhanced intersection search algorithm (*iSearch+*) that produces consecutive results suitable for MapReduce processing. We theoretically analyze the efficiency of (*iSearch+*) and find the bound on the redundant index nodes that it returns.

- Experimental evaluation: Our framework has been fully implemented, including the online sensor data segmentation, modeling, KVI-index and the hybrid query processing, and it has been thoroughly evaluated against a significant number of alternative approaches. As experimentally shown based on real sensor data, our approach significantly outperforms in terms of query response time and index updating efficiency all other ones for answering time/value point and range queries.

The second contribution of this thesis is regarding the distributed mining of correlations over streaming time series.

- We formally define the problem of continuously mining correlations in massive time series based on a distributed real-time computation engine (DisCoM problem). We propose the framework P2H, which aims to optimize both communication and computation cost for DisCoM problem.

- In P2H, we design a novel data shuffling technique parallelism-partitioning based data shuffling, referred to as P^2-data-shuffling, which is able to adaptively replicate and shuffle the sliding window of a time series only to the computing nodes containing its correlation partners, thereby dramatically decreasing the communication cost.

- We propose a δ-hypercube structure based correlation computation approach to prune unnecessary computation over the data shuffled by P^2-data-shuffling. Theoretical cost analysis of the communication and computation is provided.

- Furthermore, we design a probabilistic model for correlation inference by considering both the temporal and co-occurrence dependence of correlation occurrence in P2H.

- P2H framework and a variety of baseline approaches are implemented based on Apache Storm, a widely used distributed real-time computation engine. Extensive experiments on both synthetic and real datasets demonstrate the effectiveness and efficiency and effectiveness of our approach. in terms of peak capacity, communication cost, processing latency and pruning power.

As for robust learning over noisy and non-stationary data, we propose two approaches respectively specialized for offline and online settings as:

In the offline environment where large noisy datasets are given, we focus on the distributed regression tree, which is at the core of several highly successful distributed machine learning models and broadly used in a wide spectrum of applications. Our objective is to enhance the robustness as well as the training efficiency of the distributed regression tree over noisy datasets.

- We define the distributed robust regression tree employing robust loss functions and identify the difficulty in designing an efficient training algorithm for the distributed robust regression tree.
- We propose a novel distributed training framework for the robust regression tree, which consists an efficient data summarization method on distributed data and a tree growing approach exploiting the data summarization to evaluate robust loss functions. Such data summarization is transmitted to the master node with bounded communication cost.
- The proposed distributed robust regression tree and baselines are implemented based on Apache Spark. Extensive experiments on both synthetic and real datasets demonstrate the efficiency and effectiveness of our approach.

As for the online setting, observations of time series possibly contaminated by both outliers and change points continuously arrive and a online learning process is expected to incrementally update the learnt model using new arriving data and to provide accurate forecasting. Long short-term memory neural networks (LSTM), a class of recurrent neural networks, have been proved to be an very effective tool for time series learning and forecasting [20,51,77,92,129,142] and additionally support online learning. We present an adaptive gradient learning method for LSTM networks [50, 51, 60], to make the streaming time series forecasting robust to outliers and change points. The concrete contribution is listed as follows:

- We model time series with LSTM networks, and use a stochastic gradient descent (SGD) based method to learn the model from the streaming time series. As novel observations arrive, the model parameters are updated in an online mode according to the gradients of the loss of the newly available data. We identify that with the standard SGD method, outlier observations leads to the updated model deviating from the normal patterns and producing oscillated incorrect predictions until the adverse effect is gradually corrected by the following normal observations.

- To solve the problem, we explore the local features of time series to weight the gradients of the learning method with distributional properties of the local data in time series. Such a gradient weighting mechanism enables LSTM to avoid deviating from the normal behaviour in the presence of outliers as well as quickly adapting to changing time series.

- The experimental analysis on synthetic and real datasets demonstrates the performance of the proposed learning method for streaming time series forecasting in the presence of anomalies and change points.

1.3 Thesis Organization

We begin by describing scalable model-view time series data management and querying in the the distributed environment in Chapter 2. This is followed by a presentation of the P2H framework for online mining of correlations in massive time series streams in Chapter 3. The robust learning topic is discussed in Chapter 4 and Chapter 5. The issue of distributed robust regression tree is covered in Chapter 4. Then, we introduce the online robust learning algorithm of recurrent neural networks on non-stationary and noisy time series streams in Chapter 5. Lastly, we summarize and conclude this thesis in Chapter 6.

1.4 Selected Publications

This thesis is mainly based on the following published papers:

- T. Guo, T. G. Papaioannou and K. Aberer. Model-View Sensor Data Management in the Cloud. IEEE International Conference on Big Data 2013 (IEEE BigData), Santa Clara, California, USA, 2013. (Chapter 2)

- T. Guo, T. G. Papaioannou and K. Aberer. Efficient Indexing and Query Processing of Model-View Sensor Data in the Cloud. Journal on Big Data Research, Elsevier, 2014. (Chapter 2)

- T. Guo, S. Sathe and K. Aberer. Fast Distributed Correlation Discovery Over Streaming Time-Series Data. 24th International Conference on Information and Knowledge Management (CIKM). Melbourne, Australia, 2015. (Chapter 3)

- T. Guo, K. Kutzkov, M. Ahmed, J. Calbimonte and K. Aberer. Efficient Distributed Decision Trees for Robust Regression. European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD), Riva del Garda, Italy, 2016. (Chapter 4)

- T. Guo, Z. Xu, X. Yao, H. Chen, K. Aberer and K. Funaya. Robust Online Time Series Prediction with Recurrent Neural Networks. IEEE/ACM SIGKDD International Conference on Data Science and Advanced Analytics (DSAA) (to appear). Montreal, Canada, 2016. (Chapter 5)

2 Efficient and Scalable Querying of Model-View Time Series Data

The more comfortable we become with being stupid, the deeper we will wade into the unknown and the more likely we are to make big discoveries.
— Martin A. Schwartz, *The importance of stupidity in scientific research*

2.1 Introduction

Recent advances in sensor technology have enabled the vast deployment of sensors embedded in user devices that monitor various phenomena for different applications of interest, e.g., air/electrosmog pollution, radiation, early earthquake detection, soil moisture, permafrost melting, etc. The data streams generated by a large number of sensors are represented as time series in which each data point is associated with a time-stamp and a sensor value. These raw discrete observations are taken as the first citizen in traditional relational sensor data management systems, which leads to a number of problems. On one hand, in order to perform

analysis of the raw sensor data, users usually adopt other third-party modeling tools (e.g., Matlab, R and Mathematica), which involve of tedious and time-consuming data extract, transform and load (ETL) processes [118]. Moreover, such data analysis tools are usually used for static data set and therefore cannot be applied for online processing of sensor data streams. On the other hand, unbounded sensor data streams often have missing values and unknown errors, which also poses great challenges for traditional raw sensor data management.

To this end, various model-based sensor data management techniques [37, 103, 118, 131] have been proposed. Model-view sensor data management leverage time series approximation and modeling techniques to segment the continuous sensor time series into disjoint intervals such that each interval can be approximated by a kind of model, such as polynomial, linear or SVD. These models, for all the intervals, (or segment models) exploit the inherent correlations (e.g. with time or among data streams) in the segments of sensor time series and approximate each segment by a certain mathematical function within a certain error bound [21, 27, 38, 57, 69]. Then, one can only materialize the models of the segments instead of the raw data and harvest a number of benefit:

First, model-view sensor data achieves compression over raw sensor data and therefore requires less storage overhead [16, 36, 119]. Second, due to the sampling frequency or sensor malfunction, there may be some missing values at some time points. If one query involves such time points, then the relevant segment model can be used to estimate the values [16, 36, 119]. In some degree, model-view sensor data increases the data availability for query processing. Third, there usually exist outliers in raw sensor data, which has negative effect on the query results. Model-view sensor data removes the outliers in each interval via the segment model and historical data on upper and lower data bounds, thereby diminishing the effect of outliers in query results. Fourth, regarding the similarity search or pattern discovery in sensor time-series mining, the segment-based time series representation is a powerful tool for dimension reduction and search space pruning [27, 68, 126].

However, proposed model-based sensor data management approaches mostly employ the relational data model and process queries based on materialized views [37, 131] on top of the modeled segments of sensor data. Nowadays, the amount of data produced by sensors is exponentially increasing. Moreover, the real-time production of sensor data requires the data management system to be able to handle high-concurrent model-view sensor data from massive sensors and this is difficult for traditional relational database to realize. To this end, recent prevalent cloud store and computing techniques provide a promising way to manage model-view sensor data [3, 24, 35, 54, 55].

The main focus of this paper is on how to manage the segment models of sensor data, namely model-view sensor data with the newly emerging cloud stores and computing techniques rather than how to explore more advanced sensor data segmentation algorithms.

In our approach, we exploit key-value stores and the MapReduce parallel computing paradigm [24, 49], two significant aspects of cloud computing, to realize indexing and querying model-view sensor data in the cloud. We characterize the modeled segments of sensor time series by the time and the value intervals of each segment [54, 55]. Consequently, in order to process range or point queries on model-view sensor data, our index in the cloud store should excel in processing interval data. Current key-value built-in indices do not support interval-related operations. The interval index for model-view sensor data should not only work for static data, but it should be dynamically updated based on the new arriving segments of sensor data. If traditional batch-updating or periodical re-building strategy was applied here [71, 121], then the high speed of sensor data yielding might lead to a large size of the new unindexed data, even in short time periods and to significant index updating delay as well. As a result, the performance of queries involving both indexed and unindexed data would degenerate greatly. Therefore, the interval index in the cloud store should be able to insert an individual new modeled segment in an online manner.

The contributions of this paper are summarized as follows:

- Innovative interval index: We propose an innovative interval index for model-view sensor data management in key-value stores, referred to as *KVI-index*. *KVI-index* is a two-tier structure consisting of one lightweight and memory-resident binary search tree and one index-model table materialized in the key-value store. This composite index structure can dynamically accommodate new sensor data segments very efficiently.
- Hybrid model-view query processing: After exploring the search operations in the in-memory structure of the KVI-index for range and point queries, we introduce a hybrid query processing approach that integrates range scan and MapReduce to process the segments in parallel.
- Intersection search: We introduce an enhanced intersection search algorithm (*iSearch+*) that produces consecutive results suitable for MapReduce processing. We theoretically analyze the efficiency of (*iSearch+*) and find the bound on the redundant index nodes that it returns.
- Experimental evaluation: Our framework has been fully implemented, including the online sensor data segmentation, modeling, KVI-index and the hybrid query processing, and it has been thoroughly evaluated against a significant number of alternative approaches. As experimentally shown based on real sensor data, our approach significantly outperforms in terms of query response time and index updating efficiency all other ones for answering time/value point and range queries.

The remainder of this paper is organized as follows: Sec. 2.2 summarizes some related work

on model-view sensor data management, interval index and index-based MapReduce optimization approaches. In Sec. 2.3, we provide a brief description of sensor-data segmentation, querying model-view sensor data and the necessity to develop interval index for managing model-view sensor data in key-value stores. The detailed designs of our innovative KVI-index and the hybrid query processing approach are discussed in Sec. 2.4 and 2.5 respectively. Then, in Sec. 2.6, we present thorough experimental results to evaluate our approach with traditional query processing ones on both raw sensor data and modeled data segments. Finally, in Sec. 2.7, we conclude our work.

2.2 Related Work

Time series segmentation is an important research problem in the areas of data approximation, data indexing and data mining. A lot of work has been devoted to exploit different types of models to approximate the segments of time series, such that the pruning and refinement framework can be applied to this segment-represented time series for the pattern matching or similarity search [27, 68]. Some other researchers proposed techniques for managing the segment models that approximate sensor time series in relational databases. MauveDB designed a model-based view to abstract underlying raw sensor data; it then used models to project the raw sensor readings onto grids for query processing [37]. As opposed to MauveDB, FunctionDB only materializes segment models of raw sensor data [131]. Symbolic operators are designed to process queries using models rather than raw sensor data. However, both approaches in [37] and [131] do not take into account applying indices to improve query processing. Moreover, their proposed approach focuses on managing static dataset of time series rather than dynamic time series.

Also, each segment of time series could be characterized by its time and value intervals. Then, one should consider employing an interval index for processing queries over model-view sensor data. Two common used indices for interval data are segment tree [71] and interval tree [11]. As for segment tree, it is essentially a static data structure that needs an initialization process to construct elementary intervals based on the initial dataset [71]. Once a new interval outside of the domain of current segment tree arrives, the elementary intervals should be rebuilt, which is not suitable for the real time nature of sensor data streams [71]. Regarding the interval tree, individual interval is not divided and replicated during the insertion phase as in the segment tree and therefore the storage overhead is linear to the number of intervals to index [11, 19, 41]. However, it is a memory-oriented structure.

Some efforts [11, 73, 74, 121] have also been done to externalize these in-memory index data structures. The relational interval tree (RI-tree) [73] integrates interval tree into relational tables and transforms interval queries into SQL queries on tables. This method makes efficient use of built-in B+-tree indices of RDBMS. Nevertheless, in this paper, we aim to design an

interval index structure for model-view sensor data that is compatible with key-value stores and distributed query processing in the cloud.

In [99], they proposed an approach that enables key-value stores to support query processing of multi-dimensional data via integrating space filing order into row-keys, which is not fit for interval data. An index for multi-dimensional point data is proposed in [135] based on P2P overlay network, which is a different underlying architecture from our adopted key-value store. The latest effort to develop interval indices in the cloud utilizes MapReduce to construct a segment tree materialized in the key-value store [121]. This approach outperforms the interval query processing provided by HBase (http://hbase.apache.org/) and Hive (http://hive.apache.org/). However, the segment tree utilized in [121] is essentially a static index, as is also the case with [71]. Therefore, an index re-building phase needs to be periodically executed to include new data.

MapReduce parallel computing is an effective tool to process large scale of sensor data in cloud stores, but conventional MapReduce always conducts trivial full scan on the whole data set for the queries of any selectivity [35]. In order to enable MapReduce and indices to collaborate for query processing, many researchers proposed to integrate index techniques into MapReduce framework to avoid full data scan for low-selective queries [39, 40, 42, 62, 148]. One prior work is to construct a B+-tree over static data set in a distributed file system (e.g. HDFS) and materialize the index also in HDFS [148]. Then, the query processor can process this index file, which involves of multiple MapReduce jobs to access different layers of the tree and therefore is not efficient enough [148]. Moreover, the proposed HDFS based B+-tree is only effective for point data rather than interval data [148].

The authors in [39] integrate indices into each file split of the data file in HDFS, such that mappers can use the indices to only access predicate-qualified data records in each split. In [40], indices applicable to queries with predicates on multiple attributes were developed by indexing different record attributes in different block replicas of data files. These two kinds of index access methods in MapReduce are both on record-level. If we integrate an interval index into each file split guided by this direction, the MapReduce data preparation, starting overhead and Mapper waves are actually not decreased much and therefore the performance does not significantly improve. On the other hand, the update in one file split requires to rebuilt the whole local index of the split thereby having high latency for segment insertions. In [42], a split-level index is designed to decrease MapReduce framework overhead. As compared to above record-level indices in [39, 40], split-level indices eliminate irrelevant file splits before launching mappers, and the data transferring and starting up overheads are further decreased. This kind of approach is more efficient for decreasing the map function invocations via overriding the data reader of MapReduce [39, 40], as compared to record-level optimizations.

2.3 Overview of model-view sensor data management

In this section, we discuss certain important issues for managing model-view sensor data in a key-value store. First, we explain what is model-view sensor data. Then, we discuss possible storage schemas for model-view sensor data and explain the necessity to develop a special index for it in key-value stores. Last, we describe the query types of our focus and some particular techniques for processing model-view sensor data queries.

2.3.1 Sensor time series segmentation

Sensor time series segmentation is a type of algorithms that fragment a time series stream into disjoint segments, and then approximate each data segment by a mathematical function or model, such that a specific error bound is satisfied [21, 27, 38, 57, 69]. Query processing can then be performed on the materialized segments instead of the raw sensor data, as in [131]. Sensor data segmentation and modeling algorithms have been extensively studied in [21, 27, 38, 57, 69, 103]. As segmentation algorithms are not the focus of this paper, we

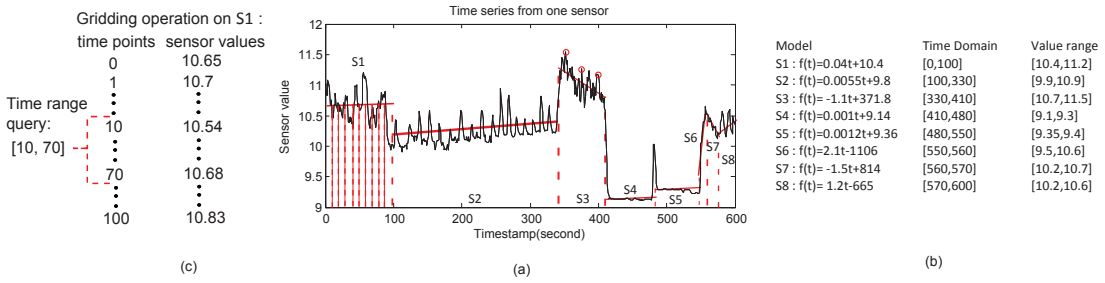


Figure 2.1 – Model-view time series data

only present a general framework in Alg. 1 for online time series segmentation [69], which is used in the dataset preparation phase of our sub-sequential experimental evaluation in Sec. 5.5. Table 2.1 summarizes the list of symbols that we employ for model segments. This

Algorithm 1 Sensor time series segmentation

Input: v_t, t, /* value and associated time point of one sensor reading */ ,$error\_bound$
1: **if** currentSeg_error(anchor , t) < error_bound **then**
2: $seg_{[anchor,t]}$ = segmentModel(v_{anchor},$\cdots$,v_t);
3: # /* approximate the sensor values between the time range $[anchor, t]$ with the specified type of model (e.g., constant, linear, polynomial, etc.) */
4: **else**
5: l_t=anchor;
6: l_r=t-1;
SegMaterialization($seg_{[anchor,t]}$)
/* output this new produced segment $seg_{[anchor,t]}$ for materialization */
anchor=t;
7: **end if**

sensor time series segmentation program is invoked each time a new sensor reading comes. It first checks if the segment model over the sensor readings from the anchor time point to current time, namely in range $[anchor, t]$, satisfies the error bound. If yes, the current segment model is updated to $seg_{[anchor,t]}$. If no, the current time t is supposed to be the starting point $anchor$ of a new segment. Then, output the segment $seg_{[anchor,t-1]}$ for the materialization process for which our following designed index approach is responsible. The segment $seg_{[anchor,t-1]}$ consists of the following information: time interval $[l_t, r_t]$, value range $[l_v, r_v]$ and model formula (for instance, it could be a sequence of coefficients for each item of one polynomial function). Under this framework, users can choose different categories of models to approximate the sensor data in segments. For instance, a time series shown in Fig. 2.1(a) is transformed into a sequence of linear functions with time dimension as the independent variable, which is depicted in Fig. 2.1(b). The associated value range in Fig. 2.1(b) indicates the values one segment model can cover within the time interval. Such set of functions, each with its associated $[l_t, r_t]$ and $[l_v, r_v]$ for the sensor data segments are referred to as the model-view sensor data over the original raw sensor data observations.

Symbol	Semantics
l_t	Beginning timestamp of one segment model
r_t	Ending timestamp of one segment model
l_v	Minimum value of one segment model's value range
r_v	Maximum value of one segment model's value range
$p_0^i \cdots p_n^i$	Coefficient of each item in polynomial model

Table 2.1 – Symbol List

2.3.2 Storage model in key-value stores

HBase overview

A table in HBase consists of a set of regions, each of which stores a sequential range partition of the row-key space of the table [24, 49]. An HBase cluster consists of a collection of servers, called region servers, which are responsible for serving a subset of regions of one table. The key-based data access method of HBase is realized by a three layered B+ tree maintained among the nodes of HBase cluster. The two upper levels referred to as the ROOT and META regions are kept in the master node of an HBase cluster. The ROOT region plays the role of root node of one B+ tree, while the META table maintains a mapping of the regions of a table

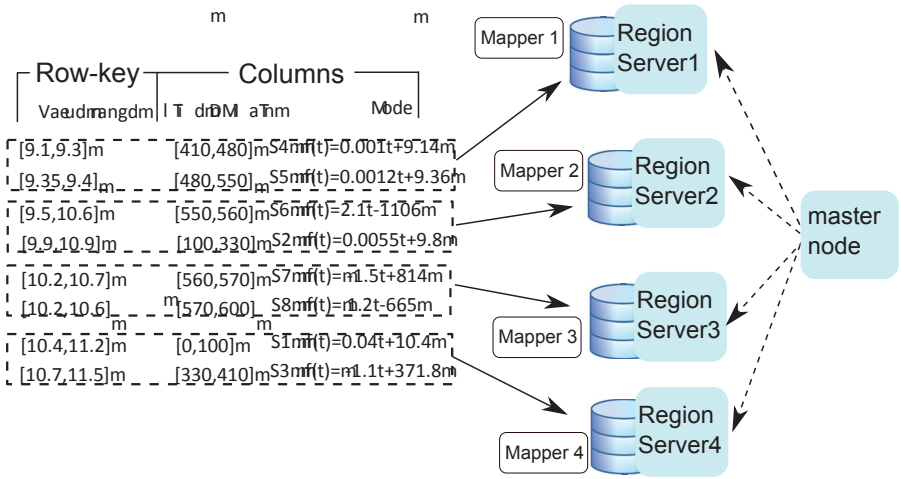


Figure 2.2 – Model-view time series in HBase key value store

to region servers. The regions distributed among the other nodes constitute the lowest level of the B+-tree index and store the real data of a table. If a region's size exceeds a configurable limit, HBase can dynamically split it into two sub-regions, which enables the system to grow dynamically as data is appended. Regarding a MapReduce job on a table in HBase, the number of mappers is equal by default to that of regions of the table to process, which means that each mapper processes one region, as shown in Fig. 2.2. The number of reducers is configurable programmatically. And HBase allows MapReduce to only process key-based table partitions of interest, which can avoid trivial full-scan on a table for low-selective queries [24, 49].

Interval-index in HBase

For the model-view sensor data in the key-value store, the time interval, the value range and the model formula (e.g. polynomial coefficients) can fully approximate one data segment. There are different possible ways to organize row keys and columns for storing and querying model-view sensor data.

One idea for storing segments in key-value stores could be to do it similarly to that of the raw sensor data management system such as Open-TSDB [3], which takes the time interval of one segment as the row key (rk). As rows are sorted on the row key in the key-value store, the starting points of time intervals are in ascending order. Therefore, although for time range or point queries the query processor knows when to stop the scan, it still needs to start the scan from the beginning of the table each time. The same problem happens to the table with value intervals as row-keys. For instance, in Fig. 2.2, when the value range is the row-key, the table is sorted according to the left boundaries of the value ranges of model-view sensor data. For a

value query range [9.2, 10], we can make sure after the fourth segment, namely $S2$ with value range [9.9, 10.9], there is no qualified segment in the table. However, still scanning has to start from the beginning of the table, since the right boundaries of the value ranges are not in order.

In summary, simply incorporating time, value interval or model coefficients into the row-key cannot contribute to accelerating the query processing. A generic index in key-value stores specialized for model-view sensor data is imperative. Moreover, considering that model-view sensor data is produced in real-time, the index for model-view sensor data in key-value stores should be able to update on the fly efficiently [49]. Since each segment of model-view sensor data is characterized by its time and value ranges, we will concentrate on employing interval index to index both the time and value dimension of one segment.

However, another issue for designing the interval index for model-view sensor data in key-value stores is where to materialize the index. One possible way would be to store the index in separated files in the distributed file system (e.g. HDFS, etc.) on top of which the key-value store HBase is also built. In this approach, as in the work [148], each layer of the tree structure is stored in one file. Consequently, index searching needs to invoke multiple MapReduce jobs to iteratively process each level of the tree, which is not efficient. Another way is to integrate one index for each region to only index the local model-view sensor data in that region, such that mappers can first load the index to locate desired data in that region and then access it. However, this kind of approach does not decrease data preparation and starting overhead of MapReduce for the whole table, since still all the regions of the table are initialized to be processed by MapReduce [39, 40]. Our proposed key-value based interval index (described in the next section) steps out of above ideas and takes advantage of both the in-memory and key-value parts to improve the query processing.

2.3.3 Querying model-view sensor data

Much work has been devoted to pattern discovery for time series data, but in this paper, we focus on the following four types of fundamental queries on model-view sensor data.

- Time point query: return the value of one sensor at a specific time point.
- Value point query: return the timestamps when the value of one sensor is equal to the query value. There may be multiple time stamps of which sensor values satisfy the query value.
- Time range query: return the values of one sensor during the query time range.
- Value range query: return the time intervals of which the sensor values are within the query value range. There may be multiple time intervals of which sensor values satisfy the query value range.

The generic process to query model-view sensor data queries comprises the following two steps:

- Searching of qualified segments: The qualified segments are the segments whose time (resp. value) intervals intersect the query time (resp. value) range or point. This step should make use of an interval index to locate all the qualified modeled segments in the segment model store.
- Model gridding: Qualified segments are too abstract and a finite set of data points are more useful as query results for end-users [131]. Therefore, model gridding is another necessary process [37, 131]. Model gridding applies three operations to each qualified segment: (i) It discretizes the time interval of the segment at a specific granularity to generate a set of time points. (ii) It generates the sensor values at the discrete time points based on the model that approximates the segment. (iii) It filters out the sensor data that does not satisfy the query predicates. The qualified time or value points that result from gridding all qualified modeled segments are returned as query results.

As shown in Fig. 2.1(c), segment one $S1$ is found as one qualified segment for time query range $[10, 70]$, model-gridding discretizes the time interval of $S1$ and evaluates the values at each time point. Then, the qualified time-value pairs constitute the query results.

2.4 Key-Value Interval Index

We will first present the design of the two-tier model index on key-value stores and then we will discuss the updating algorithm of the model index.

2.4.1 Structure of KVI-index

As each segment of model-view sensor data has a specific time and value range, instead of indexing the mathematical functions of segments, our idea is to take the time and value intervals as keys to index each segment, which allows the index to directly serve the queries proposed in Sec. 2.3. We propose the **k**ey-**v**alue represented **i**nterval index (KVI-index) to index time and value intervals of model-view sensor data. For a segment in Fig. 2.3(c), the time and value intervals are respectively indexed by the KVI-index, as depicted in Fig. 2.3(a) and (b). Our KVI-index adopts the idea from interval tree, since in-memory interval tree's primary-secondary structure is convenient for externalization to the key-value store [73]. Furthermore, the searching and scanning algorithm of the interval tree is suitable for the MapReduce computing paradigm.

Our KVI-index is a novel in-memory and key-value composite index structure. The virtual

searching tree(*vs-tree*) resides in memory, while an index-model table in the key-value store is devised to materialize the secondary structure(*SS*) of each node in *vs-tree*.

In-memory structure

The in-memory virtual searching tree(*vs-tree*) is a standard binary search tree shown in Fig. 2.3(a). Each time (or value) interval is registered on only one node of *vs-tree*: the one with which the interval first overlaps along the searching path from root. This node is defined as the registration node τ for this interval. Each node of *vs-tree* has an associated secondary structure(*SS*), materialized in the key-value store, which stores the substantial information of the modeled segments registered at this node.

We apply space-partition strategy for *vs-tree*. The height of the *vs-tree* is denoted by h. We set the value of leftmost leaf node as 0. For negative sensor data values, we use simple shifting to have the data range start from 0 for convenience. Then, the domain of the *vs-tree* is $[0, R]$ and $R = 2^{(h+1)} - 2$. The value of root node is $r = 2^h - 1$. During the whole life of KVI-index, only the root value r is kept, since, due to the lightweight computability of the space-partition, the value of each node in the searching path from the root to the node that has the queried point or interval can be calculated in run time. All the operations on *vs-tree* are performed in memory and are thus very efficient. As the domains of time and value of the sensor data are different, two *vs-tree*s one for times and another for values are kept in memory simultaneously for answering time and value queries respectively.

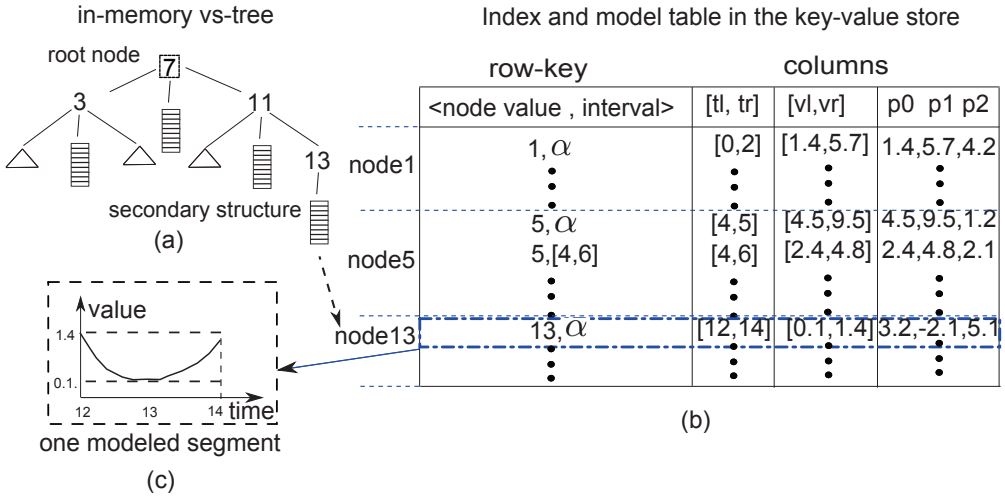


Figure 2.3 – Two-tier structure of KVI-index

Index-model table

Our novel index-model composite storage schema enables one table not only to store the modeled segments, but also to materialize the structural information of the *vs-tree*, i.e., the *SS*s for each tree node.

The index-model table is shown in Fig.2.3(b). Each row corresponds to only one modeled segment of sensor data, e.g., the data segment shown in the black dotted rectangle in Fig.2.3(c). A row key consists of the node value and the interval of an indexed modeled segment at that node. The time interval, value interval and the model coefficients are all stored in different columns of the same row. And the *SS*s of each node correspond to a consecutive range of tuples in the index-model table. For instance, the rows corresponding to the *SS*s of node 1, node 5 and node 13 in *vs-tree* are illustrated in Fig. 2.3(b). Analogously, we have two index-model tables that correspond to time and value *vs-tree*s respectively.

2.4.2 KVI-index updates

The complete modeled segment updating algorithm of KVI-index is shown in Alg. 2. It includes two processes:

(1) *Registration node searching*: locate the node τ at which one time (value) interval $[l_t, r_t]$ should be registered.

(2) *Materialization of modeled segments*: construct the row-key based on the τ and materialize one modeled segment's information into the columns of the corresponding row.

Registration node searching (rSearch)

This algorithm first involves a *domain-expansion process* to dynamically adjust the domain of the *vs-tree* according to the specific domain of the sensor data. Then, the registration node can be found on the validated *vs-tree*.

Lemma 1. *For a modeled segment M_i with time (value) interval $[l_t, r_t]$ (resp. $[l_v, r_v]$), its registration node lies in a tree rooted at $2^{\lceil \log(r_t+2) \rceil - 1} - 1$.*

Proof. The domain of one tree rooted at $2^{\lceil \log(r_t+2) \rceil - 1} - 1$ is $[0, 2^{\lceil \log(r_t+2) \rceil} - 2]$. As $2^{\lceil \log(r_t+2) \rceil} - 2 \geq 2^{\log(r_t+2)} - 2 = r_t$, the registration node of interval $[l_t, r_t]$ must be in a tree rooted at $2^{\lceil \log(r_t+2) \rceil - 1} - 1$. □

Lemma 2. *For a modeled segment M_i with time (value) interval (l_t, r_t), if the right end-point r_t satisfies $r_t > R$, the domain of vs-tree needs to expand.*

Algorithm 2 Time (or value) KVI-index updating

Input: $[l_v, r_v]$, $[l_t, r_t]$, /* value and time intervals of one modeled segment, M_i, r /* M_i denotes the coefficients of the modeled segment

```
# /* dynamic domain expansion
if (r_t > R) then
    r= 2^⌈log(r_t+2)⌉−1 -1 /* expand to new root value
end if
# /* registration node search
node = 2^⌊log r_t⌋+1 -1;
h=log(node)-1;
while (h ≥ 0) do
    if (l_t ≤ node and r_t ≥ node) then
        break; /* node is the registration node
    else
        if (l_t > node) then
            node= node+ 2^h;
        end if
        if (r_t < node) then
            node= node- 2^h;
        end if
    end if
    h=h-1;
end while
# /* materialized into the index-model table.
if the SS of node has been initialized then
    rowkey= <node|l_t|r_t>
else
    rowkey= <node|α>
end if
insert [l_v, r_v], [l_t, r_t], M_i into the table.
```

Proof. The current *vs-tree*'s height is h and $l_t \leq r$, the interval $[l_t, r_t]$ rides over the root node r. Assume that we do not expand the domain and hang $[l_t, r_t]$ on r. When a new model M_j with a time (or value) interval $[l_t^{'}, r_t^{'}]$ (or $[l_v^{'}, r_v^{'}]$) and $l_t^{'} > R$ comes, the root value has to increase to $r^{'}$=$2^{\lceil \log r_t^{'}+2 \rceil -1}$-1 (resp. $r^{'}$=$\cdots$) as $[l_t^{'}, r_t^{'}]$ (resp. $[l_v^{'}, r_v^{'}]$) intersects no node of current *vs-tree*. Then between r and $r^{'}$, there is one node with value $2^{(h+1)}$-1. The interval (l_t, r_t) is stored at node $r = 2^h$-1 and $r_t > 2^{(h+1)}$-2 $\Rightarrow r_t \geq 2^{(h+1)}$-1, therefore registering the interval $[l_t, r_t]$ on node r contradicts with the interval registration rule. □

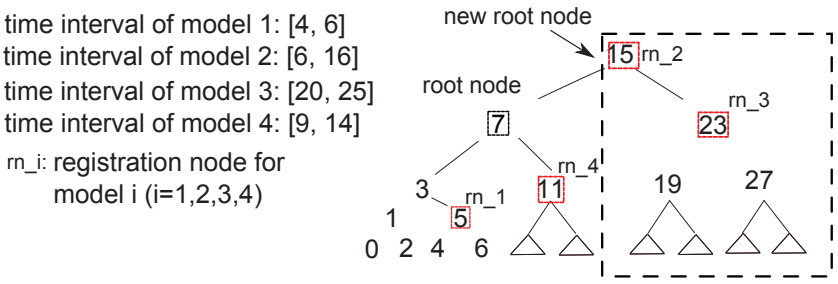


Figure 2.4 – Registration node searching of KVI-index

Using Lemma1 and Lemma2, KVI-index is able to dynamically decide when and how to adjust

the domain $[0, R]$ of *vs-tree*. The complete *rSearch* can be illustrated by Fig. 2.4. When $model1$ is to be inserted, the *vs-tree* rooted at $node7$ is still valid. $model1$ is registered at $node5$. However, when $model2$ arrives, its right end-point, i.e., 16, exceeds the domain $[0, 14]$. The *vs-tree* is expanded having 15 as new root and the new extended domain is the area enclosed by the dotted block in Fig. 2.4. Then, the sub-sequential $model2$ and $model3$ can be updated successfully.

The rSearch process can be further optimized via adaptively adjusting the starting point based on the interval to index. In this way, rSearch does not need to always search from root node, thereby shortening the length of searching path. Based on Lemma 1, for one interval (l_t, r_t), we can start to search for registration node from node $r_0 = 2^{\lfloor \log tr \rfloor + 1} - 1$ rather than root node r, which can be referred to Line 7 in Alg. 2. Also, the nodes outside of sub-domain $[0, 2^{\lfloor \log tr+1 \rfloor + 1} - 2]$ of *vs-tree* will not become the registration node, because the interval (tl, tr) does not intersect with them. For example, the $model4$ in Fig. 2.4. The adaptive searching finds that the subtree rooted at node 7 is the minimum one covering $model4$'s interval $[9, 14]$. Therefore, node 7 is the starting point for registration node searching and the length of searching path is only 1.

Materialization of modeled segment

When materializing model M_i into the *SS* of a node τ, the row-key may be chosen in two ways:

- Upon initialization of the *SS* of node τ: when no modeled segment has been stored at τ's *SS*, the row key is chosen as $< \tau, \alpha >$ for model M_i. Here, α is a postfix of row key to indicate that this row is the starting position of τ's *SS* in the table.
- Upon updating the *SS* of node τ: when the *SS* of τ has already been initialized, the time interval $[l_t, r_t]$ (resp. $[l_v, r_v]$ for value interval) to be indexed will be incorporated into the row key, i.e., $< \tau, l_t, r_t >$ (resp. $< \tau, l_v, r_v >$ in the index-model table for values). In this way, different modeled segments stored in the same *SS* of a node do not overwrite each other.

The selection of specific α should make sure that the binary representation of $< \tau, \alpha >$ is in front of any other $< \tau, l_t, r_t >$. This design is useful for query processing. For instance, if the query processor requires to access all the modeled segments stored at registration node 5, then we know that all the corresponding modeled segments lie in the rows within the row-key range $[< 5, \alpha >, < 6, \alpha >)$. For example, take the $model1$ and $model2$ in Fig. 2.5. First, the KVI-index checks whether the starting modeled segments of $node5$ and $node15$, namely rows with key $< 5, \alpha >$ and $< 15, \alpha >$, exist. Then, the row key $< 5, 4, 6 >$ is constructed for $model1$ as the *SS* of $node5$ has been initialized, whilst KVI-index constructs the row key $< 15, \alpha >$ for

model2.

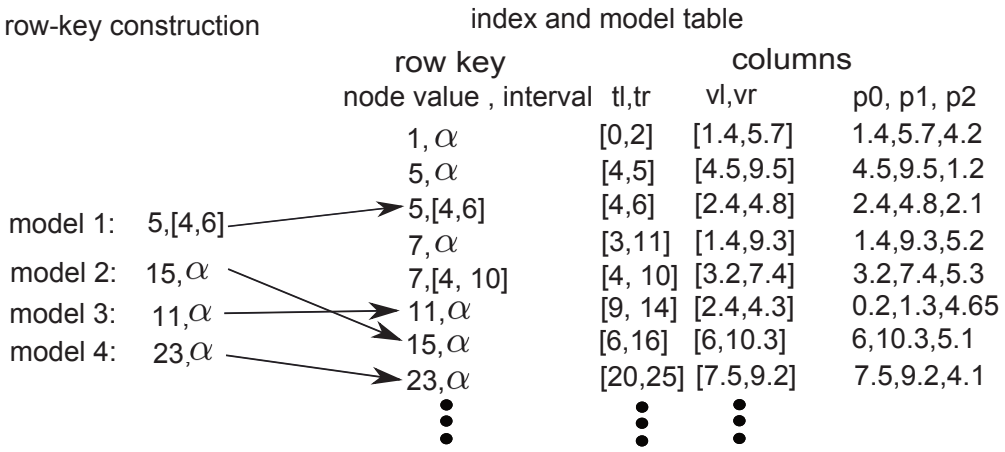


Figure 2.5 – Modeled segment materialization of KVI-index

Complexity analysis

In this section, we analyze the computational and communication complexities of updating operation of KVI-index. The complexity analysis of KVI-index includes in-memory and key-value part.

- In-memory

  The *rSearch* on *vs-tree* can run within $O(log(R))$ time. The space expansion costs $O(1)$ time. Only the root value r of *vs-tree* is kept in memory. The values of other nodes on *vs-tree* can be calculated due to the computability of *vs-tree*'s space-partition. Therefore, the space complexity of *vs-tree* is $O(1)$.

- In key-value store

  The update operation on *vs-tree* does not generate any network I/O cost. For a n-th order polynomial model of one sensor data segment, $(2+n)$ put operations are conducted to materialize one model. Therefore, the time complexity is $O(1)$ in terms of network I/O, as n is constant. Moreover, one segment model's time (or value) interval and coefficients are only materialized once into the index-model table, thus the space cost is $O(N)$ for time (or value) index.

2.5 Query Processing via KVI-index and MapReduce

For querying model-view sensor data, the searching process of qualified modeled segments (defined in Sec.2.3) in KVI-index includes two steps:

- Intersection and point search: The intersection search on *vs-tree* is used for range queries, while point search is employed for point queries. They are responsible for collecting the nodes that accommodate qualified modeled segments in their secondary structures *SS*s.
- Modeled-segment filtering: Due to the rule for interval registration at the nodes of the *vs-tree*, the *SS* of a node may contain some intervals irrelevant to queried range or point. In KVI-index, the *SS*s of all nodes found by the search operation are accessed to filter out unqualified segments.

After the above two steps, model gridding component fetches the coefficients of each qualified modeled segment and performs gridding. Next, we first describe an enhanced intersection search algorithm on *vs-tree* that benefits KVI-Scan-MapReduce query processing, introduced later in this section. We then present the point search algorithm on *vs-tree*. Subsequently, we introduce our novel hybrid KVI-Scan-MapReduce query processing. Last, we theoretically analyze the enhanced intersection search algorithm of KVI-index.

2.5.1 Intersection and point search

Enhanced interval intersection search

Alg. 3 presents the $iSearch^+$. Given a time (resp. value) range query $[l_t, r_t]$, $iSearch^+$ first calls the *rSearch* to find the registration node τ of $[l_t, r_t]$. The nodes on the searching path from the root node to the one preceding τ form a node set denoted by $\mathscr{S}_0$. The $iSearch^+$ stops at the node, which is closest to the left-end point l_t. All the nodes along the left-descending path form a node set, denoted by $\mathscr{S}_l$, while the node with the minimum value in this path is denoted by l_s. Analogously, $\mathscr{S}_r$ is the node set from the right-descending path and r_s is the node with the maximum value in this path. Any node outside the range $[l_s, r_s]$ and the set $\mathscr{S}_0$ does not have any qualified modeled segments.

The traditional intersection search would return the node set $\mathscr{C} = \mathscr{S}_0 \cup \mathscr{S}_l \cup \mathscr{S}_r \cup [l_t, r_t]$ for further modeled-segment filtering and gridding. Our $iSearch^+$ outputs the discrete node set $\mathscr{S}_0$ and a consecutive range of nodes $\mathscr{D} = [l_s, r_s]$. For example, take the range query in Fig. 2.6(a). $node7$ is the registration node of query range $[6, 10]$. The traditional $iSearch$ returns the discrete node sets shown in the solid boxes of Fig. 2.6(a), while our $iSearch^+$ returns a range of nodes $[3, 11]$ and $\mathscr{S}_0 = \{15\}$. We will see how the output of $iSearch^+$ benefits the hybrid query processing later in Subsection 2.5.2.

Algorithm 3 $iSearch^+$ of *vs-tree*

Input: time query range $[l_t, r_t]$, root value r
Output: node set $\mathscr{S}_0$ and $\mathscr{D}$

```
# /* construct S_0
node=r;
h=log(r)-1;
while (h ≥ 0) do
    if (l_t ≤ node and r_t ≥ node) then
        break; /* node is the registration node
    else
        S_0 = S_0 ∪ node
        if (l_t>node) then
            node= node + 2^h;
        end if
        if (r_t<node) then
            node= node - 2^h;
        end if
        h=h-1;
    end if
end while
# /* construct D .
l_s= 2^⌊log(l_t)⌋,
r_s= R - 2^⌊log(R−r_t)⌋,
D = [l_s, r_s]
```

Point search

We denote the point search by *sSearch* as it functions as the stabbing search of interval tree. The *sSearch* is a binary search that records the nodes along the descending path. We present the *sSearch* in Fig. 2.6(a). For example, when querying the sensor value of time point 24, the node set $\mathscr{S}_0 = \{15, 7, 11, 9, 10\}$ is returned by *sSearch*. Since there is no split searching, as in $iSearch^+$, only one node set is produced here. We denote this node set by $\mathscr{S}_0$ as well, so as to facilitate the description of the hybrid KVI-Scan-MapReduce query processing that follows next.

2.5.2 KVI-Scan-MapReduce query processing

The conventional clustered index for one-dimensional data can exactly locate a consecutive range of qualified data. Then, the query processor just needs to do range scan on these qualified data. However, as for querying model-view sensor data, the challenge is how to tackle the large amount of segment models from the *SS*s distributed across the index-model table. We first analyze the location distribution of the *SS*s of the nodes found by $iSearch^+$ and *sSearch* in the index-model table. The characteristics of this distribution inspired us to propose the hybrid KVI-Scan-MapReduce query processing approach.

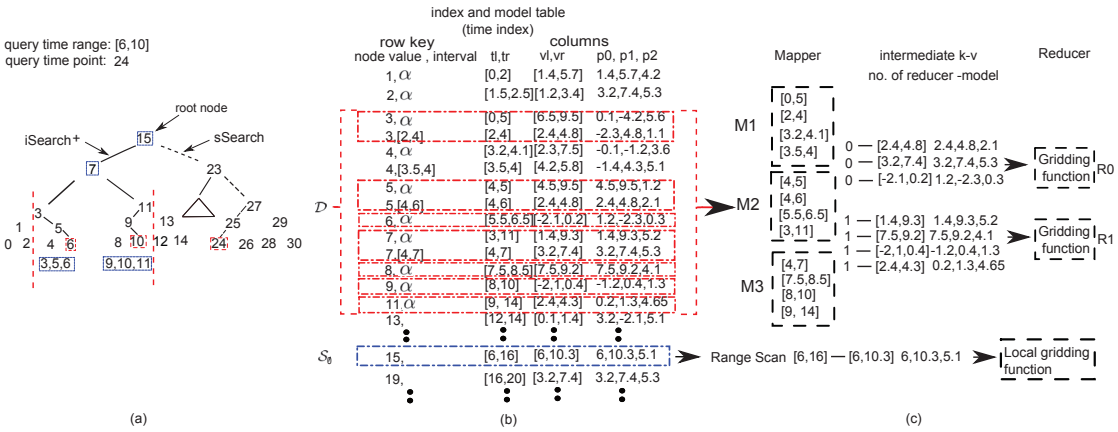


Figure 2.6 – Workflow of KVI-Scan-MapReduce approach. (a) $iSearch^+$ and *sSearch*. (b) *SS* location distribution for the range query. (c) Hybrid processing

SS location distribution

There are two cases for the *SS* distribution in the index-model table, described below.

- $\mathscr{S}_0$: The *SS*s of $\mathscr{S}_0$ are non-consecutive and sparsely distributed in the index-model table. The node value is the primary part of the row-key; thus, the distance between *SS*s of $\mathscr{S}_0$ depends on the numerical difference of node values. As $\mathscr{S}_0$ includes the nodes from root node to the one preceding the τ, the intra-distances between any consecutive nodes in $\mathscr{S}_0$ are 2^{h-i}, where $i = 0, \cdots, h - d_\tau$ is the position of the node in the descending search path $\mathscr{S}_0$ and d_τ is the depth of τ. Obviously, the intra-distances in $\mathscr{S}_0$ are greater than those for other nodes below τ in the search path.

- $\mathscr{D}$: The *SS*s of $[l_s, r_s]$ are clustered around the ones of $[l_t, r_t]$ in the index-model table. The *SS*s of $[l_t, r_t]$ are all adjacent in the index-model table. The *SS*s of $[l_s, r_s]$ are bounded by those of the sub-tree rooted at τ and the nodes in $[l_s, r_s]$ are a superset of the nodes in $[l_t, r_t]$. The deeper the registration node τ is located, the tighter the set of the *SS*s of $[l_s, r_s]$ over those of $[l_t, r_t]$.

For example, take the time (or value) query range $[6, 10]$ in Fig. 2.6(a). The registration node is $node7$. Then, $\mathscr{S}_0 = \{15\}$ and $\mathscr{D} = [3, 11]$. The sub-tree rooted at $node7$ covers the node range $\mathscr{E} = [0, 14]$ and $\mathscr{D} \subset \mathscr{E}$. From Fig. 2.6(b), the *SS*s of $\mathscr{D}$ are clustered around those of $[6, 10]$ and bounded by the *SS*s of $\mathscr{E}$. However, $node15$'s *SS* is located far away from those of $[3, 11]$.

If *SS*s of $\mathscr{S}_0$ and $\mathscr{D}$ are processed via straightforward random access and range scan provided by key-value stores, the entire modeled-segment filtering and gridding processes are conducted locally at the application side. For a table of multiple or hundreds of GBs, the communication

and computation costs are prohibitively high for the application side even for low-selective queries.

The modeled segment filtering-gridding processing matches MapReduce's filtering-aggregation paradigm. Considering the research results from [39, 40, 42], for CPU non-intensive workload, I/O cost, network latency and starting-up overhead of mappers are dominant in the execution time of MapReduce programs. If the *SS*s of $\mathscr{S}_0$ and $\mathscr{D}$ are all processed by MapReduce, a lot of time is wasted for mappers that process irrelevant *SS*s in the index-model table. This is because MapReduce will access the continuous regions of the table including the *SS*s of nodes between the $\mathscr{S}_0$ and $\mathscr{D}$ due to the sequential data feeding mechanism in the mapping phase. For example, in Fig. 2.6(b), the *SS*s of $\mathscr{D}$=[3, 11] and $\mathscr{S}_0$={15} are distant in the table. Hence, MapReduce will launch many unnecessary mappers for the irrelevant *SS*s of nodes between 11 and 15, in order to process the *SS*s of $\mathscr{S}_0$ and $\mathscr{D}$.

Hybrid model filtering and gridding

As discussed above, simply using range scan or MapReduce to process *SS*s are both problematic. Our idea is to design a hybrid KVI-Scan-MapReduce paradigm that combines range scan and MapReduce for processing *SS*s, as follows:

- (1) $\mathscr{S}_0$: the height of *vs-tree* is bounded by $log(R)$, and thus the amount of computation on $\mathscr{S}_0$ is limited. As the *SS*s of $\mathscr{S}_0$ are sparsely distributed in the index-model table and each *SS* of $\mathscr{S}_0$ can be considered to be a small range of clustered index, the random-access- and range-scan-based model filtering and gridding is suitable.

- (2) $D = [l_s, r_s]$: the successive range $[l_s, r_s]$ delimits a tight boundary of the sub-index-model table over the relevant *SS*s that are suitable for processing with MapReduce.

This hybrid paradigm eliminates the Map-phase processing of *SS*s of irrelevant nodes between $\mathscr{S}_0$ and $\mathscr{D}$ and the nodes between the elements of $\mathscr{S}_0$. Moreover, it is non-intrusive for both the key-value store and MapReduce. Regarding the time (or value) point query, it only produces the node set $\mathscr{S}_0$ without $\mathscr{D}$, hence, only range-scan-based model filtering and gridding is needed.

Suppose that the number of reducers is P and each reducer is denoted by $0, \cdots, P$-1. For range queries, the partition function f is used to assign the qualified modeled segments into different reducers. It is designed on the basis of query time (resp. value) range $[l_t, r_t]$ (resp. $[l_v, r_v]$) and the time (or value) interval $[l_i, r_i]$ of each modeled segment i. The idea is that each of the reducers is in charge of one even sub-range $\frac{r_t - l_t}{P}$. Such a partition function f is

given in Eq. 2.1.

$$f(r_i) = \begin{cases} l_t \le r_i \le r_t & \lfloor \frac{(r_i - l_t) * P}{r_t - l_t} \rfloor \\ r_i \ge r_t & P - 1 \end{cases} \tag{2.1}$$

The functionalities of mappers and reducers are depicted in detail below.

- **Mapper**: Each mapper gets the time (resp. value) interval $[l_i, r_i]$ of one modeled segment i to check whether it intersects with the query time (resp. value) range. For the qualified modeled segments, the intermediate key is derived by the partition function $f(r_i)$. The model coefficients $< p_i^1, \cdots, p_i^n >$ are the value part of the intermediate key-value pair.
- **Reducer**: One reducer receives a list of qualified modeled segments $< p_0^1, \cdots, p_0^n >$ $, < p_1^1, \cdots, p_1^n >, \cdots$. For each modeled segment $< p_i^1, \cdots, p_i^n >$, the reducer invokes a model-based gridding function to compute discrete values as query results.

Regarding the scan-based model filtering and gridding, as *SS*s in $\mathscr{S}_0$ are located in different regions of the index-model table, the query processor makes use of thread pool to process each *SS* of $\mathscr{S}_0$ in parallel. Fig. 2.6 shows the workflow of the hybrid KVI-Scan-MapReduce approach. For a time (or value) range query $[6, 10]$, $iSearch^+$ constructs the node set $\mathscr{S}_0 = \{15\}$ and $\mathscr{D} = [3, 11]$. Then, the *SS*s of the nodes in $\mathscr{D}$ are sent to MapReduce. The *SS* of $node15$, enclosed by the bottom dot-dashed block, is processed via range scan.

2.5.3 Theoretical analysis

One point to carefully consider is that $iSearch^+$ may generate redundant nodes, because the $iSearch^+$ aims to find a tight and consecutive range of *SS*s for MapReduce. For instance, in Fig. 2.6(a), the *SS* of $node4$ is not accessed by the $iSearch^+$, but is in the sub-table processed by MapReduce.

Theorem 1. *For a range query $[l_t, r_t]$, the redundant nodes in $[l_s, r_s]$ returned by $iSearch^+$ are bounded.*

Proof. Assume h to be the height of the registration node τ. Consider the left-descending path from τ to the node l_0 closest to l_t. Let d be the depth from which the descending path turns right, namely the value $w < l_t$ of the current node. Then, based on the $iSearch^+$ algorithm, w is the left boundary of accessed node range and $w = \tau - \sum_{i=1}^{d} 2_{h-i}$.

For a certain value of d, the worst case happens when the descending process continues to go right until reaching l_0, as the nodes between w and l_0 are all redundant ones. The number of

nodes returned by $iSearch^+$ under this case is given by:

$$U = \tau - (\tau - \sum_{i=1}^{d} 2^{h-i}) \tag{2.2}$$

The nodes between τ and w are all included into the output range $\mathscr{D}$ of $iSearch^+$. The number of nodes returned by the conventional *iSearch* is given by:

$$V = h - d + \{\tau - (\tau - \sum_{i=1}^{d} 2^{h-i} + \sum_{i=d+1}^{h} 2^{h-i})\} \tag{2.3}$$

Therefore, the number of redundant nodes returned by $iSearch^+$ is given by:

$$\begin{aligned} f = U - V &= d + \sum_{i=d+1}^{h} 2^{h-i} - h \\ &= d + 2^{h-d} - h - 1 \end{aligned} \tag{2.4}$$

The Eq. 2.4 is a function of d and is monotonous decreasing in d's domain $[1, h]$. Consequently, when $d = 1$, the function f reaches the maximum value, namely, the number of redundant nodes from $iSearch^+$ attains the maximum value f_{max} that is given by:

$$f_{max} = 2^{h-1} - h\,. \tag{2.5}$$

As the total number η of nodes of the sub-tree of the left child of τ is $2^h - 1$, hence

$$f \leq \frac{1}{2}\eta - log(\eta + 1) + \frac{1}{2}\,. \tag{2.6}$$

In summary, the total number of redundant nodes in the range $[l_s, r_s]$ is bounded. □

The worst case happens when the endpoints l_t and r_t are the preceding and succeeding nodes of τ, namely $l_t = \tau - 1$ and $r_t = \tau + 1$. However, for most of the cases, the redundant nodes returned from $iSearch^+$ are very limited.

2.6 Experimental Evaluation

First, we compare model-view sensor data query processing with conventional one over raw sensor data. Then, we show that our KVI-Scan-MapReduce (*KSM*) approach outperforms other model-view sensor data querying approaches. Finally, we experimentally explore the factors that affect the performance of KVI-Scan-MapReduce.

2.6.1 Setup

We employ accelerometer data from mobile phones as sensor data set. The size of raw sensor data is 22 GB including 200 million data points. After modeling, the modeled segments of the sensor data take 12 GB, while there are around 25 million modeled segments.

We developed our system using the versions of HBase and Hadoop in Cloudera CDH4.3.0. The experiments are performed on our own cluster that consists of 1 master node and 8 slaves. The master node has 64 GB RAM, 3 TB disk space (4 x 1 TB disks in RAID5) and 12 cores, each of which is 2.30 GHz (Intel Xeon E5-2630). Each slave node has 6 cores 2.30 GHz (Intel Xeon E5-2630), 32 GB RAM and 6 TB disk space (3 x 2 TB disks). Nodes are connected via 1 GB Ethernet. In the experimental results, we refer to query selectivity as the ratio of the number of qualified modeled segments over that of total modeled segments.

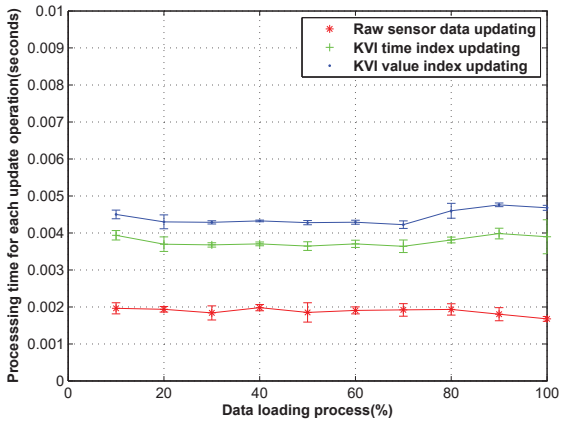
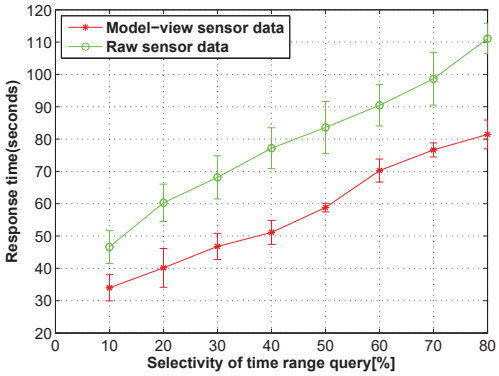


Figure 2.7 – Sensor data updating performance

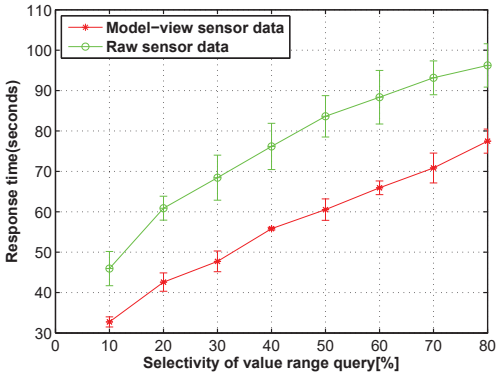
The data set contains discrete accelerometer data from mobile phones and is a sequence of tuples each of which has one timestamp and three sensor values representing the coordinates. The size of the raw sensor data set is 22 GB including 200 million data points. We simulate the sensor data emission, in order to segment and update sensor data into the KVM-index in an online manner. We implement an online sensor data segmentation component [57] applying the *PCA* (piecewise constant approximation) [118], which approximates one segment with a constant value (e.g., the mean value of the segment). Since how to segment and model sensor data is not the focus of this paper, other sensor time series segmentation approaches could also have been applied here. Provided that the segments are characterized by the time and value intervals, our KVM-index and related query-processing techniques are able to manage them efficiently in the cloud. Finally, there are around 25 million sensor data segments (nearly 15 GB) uploaded into the key-value store. Regarding the segment gridding, we choose 1 second as the time granularity which is application-specific.

2.6.2 Index updating

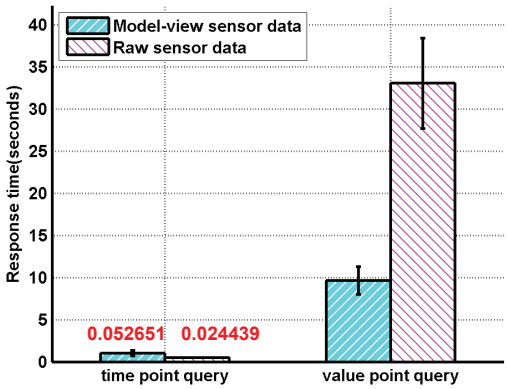
Fig. 2.7 shows the average updating time of each segment and the average insertion time of each raw sensor data point during the data uploading phase. Both time and value index keep relatively stable updating efficiency. The updating of the value KVI-index is a little slower than the time KVI-index. As discussed before, since the domain of the value *vs-tree* is smaller than that of the time *vs-tree*, the value index performs more *SS* updating operations (discussed in Sec. 2.4.2) than the time index and therefore incurs more network I/O cost. The raw sensor data insertion is the fastest but the amount of data to update is much larger than model-view approach. This is because model-view sensor data achieves data compression over the raw sensor data thereby decreasing the amount of data to upload.



(a) Time range queries



(b) Value range queries



(c) Point queries

Figure 2.8 – Query performance comparison of raw data and model-view approaches on range and point queries

2.6.3 Model-view sensor data vs. raw sensor data

We create two tables, which take the time-stamp and sensor value as the row-keys respectively, such that the query range or point can be used as keys to locate the qualified data points. Then, the query processor invokes the MapReduce to access the large size of data points for getting query results.

Fig. 2.8 (a), (b) and (c) present the query response times for time range, value range and point queries respectively. As shown in Fig. 2.8 (a) and (b), the model-view approach takes around 30% less time than the raw sensor data method for both time and value range queries. Although the raw sensor data based methods apply MapReduce to directly access the qualified tuples via the row-key based range scan, the amount of raw sensor data to process is much larger than that of the model-view approach. In Fig. 2.8(c), the processing time of the raw data based method is 2× less than that of the model-view one in time point queries, because the raw data method can use the query time point as index key to directly access the relevant data points, while our *KSM* requires to perform model filtering and gridding. For value point queries, the model-view approach has nearly 3× less time than the raw data method. As, normally, there is a large size of data points with the queried value, MapReduce is used to access this qualified sensor data set. In the model-view approach, the point query processing only uses random access and range scan to get qualified modeled segments for gridding locally, and thus it saves the time on starting MapReduce to access data.

2.6.4 Comparison of model-view approaches

There are four baseline approaches for querying model-view sensor data, namely:

MapReduce (MR). This approach utilizes MapReduce without support from any index. It always works on the whole index-model table to filter the qualified modeled segments in the mapping phase and perform the model gridding in the reduce phase.

Interval tree (IT). We implemented the traditional query processing operations of the interval tree [11, 73] by adding another table to store the *SS* of each node sorted by the right end-point of intervals. Each index, time or value, has two associated tables. During the intersection or point search on *vs-tree*, the query processor decides which table to access based on the relation between the query range (or point) and the node value. In this way, the query processor can stop scanning once it encounters one unqualified modeled segment, due to the monotonicity of end-points of modeled segments. *IT* makes use of random access and range scan to sequentially filter the qualified modeled segments and make gridding locally.

MapReduce+KVI (MRK). The idea of *MRK* is to leverage KVI-index to avoid having MapReduce to process the whole table. In *MRK*, MapReduce is designed to work over one continuous

sub-index-model table including all the *SS*s of the accessed nodes in search operations. For instance, in Fig. 2.6(a), for a time (or value) range query [6,10], *MRK* invokes MapReduce to work on the sub-table within the row-key range $[<3,\alpha>,<16,\alpha>]$. The same idea applies for point queries. As compared to our hybrid *KSM* approach, *MRK* is a lightweight indexing-MapReduce plan, as it processes many irrelevant *SS*s of nodes between $\mathscr{S}_0$ and $\mathscr{D}$.

Filter of key-value store (FKV). Some key-value stores such as HBase provide a filter functionality to support predicate-based row selection [49]. The filter transmits the filtering predicate to each region server and then all servers scan their local data in parallel. Afterwards, they return the qualified tuples. Our filter-based query processing also works on the index-model table, as the filtering predicates can be directly applied to the columns. The query processor waits until all region servers finish scans and then it retrieves each returned qualified modeled segment to conduct gridding locally.

Range Query

Fig.2.9 (a), (b) and (c) present the performance of time range queries. As depicted in Fig.2.9(a), *KSM* outperforms *MR* up to 3× for the low-selective time range queries. As the query selectivity increases, the amount of *SS*s for scan based processing decreases and that for MapReduce approaches the entire table. Therefore, the response time increases and approaches that of *MR*. The response time of *MR* increases little. As increasing query selectivity leads to ascending gridding workload in reduce phase, these results show that the overhead from model gridding is not dominant in *MR*. The response time of *MRK* is more than that of *KSM*, but less than that of *MR*. As *MRK* utilizes the KVI-index to localize a consecutive sub-index-model table covering all the *SS*s of nodes found by intersection search, it processes fewer modeled segments than *MR*'s full table scanning. Yet, as compared to *KSM*, *MRK* processes more redundant modeled segments. Moreover, as the sub-table in *MRK* covers a large range, the processing time of *MRK* increases little for low-selective queries.

Fig.2.9(b) exhibits the performance of *IT* and *FKV* approaches. As *FKV* needs to wait for each region server of HBase to finish the local data scanning, its total response time is a little longer than that of *IT* approach. They both consume much more time than all *MR*, *MRK* and *KSM*, as they apply sequential accessing of modeled segments.

We also analyze the number of modeled segments accessed by each approach in Fig. 2.9(c). These experiments show how different access methods of modeled segments affect the performance. *MR* works on the entire table, thus, the number of accessed segments is the same. From the point of view of the application, only qualified modeled segments are returned for gridding, thus *FKV* processes no redundant modeled segments and consumes the least amount of modeled segments. Since *IT* scans the *SS* of one node until encountering an un-

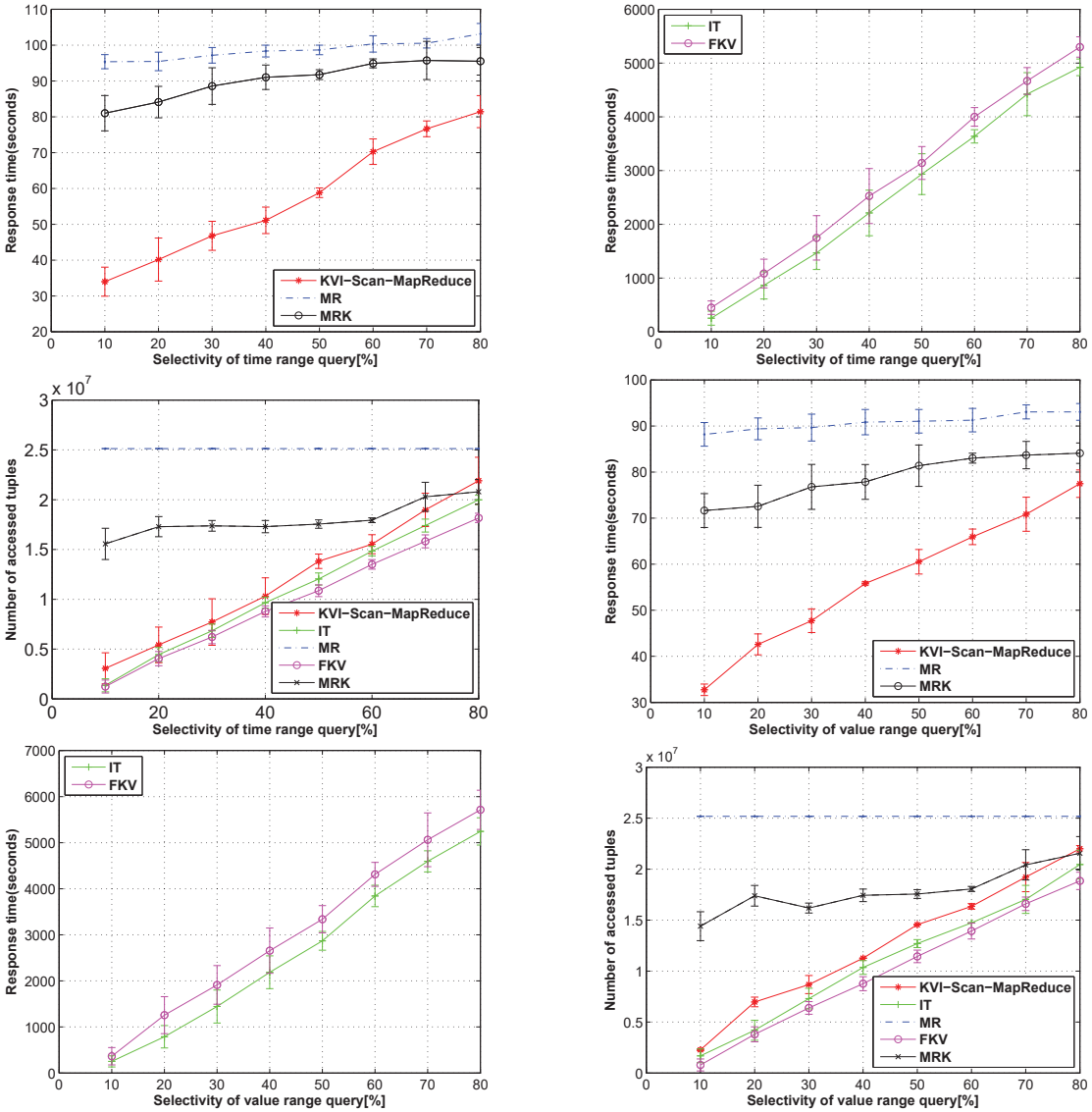


Figure 2.9 – Query performance comparison of different model-view approaches. (a)-(c): time range queries, (d)-(f): value range queries

qualified model, the total number of accessed segments is a little larger than that of *FKV*. Our *KSM* processes larger number of segments than both *IT* and *FKV* due to the continuous and redundant range of *SS*s found by $iSearch^{+}$. However, the results also verify our theoretical analysis that the amount of redundant modeled segments is bounded. *MRK* accesses more segments than *IT*, *FKV* and *KSM*, as it adds the *SS*s between $\mathscr{S}_0$ and $\mathscr{D}$ to form a continuous sub-table for MapReduce. Referring to Fig. 2.9(a) and Fig. 2.9(b), although *KSM* approach consumes more segments than *IT* and *FKV*, its hybrid paradigm is the most efficient.

Fig. 2.9 (d), (e) and (f) present the value range query performance. The different query processing approaches exhibit similar patterns as for the time range queries, so we skip the detailed analysis.

Point Query

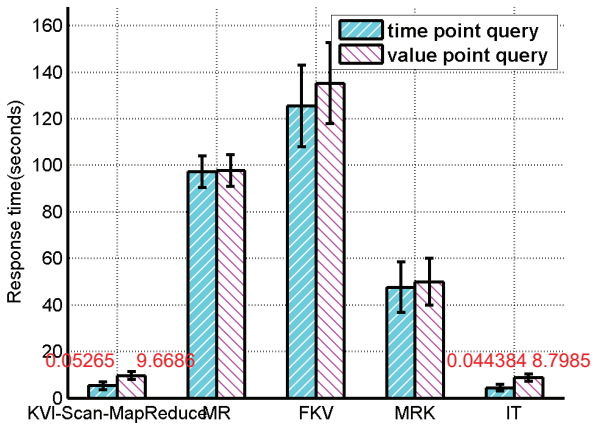


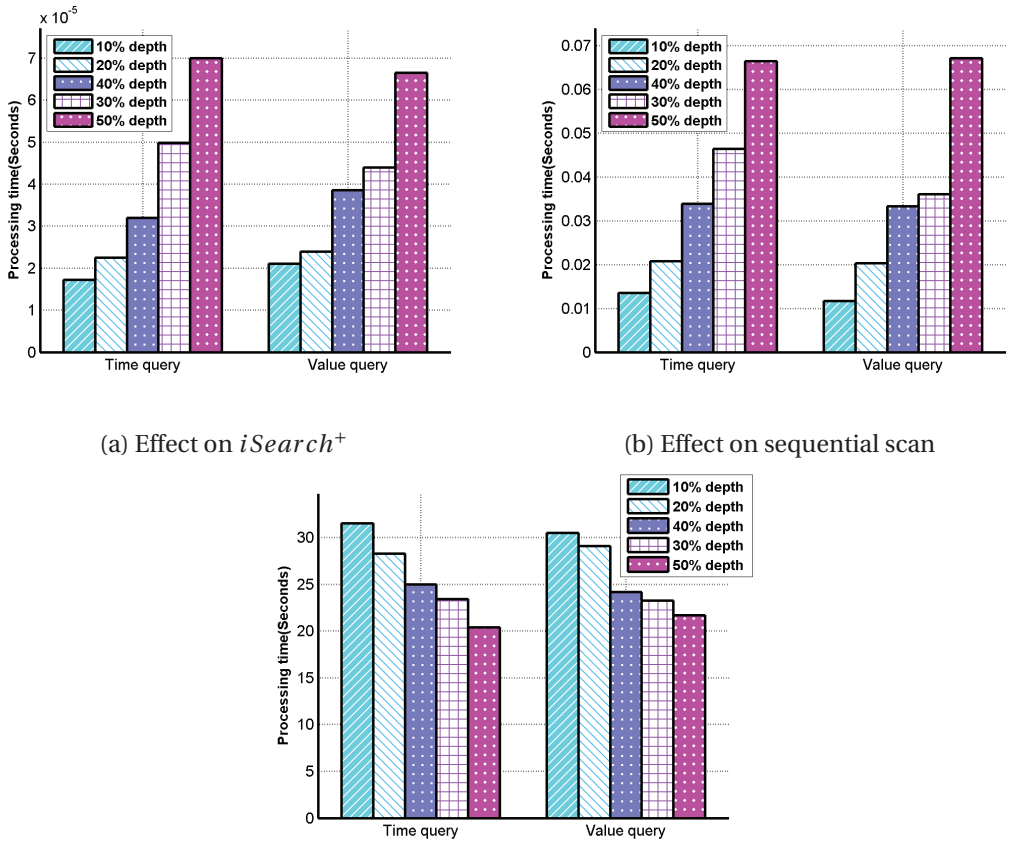
Figure 2.10 – Query performance of point queries

The time and value point query processing performance are shown in Fig. 2.10. *IT* wins both for time and value point queries. The response time of *KSM* is a little greater than *IT*, but outperforms the other approaches, because *IT* is able to access all qualified modeled segments in one *SS*. However, the *KSM* scans the whole *SS* entries of a node to find the qualified ones. Because of the invocation of MapReduce and redundant modeled segments in the sub-table, *MRK* takes more time than both *IT* and *KSM*. But, as *MRK* does not work on the entire table as *MR* does, it takes about 2× less time than *MR*. *FKV* consumes the most time as it needs to wait for server-side full table scan before gridding operations. Since the size of the domain of the sensor data values is smaller than that of the time domain, nodes in value *vs-tree* accommodate more modeled segments than time *vs-tree* nodes. Thus, the response times of value point queries of *IT*, *FKV* and *KSM* approaches are all more than those of time point queries. For *MR*

and *MRK*, the processing time differences between time and value queries are insignificant, as the time spent on model filtering and gridding is not dominant.

2.6.5 Insights into KVI-Scan-MapReduce

Searching depth



(a) Effect on $iSearch^+$

(b) Effect on sequential scan

(c) Effect on MapReduce

Figure 2.11 – Effect of searching depth on the processing time

Sec. 2.6.5 discusses effect of the searching depth on the performances of in-memory $iSearch^+$, sequential scan based and MapReduce based model filtering and gridding. At last, we will see the workload constitutions within the KVI-Scan-MapReduce paradigm in Sec. 2.6.5.

In Fig. 2.11, we present the results from time and value range queries of 10% selectivity and 10% to 50% $iSearch^+$ depth. The percentage of $iSearch^+$ depth here means the ratio of registration node searching depth over the height of *vs-tree* in $iSearch^+$. As $iSearch^+$ depth

increases, the number of *SS*s for range scan based model processing increases. Therefore, the time consumed by $iSearch^+$ and range scan based model processing both increases, shown in the Fig. 2.11(a) and Fig.2.11(b). As for the MapReduce part, the deeper the level of registration node τ is, the smaller the space covered by the sub-tree rooted at τ is. Then the range of *SS*s in $\mathscr{D}$ is tighter over the range of *SS*s of query range and results in less redundant *SS*s for MapReduce, which is theoretically analyzed in Sec. 2.5.3. From Fig. 2.11, we can see a salient decreasing trend of MapReduce processing time.

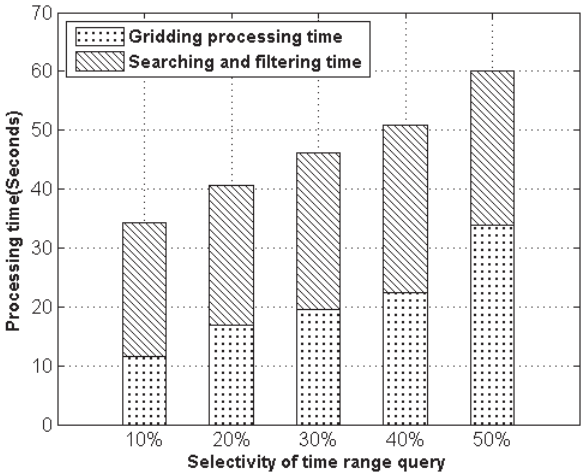


Figure 2.12 – Query processing time constitution of time range queries

Workload Constitution

This experiment aims to reveal how much time the KVI-Scan-MapReduce spends on model gridding, which is a difference of model-view approach from raw sensor data approach. Fig. 2.12 and Fig. 2.13 show the results from time and value range queries of selectivity from 10% to 50%.

From Fig. 2.12 and Fig. 2.13, the time on model gridding accounts for 1/3 - 1/2 of the total query processing time. As the majority of gridding work is done in the reduce phase and the amount of qualified segment models sent to reducers depends on the query selectivity, the time spent on model gridding increases as the query selectivity increases. If the model gridding can adapt to users' different requirements for query results, the performance of the KVI-Scan-MapReduce scheme can be further optimized.

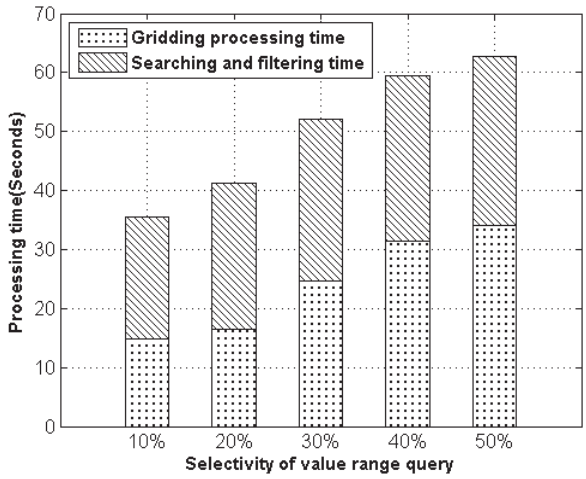


Figure 2.13 – Query processing time constitution of value range queries

2.7 Conclusion

To the best of our knowledge, this is the first work to explore the key-value representation of an interval index for model-view based sensor data management. Different from conventional external-memory index structure with complex node merging and split mechanisms, our *KVI-index*, resident partially in memory and partially materialized in the key-value store, is easy to maintain in the dynamic sensor data generation environment. Moreover, we proposed a hybrid query processing approach, namely *KVI-Scan-MapReduce*, integrating the KVI-index, range scan and MapReduce for model-view sensor data in key-value stores. Extensive experiments in a real testbed showed that our approach outperforms in terms of query response time and index updating efficiency not only query processing methods based on raw sensor data, but also all other approaches considered based on model-view sensor data for time/value range and point queries. As a future work, we plan to explore how to process time and value composite queries and join queries based on the KVI-index.

3 Distributed Mining of Correlation over Streaming Time Series

The world's most powerful graphics chip: Imagination!
— Sheldon Cooper, *The big bang theory*

3.1 Introduction

As massive powerful and diverse sensors are penetrating into our daily life, increasing amount of real-time data which is measured in the time series form are produced by a variety of devices at blazing speeds (e.g., mobile phones, sensor networks, smart meters, smart home appliances and etc.). Analogous to MapReduce ecosystems and associated distributed algorithms designed for processing large-scale static data [78], many distributed, fault-tolerant and real-time computation systems [1,2,98,152] have been developed to address the need to process massive streaming time series. Efficiently mining meaningful and timely knowledge (e.g., correlations, invariants, causal relations etc.) from time series based on such a new computation paradigm

is becoming urgent, but receives little attention in the literature.

In this paper, we concentrate on an important and fundamental time series mining problem, continuously mining and modeling correlations from massive time series via a distributed real-time computation engine (DisCoM problem). Intuitively, two time series have a significant correlation if they evolve similarly in a correlated pattern over time or when one is delayed by a certain lag, as we will provide the formal definition in Section 3.3. DisCoM problem aims to discover and model such co-evolving relations across massive time series in the real-time dynamic context. Such detected correlations can be leveraged for a variety of applications, such as correlation anomaly analysis [61], pattern discovery [154], event detection [28], leadership mining [22, 143] and so on.

Meanwhile, diverse applications need the mining and modeling of dynamic correlations as their fundamental building blocks. In the experimental research,(e.g, climate change), mining lagged correlations over sensor observations from multiple locations allows scientists to discover from which locations events originate in time [143]. In the performance monitoring for large-scale systems e.g. data centres, [75, 137], correlations among performance counters (e.g., CPU, memory usage, etc.) can be used for data-driven incident diagnosis or anomaly detection [88, 137]. For instance in Figure 3.1, time series measuring memory usage of three servers [108] is depicted. DisCoM problem aims to discover that the memory usage in Server2 and Server3 constantly follows the variation in Server1 due to some inherent data communication or function dependency among them and to model such relations. Once it is violated, anomaly is reported. In the finance area, by utilizing timely detected correlations to predict the trends of stock prices, traders can spot investment opportunities at a early stage [22]. It is also applicable in social data analysis [112].

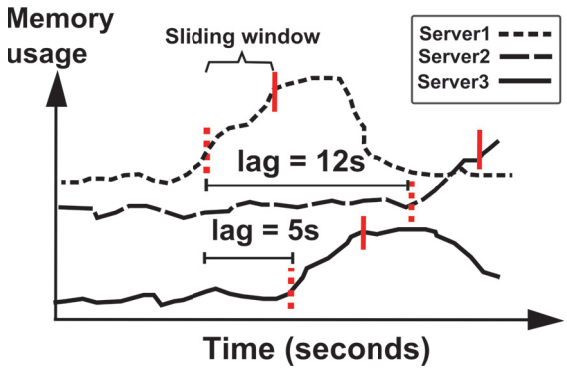


Figure 3.1 – Lagged correlation over memory usage time series

Challenges in Real-time and Distributed Mining and Modeling of Correlations: Contrary to static time series mining on a standalone machine, DisCoM problem in the highly dynamic

environment is challenging in the following aspects.

- In time series mining applications, for making decisions based on the recently observed data, users are more interested in discovering (lagged) correlations within a recent temporal range, which is known as the maximum lag. Meanwhile, in the real-time environment correlation computation is repeatedly performed over the latest observations of time series, which gives rise to the sliding window fashion. As new observations of time series continuously arrive to update sliding windows, computation at a certain time instant should be finished before the subsequent new data arrives. Otherwise it would lag further and further over time and the system cannot report timely correlations. The system has bottlenecks in such a case [2, 147]. For instance, in data center networks [137], the time interval for performance time series is very short, e.g. 2 seconds. Efficiently discovering correlations from thousands of time series in such a high velocity environment is very challenging.
- On the other hand, the distributed environment complicates the problem more. As the observations of time series are continuously distributed into different computing nodes of a distributed real-time computation cluster and each node has no knowledge about the timely characteristics of time series other nodes receive, each node has to replicate and shuffle its local time series (sliding windows) among the nodes of the cluster so as to find qualified correlations *w.r.t.* its local time series. This brute-force method could generate quadratic computation and communication costs w.r.t. the number of time series under processing at worst (i.e., similar to the idea of cross-join using MapReduce [107, 116]). High communication cost in the distributed context leads to prolonged processing latency and even bottlenecks, due to the increasing time spent on sending, receiving and parsing data [147]. Therefore, DisCoM problem requires a both communication and computation efficient solution.
- Lastly, dynamic correlations is often temporally dependent [125, 143], and it may also exhibit volatile behaviours during abnormal periods or events [28, 143]. Therefore, in addition to correlation detection, the proposed framework should entail a probabilistic model responsible for representing the dynamic nature of correlations in time series streams, such that it can be used for correlation occurrence prediction , correlation change detection and so on [7, 15, 28, 61]. How to efficiently maintain this model on-line is also non-trivial.

This paper is the first work that proposes a continuously distributed mining and modeling framework for dynamic correlations over massive time series. Previous efforts on mining correlations either focus on static time series dataset or work in a centralized way and therefore fail to tackle above challenges in the distributed and dynamic environment (refer to Section 2.2). Overall, this paper makes the following concrete contributions:

- We formally define the problem of continuously mining (lagged) correlations in massive time series based on a distributed real-time computation engine (DisCoM problem). We propose the framework P2H, which aims to optimize both communication and computation cost for DisCoM problem.
- In P2H, we design a novel data shuffling technique parallelism-partitioning based data shuffling, referred to as P^2-data-shuffling, which is able to adaptively replicate and shuffle the sliding window of a time series only to the computing nodes containing its correlation partners, thereby dramatically decreasing the communication cost.
- We propose a δ-hypercube structure based correlation computation approach to prune unnecessary computation over the data shuffled by P^2-data-shuffling. Furthermore, a quality guaranteed dimensionality-reduction method is integrated to P2H framework so as to handle DisCoM problem over long sliding windows or large lags, .
- Furthermore, we design a probabilistic model for correlation inference by considering both the temporal and co-occurrence dependence of correlation occurrence in P2H.
- Theoretical cost analysis for each phase in P2H framework is provided.
- P2H framework and a variety of baseline approaches are implemented based on Apache Storm, a widely used distributed real-time computation engine. Extensive experiments on both synthetic and real datasets demonstrate the effectiveness and efficiency and effectiveness of our approach. in terms of peak capacity, communication cost, processing latency and pruning power.

The rest of the paper is organized as follows. Section 3.3 introduces the problem definition and background knowledge. We describe the proposed P2H framework in Section 3.4 and Section 3.5. Section 3.7 provides the theoretical analysis. We perform exhaustive experimental evaluations in Section 3.8 and offer a conclusion in Section 3.9.

3.2 Related Work

Mining lead-lag relations in a centralized way has been the focus of [32, 81, 82, 95, 114, 114, 115, 115, 117, 143, 143, 155]. They do not consider the data shuffling issue in the distributed setting and modeling of dynamic lead-lag relations. [81, 95, 117] proposed index based approaches to mine correlations in time series datasets. Such approaches are not suitable for our online scenairo, where the index maintenance cost incurs high processing latency as well as communication cost in the distributed environment.

[32, 114, 115, 143, 145] focus on detecting dynamic lead-lag relations in time series streams. All these techniques perform the pair-wise computation to detect lead-lag relations between time series streams while our approach aims to avoid pair-wise computation in distributed

processing of massive time series streams. Moreover, they do not take into account the data shuffling problem in the distributed environment. [114, 115] both exploit geometric series to approximately derive the lead-lag relations. Such approximate methods can be smoothly integrated into our proposed framework. In this paper, we focus on mining exact lead-lag relations. [29] is intended for sequences of discrete values while our paper focuses on numeric time series streams. [102] proposed a dimension reduction based approach to summarise large collections of streams. The StatStream system [155] specializes in discovering correlations using a grid structure, which incurs prohibitive communication cost in the distributed environment. Our proposed framework allows computation pruning as well as discovering the lead-lag relation within a temporal range of interests.

Numerous distributed systems [1, 2, 98, 152] have been developed to process massive data streams in a high-speed environment. Although these systems provide an extensive set of operators for real-time data processing and analytics, they do not support operations on mining and modeling the interaction in data streams, i.e., lead-lag relations in this paper.

Recently, partitioning-based approaches have attracted attention for distributed data processing [46, 116, 138]. Such approaches process static datasets and need an aprori data prescanning step to estimate the data distribution for the shuffling phase. Therefore, they are not suitable for our online setting. [124] puts forward a geometric approach for monitoring function values in the sensor network, which is a different computation paradigm from the distributed real-time computation system in this paper [1, 2, 98, 152]. The data recall phase in [56] could introduce much additional communication cost for long lags relations. [53, 56] assumed time series are synchronised and failed to consider the fluctuation in time series. To the best of our knowledge, no previous work addresses the modeling of lead-lag relations in evolving entities in the distributed environment.

3.3 Preliminaries

In this section, we depict the key concepts of a distributed real-time computation engine following the terminology in [4] and how to process time series data in such a system. Then, we formulate the DisCoM problem.

3.3.1 Distributed Real-time Computation Engine

We focus on in-memory distributed real-time computation engine, where all data processing operations are performed in memory In a typical cluster of a distributed real-time computation engine [1, 2, 98, 152], there is a master node responsible for distributing programs to computing nodes of the cluster, monitoring the status of nodes and recovering nodes from failures. The

computing nodes perform the real data processing.

topology is a job submitted to the cluster, which is a program-described directed acyclic graph. The vertexes are user-defined boxes (see below) and the communication between boxes is dictated by the edges in the topology. A topology is executed indefinitely to continuously process tuples until it is killed. The important concepts are as follows.

- **Tuple** is a key-value(s) pair and the basic data unit communicated among the nodes in the cluster.
- **Box** is a processing element that continuously consumes tuples it receives, processes them according to a user-defined logic, and emits tuples to other boxes that have subscribed to it; we denote a box by $\mathscr{B}^{(x)}$. Typically, while processing a tuple, a box also modifies the key of the tuple.
- **Task** is an instance of a box that is executed in a worker node. A box could have multiple tasks run in parallel. The tuples processed by a task are referred to as the *local data* of this task (*local time series* in this paper). Note that tasks of a box process different local data using the same processing logic defined for that box.
- **Parallelism** of a box is the number of tasks executed in parallel in the cluster. This is a user-defined parameter. The parallelism of box $\mathscr{B}^{(x)}$ is denoted by $\mathrm{p}^{(x)}$. The tasks of $\mathscr{B}^{(x)}$ are denoted by $\mathscr{B}^{(x,1)}$, $\mathscr{B}^{(x,2)}$, ..., $\mathscr{B}^{(x,\mathrm{p}^{(x)})}$.
- **Shuffling function** is a function between two boxes, which determines to which task of the connected box a tuple emitted from a task of the preceding box should be sent. For instance, the by-default key-based shuffling function uses a hash function over the tuple key to send the tuple only to the task to which the hash value of this tuple's key is assigned. Tuples with a certain key are always sent to the same task.

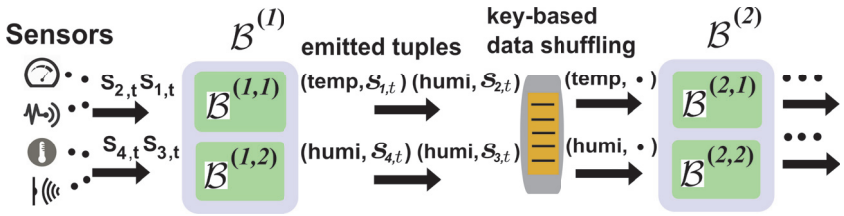


Figure 3.2 – A toy topology example of processing time series in a distributed real-time computation engine

In this paper, a time series stream is a sequence of discrete real-valued observations collected from a sensor or an entity at a regular time interval. We use n to denote the total number of time series streams fed to the cluster. Figure 3.2 illustrates a toy topology for processing time series based on a distributed real-time computation engine. It consists of two boxes $\mathscr{B}^{(1)}$ and $\mathscr{B}^{(2)}$ both with parallelism 2. Assume this topology is intended for processing various sensor

data from distributed sources based on the phenomena they measure. $\mathscr{B}^{(1)}$ is responsible for continuously fetching data from sensors and emitting tuples of the form $(phenomena, s_{i,t})$, where $phenomena$ is the key and $s_{i,t}$ is the value of time series at time instant t. Then, the tasks of $\mathscr{B}^{(2)}$ are able to process for instance temperature and humidity data in parallel.

3.3.2 Problem Statement

For time series i ($i \in \{1,\ldots,n\}$), the sliding window of length h ending at time stamp t is denoted by $\boldsymbol{s}_i^t = (s_{i,t-h+1},\cdots,s_{i,t})$ and $\boldsymbol{s}_i^t \in \mathbb{R}^h$ (t is the latest time index). For the sake of simplicity, we call $\boldsymbol{s}_i^t$ as the sliding window of time series i at t.

In this paper, we make use of Pearson correlation to define lead-lag relations and illustrate our P2H framework. Due to the page limit, we present in Section 3.10 that P2H framework is able to handle lead-lag relations based on Spearman rank correlation, cosine similarity, extended Jacard similarity as well.

Definition 3.3.1 (Pearson correlation)**.** *Pearson correlation between sliding windows $\boldsymbol{s}_i^{t_1}$ and $\boldsymbol{s}_j^{t_2}$ from time series i and j ($\boldsymbol{s}_i^{t_1}, \boldsymbol{s}_j^{t_2} \in \mathbb{R}^h$). is defined as follows:*

$$rel(\boldsymbol{s}_i^{t_1}, \boldsymbol{s}_j^{t_2}) = \frac{(\boldsymbol{s}_i^{t_1} - \mu(\boldsymbol{s}_i^{t_1}) \cdot \mathbb{1}) \cdot (\boldsymbol{s}_j^{t_2} - \mu(\boldsymbol{s}_j^{t_2}) \cdot \mathbb{1})}{(h-1)\sigma(\boldsymbol{s}_i^{t_1})\sigma(\boldsymbol{s}_j^{t_2})} \tag{3.1}$$

where $\mathbb{1}$ is an all-one vector of dimension h, $\sigma(\boldsymbol{s}_i)$ and $\mu(\boldsymbol{s}_i)$ are the sample standard deviation and mean of the elements in $\boldsymbol{s}_i$, respectively.

It is a measure of the linear correlation giving a value between +1 and -1 inclusive, where +1 is total positive correlation, 0 is no correlation, and -1 is total negative correlation. If $t_1 = t_2$, above $rel(\boldsymbol{s}_i^{t_1}, \boldsymbol{s}_j^{t_2})$ measures the synchronised correlation between sliding windows of time series i and j. When $t_1 \neq t_2$, it reflects the lagged correlation, which is formally defined as:

Definition 3.3.2 (τ-lead-lag correlation)**.** *For a pair of time series i and j ($1 \leq i, j \leq n$), given time instant t and lag τ, τ-lead-lag relation $\rho_{i,j}(t,\tau)$ between time series i and j is defined as: $\rho_{i,j}(t,\tau) = rel(\boldsymbol{s}_i^t, \boldsymbol{s}_j^{t+\tau})$ ($\boldsymbol{s}_i^t, \boldsymbol{s}_j^{t+\tau} \in \mathbb{R}^h$). i is the leader and j is the follower.*

τ-lead-lag correlation $\rho_{i,j}(t,\tau)$ measures to what extent time series i leads to a correlated variation in time series j with time lag τ.

In reality, users are more interested in the recent data for timely decision making and thus require to specify the maximum lag for lead-lag relations. Such maximum lag indicates the extent to which one has to look into the past to discover leaders for current time series and is

also interpreted as how far in advance users would like to predict the future trend based on detected lead-lag relations [22, 115].

Now, the DisCoM problem is defined as:

Definition 3.3.3 (DisCoM Problem)**.** *Assume a distributed real-time computation engine is processing n time-series, which are continuously distributed to different nodes of the cluster. For each time series i.e. i, a DisCoM problem requires to*

(1) continuously report the follower time series having $\rho_{i,j}(t,\tau) \geq \epsilon$ and τ within a maximum lag ℓ ($\ell \geq 0$ and $\tau \leq \ell$);

(2) on-line maintain a probabilistic model over the detected correlations such that the probability of correlation occurrences can be inferred.

In the above definition, ϵ is always assumed to be greater than zero in this paper. It can be shown that negative ϵ can be treated as positive [32, 155]. Thus, without loss of generality, henceforth we only focus on the positive threshold ϵ.

3.3.3 Normalization Operation

In this part, we present the normalization operation over sliding windows of time series, which is able to transform the τ-lead-lag correlation between two sliding windows to the Euclidean distance between normalized ones. Such a property will be harnessed by the following P2H framework to design communication and computation optimization methods in the Euclidean space for DisCoM problem.

For Pearson correlation, the normalized sliding window $\hat{\boldsymbol{s}}_i^t$ is obtained via $\hat{\boldsymbol{s}}_i^t = norm(\boldsymbol{s}_i^t) = \frac{\boldsymbol{s}_i^t - \mu(\boldsymbol{s}_i^t)}{\sqrt{h-1}\sigma(\boldsymbol{s}_i^t)}$ [32, 155] and $\hat{\boldsymbol{s}}_i^t \cdot \hat{\boldsymbol{s}}_i^t = 1$. It indicates $|\hat{s}_{i,k}| \leq 1$ and thus the range of variation of the normalized sliding window is known apriori, and is independent of the variation in the original sliding window.

Meanwhile, given two normalized sliding windows $\hat{\boldsymbol{s}}_1^{t_1}$ and $\hat{\boldsymbol{s}}_2^{t_2}$ of $\boldsymbol{s}_1^{t_1}$ and $\boldsymbol{s}_2^{t_2}$, their Pearson correlation can be expressed as:

$$rel(\boldsymbol{s}_1^{t_1}, \boldsymbol{s}_2^{t_2}) = \hat{\boldsymbol{s}}_1^{t_1} \cdot \hat{\boldsymbol{s}}_2^{t_2} \tag{3.2}$$

such that there exists the following equation:

$$\mathscr{D}^2(\hat{\boldsymbol{s}}_1^{t_1}, \hat{\boldsymbol{s}}_2^{t_2}) = 2 - 2 \cdot rel(\boldsymbol{s}_1^{t_1}, \boldsymbol{s}_2^{t_2}) \tag{3.3}$$

, where $\mathscr{D}^2(\hat{\boldsymbol{s}}_1^{t_1}, \hat{\boldsymbol{s}}_2^{t_2}) = (\hat{\boldsymbol{s}}_1^{t_1} - \hat{\boldsymbol{s}}_2^{t_2}) \cdot (\hat{\boldsymbol{s}}_1^{t_1} - \hat{\boldsymbol{s}}_2^{t_2})$.

Consequently, for time series i and j with correlation $\rho_{i,j}(t,\tau) \geq \epsilon$, there must be $\mathscr{D}(\hat{\boldsymbol{s}}_i^t, \hat{\boldsymbol{s}}_j^{t+\tau}) \leq \delta$, where distance δ is related to ϵ as $\delta = \sqrt{2(1-\epsilon)}$. As ϵ decreases, δ increases and vice versa. We can prove that all the supported metrics (i.e., Pearson correlation, Spearman correlation, cosine similarity, extended Jacard similarity) have associated normalization operations (limited to the space, refer to Section 3.10 for details).

3.4 Data shuffling in P2H

This section introduces our core contribution P2H framework. The topology of P2H in the distributed real-time computer engine is shown in Figure 3.3. Box $\mathscr{B}^{(pre)}$ is responsible for maintaining the sliding windows of input time series, updating the normalized sliding windows incrementally [32, 155] and emitting a tuple for each time series consisting of time series id and the normalized sliding window at the current time instant. Between box $\mathscr{B}^{(pre)}$ and $\mathscr{B}^{(cmp)}$, we design a novel **p**arallelism-**p**artitioning based data **s**huffling (P^2-data-shuffling), which is able to adaptively replicate and shuffle a tuple sent by $\mathscr{B}^{(pre)}$ only to the tasks of box $\mathscr{B}^{(cmp)}$ containing leader sliding windows of the one contained in this tuple. Consequently, the communication cost is dramatically reduced compared with the brute-force replicating of sliding windows among all the tasks of $\mathscr{B}^{(cmp)}$. Then, by exploiting the δ-hypercube structure based computation pruning, box $\mathscr{B}^{(cmp)}$ efficiently detects qualified correlations from the sliding windows shuffled to each task of $\mathscr{B}^{(cmp)}$ by P^2-data-shuffling. Finally, box $\mathscr{B}^{(mdl)}$ aggregates the detected correlations from $\mathscr{B}^{(cmp)}$ and exploits the Markov Random Field [96] to model both the co-occurrence and temporal dependency of correlation occurrence. Users can issue correlation inference queries through this model.

3.4.1 Parallelism-Partitioning based Data Shuffling

In this part, we describe the parallelism-partitioning based data shuffling (P^2-data-shuffling) in P2H framework.

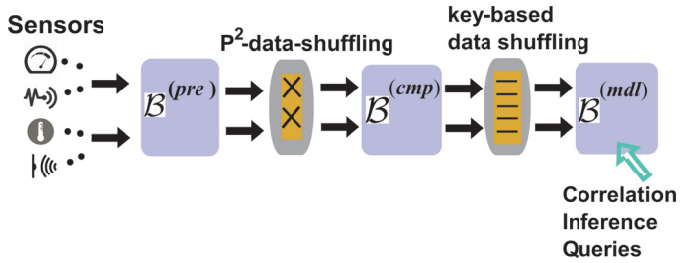


Figure 3.3 – Topology of P2H framework.

Parallelism based Space Partitioning: Recall that the normalized sliding windows are dis-

tributed in a bounded Euclidean space as is discussed in Section 3.3.3. Our motivation of partitioning the space of the normalized sliding windows in P^2-data-shuffling is from the below observation.

Observation 3.4.1. *Figure 3.4(*a*) depicts the 3-dimensional Euclidean space for normalized sliding windows of size* 3*, where a certain normalized sliding window corresponds to one point in this space. We divide the space into two parts on each dimension, thereby producing* $2^3 = 8$ *partitions. Assume that each partition is handled by a unique task of box* $\mathscr{B}^{(cmp)}$ *and the sliding windows mapped to a certain partition are all shuffled to the task responsible for this partition.*

Take the front upper-left partition with yellow-grey region for an example. For the local sliding windows in the yellow-grey region, since they are on the boundary and could have leader sliding windows in the neighbouring partitions, and therefore they should be replicated to such partitions. Meanwhile, the sliding windows outside that yellow-grey region must not have leaders from other partitions. We will formally define the yellow-grey region in this section later.

Therefore, majority of the sliding windows in a partition are shuffled once and do not need replicating, thereby greatly decreasing the communication and computation costs.

Now let us see how to formulate above idea in P^2-data-shuffling. Initially, we apply 2-way partitioning on each dimension of the space over the normalized sliding windows and thus obtain 2^h partitions (we will compare 2-way partitioning with other methods in Section 3.7). However, in order to associate partitions with $\mathrm{p}^{(cmp)}$ tasks of box $\mathscr{B}^{(cmp)}$, we should adjust the dimensionality used for space partitioning. The need for reducing the dimensionality in space partitioning is evaluated as follows: we compute $h_p = \lfloor \log_2(\mathrm{p}^{(cmp)}) \rfloor$ and if $h_p \leq h$, space partitioning only utilizes the latest h_p entries of the sliding window, (e.g., $(\hat{s}_{i,t-h_p+1}, \ldots, \hat{s}_{i,t})$ in $\hat{\boldsymbol{s}}_i^t$) which is sufficient for maintaining the one-to-one correspondence between partitions and tasks of $\mathscr{B}^{(cmp)}$.

Normally, $h_p \ll h$, because h could be up to hundreds or more for mining significant correlations (see the experiment section). It is impossible for $h_p > h$, as it requires $\mathrm{p}^{(cmp)} > 2^h$, which is a prohibitive large value for parallelism $\mathrm{p}^{(cmp)}$. In the implementation, we could set $\mathrm{p}^{(cmp)}$ as an exponential of 2 to make full use of the tasks of $\mathscr{B}^{(cmp)}$.

Based on above partitioning method, the partition vector for a normalized sliding window is defined by function $f_p : \mathbb{R}^h \to \mathbb{R}^{h_p}$:

$$\boldsymbol{p}_i^t = f_p(\hat{\boldsymbol{s}}_i^t) = (\mathrm{sgn}(\hat{s}_{i,t-h_p+1}) \lceil |\hat{s}_{i,t-h_p+1}| \rceil, \ldots, \mathrm{sgn}(\hat{s}_{i,t}) \lceil |\hat{s}_{i,t}| \rceil) \tag{3.4}$$

where $\mathrm{sgn}(x)$ extracts the sign of its argument: $\mathrm{sgn}(x) = 1$ if $x \geq 0$ and $\mathrm{sgn}(x) = -1$ if $x < 0$. Since $-1 \leq \hat{s}_{i,t} \leq 1$, each entry of the partition vector $\boldsymbol{p}_i^t$ is either -1 or 1. Partition vector $\boldsymbol{p}_i^t$ indicates the partition where $\hat{\boldsymbol{s}}_i^t$ locates.

Example 3.4.1.1. *In Figure 3.4(*b*), it depicts the space for 3D sliding windows. Suppose* $p^{(cmp)} = 4$ *then we have* $h_p = 2$*. Since* $h_p < 3$*, only dimensions* t *and* $t-1$ *are used to obtain* 4 *partitions. Each of them corresponds to one hyper-rectangle and is identified by a two dimensional partition vector, as is shown in Figure 3.4(*c*). Three normalized sliding windows (* $\hat{\mathbf{s}}_1^t$*,* $\hat{\mathbf{s}}_2^t$ *and* $\hat{\mathbf{s}}_3^t$ *) respectively locate in three partitions in Figure 3.4(*c*). They distribute on the surface of a hyper-sphere centred at the origin point in the space, since normalized sliding windows have unit norms.*

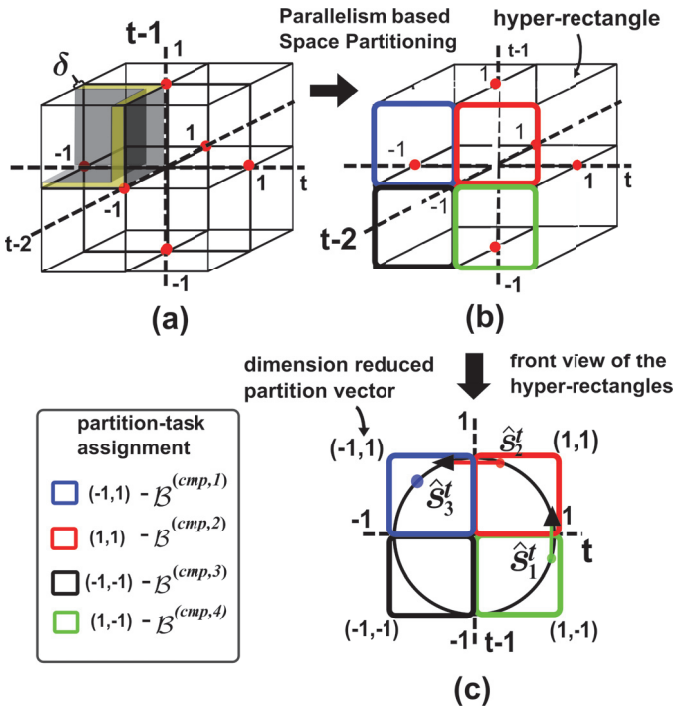


Figure 3.4 – Illustration of P^2-data-shuffling in 3D space (best viewed in color). (a) motivation example of partitioning the space of normalized sliding windows for communication optimization. (b) parallelism-partitioning: hyper-rectangles derived by 2-way partitioning the space over dimension t and $t-1$. (c) normalized sliding window replication amongst partitions; each partition is handled by a unique task of box $\mathscr{B}^{(cmp)}$.

Data Replication and Shuffling among Partitions: In this part, we elaborate on the efficient and adaptive replicating of time series sliding windows among the partitions. We first define the target-partition set for a normalized sliding window during the data shuffling phase as follows:

Definition 3.4.1 (Target-Partition Set)**.** A set of target partition(s) denoted by $\mathscr{R}_i^t$ of normalized sliding window $\hat{\mathbf{s}}_i^t$ contains all the partitions which host the leaders of $\hat{\mathbf{s}}_i^t$. $\hat{\mathbf{s}}_i^t$ should be only replicated to the tasks corresponding to the partitions in $\mathscr{R}_i^t$.

Different sliding windows have different sets of target partitions. We use normalized sliding window $\hat{\boldsymbol{s}}_i^t$ to illustrate the following data replicating approach. In order to construct $\mathscr{R}_i^t$, besides the partition $\boldsymbol{p}_i^t$ where a $\hat{\boldsymbol{s}}_i^t$ locates, the key is to find out the alternative partition(s) which could have leader of $\hat{\boldsymbol{s}}_i^t$. Our target-partition set construction algorithm consists of two steps:

First, we build a dimension subset $\mathscr{H}_i \subseteq \{t-h_p+1,\ldots,t\}$ for $\hat{\boldsymbol{s}}_i^t$ based on the following lemma. Such a dimension subset indicates the dimension(s) along which to replicate sliding window $\hat{\boldsymbol{s}}_i^t$.

Lemma 3.4.1 (Dimension Subset Selection)**.** *For a sliding window $\hat{\boldsymbol{s}}_i^t$ in partition $\boldsymbol{p}_i^t$, a dimension $k \in (t-h_p+1,\ldots,t)$ is added to the set $\mathscr{H}_i$ if and only if $(2-2\sqrt{1-\hat{s}_{i,k}^2}) \leq \delta$*

Proof. Given a dimension k, we first define an *opposite partition* of partition $\boldsymbol{p}_i^t$ on dimension k as:

$$\tilde{\boldsymbol{p}}_i^t = (p_{i,t-h_p+1},\ldots,-p_{i,k},\ldots,p_{i,t})$$

namely, $\tilde{\boldsymbol{p}}_i^t$ is on the opposite direction of partition $\boldsymbol{p}_i^t$ only on dimension k while retains the same values on other dimensions as $\boldsymbol{p}_i^t$. For the sake of simplicity, we will denote the time interval of a sliding window by $[1,h]$ and the interval for partition vector by $[1,h_p]$ and $k \in \{1,\ldots,h_p\}$ throughout this proof.

Then, we formulate an optimization problem to derive the minimum distance between sliding window $\hat{\boldsymbol{s}}_i^t$ and the one in the opposite partition $\tilde{\boldsymbol{p}}_i^t$ and then compare it with distance threshold δ so as to determine whether dimension k should be selected. If dimension k is chosen, the following described shuffling process will replicate this sliding window to the opposite partition on dimension k.

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \ & \sum_{j=1}^{h}(x_j-\hat{s}_{i,j})^2 \\ \text{s.t.:} \ & x_j\hat{s}_{i,j} \geq 0, \ j=\{1,\ldots,h_p\}\setminus k \\ & x_j\hat{s}_{i,j} \leq 0, \ j=k \\ & \sum_{1}^{h} x_j^2 = 1 \end{aligned} \tag{3.5}$$

where $\boldsymbol{x} \in \mathbb{R}^h$ represents a sliding window in $\tilde{\boldsymbol{p}}_i^t$ having the minimum distance to $\hat{\boldsymbol{s}}_i^t$. Based on the definition of opposite partition on dimension k, the k-entry of the sliding window $\boldsymbol{x}$ in $\tilde{\boldsymbol{p}}_i^t$ should have the opposite sign as $\hat{s}_{i,k}$ while $\tilde{\boldsymbol{p}}_i^t$ has the same sign as $\hat{\boldsymbol{s}}_i^t$ on the other dimensions. This gives rise to the first two constraints. The third constraint ensures that $\boldsymbol{x}$ is a normalized

sliding window (i.e., $\boldsymbol{x} \cdot \boldsymbol{x} = 1$).

Then, we obtain the Lagrange formula of 3.5 as:

$$\begin{aligned}\mathscr{L}(\boldsymbol{x}, \lambda, \upsilon) = \sum_{j=1}^{h} (x_j - \hat{s}_{i,j})^2 - \sum_{j \in \{1,\ldots,h_p\} \setminus k} \lambda_j \hat{s}_{i,j} x_j \\ + \lambda_k \hat{s}_{i,k} x_k + \upsilon(\sum_{1}^{h} x_j^2 - 1)\end{aligned} \tag{3.6}$$

Based on KKT conditions for optimization problem 3.5, we first derived the optimal $\boldsymbol{x}^*$ by the derivative of $\mathscr{L}(\boldsymbol{x}, \lambda, \upsilon)$ *w.r.t.* $\boldsymbol{x}$ as follows:

$$\begin{aligned}&2x_j^* - 2\hat{s}_{i,j} - \lambda_j \hat{s}_{i,j} + 2\upsilon x_j^* = 0,\ j = \{1, \ldots, h_p\} \setminus k \\ &2x_m^* - 2\hat{s}_{i,m} + 2\upsilon x_m^* = 0,\ m = \{h_p + 1, \ldots, h\} \\ &2x_k^* - 2\hat{s}_{i,k} + \lambda_k \hat{s}_{i,k} + 2\upsilon x_k^* = 0\end{aligned} \tag{3.7}$$

we can obtain $\boldsymbol{x}^*$ as follows: $x_j^* = \frac{2s_{i,j} + \lambda_j \hat{s}_{i,j}}{2\upsilon+2}$, $x_m^* = \frac{2s_{i,m}}{2\upsilon+2}$, $x_k^* = \frac{2s_{i,k} - \lambda_k \hat{s}_{i,j}}{2\upsilon+2}$. Then, above $\boldsymbol{x}^*$ is introduced to the KKT conditions for the constrains in problem 3.5 to derive λ_k^* and λ_j^* as:

$$\begin{aligned}-\lambda_j^* \hat{s}_{i,j} \frac{2\hat{s}_{i,j} + \lambda_j^* \hat{s}_{i,j}}{2\upsilon + 2} = 0,\ \lambda_j^* \geq 0 \\ \lambda_k^* \hat{s}_{i,k} \frac{2\hat{s}_{i,k} - \lambda_k^* \hat{s}_{i,k}}{2\upsilon + 2} = 0,\ \lambda_k^* \geq 0\end{aligned} \tag{3.8}$$

Thus, λ_j^* should be zero when $\hat{s}_{i,j} \neq 0$. For λ_k, the optimal $\boldsymbol{x}^*$ in the opposite partition of $\boldsymbol{p}_i^t$ on dimension k should on the hyperplane $x_k = 0$ to attain the minimum distance between $\hat{\boldsymbol{s}}_i^t$ and $\boldsymbol{x}$ and therefore $\lambda_k^* = 2$. Now based on the values of λ_j^* and λ_k^*, we can obtain υ^* by using the equality constraint in 3.5 as: $\upsilon^* = \sqrt{1 - \hat{s}_{i,k}^2} - 1$.

So far, we can get the values of $\boldsymbol{x}^*$ and the minimum distance between $\hat{\boldsymbol{s}}_i^t$ and $\boldsymbol{x}^*$ is $2 - 2\sqrt{1 - \hat{s}_{i,k}^2}$. Lemma 3.4.1 is proved. □

Second, dimension subset $\mathscr{H}_i$ is leveraged to derive target-partition set $\mathscr{R}_i^t$. Based on Lemma 3.4.1, since $\mathscr{H}_i$ contains the dimensions on which target partitions have opposite entries as $\boldsymbol{p}_i^t$, target-partition set $\mathscr{R}_i^t$ of $\hat{\boldsymbol{s}}_i^t$ is built as the set of all the permutations of $\boldsymbol{p}_i^t$, such that *only* the entries corresponding to the dimensions present in $\mathscr{H}_i$ are permuted and the remaining are held constant. For example, if $\boldsymbol{p}_i^t = (-1, 1, 1)$ and $\mathscr{H}_i = \{2, 3\}$, then only the 2^{nd} and 3^{rd} dimension of $\boldsymbol{p}_i^t$ are permuted to form $\mathscr{R}_i^t$ as follows: $\mathscr{R}_i^t = \{(-1, -1, -1), (-1, -1, 1), (-1, 1, -1), (-1, 1, 1)\}$.

The above target-partition construction algorithm has the following desirable property:

Theorem 3.4.1. *For a normalized sliding window $\hat{\boldsymbol{s}}_i^t$ in $\boldsymbol{p}_i^t$, the target-partitions in R_i^t constructed by above dimension subset selection and the permutation process over $\mathscr{H}_i$ contain all the leaders of $\hat{\boldsymbol{s}}_i^t$, while the partitions not present in $\mathscr{R}_i^t$ must not have leaders of $\hat{\boldsymbol{s}}_i^t$.*

Proof. The proof relies on Lemma 3.4.1 and refer to Section 4.8 for details. □

In summary, for each tuple emitted by box $\mathscr{B}^{(pre)}$, P^2-data-shuffling checks the condition given by Lemma 3.4.1 to generate the dimension subset $\mathscr{H}_i$. If $\mathscr{H}_i = \emptyset$, $\hat{\boldsymbol{s}}_i^t$ is only shuffled to the task responsible for partition $\boldsymbol{p}_i^t$. Otherwise, $\mathscr{H}_i$ is used for creating the target-partition set $\mathscr{R}_i^t$, and $\hat{\boldsymbol{s}}_i^t$ is replicated to the tasks corresponding to the partitions in $\mathscr{R}_i^t$. Moreover, due to the one-to-one correspondence between partitions and tasks of box $\mathscr{B}^{(cmp)}$, the size of target-partition set is bounded by parallelism $\mathrm{p}^{(cmp)}$, which is a system parameter independent of n and h.

Example 3.4.1.2. *Consider Figure 3.4(c), sliding window $\hat{\boldsymbol{s}}_3^t$ (in blue) is contained in the partition $(-1,1)$ and both dimension t and $t-1$ of $\hat{\boldsymbol{s}}_3^t$ are not qualified for the condition in Lemma 3.4.1, therefore it is only shuffled to partition $(-1,1)$. Sliding window $\hat{\boldsymbol{s}}_1^t$ (in green) is qualified on dimension t, therefore in addition to partition $(1,-1)$ where $\hat{\boldsymbol{s}}_1^t$ is located, it is also replicated to partition $(1,1)$. Likewise, $\hat{\boldsymbol{s}}_2^t$ is replicated to partitions $(1,1)$ and $(-1,1)$.*

Finally, we verify the correctness and completeness of P^2-data-shuffling through the following theorem.

Theorem 3.4.2 (Correctness and Completeness of P^2)**.** *Through P^2-data-shuffling, each task of box $\mathscr{B}^{(cmp)}$ receives the normalized sliding windows located in the partition corresponding to this task and the sliding windows from other partitions that are followers of this task's local ones. Therefore, the leaders (or followers) of each time series can be detected by the tasks of $\mathscr{B}^{(cmp)}$.*

Proof. Refer to Section 4.8 for details. □

Adaptive Partitioning: Our P^2-data-shuffling can be extended to support space partitioning adaptive to the change of data distribution in sliding windows. his requires that the partitioning position on each dimension could be adjusted in order to attain load balancing for partitions. We briefly give a possible solution here and leave it for our future work. First, an additional box responsible for monitoring the global distribution of the normalized sliding windows on each dimension is added to the current topology. The output of this box, partition positions on each dimension derived from the collected data distribution information, is sent to box $\mathscr{B}^{(pre)}$ which perform the operations based on the received partition information as described above.

3.5 Correlation Mining in P2H

Through above steps, each task of box $\mathscr{B}^{(cmp)}$ receives the normalized sliding windows shuffled by P^2-data-shuffling. In this section, we present how each task of box $\mathscr{B}^{(cmp)}$ efficiently discovers correlations from these sliding windows by using the δ-hypercube structure based computation pruning. First, we introduce the δ-hypercube structure in Euclidean space as well as the favourable property of it for computation pruning. A δ-hypercube structure is a h-dimensional hypercube edges of which have length δ [46]. The δ-hypercube in which a normalized sliding window (e.g., $\hat{\boldsymbol{s}}_i^t$) is contained, is identified by its δ-**hypercube vector** as: $\boldsymbol{c}_i^t = f_c(\hat{\boldsymbol{s}}_i^t) = \left(\left\lceil \frac{\hat{s}_{i,t-h+1}}{\delta} \right\rceil, ..., \left\lceil \frac{\hat{s}_{i,t}}{\delta} \right\rceil\right)$. Now we define neighbouring δ-hypercubes of $\boldsymbol{c}_i^t$ as:

Definition 3.5.1 (neighbouring δ-hypercubes)**.** The set of neighbouring δ-hypercubes of $\boldsymbol{c}_i^t$ is defined as $\mathscr{N}(\boldsymbol{c}_i^t) = \{\boldsymbol{c}_j^t | \ \forall k \in 1, \ldots, h, |c_{i,k} - c_{j,k}| \leq 1\}$ and $\boldsymbol{c}_i^t \in \mathscr{N}(\boldsymbol{c}_i^t)$.

Given the sliding windows mapped to different δ-hypercubes, the δ-hypercube structure has such a desirable property for efficiently finding the sliding windows within δ distance to a certain one (e.g. $\hat{\boldsymbol{s}}_i^t$), namely the leaders of $\hat{\boldsymbol{s}}_i^t$, without performing pair-wise computations [114, 115]:

Lemma 3.5.1. *For a sliding window $\hat{\boldsymbol{s}}_i^t$ in δ-hypercube $\boldsymbol{c}_i^t$, the sliding windows within δ distance to $\hat{\boldsymbol{s}}_i^t$ are only contained in the δ-hypercubes from $\mathscr{N}(\boldsymbol{c}_i^t)$. All the δ-hypercubes out of $\mathscr{N}(\boldsymbol{c}_i^t)$ must not contain the sliding windows close to $\hat{\boldsymbol{s}}_i^t$ within δ.*

Proof. Refer to Section 3.10 for details. □

Example 3.5.0.1. *In the partitions over the Euclidean space of Figure 3.5(*a*), a blue δ-hypercube containing normalized sliding window $\hat{\boldsymbol{s}}_i^t$ in partition* $(-1, 1)$ *(blue one) is shown. The left small figure depicts the set of the neighbouring δ-hypercubes around the blue one (the blue star represents $\hat{\boldsymbol{s}}_i^t$).*

3.5.1 Correlation Computation

In this part, we propose the δ-hypercube structure based computation pruning criteria for efficiently detecting correlations in box $\mathscr{B}^{(cmp)}$.

First, in box $\mathscr{B}^{(cmp)}$ each task maintains a time-stamped queue to organize the δ-hypercubes associated with the contained sliding windows. Each entry of this queue corresponds to a unique time instant and contains a set of δ-hypercubes having the normalized sliding windows at this time instant. We denote the set of δ-hypercubes in the entry at time t by $\mathscr{C}^t$.

Then, in each task of $\mathscr{B}^{(cmp)}$ we categorize its δ-hypercubes in $\mathscr{C}^t$ into the following two types, which will be used to derive the pruning criteria later:

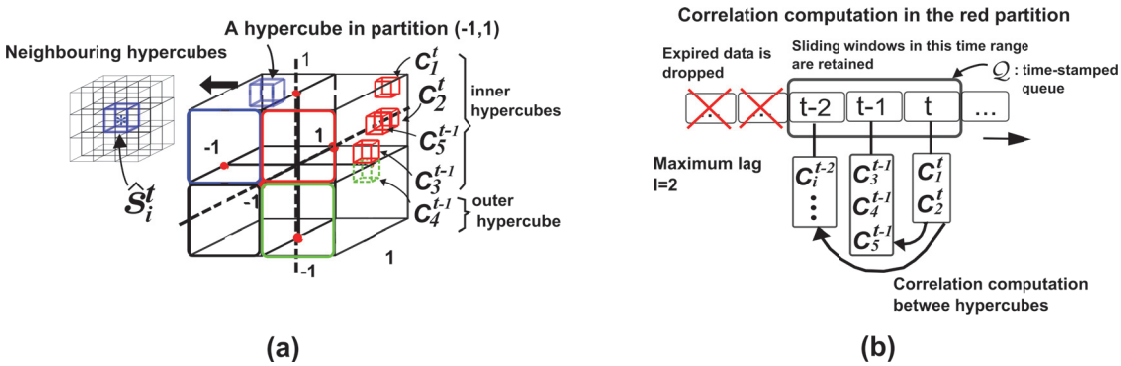


Figure 3.5 – Correlation computation. (best viewed in colour) (a) δ-hypercube (blue cubic) containing normalized sliding window $\hat{\boldsymbol{s}}_i^t$ (blue star) in the blue partition $(-1,1)$ and its neighbouring δ-hypercubes; inner and outer δ-hypercubes in the red partition. (b) The box task responsible for the red partition: δ-hypercubes prior to $t-2$ are dropped due to the maximum lag $\ell = 2$; The arrows between δ-hypercube set at different entries of the queue indicate the correlation computation between them.

Definition 3.5.2 (Inner δ-hypercube)**.** Inner δ-hypercube is the one containing the normalized sliding windows mapped to the partition corresponding to this task.

Definition 3.5.3 (Outer δ-hypercube)**.** Outer δ-hypercube is the one containing the normalized sliding windows originally mapped to different partitions from the one corresponding to this task, but replicated to this task by P^2-data-shuffling from box $\mathscr{B}^{(pre)}$.

In order to find the qualified correlations in each task at time t, the following three lemmas provide a step-by-step way to prune computation. The lemma below first points out the pairs of δ-hypercubes whose contained sliding windows should be examined.

Lemma 3.5.2. *The correlation computation should be performed over a pair of sliding windows respectively from a δ-hypercube in $\mathscr{C}^t$ and an inner δ-hypercube prior to t.*

Proof. Refer to Section 3.10 for details. □

For two δ-hypercubes qualified for Lemma 3.5.2 (e.g., $\boldsymbol{c}_i^t$ and $\boldsymbol{c}_j^{t_1}$, $t_1 < t$), the following lemma determines whether a sliding window from $\boldsymbol{c}_i^t$ has leaders in $\boldsymbol{c}_j^{t_1}$.

Lemma 3.5.3. *For a sliding window $\hat{\boldsymbol{s}}_i^t$ from δ-hypercube $\boldsymbol{c}_i^t$, its minimum distance to a δ-hypercube $\boldsymbol{c}_j^{t_1}$ is defined as:* $D_m(\hat{\boldsymbol{s}}_i^t, \boldsymbol{c}_j^{t_1}) = \sqrt{\sum_{k=0}^{h-1} \begin{cases} (\hat{s}_{i,t-k}^t - c_{i,t-k}^t \cdot \delta)^2 & : c_{i,t-k}^t < c_{j,t-k}^t \\ 0 & : c_{i,t-k}^t = c_{j,t-k}^t \\ (\hat{s}_{i,t-k}^t - c_{i,t-k}^t \cdot \delta + \delta)^2 & : c_{i,t-k}^t > c_{j,t-k}^t \end{cases}}$. *If*

Algorithm 4 Correlation computation in each task of box $\mathscr{B}^{(cmp)}$

Input: local normalized sliding windows at time instant t, δ

1: Initialize a empty entry for $\mathscr{C}^t$ in queue $\mathscr{Q}$
2: **for each** received normalized sliding window $\hat{\boldsymbol{s}}_i^t$ at time t **do**
3: derive $\boldsymbol{c}_i^t = \left(\left\lceil \frac{\hat{s}_{i,t-h+1}}{\delta} \right\rceil, ..., \left\lceil \frac{\hat{s}_{i,t}}{\delta} \right\rceil\right)$;
4: add $\hat{\boldsymbol{s}}_i^t$ to the sliding window set of δ-hypercube $\boldsymbol{c}_i^t$;
5: **end for**
6: drop δ-hypercube set $\mathscr{C}^{t-\ell-1}$ and associated sliding windows in $\mathscr{Q}$
7: **for each** $k = 1, \cdots, \ell$ **do**
8: **for each** pairs of δ-hypercubes $\boldsymbol{c}^t$ and $\boldsymbol{c}^{t-k}$ respectively from $\mathscr{C}^t$ and $\mathscr{C}^{t-k}$ **do**
9: **if** $\boldsymbol{c}^{t-k}$ is inner hypercube and $\boldsymbol{c}^t$ and $\boldsymbol{c}^{t-k}$ are neighbouring δ-hypercubes based on Definition 3.5.1 **then**
10: For $\hat{\boldsymbol{s}}_i^t$ and $\hat{\boldsymbol{s}}_j^{t-k}$ respectively from $\boldsymbol{c}^t$ and $\boldsymbol{c}^{t-k}$:
11: **if** they are not purned by Lemma 3.5.3 and Lemma 3.5.4 **then**
12: perform correlation computation over $\hat{\boldsymbol{s}}_i^t$ and $\hat{\boldsymbol{s}}_j^{t-k}$
13: **end if**
14: **end if**
15: **end for**
16: **end for**

$D_m(\hat{\boldsymbol{s}}_i^t, \boldsymbol{c}_j^{t_1}) > \delta$, there is no leader of $\hat{\boldsymbol{s}}_i^t$ in δ-hypercube $\boldsymbol{c}_j^{t_1}$ and the correlation computation for $\hat{\boldsymbol{s}}_i^t$ w.r.t. $\boldsymbol{c}_j^{t_1}$ *is pruned.*

Proof. Refer to Section 3.10 for details. □

Now if sliding window $\hat{\boldsymbol{s}}_i^t$ is not filtered out by Lemma 3.5.3 for $\boldsymbol{c}_j^{t_1}$, we further propose the probabilistic estimation of correlations between $\hat{\boldsymbol{s}}_i^t$ and any sliding window in $\boldsymbol{c}_j^{t_1}$.

Lemma 3.5.4. *Define functions $max(\boldsymbol{x})$ and $min(\boldsymbol{x})$ ($\boldsymbol{x} \in \mathbb{R}^h$), which respectively return the maximum and minimum element values in $\boldsymbol{x}$. For a sliding windows $\hat{\boldsymbol{s}}_i^t$ in $\boldsymbol{c}_i^t$, if $\hat{\boldsymbol{s}}_j^{t_1}$ in $\boldsymbol{c}_j^{t_1}$ addresses $\|\hat{\boldsymbol{s}}_i^t - min(\hat{\boldsymbol{s}}_i^t) \cdot \mathbb{1}\|_1 \cdot \sqrt{\frac{(max(\hat{\boldsymbol{s}}_j^{t_1}) - min(\hat{\boldsymbol{s}}_j^{t_1}))^2 \cdot ln(\frac{1}{\theta})}{2h}} \geq \epsilon$, then the true correlation between $\hat{\boldsymbol{s}}_i^t$ and $\hat{\boldsymbol{s}}_j^{t_1}$ is above ϵ with probability above $1-\theta$.*

Proof. (**Sketch**) Define $\boldsymbol{r} = \frac{\hat{\boldsymbol{s}}_i^t - min(\hat{\boldsymbol{s}}_i^t) \cdot \mathbb{1}}{\|\hat{\boldsymbol{s}}_i^t - min(\hat{\boldsymbol{s}}_i^t) \cdot \mathbb{1}\|_1}$ ($\boldsymbol{r} \in \mathbb{R}^h$), such that $\sum_{k=1}^{h} r_k = 1$, which can be thought of as a discrete probability mass function. Based on $rel(\hat{\boldsymbol{s}}_i^t, \hat{\boldsymbol{s}}_j^{t_1}) = \hat{\boldsymbol{s}}_i^t \cdot \hat{\boldsymbol{s}}_j^{t_1}$ and $\mathbb{1} \cdot \hat{\boldsymbol{s}}_j^{t_1} = 0$, $\frac{\hat{\boldsymbol{s}}_i^t \cdot \hat{\boldsymbol{s}}_j^{t_1}}{\|\hat{\boldsymbol{s}}_i^t - min(\hat{\boldsymbol{s}}_i^t) \cdot \mathbb{1}\|_1} = \boldsymbol{r} \cdot \hat{\boldsymbol{s}}_j^{t_1} = \sum_{k=1}^{h} r_k \hat{s}_j^{t_1}$ The rightmost formula of this equation can be considered

as the expectation of random variable $\hat{s}_j^{t_1}$. Then we apply the Hoeffding bound to derive the probabilistic bound for correlation $\hat{\boldsymbol{s}}_i^t \cdot \hat{\boldsymbol{s}}_j^{t_1}$. Refer to Section 3.10 for details. □

Based on this lemma, for finding the leaders of a sliding window $\hat{\boldsymbol{s}}_i^t$, $\|\hat{\boldsymbol{s}}_i^t - min(\hat{\boldsymbol{s}}_i^t) \cdot \mathbb{1}\|_1$ is first calculated. Then, the correlation estimate between any sliding window and $\hat{\boldsymbol{s}}_i^t$ is derived in constant time by Lemma 3.5.4. If the estimated value is above ϵ, we can directly report this correlation with high confidence. Otherwise, correlation computation is performed on this pair of sliding windows.

Algorithm 4 shows how above lemmas work together to efficiently discover correlations. Each task of $\mathscr{B}^{(cmp)}$ initializes the set of δ-hypercubes $\mathscr{C}^t$ by assigning sliding windows at time t to different δ-hypercubes and then drops the expired δ-hypercube set $\mathscr{C}^{t-l}$ and the associated sliding windows (line $1-6$). Then, each δ-hypercube in $\mathscr{C}^t$ is compared with the δ-hypercubes in $\mathscr{C}^{t-k}(k = 1,\ldots,\ell)$ by checking the conditions given in Lemma 3.5.1 and Lemma 3.5.2 (line $7-9$). For two qualified δ-hypercubes, the correlation computation is performed over the pairs of sliding windows that are not pruned by Lemma 3.5.3 and Lemma 3.5.4 (line $10-16$). and then qualified correlation are output(line $13-16$).

Example 3.5.1.1. *Figure 3.5(*b*) exhibits a correlation computation example with maximum lag $\ell = 2$. In the task of $\mathscr{B}^{(cmp)}$ corresponding to the red partition, as shown in Figure 3.5(*a*) local sliding windows at time t are mapped to two δ-hypercubes $\boldsymbol{c}_1^t$ and $\boldsymbol{c}_2^t$, while sliding windows at $t-1$ are assigned to two inner δ-hypercubes $\boldsymbol{c}_3^{t-1}$, $\boldsymbol{c}_5^{t-1}$ and a outer $\boldsymbol{c}_4^{t-1}$ from the green partition. All these δ-hypercubes are assigned to the respective entries in the queue based on their time instants, as is shown in Figure 3.5(*b*) Based on the computation pruning lemmas the correlation computation is only performed over pairs of sliding windows respectively from δ-hypercube $\boldsymbol{c}_2^t$ and $\boldsymbol{c}_5^{t-1}$.*

In the final step, all the qualified correlations that are emitted by tasks of box $\mathscr{B}^{(cmp)}$ are aggregated by box $\mathscr{B}^{(mdl)}$ to model the dynamics of correlations [22, 143].

3.5.2 Integrating Dimensionality-Reduction Techniques

The motivation of integrating dimensionality-reduction here is to overcome the curse of dimensionality [139] incurred by long sliding windows. On the other hand, as box $\mathscr{B}^{(cmp)}$ needs to maintain the sliding windows of time series from current time instant t to $t-\ell$, dimensionality-reduction can decrease the memory consumption in box $\mathscr{B}^{(cmp)}$ and avoid expensive I/O costs.

Now we present how to adapt a dimensionality-reduction technique, namely random projections to P2H framework. We denote such enhanced P2H as $\mathrm{P^2H^+}$. Random projections

have been found to be a computationally efficient, yet sufficiently accurate method for dimensionality-reduction of high-dimensional data [5, 18, 80]. Moreover, it has proved to be a very effective tool for both cooperative and uncooperative time series [18, 32]. (Our P2H is also compatible with other dimensionality-reduction methods)

The main idea of random projections comes from the Johnson-Lindenstrauss Lemma [7]. The original h dimensional sliding windows can be projected into a d ($d \ll h$) dimensional subspace through a $d \times h$ random matrix V whose columns $\boldsymbol{v}_i \in \mathbb{R}^h (i = 1, \ldots, d)$. For instance, the dimension-reduced sliding window of $\hat{\boldsymbol{s}}_i^t$ is expressed as $\check{\boldsymbol{s}}_i^t = V \cdot \hat{\boldsymbol{s}}_i^t = (\hat{\boldsymbol{s}}_i^t \cdot \boldsymbol{v}_1, \cdots, \hat{\boldsymbol{s}}_i^t \cdot \boldsymbol{v}_d)$ and $\check{\boldsymbol{s}}_i^t \in \mathbb{R}^d$. We choose the sparse random projection for our real-time environment [5], which is able to efficiently update $\check{\boldsymbol{s}}_i^t$. The elements of the random matrix V are drawn from the following distribution:

$$v_{i,j} = \begin{cases} +\sqrt{3} & with\ probability\ 1/6 \\ 0 & with\ probability\ 2/3 \\ -\sqrt{3} & with\ probability\ 1/6 \end{cases}$$

such that only 1/3 of the data as expected in a random vector $\boldsymbol{v}_i$ needs to be processed.

Meanwhile, we have the following results regarding the distances preservation between dimension-reduced normalized sliding windows.

Theorem 3.5.1 (Random Projection)**.** *[5] For any two normalized sliding windows, e.g., $\hat{\boldsymbol{s}}_i^t$ and $\hat{\boldsymbol{s}}_j^t$, when d satisfies $d \geq \frac{4+2\beta}{\varepsilon^2/2-\varepsilon^3/3} log(n)$, we have*

$$(1-\varepsilon)||\hat{\boldsymbol{s}}_i^t - \hat{\boldsymbol{s}}_j^t|| \leq ||\check{\boldsymbol{s}}_i^t - \check{\boldsymbol{s}}_j^t|| \leq (1+\varepsilon)||\hat{\boldsymbol{s}}_i^t - \hat{\boldsymbol{s}}_j^t||$$

with probability at least $1 - n^{-\beta}$

Tasks of $\mathscr{B}^{(pre)}$ perform random projections on individual normalized sliding window and send only these dimension-reduced sliding windows to box $\mathscr{B}^{(cmp)}$ (we will see how to set d in Section 5.5). Due to the distance preserving property as is shown in Theorem 3.5.1, P^2-data-shuffling between box $\mathscr{B}^{(pre)}$ and $\mathscr{B}^{(cmp)}$ is able to work in the space of the dimension-reduced normalized sliding windows. Then, box $\mathscr{B}^{(cmp)}$ is able to organize the dimension-reduced sliding windows using the δ-hypercube structure and performs the aforementioned algorithm of correlation computation.

3.6 Correlation Modeling Module

In this part, we describe the correlation relation modeling procedure for each entity. For simplicity, we use an example entity i to illustrate the proposed model.

Correlation Occurrence Model:.
For entity i, $\mathscr{B}^{(mdl)}$ maintains a set of time instants denoted by A_u^τ, which records the time

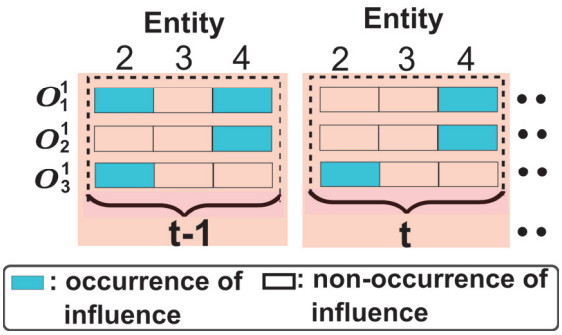


Figure 3.6 – Observations of correlation occurrence variable $\boldsymbol{o}_\tau^1$ for entity 1 and lag $\tau \in \{1,2,3\}$. (best viewed in colour)

instants when entity i has correlation on u with lag τ. Let $\boldsymbol{o}_\tau^i = \{0,1\}^{n-1}$ ($\tau \in \{1,\ldots,\ell\}$) denote the random variable of entity i's correlation occurrence with lag τ. For the elements of $\boldsymbol{o}_\tau^i$, $o_{\tau,u}^i = 1$ ($u \in \mathscr{N} \setminus i$) indicates entity i has correlation on entity u with lag τ. We say that the correlation of i on entity u and v co-occurs whenever their corresponding $o_{\tau,u}^i$ and $o_{\tau,v}^i$ are both one in $\boldsymbol{o}_\tau^i$. For instance, in the toy example in Figure 3.6, at time instant $t-1$, lead-lag detection module returns the detected occurrences in the pink area at $t-1$ for entity 1, which represent a sample of $\boldsymbol{o}_\tau^1$. Now, we model the temporal correlation occurrence process of length ℓ by the Markov chain as: $P(\boldsymbol{o}_1^i,\ldots,\boldsymbol{o}_\ell^i) = P(\boldsymbol{o}_1^i)\prod_{k=1}^{\ell-1} P(\boldsymbol{o}_{k+1}^i|\boldsymbol{o}_k^i)$.

The joint distribution of $\boldsymbol{o}_1^i$ is modeled using a Markov Random Field model parameterized through correlation co-occurrences [15, 96]:

$$P(\boldsymbol{o}_1^i) = \frac{1}{Z_1} exp\Big\{ \sum_{u,v\in\{\mathscr{N}-i\}} \alpha_{u,v}^1 \mathbb{I}(o_{1,u}^i, o_{1,v}^i) + \sum_{u\in\{\mathscr{N}-i\}} \alpha_u^1 o_{1,u}^i \Big\}$$

where Z_1 is a normalizing constant. The influence co-occurrence is assessed based on two-node cliques, namely $\mathbb{I}(o_{1,u}^i, o_{1,v}^i) = 1$ if both $o_{1,u}^i, o_{1,v}^i = 1$ and zero otherwise. The $\alpha_{u,v}^1$ is the co-occurrence potential and $\alpha_{u,v}^1 = \alpha_{v,u}^1$. α_u^1 is self-occurrence potential. Our model favors co-occurrences over non co-occurrences.

Then, the correlation occurrence variable with lag $k+1$ conditional on the occurrence variable

with lag k is modeled as:

$$P(\boldsymbol{o}^i_{k+1}|\boldsymbol{o}^i_k) = \frac{1}{Z_{k+1}} exp\Big\{ \sum_{u,v\in\{\mathcal{N}-i\}} \alpha^{k+1}_{u,v}\mathbb{I}(o^i_{k+1,u}, o^i_{k+1,v}) + \sum_{u\in\{\mathcal{N}-i\}} \beta^{k+1}_u \mathbb{I}(o^i_{k+1,u}, o^i_{k,u})\Big\} \tag{3.9}$$

The term $\beta^{k+1}_u \mathbb{I}(o^i_{k+1,u}, o^i_{k,u})$ in this model accounts for the temporal dependence of correlation on entity u. β^{k+1}_u is the temporal-dependence potential.

Training and Inference:.
Our correlation occurrence model is maintained in the real-time and distributed environment, and the potential parameters above should be efficiently and continuously updated. Therefore, we employ the empirical computation [15] to estimate the co-occurrence and temporal-dependence potentials as follows:

$$\alpha^k_{u,v} = \frac{|A^k_u \cap A^k_v|}{t},\ \alpha^1_u = \frac{|A^1_u|}{t},\ \beta^{k+1}_u = \frac{|A^{k+1}_u \cap A^k_u|}{t} \tag{3.10}$$

where t could represent the time interval at which the potentials are updated. As a result, these potentials are incrementally learnt in dynamic environment.

The normalizing constants in the denominators of above models are over all exponential possible configurations for $\boldsymbol{o}^i_k$ and are computational intractable. Therefore, we resort to Gibbs sampling, a flavor of Markov Chain Monte Carlo (MCMC) method for the inference of correlation occurrences. Gibbs sampling fixes all but one random variable, samples that one conditioned on the others, and then repeats the process for each random variable [96]. So, all we need are the conditional distributions as follows: (refer to Section 3.10 for the deduction)

$$P(o^i_{\tau,u}|\{\boldsymbol{o}^i_1,\ldots,\boldsymbol{o}^i_\tau\}\backslash o^i_{\tau,u}) = \frac{exp\Big\{ \sum_{v\in\{\mathcal{N}-i-u\}} \alpha^\tau_{u,v}\mathbb{I}(o^i_{\tau,u}, o^i_{\tau,v}) + \beta^\tau_u \mathbb{I}(o^i_{\tau,u}, o^i_{\tau-1,u})\Big\}}{1 + exp\Big\{ \sum_{v\in\{\mathcal{N}-i-u\}} \alpha^\tau_{u,v}\mathbb{I}(1, o^i_{\tau,v}) + \beta^\tau_u \mathbb{I}(1, o^i_{\tau-1,u})\Big\}} \tag{3.11}$$

So far, the correlation occurrence models of entities are respectively maintained in different tasks of $\mathscr{B}^{(mdl)}$. Such a probabilistic framework allows to answer a variety of queries by using the conditional probability.

3.7 Cost Analysis

In this part, we provide the theoretical cost analysis. **Computation Cost of Box $\mathscr{B}^{(pre)}$:** Statistics on each time series (i.e., mean, variance) are updated in constant time. The normalized sliding windows are updated in linear time *w.r.t.* the sliding window length. **Computation and Communication Cost of P^2-data-shuffling :** Between Box $\mathscr{B}^{(pre)}$ and $\mathscr{B}^{(cmp)}$, P^2-data-shuffling only uses the first h_p ($h_p \ll h$) entries of each normalized sliding window to derive relevant tasks in linear time w.r.t. h_p, which is independent of h and n.

The communication cost for the sliding window of each time series is decomposed as a product of the number of replicas produced by P^2-data-shuffling and the cost of each replica (i.e., size of a normalized sliding window or a dimension reduced sliding window). As is shown in Subsection 3.4.1, the number of replicas for a sliding window in P^2-data-shuffling is independent of n and h and is bounded by the parallelism of box $\mathscr{B}^{(cmp)}$ ($\ll n$). Overall, the communication cost in P^2-data-shuffling is dramatically decreased compared to the brute-force data communication method.

Note that when the parallelism of $\mathscr{B}^{(cmp)}$ is increased, the amount of data that each task of $\mathscr{B}^{(cmp)}$ deals with is decreased. This is because under the assumption of uniform data distribution, the number of sliding windows each task processes is approximately modelled as $n(\frac{1+\delta}{2})^{h_p}$, which declines as parallelism of $\mathscr{B}^{(cmp)}$ increases ($\frac{1+\delta}{2} < 1$) [46, 139].

Then, we provide the following theorem regarding the 2-way partitioning in P^2-data-shuffling:

Theorem 3.7.1. *The 2-way partitioning over the Euclidean space of normalized sliding windows in P^2-data-shuffling achieves the minimum overall sliding window communication cost among the partitions compared with more fine-grained partitioning (i.e., 3-way, 4-way and so on).*

Proof. For p-way partitioning over the Euclidean space of normalized sliding windows ($p \geq 2$), based on the data shuffling principle among partitions (refer Section 3.4.1) and the uniform data distribution assumption of normalized sliding windows, the amount of sliding windows communicated among the partitions for detecting qualified lead-lag relations is modeled on the basis of Minkowski sum [139] as:

$$Cost = n(\frac{1}{p} + \frac{\delta}{2})^{h_p} \frac{2^h - (2 - \frac{4}{p})^{h_p}}{(\frac{2}{p})^{h_p}} + n(\frac{1}{p} + \delta)^{h_p} \frac{(2 - \frac{4}{p})^{h_p}}{(\frac{2}{p})^{h_p}} \tag{3.12}$$

The first term corresponds to the communicated sliding window replicas in the boundary partitions. Since such partitions only replicate sliding windows to the inner partitions, the communication cost of individual partition is modeled as $n(\frac{1}{p} + \frac{\delta}{2})^{h_p}$ (which is the upper

bound) and the number of such partitions is modeled as $\frac{2^h-(2-\frac{4}{p})^h}{(\frac{2}{p})^{h_p}}$ [46, 139]. On the other hand, the inner partitions replicate sliding windows on both positive and negative directions of relevant dimensions, therefore the number of communicated replicas of such a partition is modeled as $n(\frac{1}{p}+\delta)^{h_p}$.

Then, replacing the number of replicas in the second term with $n(\frac{1}{p}+\frac{\delta}{2})^{h_p}$ leads to the following inequality: $Cost \geq n(\frac{1}{p}+\frac{\delta}{2})^{h_p}(p^{h_p})$. The right side of the inequality has positive first-derivative *w.r.t.* p and thus attains the minimum value at $p = 2$. Therefore, 2-way partitioning in P^2-data-shuffling yields the minimum number of sliding windows replicas. □

Computation Cost of box $\mathscr{B}^{(cmp)}$: Each task of box $\mathscr{B}^{(cmp)}$ performs the lead-lag relation computation by using the δ-hypercube structure to prune computation thereby circumventing brute-force lead-lag relation computation. In Section 5.5, we will experimentally show the pruning power of such methods.

The communication cost between box $\mathscr{B}^{(cmp)}$ and $\mathscr{B}^{(agg)}$ depends on the number of qualified lead-lag relations. Since this number is unknown apriori, we have to omit the analysis of $\mathscr{B}^{(agg)}$ and the computation cost of hash-set based duplicate removal in $\mathscr{B}^{(agg)}$ is negligible as well.

3.8 Experimental Evaluation

In this section, we perform extensive experimental evaluation comparing P2H and $\mathrm{P}^2\mathrm{H}^+$ with baseline approaches. First, we describe the baselines and datasets in Subsection 3.8.1 and Subsection 3.8.2. Then, the efficiency, lead-lag relation parameter sensitivity and pruning power are respectively analysed in Subsection 3.8.3, Subsection 3.8.4 and Subsection 3.8.5.

3.8.1 Baselines

The implementations of P2H, $\mathrm{P}^2\mathrm{H}^+$ and baselines are done using Apache Storm. We choose Storm here, since Storm has lower processing latency compared to other distributed real-time computation system (e.g, S4, Spark Streaming, Samza) due to the one-at-a-time data processing model of Storm [2]. Moreover, Storm provides flexible interfaces which allow to develop advanced customized data processing and shuffling logics. Boxes in P2H, $\mathrm{P}^2\mathrm{H}^+$ and baselines are implemented as bolts in Storm, which have user-specified number of tasks (i.e., parallelism). We implement the P^2-data-shuffling shuffling using a *custom grouping function* provided by Storm. (Our proposed P2H could also be implemented based on other distributed real-time computation platforms.)

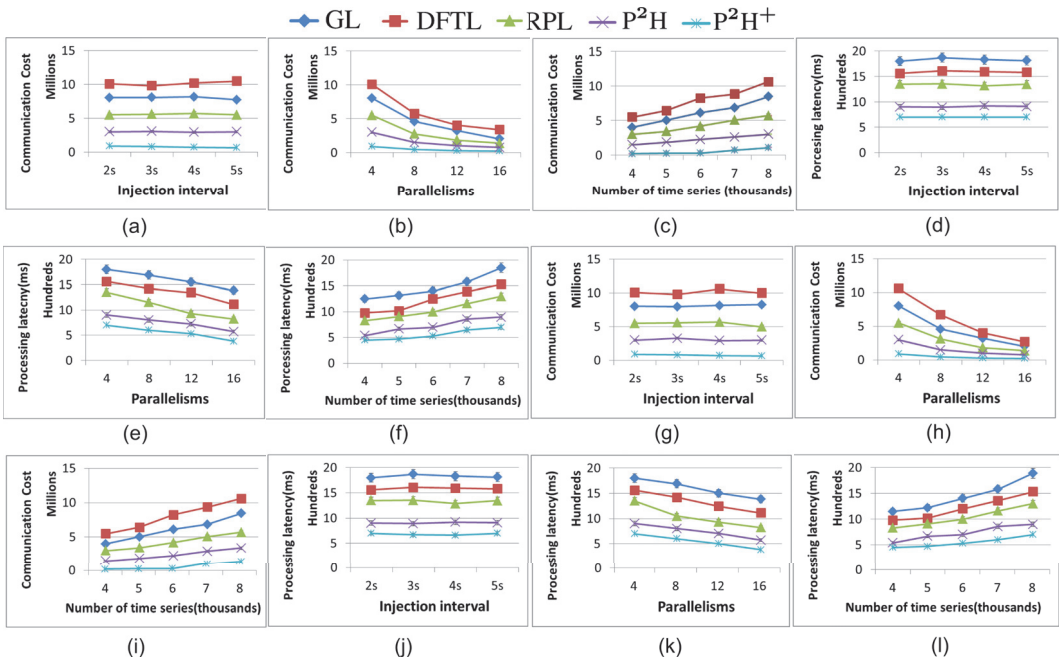


Figure 3.7 – Communication cost and processing latency as a function of injection interval, parallelisms of boxes and the number of input time series.(best viewed in colour) (a)-(f) results on cluster dataset. (g)-(l) results on stock dataset.

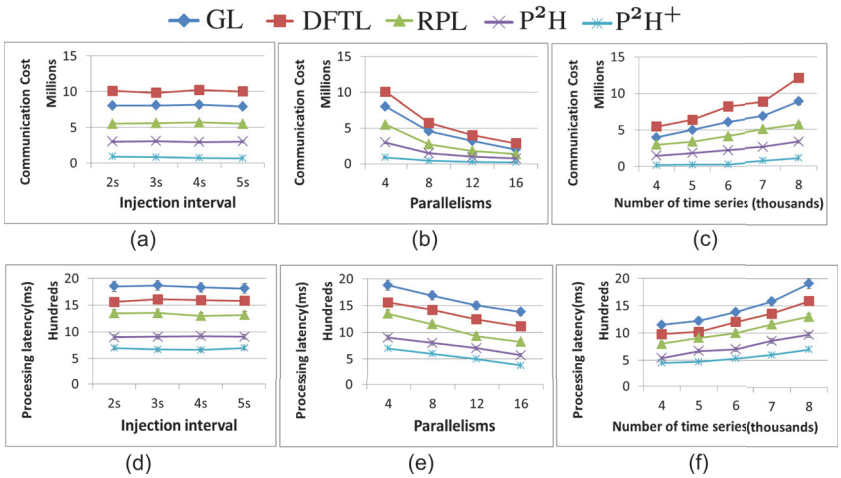


Figure 3.8 – Communication cost and processing latency as a function of injection interval, parallelisms of boxes and the number of input time series on synthetic dataset. (best viewed in colour)

The baseline approaches are as follows:

GL: it consists of three boxes. The first box maintains the sliding windows. Between the first

and second box, group-based data shuffling [107] is leveraged to reduce the communication cost. The second box incrementally computes lead-lag relations over all the pair-wise time series [115], which are aggregated and reported by the third box.

DFTL: This is a DFT (discrete Fourier Transform) based approach proposed in [155]. We applied it to the distributed setting by building a topology consisting of three boxes. The first box maintains the DFT series of sliding windows, which are shuffled according to the grid structure. The second box computes the correlations to find qualified lead-lag relations and then forwards them to the last box, where duplicate removal is performed.

RPL: This is a random projection based approach proposed in [32], which includes three boxes. The first box performs the random projections over sliding windows and shuffles the dimensionality-reduced sliding windows using a dimensionality-grouping method [32]. The second box computes the lead-lag relations over the sub-sliding-windows. Then, the last box aggregates the qualified pairs of sub-sliding-windows, checks the lead-lag relations over the full sliding windows and reports the qualified lead-lag relations.

For the fair comparison, $\mathrm{P}^{\mathbf{2}}\mathrm{H}^{+}$, DFTL and RPL have the dimension-reduced sliding windows of the same length during the experiments.

3.8.2 Datasets and Cluster Details

One synthetic and two real datasets are used for evaluations. The synthetic dataset is generated as follows. Given the required number of time series n, we first generate $\frac{n}{\alpha}$ seed time series. Each seed time series is generated using a random walk model [155]. From each seed time series $\boldsymbol{s}_i$, we produce α dataset as follows:

$$s_{j,t} = \gamma_{j,t} + \beta_j \cdot s_{i,t},$$

where $\gamma_{j,t}$ and β_j are real random numbers between $[0,100]$, and β_j is sampled once for each time series $\boldsymbol{s}_j$, while $\gamma_{j,t}$ is sampled once for *each entry* in time series $\boldsymbol{s}_j$. In our experiments, we set $\alpha = 1000$ and $n = 20000$.

The first real dataset is the *Google Cluster Usage* [108] data. It records extensive activities of 12K cluster nodes from a data center. We extract three parameters: CPU usage, memory usage and disk space usage for each cluster node. The total number of extracted time series is 36K. We denoted this dataset by *cluster-data*. The second dataset consists of intra day stock quotes from the *S&P* 500 index and ETFs (exchange traded funds). We segment the quotes *w.r.t.* each stock into 20 parts, each of which forms a time series. Totally, we obtain 19920 (996×20) time series. We refer to this dataset as *stock-data*. The data is continuously and repeatedly fed into

Storm according to a specified injection interval (we will define later) during the experiments.

Cluster Setup: The experiments are performed using a cluster consisting of 1 master and 8 slaves. The master node has 64GB RAM, 4TB disk space and 12 x 2.30 GHz cores. Each slave node has 6 x 2.30 GHz cores, 32GB RAM and 6TB disk space. All the nodes are connected via 1GB Ethernet. All the performance metrics are computed by averaging every 20 seconds for 10 times, after the cluster reaches a stable state.

3.8.3 Analysing Efficiency

In this part, we evaluate the efficiency of the approaches by measuring the following three metrics communication cost, processing latency and peak capacity (as we will define later) as functions of injection interval, number of input time series (i.e., n) and parallelisms of boxes. We choose these three parameters here, since they describe the context of a DisCoM problem. **Injection interval** Δ defines the time interval between the data points of the input time series. Parallelisms describe the available computing resource in the cluster. For the fair evaluation, all the approaches have the same parallelism.

Communication cost is measured by the amount of basic data units communicated between the front two boxes of each approach divided by the parallelism of the second box. Here a data unit is a basic data type, which could be float, integer, etc. Intuitively, it reflects the amount of data on average each task of the second box processes. As the communication cost between the last two boxes depends on the qualified lead-lag relations and is identical for all approaches, we omit it here. **Processing latency** is the average processing time for each task of boxes considered together. **Peak capacity** is the maximum number of time series that an approach can simultaneously process without causing bottlenecks in the system [2, 155]. A *bottleneck* is caused when sliding windows at the current time instant have to wait (in memory) for the sliding windows at a previous time instant to finish processing [2, 147, 155]. Bottlenecks caused by any tasks are detected and reported by the Storm cluster UI [2].

These three metrics are measured by varying one of injection interval, number of input time series and parallelisms within a pre-defined range, while setting the others to their basic set-up values, namely Δ = 2sec, n = 8000 and p = 4. (the parameters of lead-lag relation, relation threshold ϵ, sliding window size h and maximum lag ℓ are set as $\epsilon = 0.7, h = 500, \ell = 300$ in this group of experiments. We will analysis their effects later.). As we will see below, our distributed approach is able to process much more high-speed time series compared with previous work [155] on a standalone machine, which processes 4000 time series with 15 seconds time interval (i.e., basic windows in [155]).

Communication Cost and Processing Latency: In Figure 3.7(*a*) the communication costs of

all approaches are relatively stable *w.r.t.* injection interval Δ. On average, P2H and P^{2}H$^+$ respectively exhibits 70% and 90% less communication cost than DFTL. Due to the adjacent-cell data shuffling base on the grid [155], DFTL yields the most communication cost. In Figure 3.7(*b*) the increase of parallelism enables to have more tasks and therefore the sliding windows distributed to each task is decreased. At the highest level of number of input time series in Figure 3.7(*c*), P^2-data-shuffling enable P2H to save up to 3× less communication cost than DFTL. Because of the similar patterns shown in Figure 3.7(*h-i*) on stock dataset, we omit the description here.

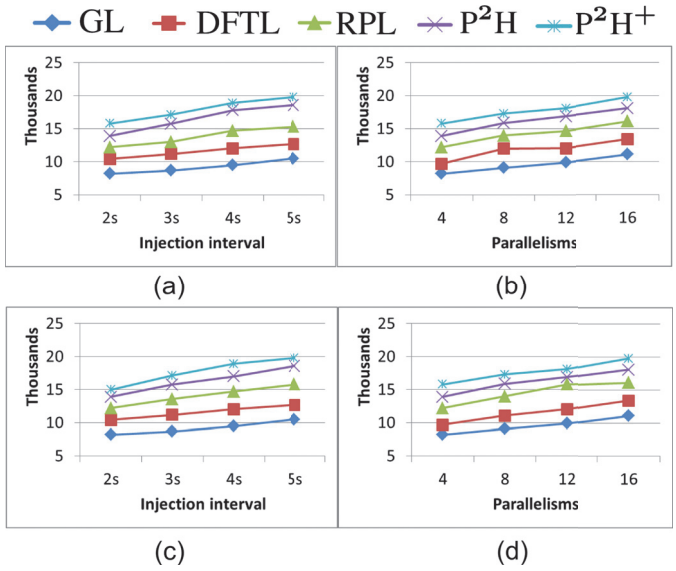


Figure 3.9 – Peak capacity as a function of time series injection interval and parallelisms on real datasets. (best viewed in colour.) (a)-(b) results on cluster dataset. (c)-(d) results on stock dataset.

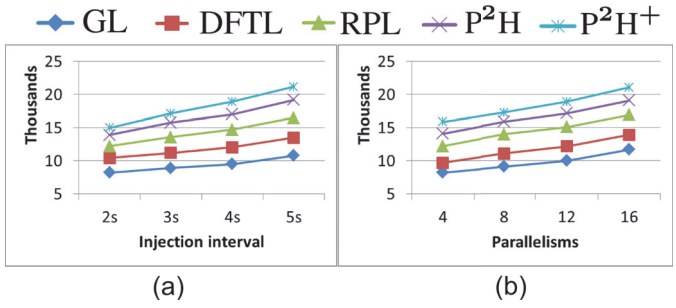


Figure 3.10 – Peak capacity as a function of time series injection interval and parallelisms on synthetic dataset. (best viewed in colour)

As for the processing latency, in Figure 3.7(*d*), P2H approach has nearly 1× lower latency as compared to GL at the maximum injection interval. Because of the brute-force lead-lag

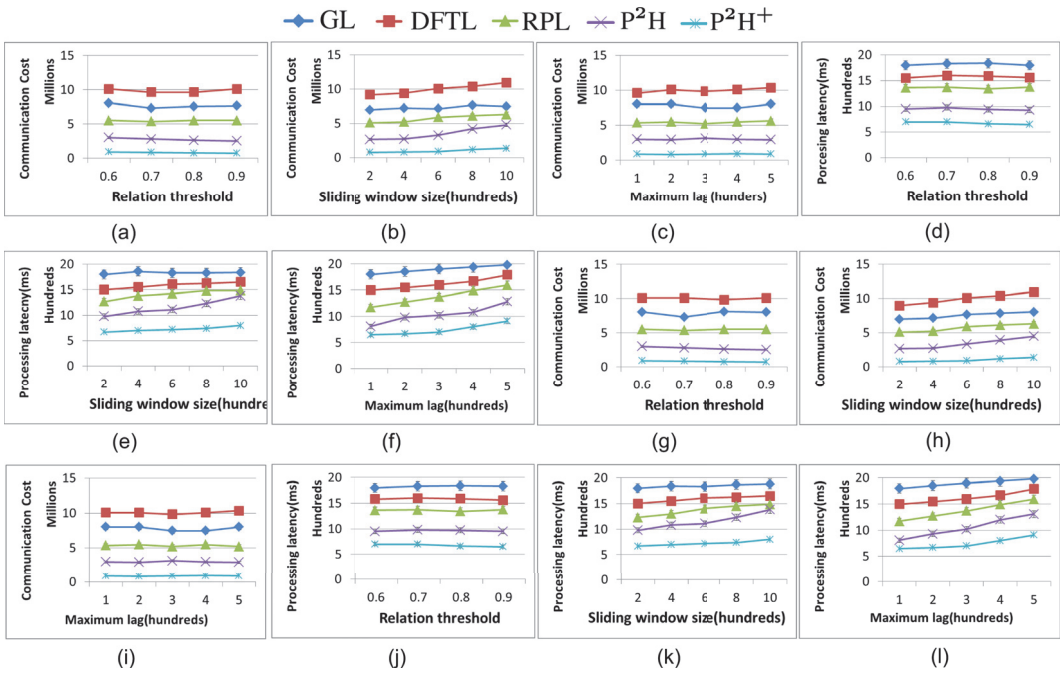


Figure 3.11 – Sensitivity analysis of communication cost and processing latency *w.r.t.* lead-lag relation threshold, sliding window size and maximum lag. (best viewed in colour). (a)-(f) results on cluster dataset. (g)-(l) results on stock dataset.

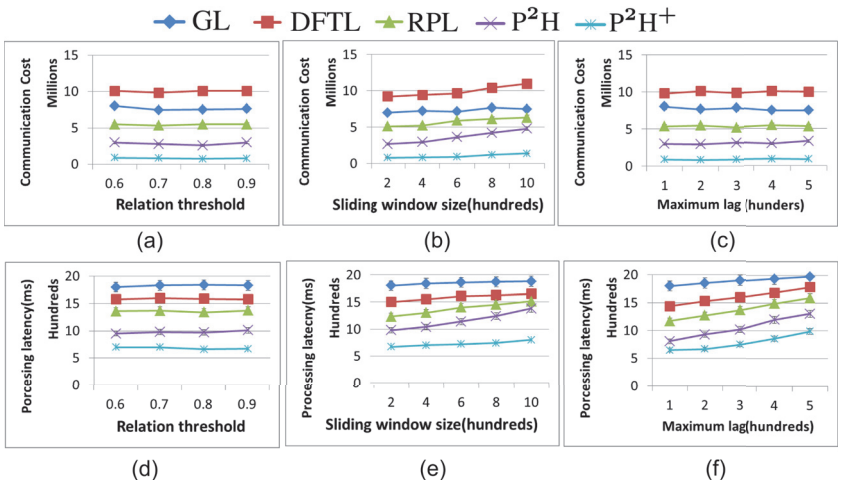


Figure 3.12 – Sensitivity analysis of communication cost and processing latency *w.r.t.* lead-lag relation threshold, sliding window size and maximum lag on synthetic dataset. (best viewed in colour)

relation computation over pair-wise time series, GL has the maximum processing latency. In Figure 3.7(*e*), as the increase of parallelisms lowers down the average amount of data each

task processes, the processing latencies of all approaches decrease. When the number of time series increases, the approaches with computation pruning (i.e., P2H, P^2H^+, DFTL and RPL) show lower processing latencies than GL. Specifically, P2H and P^2H^+ respectively have 1× and 2× lower processing latency than GL at the maximum number of time series. Refer Figure 3.7(*h*-*i*) for the results on stock dataset.

Peak Capacity: In this part, we attempt to answer the following question: how many time series at the maximum can each approach handle in real time without causing bottlenecks given a certain context? Therefore, this set of experiments demonstrates how peak capacity of each approach varies with the injection interval and parallelisms in Figure 3.9.

In Figure 3.9(*a*) and (*b*), the peak capacities of all approaches increase as the injection interval and parallelism increase. This is because the increases of these two parameters lead to more computing resources and more available processing time interval. At the highest level of parallelism and injection interval, P2H respectively exhibits 100% and 90% more peak capacities than GL. As is shown in Figure 3.7, given the same number of time series P2H and P^2H^+ are able to process it with less latency and therefore compared with baselines they can process more time series under the same context.

3.8.4 Analysing Sensitivity of Lead-lag Relation Parameters

In real applications, given a certain number of time series input to the distributed real-time computation engine with specific parallelisms, users often tune the parameters of lead-lag relations (i.e., relation threshold ϵ, maximum lag ℓ and sliding window size h) to explore application-specific lead-lag relations. For instance, lead-lag relations above 0.6 could be considered as strong [117, 155]. Several hundreds are enough for h so as to obtain significant lead-lag relations [22, 107]. ℓ depends on certain applications.

Therefore, in this set of experiments we will study how sensitive the communication and computation cost are to the changes in the lead-lag relation parameters by varying one of relation threshold ϵ, maximum lag ℓ and sliding window size h and keeping the others as their basic set-up values under a certain context (i.e., injection interval, number of time series and parallelisms are set as the basic set-up as is shown in Subsection 3.8.3).

In Figure 3.11(*a*) and (*c*) the communication costs of all approaches are relatively stable *w.r.t.* query threshold ϵ and maximum lag ℓ. P2H and P^2H^+ present a little decreasing trend of communication cost *w.r.t.* ϵ, since the amount of sliding windows shuffled among partitions declines as ϵ increases, as is shown in Subsection 3.4.1. For the sliding window size h in Figure 3.11(*b*), P^2H^+ has nearly 5× lower cost as compared to DFTL at the highest level of sliding window size. The communication cost of P2H presents an increasing trend due to the

lack of dimensionality-reduction.

As for the processing latency, in Figure 3.11(*d*) about lead-lag relation threshold ϵ, average improvement in the latency of P2H *w.r.t.* GL is approximately 1×. When sliding window length increases in Figure 3.11(*e*), the processing latencies of DFTL, RPL, P2H and P^2H^+ increase a little. This is because dimension-reduction methods in these approaches need to retain more entries when h increases [32]. Due to the incremental lead-lag relation computation, GL has a relatively stable, but higher processing latency. In Figure 3.11(*f*), all the approaches have increasing processing latencies as maximum lag ℓ increases. Specifically, the computation pruning allows P2H to have about 50% lower latency as compared to GL at the maximum lag. We omit the detailed description about the results on stock dataset, as is shown in Figure 3.11(*h*-*i*).

3.8.5 Analysing Pruning Power

As P2H, DFTL and RPL approaches all contain computation pruning components, we evaluates the pruning power (the higher, the better) of them in this set of experiments. Since P^2H^+ and P2H share the same computation pruning mechanism, only P2H is assessed here. We also omit GL here, since it always computes pair-wise lead-lag relations. **Pruning power** is defined as the ratio of the number of leader and follower sliding window pairs that are pruned (without having to compute values of lead-lag relations) to the total number of lead-lag relations (i.e., the total number of time-series pairs by the maximum lag). Higher values of pruning power are considered better.

We consider the pruning power under a certain context set-up (i.e. injection interval, number of time series and parallelisms are set as basic values.) From the results in Figure 3.11, we observe that sliding window size affects the processing latency and therefore the pruning power could vary as it. On the other hand, ϵ directly affects the size of δ-hypercube and thus we also to reveal its effect on the pruning power. The pruning power of an approach is relatively stable across different maximum lags and we set it as the basic value.

In Table 3.1, the upper, middle and lower values in each cell respectively correspond to DFTL, RPL and P2H. In P2H, based on the relation between ϵ and δ (refer Subsection 3.3.3), higher values of ϵ lead to shrinking δ-hypercubes and therefore more pairs of sliding windows are pruned. On the other hand, higher h leads to more uniform value distribution of the entries in the normalized sliding windows [139] thereby decreasing the pruning power. We observe in the table at the maximum ϵ and minimum h, P2H achieves the maximum pruning power 0.647, which is around 40% better than DFTL. Likewise, the experiments on the stock dataset demonstrate similar results in Table 3.3.

Table 3.1 – Pruning powers of DFTL, RPL and P2H *w.r.t.* lead-lag relation threshold ϵ and sliding-window length h for cluster dataset.

ϵ \ h	200	400	600	800	1000
0.7: DFTL	0.421	0.431	0.423	0.354	0.327
RPL	0.525	0.514	0.509	0.442	0.435
P2H	0.545	0.544	0.512	0.442	0.415
0.8: DFTL	0.447	0.445	0.452	0.426	0.397
RPL	0.535	0.544	0.511	0.448	0.435
P2H	0.632	0.587	0.548	0.559	0.442
0.9: DFTL	0.461	0.442	0.433	0.372	0.342
RPL	0.529	0.549	0.439	0.442	0.445
P2H	**0.646**	0.548	0.528	0.518	0.467

Table 3.2 – Pruning powers of DFTL, RPL and P2H *w.r.t.* lead-lag relation threshold ϵ and sliding-window length h for stock dataset.

ϵ \ h	200	400	600	800	1000
0.7: DFTL	0.403	0.331	0.323	0.354	0.327
RPL	0.542	0.514	0.439	0.442	0.415
P2H	0.602	0.543	0.512	0.502	0.411
0.8: DFTL	0.452	0.445	0.352	0.326	0.315
RPL	0.563	0.542	0.461	0.448	0.425
P2H	0.623	0.587	0.558	0.509	0.482
0.9: DFTL	0.457	0.442	0.363	0.374	0.338
RPL	0.593	0.549	0.439	0.422	0.415
P2H	**0.676**	0.648	0.528	0.454	0.457

3.8.6 Precision Measurement

In this part, we evaluate the precision and recall of P2H and $\mathbf{P^2H^+}$ under different relation thresholds [155]. As baseline approaches perform trivial lead-lag relation computation, they are omitted in this group of experiments. In Table 3.4 and Table 3.5, each cell is of the form (precision/recall). From the results, both P2H and $\mathbf{P^2H^+}$ can achieve high precision and recall. Due to the dimensionality-reduction, the precision of $\mathbf{P^2H^+}$ is slightly lower than P2H. As ϵ increases, more lead-lag relation computation is needed and therefore the precision increases.

3.9 Conclusion

In this paper, we investigated the DisCoM problem. by proposing P2H framework. It harnesses $\mathbf{P^2}$-data-shuffling and δ-hypercube to optimize communication and computation efficiency. P2H framework and baseline approaches are implemented based on Apache Storm. We

Table 3.3 – Pruning powers of DFTL, RPL and P2H *w.r.t.* lead-lag relation threshold ϵ and sliding-window length h for synthetic dataset.

ϵ \ h	200	400	600	800	1000
0.7: DFTL	0.433	0.431	0.353	0.354	0.333
RPL	0.522	0.504	0.433	0.423	0.405
P2H	0.613	0.554	0.521	0.443	0.413
0.8: DFTL	0.453	0.442	0.343	0.346	0.302
RPL	0.575	0.554	0.451	0.423	0.402
P2H	0.635	0.567	0.543	0.502	0.465
0.9: DFTL	0.448	0.443	0.354	0.365	0.323
RPL	0.585	0.533	0.435	0.413	0.414
P2H	**0.678**	0.645	0.565	0.535	0.419

Table 3.4 – Precision and recall of P2H and P^2H^+ for cluster dataset.

ϵ	0.6	0.7	0.8	0.9
P2H	0.914/0.902	0.923/0.937	0.945/0.991	0.985/0.992
P^2H^+	0.822/0.912	0.891/0.942	0.902/0.948	0.934/0.982

demonstrate the effectiveness and efficiency of our approach through extensive experiments.

3.10 Appendixes

3.10.1 Alternative Correlation Measures

Statistical Correlation: We define a generic correlation function, based on which the definitions of Pearson and Spearman correlations are given later. For two vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ ($\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^h$, h is the sliding window size for time series), the generic correlation function is defined as:

$$corre(\boldsymbol{x}_1, \boldsymbol{x}_2) = \frac{(\boldsymbol{x}_1 - \mu(\boldsymbol{x}_1)\mathbb{1}) \cdot (\boldsymbol{x}_2 - \mu(\boldsymbol{x}_2)\mathbb{1})}{(h-1)\sigma(\boldsymbol{x}_1)\sigma(\boldsymbol{x}_2)} \tag{3.13}$$

where $\mathbb{1}$ is all one vector ($\mathbb{1} \in \mathbb{R}^h$), $\sigma(\boldsymbol{x})$ and $\mu(\boldsymbol{x})$ are the sample standard deviation and mean of the elements in $\boldsymbol{x}$, respectively.

Then, the Pearson correlation coefficient $\rho^p_{i,j}$ between sliding windows $\boldsymbol{s}^t_i$ and $\boldsymbol{s}^t_j$ of time series i and j, which evaluates the linear relationship between two variables, is defined as follows:

$$\rho^p_{i,j} = corre(\boldsymbol{s}^t_i, \boldsymbol{s}^t_j) \tag{3.14}$$

Table 3.5 – Precision and recall of P2H and P^2H^+ for stock dataset.

ϵ	0.6	0.7	0.8	0.9
P2H	0.902/0.973	0.919/0.981	0.931/0.993	0.982/0.993
P^2H^+	0.843/0.903	0.882/0.936	0.901/0.939	0.944/0.987

Table 3.6 – Precision and recall of P2H and P^2H^+ for synthetic dataset.

ϵ	0.6	0.7	0.8	0.9
P2H	0.922/0.943	0.902/0.961	0.933/0.989	0.964/0.989
P^2H^+	0.854/0.913	0.884/0.932	0.912/0.942	0.944/0.979

Additionally, the non-parametric Spearman's rank-order correlation measures the strength of monotonic relationship between two ranked variables. Compared with Pearson correlation, Spearman's rank correlation coefficient is more robust to outliers. We define rank vector $\boldsymbol{r}_i^t$ of sliding window $\boldsymbol{s}_i^t$ is a vector of size h, the entries of which are the ranks of the corresponding entries in the original sliding window $\boldsymbol{s}_i^t$. For instance, given $\boldsymbol{s}_i^t = (1.3, 4.6, 3.7)$, its $\boldsymbol{r}_i^t$ is $(1,3,2)$, since sorted elements in $\boldsymbol{s}_i^t$ are $(1.3, 3.7, 4.6)$. Then, Spearman correlation $\rho_{i,j}^s$ for sliding windows $\boldsymbol{s}_i^t$ and $\boldsymbol{s}_j^t$ of time series i and j is defined on $\boldsymbol{r}_i^t$ and $\boldsymbol{r}_j^t$ as:

$$\rho_{i,j}^s = corre(\boldsymbol{r}_i^t, \boldsymbol{r}_j^t) \tag{3.15}$$

Now, we define a normalization function over a vector $\boldsymbol{x}$ ($\boldsymbol{x} \in \mathbb{R}^h$) as [155]:

$$\hat{\boldsymbol{x}} = norm(\boldsymbol{x}) = \frac{(\boldsymbol{x} - \mu(\boldsymbol{x})\mathbb{1})}{\sqrt{(h-1)}\sigma(\boldsymbol{x})} \tag{3.16}$$

where $\mathbb{1}_h$ is an all-one vector of size h. The vector $\hat{\boldsymbol{x}}$ is of unit length, namely $\hat{\boldsymbol{x}} \cdot \hat{\boldsymbol{x}} = 1$. Then the normalized sliding windows for Pearson and Spearman correlation are respectively defined as

$$\hat{\boldsymbol{s}}_i^t = norm(\boldsymbol{s}_i^t) \text{ and } \hat{\boldsymbol{r}}_i^t = norm(\boldsymbol{r}_i^t) \tag{3.17}$$

The correlation can also be written using the normalized sliding windows as follows: $\rho_{i,j}^p = \hat{\boldsymbol{s}}_i^t \cdot \hat{\boldsymbol{s}}_j^t$ and $\rho_{i,j}^s = \hat{\boldsymbol{r}}_i^t \cdot \hat{\boldsymbol{r}}_j^t$.

Note that the statistical significance test can be performed in box $\mathscr{B}^{(cmp)}$ over the detected lead-lag relations and $\mathscr{B}^{(cmp)}$ only emits the significant ones.

Cosine correlation: The cosine correlation between time series $\boldsymbol{s}_i^t$ and $\boldsymbol{s}_j^t$ is widely used in information retrieval and text mining. We adopt the similar normalization process as Pearson

correlation to $\boldsymbol{s_i^t}$ and $\boldsymbol{s_j^t}$ and obtain:

$$\hat{\boldsymbol{s}}^{\boldsymbol{t}}_{\boldsymbol{i(or\ j)}} = \frac{\boldsymbol{s}^{\boldsymbol{t}}_{\boldsymbol{i(or\ j)}}}{\|\hat{\boldsymbol{s}}^{\boldsymbol{t}}_{\boldsymbol{i(or\ j)}}\|_2} \tag{3.18}$$

such that $cos(\theta(\boldsymbol{s_i^t}, \boldsymbol{s_j^t})) = (\hat{\boldsymbol{s}}_{\boldsymbol{i}}^{\boldsymbol{t}})^T \cdot (\hat{\boldsymbol{s}}_{\boldsymbol{j}}^{\boldsymbol{t}})$ and $(\hat{\boldsymbol{s}}^{\boldsymbol{t}}_{\boldsymbol{i(or\ j)}})^T \cdot \hat{\boldsymbol{s}}^{\boldsymbol{t}}_{\boldsymbol{i(or\ j)}} = 1$. Therefore,

$$\mathscr{D}^2(\hat{\boldsymbol{s}}_{\boldsymbol{i}}^{\boldsymbol{t}}, \hat{\boldsymbol{s}}_{\boldsymbol{j}}^{\boldsymbol{t}}) = 2 - 2 \cdot cos(\theta(\boldsymbol{s_i^t}, \boldsymbol{s_j^t})) \tag{3.19}$$

Extended Jaccard Coefficient: We firs do the following deduction:

$$\begin{aligned} J(\boldsymbol{s_i^t}, \boldsymbol{s_j^t}) &= \frac{(\boldsymbol{s_i^t})^T \cdot \boldsymbol{s_j^t}}{\|\boldsymbol{s_i^t}\|_2^2 + \|\boldsymbol{s_j^t}\|_2^2 - (\boldsymbol{s_i^t})^T \cdot \boldsymbol{s_j^t}} \\ &\leq \frac{(\boldsymbol{s_i^t})^T \cdot \boldsymbol{s_j^t}}{\|\boldsymbol{s_i^t}\|_2^2 + \|\boldsymbol{s_j^t}\|_2^2 - \|\boldsymbol{s_i^t}\| \cdot \|\boldsymbol{s_j^t}\|} \leq \frac{(\boldsymbol{s_i^t})^T \cdot \boldsymbol{s_j^t}}{\|\boldsymbol{s_i^t}\| \cdot \|\boldsymbol{s_j^t}\|} \end{aligned} \tag{3.20}$$

since $\|\boldsymbol{s_i^t}\|_2^2 + \|\boldsymbol{s_j^t}\|_2^2 - \|\boldsymbol{s_i^t}\| \cdot \|\boldsymbol{s_j^t}\| \geq \|\boldsymbol{s_i^t}\| \cdot \|\boldsymbol{s_j^t}\| \geq 0$.

Applying the normalization process Eq. (3.18) to the bound in Eq. (3.20), we transform the correlation query $J(\boldsymbol{s_i^t}, \boldsymbol{s_j^t}) \geq \epsilon$ to $\epsilon \leq J(\boldsymbol{s_i^t}, \boldsymbol{s_j^t}) \leq \frac{2-\mathscr{D}^2(\hat{\boldsymbol{s}}_{\boldsymbol{i}}^{\boldsymbol{t}}, \hat{\boldsymbol{s}}_{\boldsymbol{j}}^{\boldsymbol{t}})}{2}$. We can see that Euclidean distance between normalized sliding windows provides upper bound for Extended Jaccard Coefficient. Based on this this property, we can use proposed P2H plus a posting filtering to exactly process the query on Extended Jaccard Coefficient.

Euclidean distance: in processing queries on Euclidean distance, P2H works directly on the original sliding windows without sliding window normalization process. The query threshold is directly used for sliding window grouping and computation pruning.

3.10.2 Proof of lemmas and theorems

THEOREM 4.2 *For a normalized sliding window $\hat{\boldsymbol{s}}_i^t$ in $\boldsymbol{p}_i^t$, the target-partitions in R_i^t constructed by above dimension subset $\mathscr{H}_i$ selection and the permutation process over $\mathscr{H}_i$ contain the leaders of $\hat{\boldsymbol{s}}_i^t$, while the partitions not present in $\mathscr{R}_i^t$ must not have leaders of $\hat{\boldsymbol{s}}_i^t$.*

Proof. The proof relies on Lemma 3.4.1. Based on Lemma 3.4.1, partitions in $\mathscr{R}_i^t$ derived from $\mathscr{H}_i$ have leader sliding windows of $\hat{\boldsymbol{s}}_i^t$. Through the proof of Lemma 3.4.1, the sliding windows in the opposite partitions on the dimensions not present in $\mathscr{H}_i$ have the distance to $\hat{\boldsymbol{s}}_i^t$ larger than δ, therefore the partitions not present in $\mathscr{R}_i^t$ have no leader sliding windows for $\hat{\boldsymbol{s}}_i^t$. □

THEOREM 4.3 *[Correctness and Completeness of P^2] Through P^2-data-shuffling, each task of box $\mathscr{B}^{(cmp)}$ receives the normalized sliding windows located in the partition corresponding to this task and the sliding windows from other partitions that are followers of this task's local sliding windows. Therefore, the leaders (or followers) of each time series can be found by the tasks of $\mathscr{B}^{(cmp)}$.*

Proof. As the sliding windows in a partition are shuffled to the same task of box $\mathscr{B}^{(cmp)}$, the task of $\mathscr{B}^{(cmp)}$ responsible for this partition is able to find the local qualified lead-lag relations. On the other hand, for some sliding windows whose leaders are in the partition other than where they locate, as is shown in Lemma 3.5 and Theorem 3.4.1, P^2-data-shuffling is able to send such sliding windows to the partitions having their leaders. Therefore, all the leaders of each sliding window can be found by tasks of $\mathscr{B}^{(cmp)}$. □

LEMMA 5.1 *For a sliding window $\hat{\mathbf{s}}_i^t$ in δ-hypercube $\mathbf{c}_i$, the sliding windows within δ distance to $\hat{\mathbf{s}}_i^t$ are only contained in the δ-hypercubes from $\mathscr{N}(\mathbf{c}_i)$. All the δ-hypercubes out of $\mathscr{N}(\mathbf{c}_i)$ must not contain the sliding windows close to $\hat{\mathbf{s}}_i^t$ within δ.*

Proof. For the hypercubes not present in $\mathscr{N}(\mathbf{c}_i^t)$, on certain dimension(s) the distances between $\hat{\mathbf{s}}_i^t$ and such hypercubes are δ at minimum. This means the distance between $\hat{\mathbf{s}}_i^t$ and the normalized sliding windows in such hypercubes are above δ. □

LEMMA 5.2 *The lead-lag relation computation should be performed over a pair of sliding windows respectively from a δ-hypercube in $\mathscr{C}^t$ and an inner δ-hypercube prior to t.*

Proof. The lead-lag relation computation is only performed over a pair of sliding windows respectively from a δ-hypercube in $\mathscr{C}^t$ and an inner δ-hypercube prior to t. This is because based on Theorem 4.3 the followers of sliding windows in outer δ-hypercubes prior to t in this task are shuffled to the tasks of box $\mathscr{B}^{(cmp)}$ where these outer δ-hypercubes are inner ones and they can be detected there. □

LEMMA 5.3 *For a sliding window $\hat{\mathbf{s}}_i^t$ from δ-hypercube $\mathbf{c}_i^t$, its minimum distance to a δ-hypercube $\mathbf{c}_j^{t_1}$ is defined as:* $D_m(\hat{\mathbf{s}}_i^t, \mathbf{c}_j^{t_1}) = \sqrt{\sum_{k=0}^{h-1} \begin{cases} (\hat{s}_{i,t-k}^t - c_{i,t-k}^t * \delta)^2 & : c_{i,t-k}^t < c_{j,t-k}^t \\ 0 & : c_{i,t-k}^t = c_{j,t-k}^t \\ (\hat{s}_{i,t-k}^t - c_{i,t-k}^t * \delta + \delta)^2 & : c_{i,t-k}^t > c_{j,t-k}^t \end{cases}}$.

If $D_m(\hat{\mathbf{s}}_i^t, \mathbf{c}_j^{t_1}) > \delta$, there is no leader of $\hat{\mathbf{s}}_i^t$ in δ-hypercube $\mathbf{c}_j^{t_1}$ and the lead-lag relation computation for $\hat{\mathbf{s}}_i^t$ w.r.t. *$\mathbf{c}_j^{t_1}$ is pruned.*

Proof. The minimum distance between a sliding window $\hat{\boldsymbol{s}}_i^t$ and a δ-hypercube $\boldsymbol{c}_j^{t_1}$ is evaluated by comparing δ-hypercube $\boldsymbol{c}_i^t$ where this sliding window locates and $\boldsymbol{c}_j^{t_1}$. The computation of $D_m(\hat{\boldsymbol{s}}_i^t, \boldsymbol{c}_j^{t_1})$ can be illustrated by Figure 3.13. The grey δ-hypercube represents δ-hypercube $\boldsymbol{c}_i^t$ where $\hat{\boldsymbol{s}}_i^t$ (blue point) locates. The blue δ-hypercube represents $\boldsymbol{c}_j^{t_1}$.

In Figure 3.13(a), $\boldsymbol{c}_i^t$ and $\boldsymbol{c}_j^{t_1}$ have the same coordinate on dimension $t-1$, so the point in $\boldsymbol{c}_j^{t_1}$ with the minimum the distance to $\hat{\boldsymbol{s}}_i^t$ also has the same coordinate on dimension $t-1$ as $\hat{\boldsymbol{s}}_i^t$ and the distance at this dimension is zero. In Figure 3.13(b), $\boldsymbol{c}_i^t$ and $\boldsymbol{c}_j^{t_1}$ have no same coordinates and all dimensions are used to compute $D_m(\hat{\boldsymbol{s}}_i^t, \boldsymbol{c}_j^{t_1})$. If $D_m(\hat{\boldsymbol{s}}_i^t, \boldsymbol{c}_j^{t_1}) > \delta$, no sliding windows in

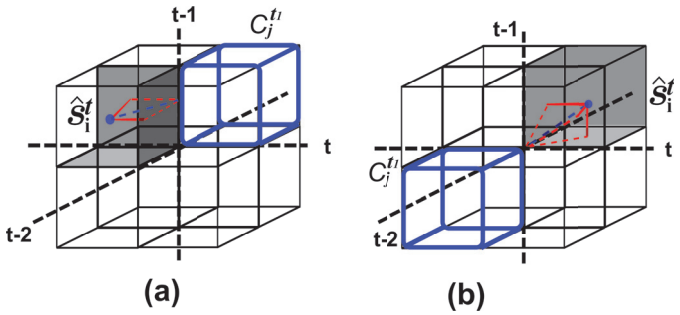


Figure 3.13 – Distance between a sliding window and a δ-hypercube

$\boldsymbol{c}_j^{t_1}$ are close to $\hat{\boldsymbol{s}}_i^t$ within δ. □

LEMMA 5.4 *Define functions $max(\boldsymbol{x})$ and $min(\boldsymbol{x})$ ($\boldsymbol{x} \in \mathbb{R}^h$), which respectively return the maximum and minimum element values in $\boldsymbol{x}$. For a sliding windows $\hat{\boldsymbol{s}}_i^t$ in $\boldsymbol{c}_i^t$, if $\hat{\boldsymbol{s}}_j^{t_1}$ in $\boldsymbol{c}_j^{t_1}$ addresses $\|\hat{\boldsymbol{s}}_i^t - min(\hat{\boldsymbol{s}}_i^t)\cdot\mathbb{1}\|_1 \cdot \sqrt{\frac{(max(\hat{\boldsymbol{s}}_j^{t_1}) - min(\hat{\boldsymbol{s}}_j^{t_1}))^2 \cdot ln(\frac{1}{\theta})}{2h}} \geq \epsilon$, then the true lead-lag relation between $\hat{\boldsymbol{s}}_i^t$ and $\hat{\boldsymbol{s}}_j^{t_1}$ is above ϵ with probability above $1-\theta$.*

Proof. Define $\boldsymbol{r} = \frac{\hat{\boldsymbol{s}}_i^t - min(\hat{\boldsymbol{s}}_i^t)\cdot\mathbb{1}}{\|\hat{\boldsymbol{s}}_i^t - min(\hat{\boldsymbol{s}}_i^t)\cdot\mathbb{1}\|_1}$ ($\boldsymbol{r} \in \mathbb{R}^h$), such that $\sum_{k=1}^{h} r_k = 1$, which can be thought of as a discrete probability mass function. Based on $rel(\hat{\boldsymbol{s}}_i^t, \hat{\boldsymbol{s}}_j^{t_1}) = \hat{\boldsymbol{s}}_i^t \cdot \hat{\boldsymbol{s}}_j^{t_1}$ and $\mathbb{1}\cdot\hat{\boldsymbol{s}}_j^{t_1} = 0$, $\frac{\hat{\boldsymbol{s}}_i^t \cdot \hat{\boldsymbol{s}}_j^{t_1}}{\|\hat{\boldsymbol{s}}_i^t - min(\hat{\boldsymbol{s}}_i^t)\cdot\mathbb{1}\|_1} = \boldsymbol{r}\cdot\hat{\boldsymbol{s}}_j^{t_1} = \sum_{k=1}^{h} r_k \hat{s}_j^{t_1}$ The rightmost formula of this equation can be considered as the expectation of random variable $\hat{s}_j^{t_1}$. Then we apply the Hoeffding bound to derive the probabilistic bound for correlation $\hat{\boldsymbol{s}}_i^t \cdot \hat{\boldsymbol{s}}_j^{t_1}$.

Based on Hoeffding bound [97], $\Pr(\mathbb{1}\cdot\hat{\boldsymbol{s}}_j^{t_1} - \mu \geq a) \leq e^{\frac{-2ha^2}{R^2}}$ where $R = max(\hat{\boldsymbol{s}}_j^{t_1}) - min(\hat{\boldsymbol{s}}_j^{t_1})$ and $\mu = \boldsymbol{r}\cdot\hat{\boldsymbol{s}}_j^{t_1} = \frac{\hat{\boldsymbol{s}}_i^t \cdot \hat{\boldsymbol{s}}_j^{t_1}}{\|\hat{\boldsymbol{s}}_i^t - min(\hat{\boldsymbol{s}}_i^t)\cdot\mathbb{1}\|_1}$

Then, assume $\theta = e^{\frac{-2ha^2}{R^2}}$, we can derive $a = -\sqrt{\frac{R^2 log(\frac{1}{\theta})}{2h}}$. Therefore, the probability of $\frac{\hat{\boldsymbol{s}}_i^t \cdot \hat{\boldsymbol{s}}_j^{t_1}}{\|\hat{\boldsymbol{s}}_i^t - min(\hat{\boldsymbol{s}}_i^t)\cdot \mathbb{1}\|_1} \leq a$ is less than or equal to θ and the lemma is proved. □

Induction of the conditional probability in Gibbs sampling:

$$P(o_{\tau,u}^i | \{\boldsymbol{o}_1^i, \ldots, \boldsymbol{o}_\tau^i\} \backslash o_{\tau,u}^i) = \frac{exp\Big\{\sum\limits_{v \in \mathcal{N}-i} \alpha_{u,v}^\tau \mathbb{I}(o_{\tau,u}^i, o_{\tau,v}^i) + \beta_u^\tau \mathbb{I}(o_{\tau,u}^i, o_{\tau-1,u}^i)\Big\}}{1 + exp\Big\{\sum\limits_{v \in \mathcal{N}-i} \alpha_{u,v}^\tau \mathbb{I}(1, o_{\tau,v}^i) + \beta_u^\tau \mathbb{I}(1, o_{\tau-1,u}^i)\Big\}}$$

Proof.

$$P(o_{\tau,u}^i | \{\boldsymbol{o}_1^i, \ldots, \boldsymbol{o}_\tau^i\} \backslash o_{\tau,u}^i) = \frac{P(\boldsymbol{o}_1^i, \ldots, \boldsymbol{o}_\tau^i)}{P(o_{\tau,u}^i = 1, \{\boldsymbol{o}_1^i, \ldots, \boldsymbol{o}_\tau^i\} \backslash o_{\tau,u}^i) + P(o_{\tau,u}^i = 0, \{\boldsymbol{o}_1^i, \ldots, \boldsymbol{o}_\tau^i\} \backslash o_{\tau,u}^i)}$$

,where

$$P(\boldsymbol{o}_1^i, \ldots, \boldsymbol{o}_\tau^i) = \frac{1}{Z_1} exp\Big\{\sum_{u,v \in \mathcal{N}-i} \alpha_{u,v}^1 \mathbb{I}(o_{1,u}^i, o_{1,v}^i) + \sum_{u \in \mathcal{N}-i} \alpha_u^1 o_{1,u}^i\Big\} \cdot \prod_{k=1,\ldots,\tau-1} \frac{1}{Z_{k+1}} exp\Big\{\sum_{u,v \in \mathcal{N}-i} \alpha_{u,v}^{k+1} \mathbb{I}(o_{k+1,u}^i, o_{k+1,v}^i) + \sum_{u \in \mathcal{N}-i} \beta_u^{k+1} \mathbb{I}(o_{k+1,u}^i, o_{k,u}^i)\Big\}$$

,

$$P(o_{\tau,u}^i = 1, \{\boldsymbol{o}_1^i, \ldots, \boldsymbol{o}_\tau^i\} \backslash o_{\tau,u}^i) = \frac{1}{Z_1} exp\Big\{\sum_{u,v \in \mathcal{N}-i} \alpha_{u,v}^1 \mathbb{I}(o_{1,u}^i, o_{1,v}^i) + \sum_{u \in \mathcal{N}-i} \alpha_u^1 o_{1,u}^i\Big\} \cdot \prod_{k=1,\ldots,\tau-2} \frac{1}{Z_{k+1}} exp\Big\{\sum_{u,v \in \mathcal{N}-i} \alpha_{u,v}^{k+1} \mathbb{I}(o_{k+1,u}^i, o_{k+1,v}^i) + \sum_{u \in \mathcal{N}-i} \beta_u^{k+1} \mathbb{I}(o_{k+1,u}^i, o_{k,u}^i)\Big\} \cdot \frac{1}{Z_\tau} exp\Big\{\sum_{v \in \mathcal{N}-i} \alpha_{u,v}^\tau \mathbb{I}(1, o_{\tau,v}^i) + \beta_u^\tau \mathbb{I}(1, o_{\tau-1,u}^i) + \sum_{y,v \in \mathcal{N}-i-u} \alpha_{y,v}^\tau \mathbb{I}(o_{\tau,y}^i, o_{\tau,v}^i) + \sum_{v \in \mathcal{N}-i-u} \beta_v^\tau \mathbb{I}(o_{\tau,v}^i, o_{\tau-1,v}^i)\Big\}$$

and

$$
\begin{aligned}
&P(o^i_{\tau,u} = 0, \{\boldsymbol{o}^i_1, \ldots, \boldsymbol{o}^i_\tau\} \backslash o^i_{\tau,u}) = \\
&\frac{1}{Z_1} exp\Big\{ \sum_{u,v \in \mathcal{N}-i} \alpha^1_{u,v} \mathbb{I}(o^i_{1,u}, o^i_{1,v}) + \sum_{u \in \mathcal{N}-i} \alpha^1_u o^i_{1,u} \Big\} \cdot \\
&\prod_{k=1,\ldots,\tau-2} \frac{1}{Z_{k+1}} exp\Big\{ \sum_{u,v \in \mathcal{N}-i} \alpha^{k+1}_{u,v} \mathbb{I}(o^i_{k+1,u}, o^i_{k+1,v}) \\
&+ \sum_{u \in \mathcal{N}-i} \beta^{k+1}_u \mathbb{I}(o^i_{k+1,u}, o^i_{k,u}) \Big\} \cdot \\
&\frac{1}{Z_\tau} exp\Big\{ \sum_{y,v \in \mathcal{N}-i-u} \alpha^\tau_{y,v} \mathbb{I}(o^i_{\tau,y}, o^i_{\tau,v}) + \sum_{v \in \mathcal{N}-i-u} \beta^\tau_v \mathbb{I}(o^i_{\tau,v}, o^i_{\tau-1,v}) \Big\}
\end{aligned}
$$

Therefore, we put them together and eliminate the terms in common to obtain the conditional probability. □

4 Efficient Distributed Decision Trees for Robust Regression

It's easy to predict the future. Just look at what you've been doing.
— Dandapani

4.1 Introduction

Decision trees are at the core of several highly successful machine learning models for both regression and classification, since their introduction by Quinlan [106]. Their popularity stems from the ability to (a) select, from the set of all attributes, a subset that is most relevant for the regression and classification problem at hand; (b) identify complex, non-linear correlations between attributes; and to (c) provide highly interpretable and human-readable models [45, 101, 106, 132]. Recently due to the increasing amount of available data and the ubiquity of distributed computation platforms and clouds, there is a rapidly growing interest in designing distributed versions of regression and classification trees [12, 14, 101, 122, 133, 150], for instance,

the decision/regression tree in Apache Spark MLlib machine learning package[1]. Meanwhile, since many of the large datasets are from observations and measurements of physical entities and events, such data is inevitably noisy and skewed in part due to equipment malfunctions or abnormal events [65, 67, 136].

With this paper, we propose an efficient distributed and regression tree learning framework that is robust to noisy data with outliers. This is a significant contribution since the effect of outliers on conventional regression trees based on the mean squared error criterion is often disastrous. Noisy datasets contain outliers (e.g., grossly mis-measured target values), which deviate from the distribution followed by the bulk of the data. Ordinary (distributed) regression tree learning minimizes the squared mean error objective function and outputs the mean of the data points in the leaf nodes as predictions, which is especially problematic and sensitive to noisy data in two aspects [65, 67, 132]. First, during the tree growing phase (the learning phase), internal tree nodes are split so as to minimize the square-error loss function, which places much more emphasis on observations with large residuals [45, 65, 132]. As a result, bias on the split of a tree node due to noisy and skewed data will propagate to descendent nodes and derail the tree building process. Second, outliers drag the mean predictions away from the true values on leaf nodes, thereby leading to highly skewed predictors. Consequentially, the distributed regression tree trained on noisy data can neither identify the true patterns in data, nor provide reliable predictions [64, 65, 67, 132, 136].

Previous methods to address robustness in the distributed regression tree fail to prevent noisy data from deviating the splits and predictions of tree nodes. For regression problems, it can be very difficult to spot noise or outliers in the data without careful investigation and even harder in multivariate data sets with both categorical and numerical features [72]. Overfitting avoidance, known as the node pruning in the context of regression trees, is a general way to allow robustness for unseen data by penalizing the tree for being too complex. But pruning operations cannot correct the biased splits of tree nodes [65, 67]. Ensemble methods like RandomForest [83], RotationForest [111] and Gradient Boosted Tree [45] produce superior results by creating a large number of trees. But outliers distributed across attributes (or features) would still bias individual trees as well as the predictions aggregated from them.

Contributions: In this paper, we focus on enhancing the robustness of a distributed regression tree as well as the training efficiency. Concretely, this paper makes the following contributions:

- We define the distributed robust regression tree employing robust loss functions and identify the difficulty in designing an efficient training algorithm for the distributed robust regression tree.
- We propose a novel distributed training framework for the robust regression tree, which

[1] http://spark.apache.org/docs/latest/mllib-decision-tree.html

consists an efficient data summarization method on distributed data and a tree growing approach exploiting the data summarization to evaluate robust loss functions.

- The proposed distributed robust regression tree and baselines are implemented based on Apache Spark. Extensive experiments on both synthetic and real datasets demonstrate the efficiency and effectiveness of our approach.

The organization of the paper is as follows: Section 4.2 summarizes the related work. Section 4.3 presents the necessary background and the problem definition. Then, Section 4.4 and Section 4.5 present proposed framework and experiment results. We discuss the possible extension in Section 4.6 and conclude the work in Section 4.7.

4.2 Related Work

To the best of our knowledge, there is no existing work on the robust loss function based distributed regression trees in the literature, and thus we first summarize previous efforts to handle noisy data for regression/classification trees in centralized environments, and then the work on the distributed regression/classification trees. and the data summarization techniques utilized in the distributed regression trees.

Robust classification/regression trees: Many methods have been proposed to handle noisy data, but most of them concentrate on refining leaf nodes after training or purely on the classification problem. [151] applies "smoothing" on the leaves of a decision tree but not inner nodes. [43] assigns a confidence score to the classifier predictions rather than improving the classification itself. Zadorny and Elkan [151], Provost and Domingos [105] and [30] improve the classification probabilities by using regression in the leaves. Another well-known method for dealing with noisy data is fuzzy decision trees [64, 100]. The fuzzy function may be domain specific and require a human expert in order to correctly define it. The other type of approaches is based on post-processing applied after a decision tree has already been built on noisy data. John [65] proposed iterative removal of instances with outlier values. [67] requires to perform back-ward path traversal for examined instances.

Our paper aims to improve the robustness of distributed regression trees by preventing the outliers from influencing the tree induction phase based on robust loss functions. Above post-processing methods can be smoothly integrated into our framework.

Distributed classification/regression trees: Our proposed approach borrows ideas from previous distributed regression tree algorithms to improve the training efficiency. But the previous algorithms do not consider the effect of data noise and outlier issues.

Parallel and distributed decision tree algorithms can be grouped into two main categories:

task-parallelism and data-parallelism. Algorithms in the first category [34, 127] divide the tree into sub-trees, which are constructed on different workers, e.g. after the first node is split, the two remaining sub-trees are constructed on separate workers. The downside of this approach is that each worker should either have a full copy of data. For large data sets, this method would lead to slowdown rather than speed-up.

In the data-parallelism approach, the training instances are divided among the different nodes of the cluster. Dividing data by features [44] requires the workers to coordinate which input data instance falls into which tree-node. This requires additional communication, which we try to avoid as we scale to very large data sets. Dividing the data by instances [122] avoids this problem. Instance-partitioning approach PLANET [101] selects splits using histograms with fixed bins constructed over the value domain of features. Such static histograms overlooks the variation of underlying data distribution as the tree grows and therefore could lead to biased splits. [14, 133] put forward to construct dynamic histograms rebuilt for each layer of tree nodes and used for deliberately approximating the exact splits. [14, 133] communicate the histograms re-built for each layer of tree nodes to a master worker for tree induction. [12] is a MapReduce algorithm which builds multiple random forest ensembles on distributed blocks of data and merges them into a mega-ensemble. In [66] ScalParC employs a distributed hash table to implement the splitting phase for classification problems. [90] approach uses sampling to achieve memory efficient processing of numerical attributes for Gini impurity in the classification tree.

In this paper, our approach falls into the instance-partition category and we build dynamic histograms to summarize the value distribution of the target variable for the robust loss estimation.

Data summarization in distributed classification/regression trees: Data summarization in distributed regression trees [14, 101, 123, 133] serves for data compression to facilitate the communication between workers and the master and supports mergeable operations for building a global picture about the data distribution on the master to grow the tree. Meanwhile, [14, 101, 133] build histograms over the feature value domain to provide splits candidates in growing the tree. Our proposed data summarization borrows ideas from [14] and is able to support efficient estimation of robust loss criteria.

4.3 Preliminaries and Problem Statement

In this part, we first present the regression tree employing robust loss functions. Then, we describe the robust regression tree in the distributed environment and formulate the problem of this paper.

4.3.1 Robust Regression Tree

In the regression problem, define a dataset $\mathscr{D} = \{(\boldsymbol{x}_i, y_i)\}$, where $\boldsymbol{x}_i \in \mathbb{N}^d$ is a vector of predictor features of a data instance and $y_i \in \mathbb{R}$ is the target variable. d is the number of features. Let $D^n \in \mathscr{D}$ denote the set of instances falling under tree node n.

Regression tree construction [106, 132] proceeds by repeated greedy expansion of tree nodes layer by layer until a stopping criterion, e.g. the tree depth is met. Initially, all data instances belong to the root node of the tree. An internal tree node (e.g., D^n) is split into two children nodes respectively with data subsets $D_L (D_L \subset D^n)$ and $D_R (D_R = D^n - D_L)$ by using a predicate on a feature, so as to minimize the weighted loss criteria: $\frac{|D_L|}{|D^n|} L(D_L) + \frac{|D_R|}{|D^n|} L(D_R)$, where $L(\cdot)$ is a loss function (or error criteria) defined over a set of data instances.

This paper proposes the distributed regression tree employing robust loss functions to handle noisy datasets with outliers on the target variable (the regression tree is robust to outliers in feature space [45]). In robust regression, there are two main types of robust loss functions: accommodation and rejection [45, 65, 128]. Accommodation approach is to define a loss function that lessens the impact of outliers The least absolute deviation, referred to as LAD, is an accommodation method [45, 128, 132]. It is defined on a set of data instances D as: $L_l(D) = \frac{1}{|D|} \sum_{(\boldsymbol{x}_i y_i) \in D} |y_i - \hat{y}|$, and $\hat{y} = median_{(\boldsymbol{x}_i y_i) \in D}(\{y_i\})$, which returns the median of a set of values [132]. On the other hand, rejection approach aims to restrict the attention only to the data that seems "normal" [65]. The loss function of the rejection type is the trimmed least absolute deviation, referred to as TLAD. It is defined as $L_l(\tilde{D})$, where $\tilde{D}$ is the trimmed dataset of D derived by removing data instances with the τ% largest and τ% smallest target values $(0 < \tau < 1)$ from D and thus in TLAD $\hat{y} = median_{y_i \in \tilde{D}}(\{y_i\})$.

Then, the robust regression tree in this paper is defined as:

Definition 4.3.1 (Robust Regression Tree)**.** In a robust regression tree, an internal tree node is split so as to minimize the weighted robust loss function $\frac{|D_L|}{|D^n|} L_l(D_L) + \frac{|D_R|}{|D^n|} L_l(D_R)$, where D_L and D_R are two (trimmed) data subsets corresponding to the children nodes. The leaf nodes take the median of target values in the leaf node as the prediction value.

4.3.2 Robust Regression Tree in the Distributed Environment

In contemporary distributed computation systems [79, 123], one node of the cluster is designated as the master processor and the others are the workers. Denote the number of workers by P. The training instance set is instance-divided into P disjoint subsets stored in different workers and each worker can only access its local data subset. Let D_p be the set of data instances stored at worker p, such that $\cup_{p=1}^{P} D_p = \mathscr{D}$. For $p, q \in \{1, \ldots, P\}$, $D_p \cap D_q = \emptyset$ and $|D_p| \approxeq |\mathscr{D}|/P$. Denote the data instances in D_p belonging to a tree node n by D_p^n. A straightforward way to

grow the robust regression tree layer by layer on the master is inefficient [14, 101, 123], because splitting an internal tree node requests to repeatedly access distributed data and calculate LAD (or TLAD) via expensive distributed sorting [14, 123], for each trial split predicate per feature. Such a solution incurs dramatic communication and computation overheads , thereby degrading the training efficiency and scalability [14, 132].

To this end, our following proposed distributed robust regression tree will exploit data summarization [14, 101, 133], which is able to provide compact representations of the distributed data, to enhance the training efficiency.

4.3.3 Problem Statement

As is presented above, it is non-trivial to design an efficient training approach for distributed robust regression tree. Therefore, the problem this paper aims to solve is defined as:

Definition 4.3.2 (Training a Distributed Robust Regression Tree)**.** Given robust lost functions (LAD or TLAD) and training instance partitions $D_1, \ldots, D_p$ of a data set $\mathscr{D}$ distributed across the workers $1, \ldots, p$ of a cluster, training a robust regression tree in such a distributed setting involves two sub-problems: (1) to design an efficient data summarization method for the workers to extract sufficient information from local data and to transmit only such data summarization to the master with bounded communication cost. (2) to grow a robust regression tree on the master by estimating the robust loss function based on the data summarization.

To keep things simple, we assume that all the features are discrete or categorical. However, all the discussion below can be easily generalized to continuous features [45], which is discussed in Section 4.6. Therefore, a split predicate on a categorical feature is a value subset. Let $\mathcal{V}_k$ represents the value set of feature k and $k \in \{1, \ldots, d\}$. For instance, given the set of data instances D^n on a tree node n and a value subset on feature k, $\mathcal{V}_k^- \subset \mathcal{V}_k$, two data subsets partitioned by $\mathcal{V}_k^-$ are $D_L = \{(\boldsymbol{x}_i, y_i) | (\boldsymbol{x}_i, y_i) \in D^n, x_{i,k} \in \mathcal{V}_k^-\}$ and $D_R = D^n - D_L$.

Often, regression tree algorithms also include a pruning phase to alleviate the problem of overfitting the training data. For the sake of simplicity, we limit our discussion to regression tree construction without pruning. However, it is relatively straightforward to modify the proposed algorithms to incorporate a variety of pruning methods [14, 45].

4.4 Distributed Robust Regression Tree

In this part, we introduce the key contribution, the distributed robust regression tree, referred to as DR2-Tree.

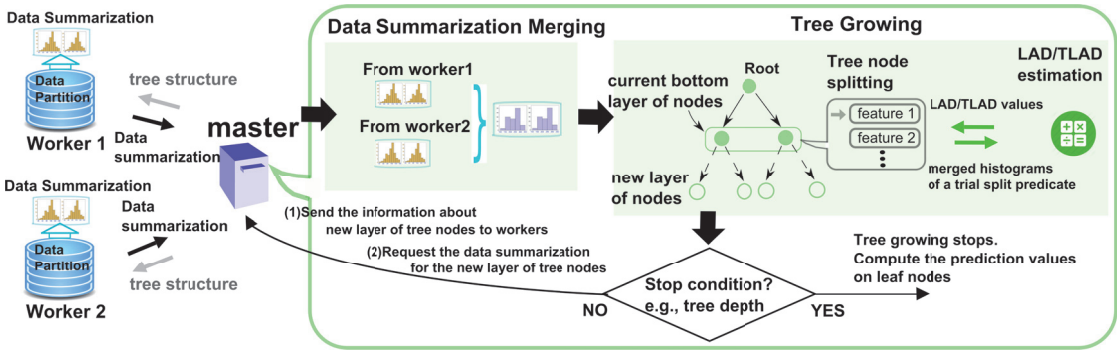


Figure 4.1 – Framework of the distributed robust regression tree (best viewed in colour).

Overview: As is shown in Figure 4.1, in DR2-Tree the master grows the regression tree layer by layer in the top-down manner. Each worker retains the split predicates of the so-far trained tree nodes for data summarization. An efficient dynamic-histogram based data summarization approach is designed for workers to communicate with the master (refer to Section 4.4.1). Then, by using such approximate descriptions of data, the master is able to efficiently evaluate robust loss functions for determining the best split of each internal tree node, thereby circumventing expensive distributed sorting for deriving LAD/TLAD (refer to Section 4.4.2). Finally, the master sends the new layer of tree nodes to each worker for the next round of node splitting.

4.4.1 Data Summarization on Workers

Our data summarization technique adopts the dynamic histogram, a concise and effective data structure supporting mergable operations in the distributed setting [14, 133]. The one-pass nature of our proposed data summarization algorithm enables it to be adaptable to the distributed streaming learning [14] as well. Moreover, we will derive efficient robust loss function estimation algorithm based on such data summarization in the next subsection.

During the data summarization process, worker p builds a *histogram set* denoted by $\mathscr{H}_p^n = \{H_{r,v_r}^n\}$, for each tree node on the bottom layer, e.g., node n. It summarizes the target value distributions of D_p^n, the data instances belonging to tree node n in data partition D_p. H_{r,v_r}^n is a histogram describing the target value distribution of data instances having value v_r on feature r in D_p^n. H_{r,v_r}^n is a space bounded histogram of maximum β bins ($|H_{r,v_r}^n| \leq \beta$), e.g. $H_{r,v_r}^n = \{b_1, \ldots, b_\beta\}$. Let $count(H)$ (or $count(\mathscr{H})$) be the number of data instances summarized by a histogram H (or a histogram set $\mathscr{H}$). Each bin of a histogram is represented by a quad, e.g. $b_i = (l, r, c, s)$, where l and r are the minimum and maximum target values in this bin, c is the number of target values falling under this bin and s is the sum of the target values. We will see how such quad elements are used in growing the tree in the next subsection. The number of bins β in the histograms is specified through a trade-off between accuracy and computational

Algorithm 5 Data summarization on a worker

Input: data partition in this worker, e.g., D_p

Output: histogram sets $\{\mathscr{H}_p^n\}$ describing the target value distribution in internal tree node n

Bins in each histogram are maintained according to the order of bin boundaries.

$T_{v_r}^{n_i}$: a priority queue recording the distances between neighbouring bins.

for each data sample $(\boldsymbol{x}_i, y_i)$ in D_p **do**

  search the tree built so far to locate the leaf node, e.g. n_i, to which sample $(\boldsymbol{x}_i, y_i)$ belongs.

  for each feature value $x_{i,k}$ of $\boldsymbol{x}_i$ **do**

    search for the bin b_{incl} such that $y_i \in [b_{incl}.l, b_{incl}.r]$ by the binary search over bins of $H_{k,x_{i,k}}^{n_i}$

    if there exits such a bin b_{incl} for y_i **then**

      only update the bin b_{incl} by $b_{incl}.c = b_{incl}.c + 1$, $b_{incl}.s = b_{incl}.s + y_i$

    else

      # b_{lower} and b_{upper} are obtained during the above search process for b_{incl}.

      $b_{lower} = \underset{b_j \in \{b_k | b_k.r \le y_i\}}{\operatorname{argmax}} b_j.r$

      $b_{upper} = \underset{b_j \in \{b_k | b_k.l \ge y_i\}}{\operatorname{argmin}} b_j.l$

      insert a new bin $(y_i, y_i, 1, y_i)$ into $H_{k,x_{i,k}}^{n_i}$ between bin b_{lower} and b_{upper}

      insert two new neighbour-bin distances $|b_{lower}.r - y_i|$ and $|b_{upper}.l - y_i|$ to the $T_{v_r}^{n_i}$

      if current $|H_{k,x_{i,k}}^{n_i}|$ > histogram space bound β **then**

        for the pair of bins b_u and b_v with the minimum distance in $T_{v_r}^{n_i}$, replace the bins b_u and b_v in $H_{k,x_{i,k}}^{n_i}$ by the merged bin as:

        $(\min(b_u.l, b_v.l), \max(b_u.r, b_v.r), b_u.c + b_v.c, b_u.s + b_v.s)$

      end if

    end if

  end for

end for

and communication costs: a large number of bins gives a more accurate data summarization, whereas small histograms are beneficial for avoiding time, memory, and communications overloads.

Algorithm 5 presents the data summarization procedure on each worker, which updates the local data instances one by one to the corresponding histogram set. First, the tree node n_i in the bottom layer of the tree for a data instance $(\boldsymbol{x}_i, y_i) \in D_p$ is found (line $1-2$) and its associated $\mathscr{H}_p^{n_i}$ will be updated. For each feature value of $(\boldsymbol{x}_i, y_i)$, y_i is inserted to the corresponding histogram in $\mathscr{H}_p^{n_i}$ by either updating an existing bin having the value range covering y_i (line $3-6$) or inserting a new bin $(y_i, y_i, 1, y_i)$ to the histogram (line $7-12$). Second, if the size of the histogram exceeds the predefined maximum value β then the nearest bins are continuously merged until addressing the limit β (line $13-16$). A temporary priority structure (e.g., $T_{v_r}^{n_i}$) is maintained for efficiently finding closest bins to merge (line $13-16$). Finally, workers only send such data summarization to the master.

Complexity Analysis: In line $2-6$, the binary search over bins of a histogram takes $\log\beta$ time. Then the priority structure can support in finding the nearest bins and updating bin distances in $\log\beta$ time (line $13-16$). Overall, the time complexity of Algorithm 5 is $\mathscr{O}(|D_p| d \log\beta)$. Compared with the histogram building approach in [14, 133], our method circumvents the sorting operation for updating individual data instances and improves the efficiency, as is demonstrated in Section 5.5. The communication complexity for transmitting data summarization of the bottom layer of nodes between the worker and master is bounded by $\mathscr{O}(\max_r(|\mathscr{V}_r|) d \beta)$ independent of the size of the data partitions. For the features with high cardinality, our data summarization can incorporate extra histograms over feature values to decorrelate the communication cost and the feature cardinality [14, 133].

4.4.2 Tree Growing on the Master

In this part, we will first outline the tree node splitting process using the data summarization in growing the tree. Then, we present the involved two fundamental operations in detail, namely the histogram merging and LAD/TLAD estimation.

Tree Node Splitting: In order to find the best split of a tree node, we need a histogram set summarizing all the data instances falling under this node. Therefore, as is presented in Algorithm 6, a unified histogram set is built by using the histogram merging operation, which will be described in Algorithm 7. Then, it iterates over each feature to find a split predicate (Line 4-6), i.e., a feature value subset, so as to minimize the weighted loss as:

$$\{v*, \mathscr{V}^{+*}, \mathscr{V}^{-*}\} = \underset{v_i, \mathscr{V}^+, \mathscr{V}^-}{argmin} \hat{L}_l(H^-) \frac{count(H^-)}{count(\mathscr{H}^n)} + \hat{L}_l(H^+) \frac{count(H^+)}{count(\mathscr{H}^n)} \tag{4.1}$$

Algorithm 6 Tree node splitting

Input: histogram sets of tree node n from all data partitions, $\mathscr{H}_1^n, \ldots, \mathscr{H}_P^n$.
Output: the split feature and associated value set for tree node n.

1: build a unified histogram set summarizing the overall target value distribution for this tree node $\mathscr{H}^n = merge(\mathscr{H}_1^n, \ldots, \mathscr{H}_P^n)$ by using the histogram merging operation presented in Algorithm 7
2: **for each** feature $k \in \{1, \ldots, d\}$ **do**
3: Sort the feature values in $\mathscr{V}_k$ according to the median estimations of data in the corresponding histograms [132].
4: $\tilde{\mathscr{V}}_k$: the sorted feature values in $\mathscr{V}_k$.
5: iterate over $\tilde{\mathscr{V}}_k$ to find a v_j and the associated feature value subsets $\mathscr{V}^- = \{v_j | j \leq i\}$ and $\mathscr{V}^+ = \mathscr{V}_k - \mathscr{V}^-$, so as to minimize the weighted robust loss function.
6: **end for**
7: return the feature and value subsets, which achieve the minimum robust loss.

where $\hat{L}_l(\cdot)$ is the histogram based estimation of robust loss functions (LAD/TLAD), which is presented in Algorithm 8. $\mathscr{H}^n = merge(\mathscr{H}_1^n, \ldots, \mathscr{H}_P^n)$, which will be described in Algorithm 7. For a trial feature value subset, e.g. $\mathscr{V}^- = \{v_j | j \leq i\}$ and $\mathscr{V}^+$, we need to estimate the LAD/TLAD over the data subsets defined by $\mathscr{V}^-$ and $\mathscr{V}^+$. Therefore, two temporary histograms, e.g., H^- and H^+ are built by merging the histograms in $\mathscr{H}^n$ corresponding to the feature values present in $\mathscr{V}^-$ and $\mathscr{V}^+$, i.e., $H^- = merge(\{H_{v_j}^n | j \leq i\})$ and $H^+ = merge(\{H_{v_j}^n | j > i\})$ approximating the distributions of two data subsets defined by $\mathscr{V}^-$ and $\mathscr{V}^+$.

Finally, when the tree reaches the stopping depth, the predictions on the leaf nodes can be exactly derived by accessing the distributed dataset. This step is only performed when the tree growing phase is finished.

Histogram Merging: Our proposed histogram merging operation is a one-pass method over the bins of histograms and creates a histogram summarizing the union of data distribution of the two histograms. As is presented in Algorithm 6, it is mainly used in two cases: (1) build a unified histogram set for each tree node on the bottom layer; (2) build temporary histograms to approximate the target value distributions of two data subsets defined by a trial feature value subset. Algorithm 7 presents the histogram merging algorithm. Two histograms H_1 and H_2 are first combined in the merge-sort way. During this process, a heap is maintained to record the neighbour-bin distances. Then, bins which are closest are merged together to form a single bin. The process repeats until the histogram has β bins.

Complexity Analysis: In Algorithm 6 Line 2-11 scans the bins in histograms H_1 and H_2 once and thus takes $\mathscr{O}(\beta)$. Line 12-14 combines the redundant bins by using the heap, which takes

Algorithm 7 Histogram merging

Input: Two histograms, e.g., H_1 and H_2.
Output: A histogram H summarizing the union of data distribution in H_1 and H_2.
E is a priority queue recording the distances between neighbouring bins.
H: merged histogram.
while H_1 and H_2 have bins **do**
  b_i and b_j: current popped bins from H_1 and H_2
  if $b_i.l < b_j.l$ **then**
    insert b_i in H
  else
    insert b_j in H
  end if
  insert the new neighbour bin distance in E.
end while
insert the remaining bins in H_1 or H_2 to H.
while $|H| >$ histogram space bound β **do**
  pop from E the pair of bins b_u and b_v with the minimum bin distance
  replace the bins b_u and b_v in H by the merged bin
  $(\min(b_u.l, b_v.l), \max(b_u.r, b_v.r), b_u.c + b_v.c, b_u.s + b_v.s)$
end while

$\mathcal{O}(\beta log(\beta))$.

LAD/TLAD Estimation: A straightforward method to estimate LAD (or TLAD) based on a histogram is to first make a median estimate and then to sample data in each bin of the histogram to approximate individual absolute deviations [14,133]. Both the median estimation and data sampling process introduce errors into the LAD (or TLAD) estimation [132].

To this end, we propose a more efficient and precise algorithm to approximate LAD and TLAD in one-pass way. Before giving the details, we first define some notations.

Definition 4.4.1 (Quantile Bin of a Histogram)**.** Given a histogram $H = \{b_1, \ldots, b_\beta\}$, $count(H)$ the number of values this histogram summarizes and a quantile q over the summarized values, the quantile bin b_q addresses $\sum_{b_i<b_q} b_i.c < count(H) \cdot q$ and $\sum_{b_i \leq b_q} b_i.c \geq count(H) \cdot q$

Definition 4.4.2 (R-Partial-Sum of a Bin)**.** Given a bin $b = (l, r, c, s)$ of a histogram, R-Partial-Sum of bin b, $S_p(b, R)$ is defined as the sum of the R smallest values summarized in this bin.

Recall that in the data summarization in Algorithm 5, the histogram updating process unites neighbouring bins according to the distance of bin boundaries. This allows the bins to adapt to the data distribution. Regarding the values summarized by a bin in a histogram (e.g., b), we can

safely assume that they are uniformly distributed in range $[b.l, b.r]$ [14]. Therefore, we provide the lemma below, which will be used for LAD estimation, to approximate R-Partial-Sum:

Algorithm 8 LAD / TLAD Estimation

Input: A histogram H, trim ratio τ
Output: LAD or TLAD estimation.

1: s: variable to record $\sum_{b_i>b_m} b_i.s - \sum_{b_i<b_m} b_i.s$, s_t: variable to record $\sum_{b_i>b_{\overline{q}}} b_i.s - \sum_{b_q<b_i<b_m} b_i.s$.
   # m is the index of 1/2-quantile bin; q and $\overline{q}$ are the indices of τ-quantile and (1-τ)-quantile bins.
2: c: variable to record the current count of data instances.
3: **for each** bin b_i in H **do**
4: $c = c + b_i.c$
5: record quantile bin m, q and $\overline{q}$ when c address the corresponding conditions.
6: **if** 0.5-quantile bin m is found **then**
7: $s = s + b_i.s$
8: **else**
9: $s = s - b_i.s$
10: **end if**
11: **if** τ-quantile bin b_q is found and 1/2-quantile bin b_m is not found **then**
12: $s_t = s_t - b_i.s$
13: **else if** (1-τ)-quantile bin $b_{\overline{q}}$ is not found and 1/2-quantile bin q is found **then**
14: $s_t = s_t + b_i.s$
15: **end if**
16: **end for**
17: LAD: $\hat{L}_l(H) = s + b_m.s - 2\hat{S}_p(b_m, R)$,
18: TLAD: $\hat{L}_l(H,\tau) = s_t + \hat{S}_p(b_q, R_1) - b_q.s + \hat{S}_p(b_{\overline{q}}, R_2) + b_m.s - 2\hat{S}_p(b_m, R)$

Lemma 4.4.1. *For a bin $b = (l, r, c, s)$ of a histogram and an integer R ($R \leq b.c$), under the assumption of the uniform distribution of values in the bin, R-Partial-Sum of bin b can be approximated by* $S_p(b,R) \approxeq \hat{S}_p(b,R) = \begin{cases} b.s & : R = b.c \\ R \cdot b.l + R(R-1)\delta & : otherwise \end{cases}$, *where* $\delta = \frac{(b.s - b.r - b.c \cdot b.l + b.l)}{(b.c-2)(b.c-1)}$.

Proof. □

Now we provide the following lemma for estimating the LAD/TLAD based on a histogram as:

Lemma 4.4.2. *Given a histogram $H = \{b_1, \ldots, b_\beta\}$, the LAD/TLAD over the data summarized by histogram H can be exactly computed by:*

(1) $L_l(H) = \sum_{b_i>b_m} b_i.s - \sum_{b_i<b_m} b_i.s + b_m.s - 2S_p(b_m, R)$, *where* $R = \lceil \frac{C}{2} \rceil - \sum_{b_i<b_m} b_i.c$, $C = count(H)$ *is the total number of data instances covered in histogram H, and an b_m is the $\frac{1}{2}$-quantile bin.*

(2) $L_l(H,\tau) = \sum_{b_m<b_i<b_{\overline{q}}} b_i.s - \sum_{b_q<b_i<b_m} b_i.s + S_p(b_q,R_1) - b_q.s + S_p(b_{\overline{q}},R_2) + b_m.s - 2S_p(b_m,R)$, *where* b_m, b_q *and* $b_{\overline{q}}$ *are respectively the* $\frac{1}{2}$, τ *and* $(1-\tau)$*-quantile bins,* $R = \lceil \frac{C}{2} \rceil - \sum_{b_i<b_m} b_i.c$, $R_1 = C\cdot\tau - \sum_{b_i<b_q} b_i.c$, $R_2 = C\cdot(1-\tau) - \sum_{b_i<\overline{q}} b_i.c$.

Proof. Limited by the space, refer to Section 4.8 for the proof details. □

Lemma 4.4.2 suggests that in estimating LAD/TLAD based on a histogram, the median estimation step is circumvented. Meanwhile, given the histograms LAD/TLAD can be estimated through replacing $S_p(\cdot)$ in Lemma 4.4.2 by $\hat{S}_p(\cdot)$ defined in Lemma 4.4.1 and exactly computing the remaining terms. In summary, the histogram based estimation of robust loss functions can be expressed as:

$$\hat{L}_l(H) = \sum_{b_i>b_m} b_i.s - \sum_{b_i<b_m} b_i.s + b_m.s - 2\hat{S}_p(b_m,R) \tag{4.2}$$

and

$$\begin{aligned}\hat{L}_l(H,\tau) = \sum_{b_m<b_i<b_{\overline{q}}} b_i.s - \sum_{b_q<b_i<b_m} b_i.s + \hat{S}_p(b_q,R_1) - b_q.s + \\ \hat{S}_p(b_{\overline{q}},R_2) + b_m.s - 2\hat{S}_p(b_m,R)\end{aligned} \tag{4.3}$$

where R, R_1 and R_2 are defined as Lemma 4.4.2.

On the basis of Lemma 4.4.2, we put forward the LAD/TLAD estimation algorithm, as is shown in Algorithm 7. It is able to estimate LAD or TLAD in one-pass over the bins of the given histogram. In our LAD/TLAD estimation algorithm, the only approximate part is $\hat{S}_p(b,R)$. Now we provide the theoretical error bound on it:

Theorem 4.4.1. *Given a bin* $b = (l,r,c,s)$ *of a histogram and* R *(*$R \le b.c$*), if* $R = 1$ *or* $R = b.c$, $\hat{S}_p(b,R)$ *provided in Lemma 4.4.1 is the exact* R*-Partial-Sum. Otherwise, the approximation error of* R*-Partial-Sum of bin* b, $S_p(b,R) - \hat{S}_p(b,R)$ *is bounded within* $[(R-b.c)\cdot(b.r-b.l), b.s - b.c\cdot b.l]$.

Proof. Refer to Section 4.8. □

Therefore, the LAD/TLAD estimation has bounded errors as well.

4.5 Experimental Evaluations

In this section, we perform extensive experiments to demonstrate the efficiency and effectiveness of DR2-Tree. we first present the setup of the experiments including datasets, baselines and the implementation environment. Then, we report the results focusing on three aspects: the efficiency in terms of training time and speedup, the effectiveness in terms of prediction accuracy and the data summarization performance in DR2-Tree.

4.5.1 Setup

Dataset: In the experiments, we use one synthetic and two real datasets.

Synthetic Data: Our synthetic data generator[2] produces data instances with specified number of features. For each distinct feature value combination e.g., $(v^1, \ldots, v^d)$, where v^1 is the value of the first feature and d is the number of features, it generates several data instances having such feature values and the target values sampled from a Gaussian distribution. Such Gaussian distributions are specific *w.r.t.* feature-value combinations. Meanwhile, data instances with outliers on the target variable are injected based on a Bernoulli distribution. The probability of the Bernoulli distribution is specified through the *percentage of outliers* in the produced dataset and it is set as 0.05 initially, i.e., 5% of data instances have outlier target values. *The magnitude of outlier target values* is defined as the times of the Gaussian distribution mean. By default, the magnitude is 3, which means that the target value of an outlier data instance is sampled from a Gaussian distribution with 3 times larger mean than the mean of the corresponding feature value combination's distribution. The percentage and magnitude of outliers will be tuned later in Subsection 4.5.3.

Flight Dataset: It contains the scheduled and actual departure and arrival times of flights reported by certified U.S. air carriers from 1987-2008 [3]. It contains data instances with abnormal values on the "arrival delay" and "departure delay" attributes, due to abnormal events, e.g., weather, security, etc. In our experiments, we use the attribute "ArrDelay" as the target variable and the categorical features as the independent variables. The cardinalities of these categorical features vary from 10 to 1032.

Network Dataset: It is a dataset provided by a major European telecommunication service provider consisting of active measurements from probes within a residential ISP network. The probes measure various performance fields such as the throughput, jitter and delay between their location and chosen end-points. Furthermore, each probe and end-point are associated with various categorical and continuous features, such as the time of the measurement, the

[2] https://github.com/weilai0980/DRSquare_tree/tree/master/dr2tree_src
[3] http://stat-computing.org/dataexpo/2009/the-data.html

location of the endpoints and the configuration of the lines. Finally the tests cover a period of 2 days and involve 124 probes and 1314 targets. This dataset is noisy in the sense that due to network anomalies and events, the measurements could have huge outlier values.

Baselines: ER2T is a distributed robust regression tree. SRT and DHRT are two representative distributed regression trees in the literature [14,101,133]. ER2T: It refers to the **e**xact distributed **r**obust **r**egression **t**ree. It builds the robust regression tree on the master by exactly calculating the robust loss functions in a distributed way. SRT: It refers to the distributed regression tree based on square error criteria [101] in Apache Spark machine learning tool set [4]. Prior to the tree induction, a pre-processing step is performed to obtain static and equidepth histograms for each feature and the split points are constantly selected from the bins of such histograms in the training phase. DHRT: It implements a single distributed regression tree based on [133], which employs **d**ynamic **h**istograms [14] to summarize statistics in distributed data for evaluating the square error split criterion on the master. In building histograms, it requires to sort the bins each time a data instance is added to the set already represented by the histogram [14].

In Subsection 4.5.3, we will also use random forests (RF) and gradient boosted regression trees (GBT) in the distributed machine learning library Spark MLlib to compare with our robust regression tree in terms of accuracy.

Implementation: Our proposed DR2-Tree and baselines are all implemented on Apache Spark, a popular distributed data processing engine. The engine is deployed on a cluster of 23 servers, each with 16 cores (2.8GHz) and 32G of RAM.

4.5.2 Efficiency

In this group of experiments, we evaluate the efficiency of growing regression trees under different conditions. The training time is measured as the total time for growing a tree from the root node until the specified depth. To mitigate the effects of varying cluster conditions, all the results have been averaged over multiple runs.

We consider four parameters to tune in this set of experiments, namely the tree depth, training data size, maximum number of bins in data summarization and the number of workers. They have direct effect on the training time [14, 101, 123, 133]. The experiments are performed by varying one parameter while keeping the others as default values. By default, the maximum number of bins is set as 500, the number of workers is 5, the depth is 6 and the size of the training dataset is 10 million initially.

[4] http://spark.apache.org/docs/latest/mllib-decision-tree.html

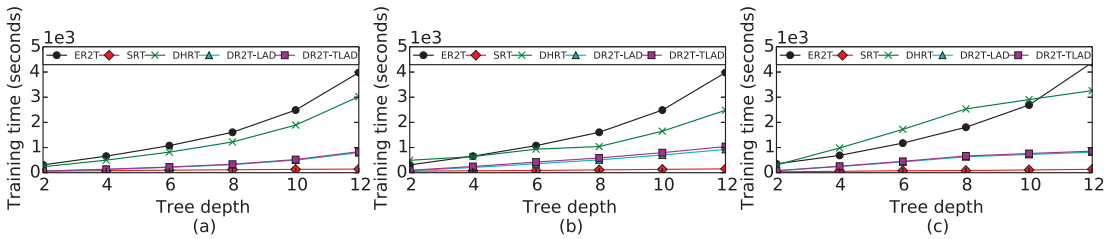


Figure 4.2 – Training time *w.r.t.* the depth of the tree. (a) synthetic dataset (b) flight dataset (c) network dataset. (best viewed in colour)

Depth: Figure 4.2 presents the training time as a function of the training depth of the regression tree. DR2-Tree outperforms ER2T and DHRT by 3× and 2× faster in average. In ER2T the training time consistently takes the longest, as it computes the expensive exact median and LAD (TLAD) in the distributed setting. SRT takes 0.5 times less time than DR2-Tree. This is because SRT constantly summarizes the data using fixed bins and thus takes less time to extract statistics in bins from distributed data during the training phase. But square-error based SRT and DHRT are less robust to noisy data than DR2-Tree, which will be shown in the next subsection.

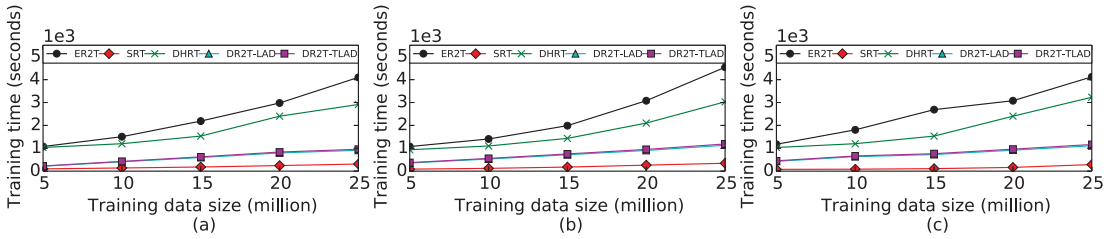


Figure 4.3 – Training time *w.r.t.* the size of training dataset. (a) synthetic dataset (b) flight dataset (c) network dataset. (best viewed in colour)

Data size: In Figure 4.3, we present the training time as a function of the size of the training dataset, i.e., the number of data instances. Due to the one-pass nature of data summarization and tree growing processes, the training time of DR2-Tree increases linearly, highlighting the scalability. DHRT has a quickly increasing training time in part due to the quadratic computation in updating histograms. DR2-Tree takes 3× and 2× less training time in average than ER2T and DHRT.

Maximum number of bins: In Figure 4.4, we investigate the effect of the maximum number of bins in data summarization on the training time. ER2T employs no data summarization. In SRT, bins are built according to the cardinality of features in data. Therefore, varying the number of bins has no effect on ER2T and SRT and in Figure 4.4 only the results of DR2-Tree

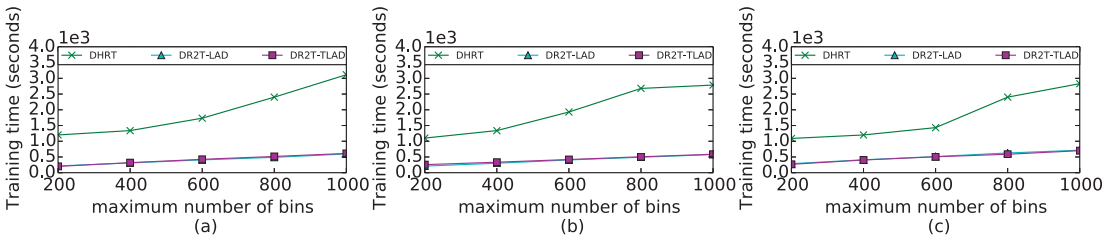


Figure 4.4 – Training time *w.r.t.* the maximum number of bins in data summarization. (a) synthetic dataset (b) flight dataset (c) network dataset. (best viewed in colour)

and DHRT are reported. The number of bins affects both the efficiency of data summarization on workers and tree growing on the master, and thus in general the training time is positively correlated with the maximum number of bins. At the highest level of bin numbers, the training time of DR2-Tree is average 4 times less than DHRT.

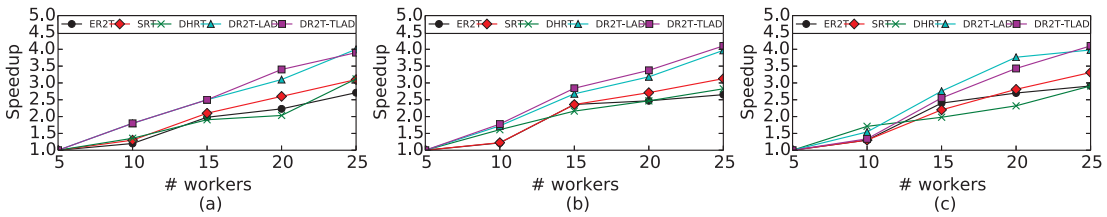


Figure 4.5 – Training time *w.r.t.* the number of workers. (a) synthetic dataset (b) flight dataset (c) network dataset. (best viewed in colour)

Number of workers: In Figure 4.5, we proceed to investigate the speedup for different numbers of workers. For large datasets, the communication between workers and the master is negligible relative to the gain in the data summarization building phase. Therefore, increasing the number of workers is beneficial for speeding up the training process [14]. DR2-Tree presents 2× higher speedup than ER2T at the highest level of number of workers.

4.5.3 Effectiveness

In this part, we evaluate the prediction accuracy of DR2-Tree and baselines under different dataset properties and regression tree set-ups. Specifically, we aim to study the effect of the maximum number of bins, the outlier percentage and magnitude in the training dataset. The prediction accuracy is measured by normalized root mean square error (NRMSE), so as to facilitate the comparison between datasets. Lower values of NRMSE are considered better. The trim ratio in DR2-Tree-TLAD is chosen within the range of $[0.05, 0.15]$ by cross-validation in this group of experiments. The un-tuned parameters in each group of experiments are set

as the default values in Section 4.5.2.

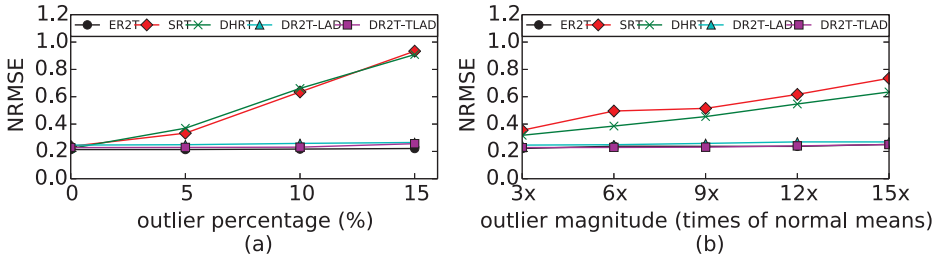


Figure 4.6 – Prediction accuracy *w.r.t.* the (a) outlier percentage and (b) magnitude in the training dataset (best viewed in colour).

Outlier properties in the dataset: In this group of experiments, we investigate the effect of the noise level of the training dataset, namely the outlier percentage and magnitude, on the prediction accuracy. Since we can only manipulate the noise level of the synthetic dataset, only the results on the synthetic dataset are reported in Figure 4.6.

In Figure 4.6(a), we increase the percentage of data instances with outlier target values while keeping the magnitude of the outlier values as 3× of the target value mean. It is observed that initially when the training dataset has no outliers, the accuracies of all approaches are highly close. As the percentage of outliers increases, the accuracy difference between the square error and robust error criterion based approaches becomes significant.

In Figure 4.6(b), we study the effect of the outlier magnitude on the accuracy. In this group of experiments, 5% data instances have outlier target values. When the magnitude of outliers increases, ER2T and DR2-Tree demonstrate stable accuracy and have 2 times less errors than SRT and DHRT at the highest level of the outlier magnitude. Compared with Figure 4.6(a), we also observe that square error based approaches are more sensitive to the outlier percentage than to the outlier magnitude in the dataset.

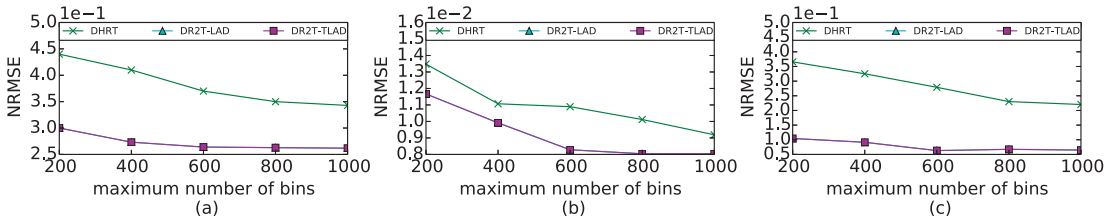


Figure 4.7 – Prediction accuracy *w.r.t.* the maximum number of bins in data summarization. (a) synthetic dataset (b) flight dataset (c) network dataset. (best viewed in colour)

Maximum number of bins: Figure 4.7 displays the prediction accuracy for different number

of bins in the data summarization. As is presented in Section 4.4, the number of bins affects the precision of error criterion estimation in DR2-Tree. Meanwhile, it should avoid setting the number of bins too large, otherwise the training efficiency would degrade, as is shown in Figure 4.4. Since only DR2-Tree and DHRT have tunable dynamic histograms, only the results of them are shown.

For the noisy synthetic and network datasets, as the number of bins increases, the master in DR2-Tree can obtain more precise data summarization based LAD/TLAD estimation and thus yields decreasing prediction errors. DR2-Tree outperforms DHRT by around 3 times. For the flight data, they present comparable accuracies.

Table 4.1 – Overall accuracy comparison (NRMSE).

datasets	ER2T	SRT	DHRT	DR2-Tree-LAD	DR2-Tree-TLAD	RF	GBT
Synthetic data	0.225	0.481	0.493	0.224	0.219	0.481	0.476
Flight data	0.00882	0.00874	0.00908	0.00889	0.00890	0.00836	0.00835
Network data	0.061	0.148	0.153	0.0629	0.0581	0.145	0.181

Overall accuracy comparison: In this part, we run all the approaches by cross-validation to achieve the best prediction accuracies for the three datasets and report the results in Table 5.2. The synthetic dataset is set to have 5% outlier target values with magnitude 3. It shows that robust error criterion based approaches have around 50% less error than square error based approaches, i.e., SRT, DHRT, RF and GBT. For not so noisy data, i.e. the flight data, two types of approaches have very comparable accuracy. Such results also demonstrate the wide applicability of our DR2-Tree.

4.5.4 Data Summarization Performance in DR2-Tree

Communication cost of data summarization: Table 4.2 shows the average communication cost of data summarization for a tree node between the worker and master. It is measured by the amount of basic data units (e.g., integer, floats, etc.). We study the communication cost variation as tree grows, since DR2-Tree employs dynamic histograms, which are rebuilt for the new-grown bottom layer of tree nodes. From Table 4.2, we observe that the communication

Table 4.2 – Data summarization communication cost as the tree grows.

Tree depth	2	4	6	8	10
Synthetic data	12500	25000	25000	18359	17871
Flight data	560000	1120000	1120000	1225000	1050000
Network data	200000	400000	437500	412500	411500

cost increases first, then keeps relatively constant or slightly decreases, as the tree grows. This

is because initially the data instances are assigned to small number of tree nodes and each tree node has data summarization with histograms of full bins to summarize the data. As the tree grows and has more nodes, the amount of data instances in each node is decreasing and the data summarization requires less bins in histograms and therefore the communication cost keeps relatively stable or decreases. The communication cost is different across datasets. This is because the number of features and the size of feature value set are different for the three datasets.

Accuracy of data summarization: In this part, we proceed to further investigate the accuracy of the data summarization based LAD/TLAD estimation in DR2-Tree. In order to understand the accuracy under different data distributions [14], we generate seven synthetic sets via different kinds of probability distributions and summarize each data set by using the histogram building approach in DR2-Tree. The probability distributions and associated parameters are listed in Table 4.3 and Table 4.4. For each distribution, 10^5 data points are generated. Then, we compare the histogram based LAD/TLAD estimation with the exact values by using the mean absolute percentage error. It expresses the accuracy as a percentage and fits for the datasets of different value scales. We run the process of data generation, histogram building and LAD/TLAD estimation for several times to report average results in Table 4.3 and Table 4.4. We

Table 4.3 – LAD/TLAD estimation error *w.r.t.* the maximum number of bins in histograms (two numbers in each cell respectively correspond to the mean absolute percentage errors of LAD and TLAD.)

#Bins / Distribution	200	400
Uniform ([0, 100])	(0.0000410, 0.347)	(0.0000411, 0.216)
Normal ($\mu = 0, \sigma = 1$)	(0.571, 0.844)	(0.450, 0.690)
Exponential($\mu = 0.5$)	(0.243, 0.0753)	(0.221, 0.0776)
Beta ($a = 0.5, b = 0.5$)	(0.0000316, 0.198)	(0.0000309, 0.123)
Gama ($a = 3, b = 1$)	(0.118, 0.381)	(0.0909, 0.184)
Lognormal ($\mu = 1, \sigma = 0.5$)	(0.138, 0.201)	(0.0940, 0.135)
Chisquare ($v = 10$)	(0.243, 0.130)	(0.196, 0.115)

observe that the error significantly decreases as the number of bins in histograms is increased. For 800 number of bins, most of the errors are below 0.1. The use of a memory-bounded data summarization in our framework naturally comes at a cost in accuracy. When the data distribution is highly skewed, practitioners can apply alternative R-partial sum approximate estimation based on the assumed distribution. This only changes the LAD/TLAD estimation component while keeping the training framework intact.

Table 4.4 – LAD/TLAD estimation error *w.r.t.* the maximum number of bins in histograms (two numbers in each cell respectively correspond to the mean absolute percentage errors of LAD and TLAD.)

#Bins / Distribution	600	800
Uniform ([0, 100])	(0.0000393, 0.122)	(0.0000392, 0.029)
Normal ($\mu = 0$, $\sigma = 1$)	(0.173, 0.403)	(0.0712,0.213)
Exponential($\mu = 0.5$)	(0.205,0.0761)	(0.143, 0.0678)
Beta ($a = 0.5$, $b = 0.5$)	(0.0000315, 0.0721)	(0.0000313, 0.0612)
Gama ($a = 3$, $b = 1$)	(0.0890, 0.114)	(0.0772, 0.0983)
Lognormal ($\mu = 1$, $\sigma = 0.5$)	(0.0862, 0.101)	(0.0723, 0.0975)
Chisquare ($\nu = 10$)	(0.138, 0.102)	(0.078, 0.083)

4.6 Discussion

Our current version of DR2-Tree focuses on the robust regression with categorical features. It can be smoothly extended to handle numeric or mixed features. For numeric features, besides the histograms built on the target values in current DR2-Tree, we can integrate additional histograms on the domains of numeric features [14, 101, 133] to form two-dimensional histogram based data summarization, such that these histograms respectively provide split candidates and error criterion estimation. Meanwhile, our data summarization has potential to support alternative robust error criterion such as Huber loss in the future. Lastly, note that our robust regression tree learner also supports binary (0-1) classification tasks by modeling them as instances of logistic regression.

4.7 Conclusion

Due to the advance in data collection techniques, the availability of massive volumes of data fuels the developments of many distributed machine learning algorithms. this information cannot be practically analyzed on a single commodity computer because the data is too large to fit in memory. On the other hand, such large datasets inevitably bear outliers, which could dramatically deteriorate the performance of some data analytic approaches. To this end, In this paper, we propose an efficient distributed robust regression tree for handling large and noisy data. Our DR2-Tree employs a novel data summarization technique, which is able to support both distributed information extraction and robust error criterion estimation for growing the regression tree. Extensive experiments reveal that: (1) Our proposed DR2-Tree is robust to datasets with various outlier percentages and magnitudes. (2) DR2-Tree exhibits comparable accuracy as the conventional distributed regression tree for relatively clean datasets with rare outliers. (3) DR2-Tree is much more efficient than exact robust regression and the dynamic

histogram based regression tree [14, 133]. Such results verifies the broad applicability of our DR2-Tree framework.

4.8 Appendixes

4.8.1 Proof of lemmas and theorems

Lemma 1. *For a bin $b = (l, r, c, s)$ of a histogram and an integer R ($R \leq b.c$), under the assumption of the uniform distribution of values in the bin, R-Partial-Sum of bin b can be approximated by* $S_p(b,R) \approxeq \hat{S}_p(b,R) = \begin{cases} b.s & : R = b.c \\ R \cdot b.l + R(R-1)\delta & : otherwise \end{cases}$, *where* $\delta = \frac{(b.s - b.r - b.c \cdot b.l + b.l)}{(b.c-2)(b.c-1)}$.

Proof. We assume that the values in a bin (e.g., b) are uniformly distributed in the value range $[b.l, b.r]$. Then, since the sum of values in the bin is given by $b.s$, we can obtain the sum of values between $b.l$ and $b.r$ is $b.s - b.r - b.l$. Meanwhile, the values in $(b.l, b.r)$ can be derived by adding incremental values over $b.l$. Therefore, the increments are considered as an arithmetic series whose elements are the products of the sequence term (e.g., $1, \ldots, b.c-1$) and the unit increment value. The sum of such increments is $(b.s - b.r - b.c \cdot b.l + b.l)$. The unit increment value is calculated as $\frac{(b.s-b.r-b.c \cdot b.l+b.l)}{(b.c-2)(b.c-1)/2}$, where $(b.c-2)(b.c-1)/2$ is the number of unit increments derived by the sum of the arithmetic series. As a result, the R-partial sum is approximated by the sum of the sequence of the first R increments plus the base value $b.l$, i.e., $R \cdot b.l + R(R-1) \cdot \frac{(b.s-b.r-b.c \cdot b.l+b.l)}{(b.c-2)(b.c-1)}$. □

Lemma 2. *Given a histogram $H = \{b_1, \ldots, b_\beta\}$, the LAD/TLAD over the data summarized by histogram H can be exactly computed by:*
(1) $L_l(H) = \sum_{b_i > b_m} b_i.s - \sum_{b_i < b_m} b_i.s + b_m.s - 2S_p(b_m, R)$, *where* $R = \lceil \frac{C}{2} \rceil - \sum_{b_i < b_m} b_i.c$, $C = count(H)$ *is the total number of data instances covered in histogram H, and an b_m is the $\frac{1}{2}$-quantile bin.*
(2) $L_l(H, \tau) = \sum_{b_m < b_i < b_{\overline{q}}} b_i.s - \sum_{b_q < b_i < b_m} b_i.s + S_p(b_q, R_1) - b_q.s + S_p(b_{\overline{q}}, R_2) + b_m.s - 2S_p(b_m, R)$, *where b_m, b_q and $b_{\overline{q}}$ are respectively the $\frac{1}{2}$, τ and $(1-\tau)$-quantile bins,* $R = \lceil \frac{C}{2} \rceil - \sum_{b_i < b_m} b_i.c$, $R_1 = C \cdot \tau - \sum_{b_i < b_q} b_i.c$, $R_2 = C \cdot (1-\tau) - \sum_{b_i < \overline{q}} b_i.c$.

Proof. Assume that the target values of data instances summarized by histogram H are given as a sorted list y_i and $C = count(H)$ is even. The median is $m = \frac{y_{C/2} + y_{C/2+1}}{2}$. The exact LAD over $\{y_i\}$ is computed as: $\sum_{i=1}^{C} |y_i - m| = \sum_{i=1}^{C/2} (m - y_i) + \sum_{i=C/2+1}^{C} (y_i - m) = \sum_{i=1}^{C/2} (-y_i) + \sum_{i=C/2+1}^{C} y_i$. If $\{y_i\}$ are grouped according to the bins in histogram H, we can obtain $\sum_{i=1}^{C/2} (-y_i) + \sum_{i=C/2+1}^{C} y_i =$

$\underbrace{\sum_{j=b_1}^{j=b_{m-1}} \sum_{y_i \in j} (-y_i)}_{(a)} + \underbrace{\sum_{j=b_{m+1}}^{j=b_\beta} \sum_{y_i \in j} y_i}_{(b)} + \underbrace{\sum_{y_i \in b_m \text{and } i \leq R} (-y_i)}_{(c)}$
$+ \underbrace{\sum_{y_i \in b_m \text{and } i > R} y_i}_{(d)}$. Item (a) and (b) in above formula can be exactly calculated by the sum element in the corresponding bin of the histogram. Item (c) and (d) can be computed through the R-partial-sum of the 0.5-quantile bin, namely, $-S_p(b_m, R)$ and $b_m.s - S_p(b_m, R)$. Above derivation also applies to the case that C is odd. The lemma is proved.

Likewise, as for TLAD assume that the target values of data instances summarized by histogram H are given as a sorted list $\{y_i\}$ and $C = count(H)$ is even. The exact Trimmed-LAD for k trimmed dataset ($k = \tau C$) is expressed as $\sum_{i=k+1}^{C/2} (-y_i) + \sum_{i=C/2+1}^{C-k} y_i$. By grouping the individual values into the corresponding bins in the histogram, we can obtain $\sum_{i=k+1}^{C/2} (-y_i) + \sum_{i=C/2+1}^{C-k} y_i =$
$\underbrace{\sum_{y_i \in b_q \text{and } i > R_1} (-y_i)}_{(a)} + \underbrace{\sum_{j=b_{q+1}}^{j=b_{m-1}} \sum_{y_i \in j} (-y_i)}_{(b)} + \underbrace{\sum_{j=b_{m+1}}^{j=b_{\overline{q}-1}} \sum_{y_i \in j} y_i}_{(c)} + \underbrace{\sum_{y_i \in b_m \text{and } i \leq R} (-y_i)}_{(d)} + \underbrace{\sum_{y_i \in b_m \text{and } i > R} (y_i)}_{(e)} +$
$\underbrace{\sum_{y_i \in b_{\overline{q}} \text{and } i \leq R_2} (y_i)}_{(f)}$. Then, item (a), (d), (e) and (f) can be respectively derived by R-partial sums of the corresponding bins and (b) and (c) can be exactly calculated in the histogram. □

Theorem 1. *Given a bin $b = (l, r, c, s)$ of a histogram and R ($R \leq b.c$), if $R = 1$ or $R = b.c$, $\hat{S}_p(b, R)$ provided in Lemma 4.4.1 is the exact R-Partial-Sum. Otherwise, the approximation error of R-Partial-Sum of bin b, $S_p(b, R) - \hat{S}_p(b, R)$ is bounded within $[(R - b.c) \cdot (b.r - b.l), b.s - b.c \cdot b.l]$.*

Proof. We first derive the bounds of the exact R-partial sum and the approximate one. Based on Lemma 4.4.1, when $R = b.c$, the approximate partial sum is equal to the exact value. For the case of $R < b.c$, $\hat{S}_p(b, R) = R \cdot b.l + R(R-1)\delta$, where $\delta = \frac{(b.s - b.r - b.c \cdot b.l + b.l)}{(b.c-2)(b.c-1)}$. Therefore, we can derive that $R \cdot b.l \leq \hat{S}_p(b, R) \leq R \cdot b.l + b.s - b.c \cdot b.l$. For the exact value $S_p(b, R)$, we have the first bound below: $b.s - S_p(b, R) \leq (b.c - R) \cdot b.r$. It holds because the sum of the remaining values, i.e. $b.s - S_p(b, R)$ is maximized when all the remaining values locate on the right boundary of the bin $b.r$. As a result, we can obtain $b.s - (b.c - R) \cdot b.r \leq S_p(b, R)$. Similarly, the upper bound is derived by $S_p(b, R) \leq b.s - (b.c - R) \cdot b.l$. Now, merging the bounds of $\hat{S}_p(b, R)$ and $S_p(b, R)$ leads to the theorem. □

5 Robust Online Time Series Prediction with Recurrent Neural Networks

Have we vanquished an enemy? None but ourselves.
—George Mallory

5.1 Introduction

Time series forecasting [10, 25, 94] is an important task in machine learning with a variety of applications, such as cyber-network traffic prediction, web user access estimation, and pedestrian flow prediction. Forecasting a time series is useful in e.g. reliability monitoring, predictive maintenance and intrusion detection, and can effectively improve availability, security, and the overall service experience. On the other hand, with the rapid growth of various tracking and monitoring technologies, time series data in many real-world systems arrives in a streaming fashion. Online learning methods are expected to extract the underlying patterns from the observed time series for real-time learning and prediction.

Due to its practical importance and technical challenges, some online forecasting methods have been investigated in the literature, including linear methods [8, 85], ensemble methods [93], kernel based methods [109], and Gaussian processes [144]. Recurrent neural networks (RNNs), a class of deep learning methods particularly designed to model sequential data, currently receive increasing attention due to the capacity on learning insightful representations of sequences [51, 77, 129], and have been successfully applied to time series forecasting [20, 92, 142]. However in real-world applications, the time series is usually contaminated by anomalies or outliers that are abrupt observations deviating from the behaviour of the majority. A typical example might be cyber-attacks, which are often shown as anomalies in time series monitoring some measurements of network traffic. In addition, the underlying patterns of time series generally keep changing over time. For example, a newly released online service or functionality of a website can change the probabilistic distribution of time series of user access traffic in terms of mean, variance, correlation and period. With these challenges, the learning methods are expected to capture the true pattern and trend of time series, namely being robust against outliers, and enabling to quickly fit new patterns with potential change points.

In this paper, we present an adaptive gradient learning method for long short-term memory (LSTM) recurrent networks [50, 51, 60], to make the streaming time series forecasting robust to outliers and change points. We model time series with LSTM networks, and use a stochastic gradient descent (SGD) based method to learn the model from the streaming time series. As novel observations arrive, the model parameters are updated in an online mode according to the gradients of the loss of the newly available data. With the standard SGD method, outlier observations will make the updated model deviate from the normal patterns and produce oscillated incorrect predictions until the adverse effect is gradually corrected by the following normal observations. To solve the problem, we explore the local features of time series to weight the gradients of the learning method with distributional properties of the local data. In particular, if a newly available observation is a potential outlier behaving differently from the regular patterns, the corresponding gradient will be downweighted, so as to avoid leading the online model to abruptly drift from the underlying normal patterns. Notice that the newly observed unusual point can also be a change point. Although it behaves differently from the majority like an anomaly, it indicates emerging patterns of time series required to learn quickly. In this case, the corresponding gradient will be retained by a high weight and lead the model to fit the new data in real time. Technically we introduce a weight function based on distributional characteristics of the local data to adapt the gradients of the online SGD to the complicated streaming time series automatically. The weight function is composed of two components: suspicion ratio and difference value drift formulating the statistical properties of the local data, which make the learning procedure robust to outliers and change points. The experimental analysis on synthetic and real datasets demonstrates the performance of the proposed learning method for streaming time series forecasting in the presence of anomalies

and change points.

The rest of the paper is organized as follows. Section 5.2 presents related work, while Section 5.3 defines the problem to be solved and introduces the notations. In Section 5.4, we present the proposed robust online learning method for RNN. Section 5.5 demonstrates the performance of our method on synthetic and real datasets. Finally, the paper is concluded in Section 5.6.

5.2 Related Work

5.2.1 Modeling Time Series in the Presence of Outliers and Change Points

Many of the existing machine learning models for time series analysis focus exclusively on either change point or outlier detection. Our paper aims to provide a recurrent neural network based approach to handle both change points and outliers simultaneously for online learning of time series.

[6] introduced the Bayesian change point detection framework and [113] extended it by using the Gaussian process as the underlying predictive model. In [23] a Gaussian process based nonparametric time series prediction model was proposed specifically for periodic time series. [87] detected changes by utilizing relatively density ratio estimation in a batch mode, i.e., they require the entire time series to be present. [146] employed an online discounting learning algorithm to incrementally learn mixture and regression models. [76, 134] put forward scalable anomaly/outlier detection frameworks integrating various predictive models. [47] and [58] provided detailed surveys on change and outlier detection.

Another line of research is to build robust models over noisy and non-stationary time series. [48] proposed a new sequential algorithm for making robust predictions based on Gaussian Process in the presence of change points. [9] extended traditional autoregressive moving average (ARMA) models to the case that less strict assumptions are on the noise term for the purpose of online time-series modeling. [110] developed time-dependent loss function to include the information about the distribution change in time series directly in the SVM learning process.

5.2.2 Recurrent Neural Networks for Time Series Analysis

Recurrent neural networks have recently shown promising results in a variety of applications, especially when there exist sequential dependencies in data [31, 89, 130]. [33] proposed a robust learning algorithm for recurrent neural networks on time series. This algorithm is based on filtering outliers from the data first and then training the neural network on the filtered data. Such a filtering-then-learning scheme focuses on dealing with outliers and is not suitable for

online training and prediction.

Long short-term memory (LSTM) [60, 89, 141], a class of recurrent neural networks with sophisticated recurrent hidden and gated units, are particularly successful and popular due to its ability to learn hidden long-term sequential dependencies and to allow online training of sequence data of indefinite duration. Some recent work [26, 84, 91] explored the performance of LSTM in time series modeling. [84] used LSTMs to recognize patterns in multivariate time series, especially multi-label classification of diagnoses. [26, 91] evaluated the ability of LSTMs to detect anomalies in ECG time series. In this paper we present an LSTM online learning method for time series forecasting in presence of both outliers and change points.

5.3 Problem Formulation

In this section, we discuss the properties and challenges of time series containing outliers and change points, and formulate the problem to be resolved.

Time series, a sequence of data points consisting of successive measurements made over time, often arises when monitoring dynamic processes in a variety of applications, e.g., Internet of Things, sensor networks, and mobile computing. We denote a univariate time series by $\{x_1, \ldots, x_T\}$, where the subscript represents the time instant and each data point x_t is a real value. In this paper, we focus on univariate time series. The proposed method can be naturally generalized to multivariate time series as well.

Since many time series are generated from observations and measurements of physical entities and events, the data is inevitably anomaly-contaminated and non-stationary in the sense that it contains both outliers and change points. An *outlier* in time series is a data point, which is significantly different from the behaviour of the majority of the time-series. A *change point* indicates that the behaviour and hidden data distribution of the time-series are significantly different before and after this point. For instance, a time series might undergo a sudden shift in its mean at a certain time. The major difference between change points and outliers is that change points correspond to more sustained, long-term changes in time series compared to volatile and abrupt outliers. The presence of outliers and change points can adversely affect the time series analysis and complicate the learning process [47, 58, 76, 134].

In many cases, time series are continuously observed, and often need to be analyzed in real time. The online learning methods are thus required, which learn and update models incrementally as new data arrives, in order to better capture the timely trend and the underlying patterns in the time series [9, 47, 63].

In this paper we focus on online time series prediction in the presence of both change points and outliers. The work aims to incrementally learn the time series and provide robust predic-

tions in real time. Given the observed time series of length $t-1$, the predicted new data point $\hat{x}_t$ at the next time t can be computed as:

$$\hat{x}_t = g(x_{t-1}, x_{t-2}, \ldots, x_1; \theta^*_{t-1}) \tag{5.1}$$

$g(\cdot)$ is a continuous function which captures the dynamic patterns of the time series. In this paper $g(\cdot)$ is modeled according to the recurrent neural network that can approximate any function with sufficient hidden layers and nonlinear units to arbitrary precision. θ^*_{t-1} denotes the parameters of the recurrent network learned with the $t-1$ sequential observations.

As the new data point x_t arrives, we update the model by minimizing the loss function, e.g., the squared error:

$$\theta^*_t = \arg\max_{\theta_t}(x_t - g(x_{t-1}, x_{t-2}, \ldots, x_1; \theta_t))^2 \tag{5.2}$$

Recursivly, the new parameters θ^*_t will be used to predict the next data $\hat{x}_{t+1}$. The online learning process is to continuously minimize the loss function over the newly available data.

We notice that, in the learning process, an outlier at time t can lead the model to deviate from the normal patterns. Given the current model θ^*_{t-1}, the loss of the outlier will be high. For reducing the loss (5.2), θ^*_t will deviate from the normal ones to best fit the outlier, and produces oscillated predictions untile the adverse effect of the outlier is gradually corrected by the following observations [33, 58, 146]. On the other hand, since a change point often leads to a shift in data distribution and makes the current model unfit to the new data, high loss is also expected [48, 76]. In this case, the high loss will take positive effect and lead the model to quickly fit to the new patterns.

To meet these challenges, the online learning method is supposed to react to outliers and change points differently. On the first hand, the online learning model should be robust to outliers by mitigating the model deviation caused by outliers. On the other hand, it should be able to promptly adapt to the new changed data by updating the model using the latest data, so as to provide timely and precise predictions.

5.4 Weighted Gradient Online Learning for LSTM Neural Networks

In this section, we present the proposed online learning method for Long Short Term Memory (LSTM) neural networks, a widely used variant of Recurrent Neural Networks (RNN). The method is capable of performing robust predictions over time series in the presence of both outliers and change points.

Table 5.1 – Notations

Symbols	Meaning
x_t	the data point at time t of the time series
$\hat{x}_t$	the predicted value of x_t by the neural network
p_t	p-value of x_t
α	significance level
w	the size of the sliding window for feature extraction
h_t	the output of a LSTM layer at time t
o_t	the output gate of a LSTM layer at time t
c_t	the memory unit of a LSTM layer at time t
f_t^j	the forget gate of a LSTM neuron j at time t
i_t^j	the input gate of a LSTM neuron j at time t
W_t	the set of parameters of all layers in the neural network at time t
β_t	gradient weight for updating the model at time t
d_t	difference value drift
s_t	suspicious ratio

5.4.1 Online learning of time series

In the online process at each time instant, e.g. t, a desirable model should be able to automatically retain useful past information in $\{x_1, \ldots, x_{t-1}\}$, discover dependences for making prediction $\hat{x}_t$ and then update itself using x_t. Recurrent neural networks (RNNs) are rising as a powerful tool to model sequence data [51, 63]. They are suitable for dealing with sequential problems due to the internal hidden states and short term memory realized by recurrent feedback connections.

Among the variants of RNNs, it is well established that the Long Short-Term Memory (LSTM) based networks work well on sequence-based tasks with long-term dependencies [60, 63] by using specialized gates and memory cells in the neuron structure. The LSTM achieves state-of-the-art results for problems spanning natural language processing, image captioning, handwriting recognition, and genomic analysis [63, 84, 86, 130].

We present the formal definition of a neuron of a LSTM layer as follows [31, 60]. Each j-th neuron in the LSTM layer maintains a memory c_t^j at time t. The output h_t^j or the activation of this neuron is then expressed as:

$$h_t^j = o_t^j tanh(c_t^j) \tag{5.3}$$

where o_t^j is an output gate that modulates the amount of memory content exposure. The

output gate is calculated by

$$o_t^j = \sigma(W_o x_t + U_o h_{t-1} + V_o c_t)^j \tag{5.4}$$

where σ is a logistic sigmoid function. h_{t-1} and c_t are respectively the vectorization of h_{t-1}^j and c_t^j, i.e. $h_{t-1} = \{h_{t-1}^j\}$ and $c_t = \{c_t^j\}$. V_o is a diagonal matrix. Then, the memory cell c_t^j is updated through partially forgetting the existing memory and adding a new memory content $\tilde{c}_t^j$:

$$c_t^j = f_t^j c_{t-1}^j + i_t^j \tilde{c}_t^j \tag{5.5}$$

where the new memory content at time t is expressed as:

$$\tilde{c}_t^j = tanh(W_c x_t + U_c h_{t-1})^j \tag{5.6}$$

The extent to which the existing memory (i.e., at time $t-1$) is forgotten is modulated by a forget gate f_t^j, and the degree to which the new memory content (i.e., at time t) is added to the memory cell is modulated by an input gate i_t^j. Then, such gates are computed by

$$f_t^j = \sigma(W_f x_t + U_f h_{t-1} + V_f c_{t-1})^j \tag{5.7}$$

$$i_t^j = \sigma(W_i x_t + U_i h_{t-1} + V_i c_{t-1})^j \tag{5.8}$$

Note that V_f and V_i are diagonal matrices.

Unlike the traditional recurrent neuron which overwrites its content at each time instant [31, 89], a LSTM neuron is able to decide whether to keep the existing memory via the introduced gates. Intuitively, if it detects an important feature from an input sequence at early stage, it easily carries this information (the existence of the feature) over a long period, thereby capturing the hidden long-term dependencies [60, 89].

In our model for online learning of time series, a dense layer is built to connect the LSTM layer so as to map the outputs of LSTM neurons to the target prediction. Denote by W_d and b_d the weights and biases of the dense layer. The output of the neural network is formulated as:

$$y = g(W_d h_t + b_d) \tag{5.9}$$

where $g(\cdot)$ is the activation function of the dense layer, e.g., $tanh$, $sigmoid$, $linear$, etc. h_t is the vector of the outputs of LSTM neurons, i.e., $h_t = \{h_t^j\}$. Online learning of a neural network is to continuously update the set of parameters in each layer, i.e., $W_t = \langle W_o, U_o, V_o, W_c, U_c, W_f, U_f, W_i, U_i, W_d, b_d \rangle$ at each time instant to minimize a loss function. Then, the prediction at time t

of the time series by the LSTM neural network can be expressed as:

$$\hat{x}_t = g(x_{t-1}, \ldots, x_1; W_{t-1}) \tag{5.10}$$

LSTM neural network employs gradient decent based iterative approaches to update the network parameters at each time instant [60, 89, 141]. The general update process can be expressed as:

$$W_t = W_{t-1} - \eta \bigtriangledown E_t(W_{t-1}) \tag{5.11}$$

where W_t and W_{t-1} represent the set of parameters W at different time instants. η is the learning rate, $E_t(W_{t-1})$ is the loss function, i.e. squared-error function, and $\bigtriangledown E_t(W_{t-1})$ is the gradient of the loss function at W_{t-1}. There are many variants of gradient descent based methods for updating the network [70, 120, 153],

Since online learning aims to continuously update the model using new arrived data, the loss function is specified as the squared-error of one training instance without accumulating the errors [63, 141]:

$$E_t(W_{t-1}) = e_t^2 = (x_t - \hat{x}_t)^2 \tag{5.12}$$

Our proposed framework below could be extended to situations where data comes on mini-batches of data points as well.

Then, the gradient is derived as:

$$\bigtriangledown E_t(W_{t-1}) = -2(x_t - \hat{x}_t) \bigtriangledown g(x_{t-1}, \ldots, x_1; W_{t-1}) \tag{5.13}$$

where the detailed derivation of $\bigtriangledown g(x_{t-1}, \ldots, x_1; W_{t-1})$ is referred to [60, 141]. Such a gradient descent scheme to incrementally update parameters enables the LSTM neural network to naturally evolve with time series in the error-driven way and to adapt to changing time series without explicit detection of change points [47].

5.4.2 Effect of Outliers and Change Points on LSTM neural networks

In this part, we first define two basic concepts. Based on statistical time series modeling theory [10, 26, 33, 140], the set of errors $\{e_t\}$ can be fitted to a parametric distribution, referred to as the error reference distribution and maintained online by moving average style methods [13, 23, 140]. During our experiments, we observe that it is reasonable to fit the errors to a Gaussian distribution, though alternative distributions can be chosen based on applications. The associated mean and variance are respectively denoted by μ_t and v_t.

Definition 5.4.1 (p-value)**.** *The p-value of data point x_t, denoted by p_t, is defined as the probability of obtaining a value less or equal to e_t under the error reference distribution $\mathcal{N}(\mu_t, v_t)$, where $e_t = x_t - \hat{x}_t$. Formally, $p_t = \Phi(\frac{e_t - \mu_t}{\sqrt{v_t}})$, where $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution.*

For the predictive model based time series analysis [23, 26, 33, 58], p-value is a fundamental indicator for outliers or changes. Then, we further define:

Definition 5.4.2 (Suspicious and Normal Points)**.** *In the online learning process of LSTM neural network over time series, if the p-value of a data point, e.g., x_t, is lower than α or higher than $1-\alpha$ ($\alpha \in (0,1)$), x_t is a suspicious data point indicating that it could be an outlier or a change point. Otherwise, we call x_t as a normal point. α is the confidence level.*

Straightforward solutions help little:

Given the neural network learnt over time series, outliers could lead to dramatical large prediction errors. Consequently using such outliers to update the neural network could drag away the gradient from the previous direction and derail the so-far trained neural network, thereby incurring unreliable prediction [33].

For instance, we use an example experiment to illustrate the effect of outliers on the LSTM neural network. We perform the conventional online learning of LSTM neural network [141] on a real time series from Yahoo S5 dataset[1]. We adopt the interleaved test-then-train framework [47], which means that at each time instant a prediction for the next data point is performed using the current neural network and then the network is updated when the next data point arrives. The experiment is run at the learning rate (0.005), which achieves the least prediction error (i.e., RMSE). For the details of the experiment setup, refer to Section 5.5.

We can observe in Figure 5.1(*b*) that outliers incur oscillated prediction around them. One possible solution might be to skip the neural network update [33] when a new coming data point is considered as a suspicious point by Definition 5.4.2. However, such an approach would not necessarily help for the reasons below.

Recall that we focus on modeling time series with both outliers and change points. Both an outlier and a change point could lead to the identification of suspicious points. For instance, Figure 5.2 shows the p-values at each time instant during the learning process of LSTM neural network in the experiment as Figure 5.1. It is observed that outliers, change points and some regular points are flagged as suspicious points. It is difficult to distinguish them from each other in such a noisy and non-stationary environment [13, 47].

[1] https://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70

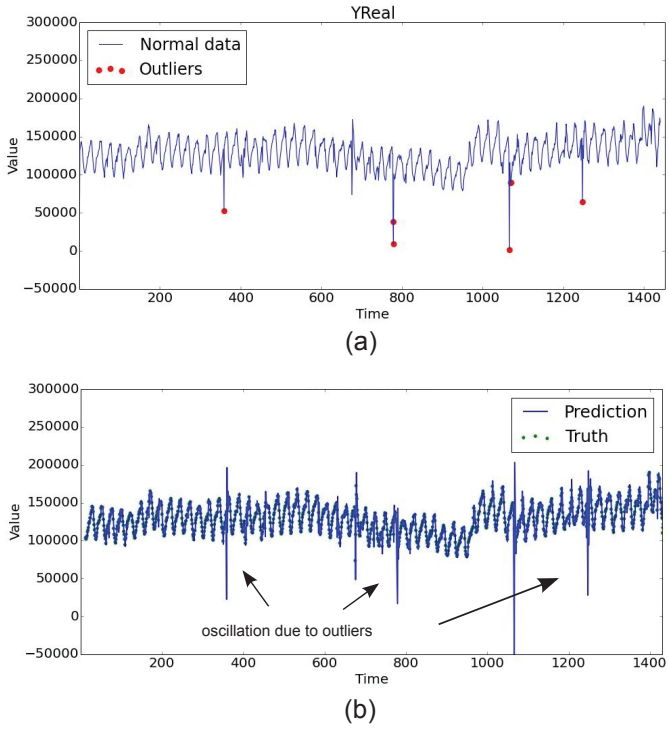


Figure 5.1 – Effect of outliers on the LSTM neural network. (a) the original time series (b) online prediction results. (best viewed in colour)

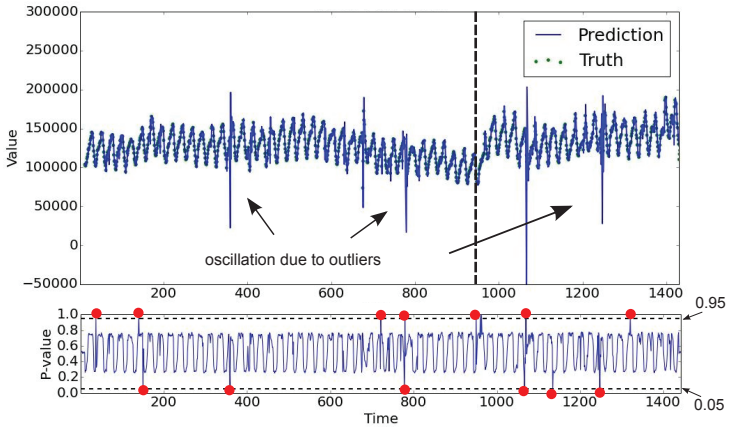


Figure 5.2 – The observed time series (top) and p-values of data points in the time series modeled by conventional online learning of LSTM neural network (bottom) (best viewed in colour). The vertical dotted line in the top figure indicates the possible change point. Red points in the bottom figure represent the time instants identified as suspicious points.

Meanwhile, when a real change point is misclassified as an outlier and removed from the neural network update, the data points immediately following the change point would then likely be regarded as outliers as well, since they follow the new distribution after the change point [13, 17]. Simply skipping such data points results in important information being lost to the LSTM updating. It could delay the learning towards the new data distribution after the change point and significantly degrade the performance of LSTM neural network afterwards.

Integrating change point detection into LSTM will hardly help as well. Most of change point detection techniques are the instances of the following framework: at time instant t statistical tests [13, 17, 47, 87] are performed over two data subsets, the intermediate past data points $\{x_i\}_{i=t-m_1,\ldots,t-1}$ and the intermediate future data points $\{x_k\}_{k=t+1,\ldots,t+m_2}$ to measure the dissimilarity, so as to determine whether t is a change point. Therefore, it requires to accumulate data to locate a change point in the history. In this paper, we consider the setting where the learning process of LSTM neural network is supposed to response to individual data points online, so as to adapt to data changes promptly and provide timely prediction.

5.4.3 Weighted Gradient Online learning of LSTM for time series

In this part, we present a novel online learning algorithm for LSTM neural network to be capable of making robust prediction over time series in the presence of both outliers and change points.

Overview: We propose the **w**eighted **g**radient online learning algorithm, referred to as WG-Learning. The key idea is to leverage local features, which are extracted from a sliding window over time series, to dynamically weight the gradient, i.e. Eq. (5.13), at each time instant for updating the current neural network. Intuitively, if a suspicious point is highly possible to be an outlier based on local features, the corresponding gradient is down-weighted, so as to avoid deviating the so-far trained LSTM neural network from the current underlying data distribution. Otherwise, if it is believed to be a change point, the associated gradient is retained by a high weight, such that the LSTM model can quickly adapt to the new data.

A common characteristic of time series analysis is that temporal continuity plays a key role [76, 110, 134, 140]. In this sense, time forms the contextual variable with respect to which all analysis is performed. Therefore, our WG-Learning maintains a sliding window over the time series and associated p-values for the data points in the sliding window.

As is shown in Figure 5.3 which outlines the work-flow of our WG-Learning, if the data point at the current time instant is a suspicious point based on Definition 5.4.2, two features presented below are extracted from the sliding window and took into account to weight the gradient used in updating the LSTM neural network at this time instant. For instance, at time t, the local fea-

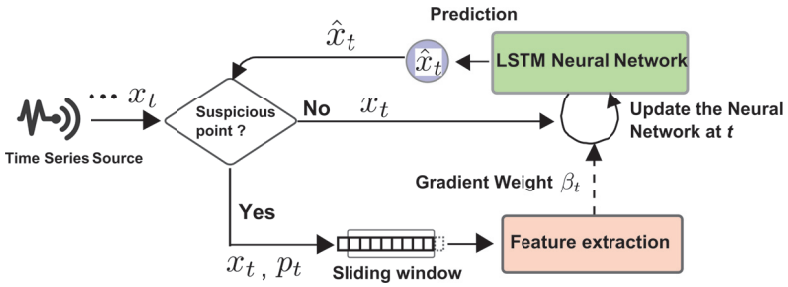


Figure 5.3 – Work-flow of WG-Learning based LSTM neural network in online learning of time series

tures *w.r.t.* data point x_t are discovered in $\{x_{t-w}, x_{t-w+1}, \ldots, x_{t-1}\}$ and $\{p_{t-w}, p_{t-w+1}, \ldots, p_{t-1}\}$. w is the length of the sliding window (for the extension of adaptive windows, please refer to Section 5.4.4).

Then, in WG-Learning formally the neural network is updated by the following formula:

$$W_t = W_{t-1} - \eta \cdot \{\beta_t \bigtriangledown E(W_{t-1})\} \tag{5.14}$$

where β_t is the time-varying gradient weight based on the features extracted from data, which will be defined below. Previous adaptive learning rate approaches [120, 153] fail to take into account the data characteristics in updating parameters of the neural network. Before presenting the details, we first summarize the benefits of our approach for learning noisy and non-stationary time series as follows:

- Reduced impact of outliers as a sudden increase of the prediction error - our WG-Learning provides reliable and smooth prediction robust to outliers as well as avoiding volatile huge prediction errors.
- Adaptability to changing data - the dynamic gradient weighting mechanism in WG-Learning enables to retain useful information in updating the neural network for converging to changed data.
- Reduction of the overall prediction errors.

Next, we will first present the features and then how to leverage them to weight the gradient.

Suspicion Ratio.
The suspicion ratio is formulated based on the following observation: outliers are rare events in time series [58, 140, 146] and would lead to the limited appearance of suspicious points in the sliding window; however, when change point happens, the neural network needs to

consume some data after the change point for adapting to the new hidden distribution and therefore may yield high proportion of suspicious points in the beginning. Therefore, the ratio of suspicious points in the sliding window can be a factor in weighting the gradient.

Now we give the formal definition. In the sliding window *w.r.t.* time t, $\{x_{t-w}, x_{t-w+1}, \ldots, x_{t-1}\}$, the suspicion ratio is defined as:

$$s_t = \frac{\sum_{i=t-1}^{t-w} \mathbb{I}_{\{p_i > 1-\alpha \text{ or } p_i < \alpha\}}}{w} \tag{5.15}$$

where $\mathbb{I}_{\{\cdot\}}$ is the indicator function.

Suspicion ratio is positively correlated with the gradient weight. On the first hand, the less the suspicion ratio in the sliding window, the more likely the suspicious point x_t is to be an outlier. Even if it is a change point in reality, the subsequent data points after this change point are highly possible to be suspicious points and thus lead to increased gradient weight in updating the model. It will not significantly delay the convergence of our neural network on changed data.

On the other hand, when the suspicion ratio is high, i.e., the suspicious points appear frequently in the sliding window, further investigation is required. This is because it is possible that some outliers just occur in the sliding window or such suspicious points are due to a real change point. In such a case, additional features should be considered for weighting the gradient weight.

Difference Value Drift.
In this part, we introduce the second feature, difference value drift. This novel feature helps to "amplify" the difference between outliers and change points from the perspective of the value domain of time series, thereby providing enhanced information to adjust the gradient weight.

First, we extract the normal and suspicious points from the sliding window at time t, i.e, $\{x_{t-w}, \ldots, x_{t-1}\}$ and respectively denote them by a set $\mathcal{N}_t = \{x_k | k \in \{t-w, \ldots, t-1\}, \alpha \leq p_k \leq 1-\alpha\}$ and a set $\mathcal{S}_t = \{x_k | k \in \{t-w, \ldots, t-1\}, p_k > 1-\alpha \text{ or } p_k < \alpha\}$. $|\mathcal{N}_t|$ and $|\mathcal{S}_t|$ represent the numbers of data points in the sets.

Then, we define the neighbours of a normal point in the sliding window as:

Definition 5.4.3 (Neighbours of a Normal Point)**.** *The left and right neighbours of a normal point x_i are the most recent two normal data points before and after x_i in the sliding window: $f_l(x_i) = x_{\max\{k:k<i, x_k \in \mathcal{N}_t\}}$ and $f_r(x_i) = x_{\min\{k:k>i, x_k \in \mathcal{N}_t\}}$.*

Now we are ready to introduce the proposed *difference value*, which is novel in the sense that

it is derived by a bi-directional differencing process and has two forms respectively specific for normal and suspicious points as follows:

- For a suspicious point (e.g., x_k) in the sliding window, the difference value is the average of absolute changes between x_t and its intermediate preceding and succeeding data points, i.e. x_{k-1} and x_{k+1}, no matter whether they are normal points or not. (if any of the two points are out of the sliding window, only one-side change is taken as the difference value).

- For a normal points (e.g., x_i) in the sliding window, the difference value is evaluated as the average difference between x_i and its two neighbours, which are defined in Definition 5.4.3, such that it captures the normal pattern of time series without the effect of potential outliers and change points.

The motivation behind the difference value is as follows. Differencing is an important tool in time series analysis [140], which helps to stabilize the mean of a time series by removing changes in the level of a time series. On the first hand, outliers are often abrupt and violate changes *w.r.t.* the intermediate preceding and succeeding normal points, thereby yielding high difference values [58, 110]. Even if outliers occur consecutively, their difference values could present abnormal fluctuating patterns compared with the normal data points. On the other hand, for the suspicious points resulting from a change point, they are under the new data distribution and expected to adhere to the property of normal time series as well, i.e., having reasonable magnitudes in difference values.

Based on above analysis, we give the formal definition of the difference value drift. At time instant t, *difference value drift* d_t between suspicious and normal points in the sliding window is defined as:

$$\frac{(x_t - x_{t-1}) + \sum\limits_{x_k \in \mathscr{S}_t} (|x_k - x_{k-1}| + |x_k - x_{k+1}|)}{\sum\limits_{x_i \in \mathscr{N}_t} (|x_i - f_l(x_i)| + |x_i - f_r(x_i)|)} \frac{|\mathscr{N}_t|}{|\mathscr{S}_t| + 1} \tag{5.16}$$

This is a relative metric to evaluate the discrepancy between the average difference values of normal and suspicious points in the sliding window. The higher the value of the difference value drift, the more probable the suspicious points are to be outliers.

The proposed two features above are complementary by providing different views to weight the gradient in updating the neural network. Now we can define the gradient weight β_t in

(5.14) integrating the suspicion ratio and the difference value shift as:

$$\beta_t = \lambda \cdot e^{-d_t \mathbb{I}_{\{d_t \geq \gamma\}}} + (1-\lambda) \cdot s_t \tag{5.17}$$

where $\lambda \in [0,1]$ is the parameter to control the importance of the two features in the weight. The term $e^{-d_t \mathbb{I}_{\{d_t \geq \gamma\}}}$ models the difference value drift in a bounded range $[0,1]$. Parameter γ represents the prior belief about the normal difference value drift. $e^{-d_t \mathbb{I}_{\{d_t \geq \gamma\}}}$ gives full weight, i.e., 1 when d_t is below γ. Otherwise, the more the value of d_t, the more probable it is that outliers occur and thus the decreasing weight is given to the gradient, as d_t increases.

As for online prediction, a particular feature of time series modeling is that a target data point at a certain time instant will become the predictor of future data points. For instance, at time t, predictors $(x_{t-1}, x_{t-2}, \ldots, x_1)$ are used to make prediction on t and update the model together with x_t. Then, x_t will be used to predict values on time $t+1, \ldots$ and so on. The problem with such a procedure is that if x_t is an outlier and thus flagged as a suspicious point, the prediction based on x_t is biased, though the model is maintained properly in WG-Learning.

Therefore, we propose to use the forecasting from the learnt neural network to replace suspicious points for the subsequent prediction based on the idea of sequence modeling with RNN [52]. Formally, the prediction process is now expressed as:

$$\hat{x}_t = g(\tilde{x}_{t-1}, \tilde{x}_{t-2}, \ldots, \tilde{x}_1; W_{t-1}) \tag{5.18}$$

where $\forall k \in \{1, \ldots, t-1\}$,

$$\tilde{x}_k = \begin{cases} \hat{x}_k \text{ as (5.10)} & x_k \text{ is a suspicous point} \\ x_k & \text{otherwise} \end{cases}$$

5.4.4 Discussion

Our proposed WG-Learning can be extended to perform outlier and change point detection with ease. Since our WG-Learning is able to provide robust prediction, we can integrate additional sub-windows into the sliding window to model the error distributions of normal behaviours in time series [13]. Then, the statistics extracted from sub-windows is leveraged to perform various statistical test [13, 17, 47] to locate the occurrences of change points. For the suspicious points not identified as change points, they are reported as outliers.

As for the size of the sliding window, we can adapt time-varying windowing techniques [17, 47] to our WG-Learning, so as to dynamically adjust the window size sufficient for summarizing the error distribution of the current neural network.

5.5 Experimental Analysis

In this section, we conduct extensive experiments to demonstrate the prediction performance of the proposed method WG-Learning.

5.5.1 Experiment Setup

Datasets: we use both synthetic and real datasets for the evaluation.

Synthetic: this synthetic dataset is produced based on a time series generator [6]. It generates segments of time series by randomly sampling the mean μ_i, variance v_i, trend slop u_i and length l_i specific for each segment i from the prior, $\mu_i \in [0,100]$, $v_i \in [10,30]$, $u_i \in [-0.5,0.5]$ and length $l_i \in [500,1000]$. The data points in each segment are then the samples from the Gaussian distribution with the corresponding mean and variance plus the trend component. The number of points in each segment is the sampled length. In this way, we obtain time series with change points, which are the boundaries between segments. Outliers are injected into each segment based on a Bernoulli distribution specified by β and thus $l_i \cdot \beta$ is the expected number of outliers in the segment of time series. β is chosen as 0.01. The magnitude of outliers is defined as the times of the variance of the segment. By default, the magnitude is 10, which means that the value of an outlier data point is sampled from a Gaussian distribution with 10 times larger mean than the mean of the corresponding segment.

YSyn: the synthetic time series is from Yahoo's S5 dataset[2]. The dataset consists of both real and synthetic time-series. We use the time series in A4Benchmark containing synthetic outliers and change-points. The synthetic time-series have varying noise and trends with pre-specified seasonalities.

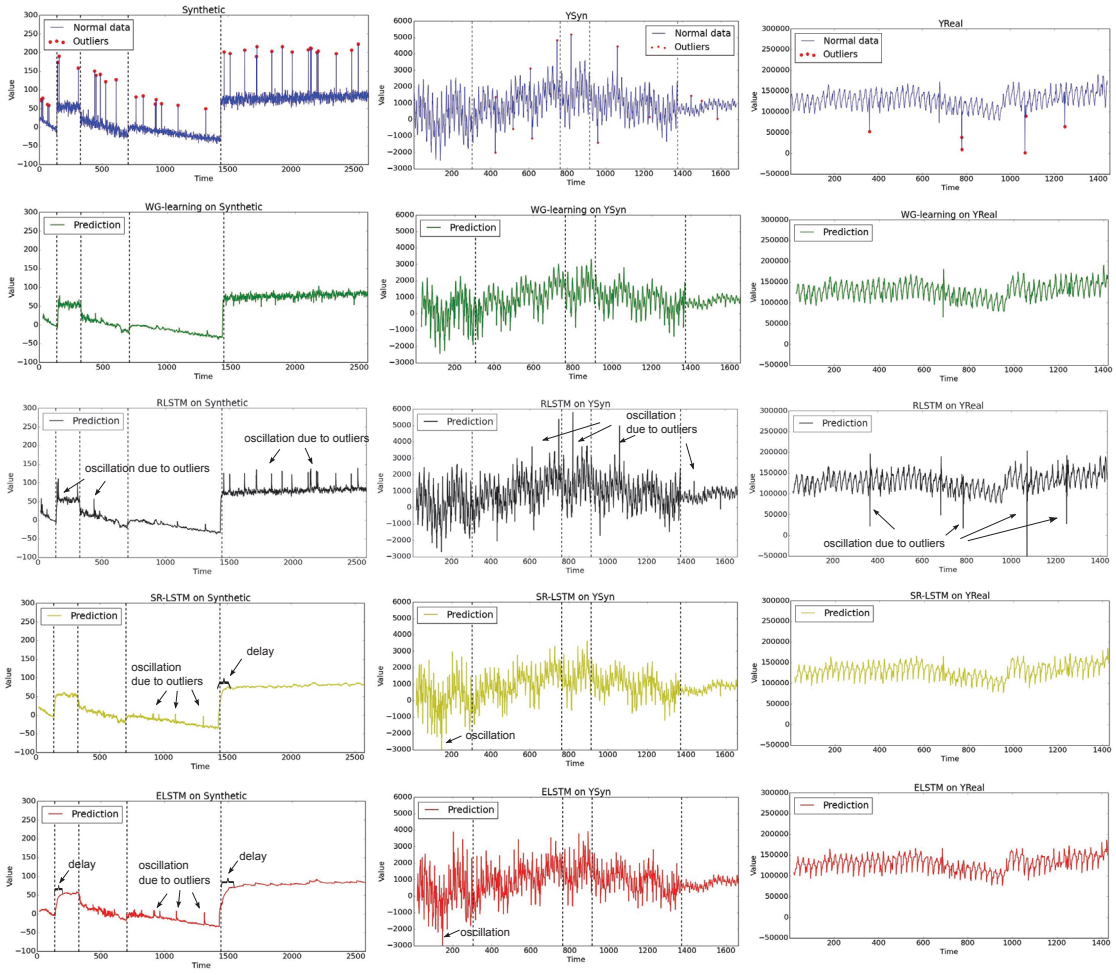
YReal: the real time series is from the A1Benchmark data in Yahoo's S5 dataset. The time series representing the metrics of various Yahoo services contains both outliers and change points.

Baselines: We compare the proposed method with three baselines.

RLSTM: this refers to the conventional **r**eal-time current learning (RTRL) of LSTM neural networks [141]. It updates the network directly using the newly available data without considering the effect of outliers and change points.

SR-LSTM: it stands for the online learning of LSTM with **s**uspicious point **r**emoval based on the idea of [33]. Specifically, once a suspicious point is detected, SR-LSTM skips the model update on this point. This method may circumvent the adverse effect of outliers, but could lead to loss of useful information for changing patterns.

[2] https://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70



(a) Synthetic dataset (b) YSyn dataset (c) YReal dataset

Figure 5.4 – Online one-step ahead prediction (best viewed in colour). The figures in each column respectively visualize the observed time series, and the predicted ones with the baselines and the WG-Learning. The proposed method provides a smooth prediction resistant to outliers and change points.

ELSTM: this baseline is based on [23] to integrate both outlier and change point detection into LSTM. It uses the statistical control chart (e.g., EWMA) [13, 23] to monitor the distribution of prediction errors. Based on the monitored statistics, for a suspicious outlier, the model is updated by using a recent normal point, while it is trained as usual when the point is identified as a change point.

SR-LSTM and ELSTM have the confidence level parameter in common with WG-Learning, and they are all set to 0.05 as usual in typical statistical tests [13, 76]. Additionally, ELSTM has a parameter, the moving average weight with the usual recommendation range [0.1, 0.3] [13]. In the experiments below, we set it to 0.2. The size of sliding window in WG-Learning is set as 20. λ and γ are respectively set to 0.8 and 5

We use the same LSTM model for the baselines and the proposed method in the experiments. In particular, the number of layers is 3, and the number of nodes (neurons in LSTM layer) is 400. The mean squared error is chosen as the loss function. The L2 regularization with 0.0001 penalty is used. Nesterov momentum is adopted as the optimizer. Our WG-Learning and baselines are implemented based on Theano[3].

Evaluation Metric: We evaluate the predictive performance of WG-Learning and baselines in terms of root mean square error (RMSE) excluding the outliers. Lower values of RMSE are considered better. At each time instant, we perform one-step ahead prediction using the online model and then update the model when the new data point arrives.

5.5.2 Experiment Results

Table 5.2 shows the prediction results on each dataset. WG-Learning outperforms the baselines in all datasets by tracking the variation of time series and resisting to outliers. We observe that for the datasets with large outliers e.g., YReal, WG-Learning significantly outperforms RLSTM. For the datasets (Synthetic and YSyn) having medium outliers, WG-Learning still achieves better results. SR-LSTM and ELSTM occasionally get better than RLSTM depending on datasets. For the very dynamic dataset, e.g., YSyn and YReal, SR-LSTM has higher errors than RLSTM, this could be because the aggressive update skipping mechanism of SR-LSTM leads to missing useful information.

Figure 5.4 visualizes the predictive performance of the online model for the time period with complex outliers and change points in each dataset. The top plots of Figure 5.4 visualize the observed time series of the datasets Synthetic, YSyn and YReal. The red points represent the outliers and the vertical dotted lines indicate change points. YReal has no explicit change points and thus no vertical line is shown in the corresponding figure. The other plots of

[3] http://deeplearning.net/software/theano/

Table 5.2 – Prediction accuracy on synthetic and real datasets (RMSE).

approach \ dataset	Synthetic	YSyn	YReal
RLSTM	11.01	562.56	10104.24
SR-LSTM	10.03	669.63	13816.99
ELSTM	10.31	660.02	13789.21
WG-Learning	**9.62**	**502.79**	**8928.92**

Figure 5.4 show the predicted time series using the three baselines and WG-Learning.

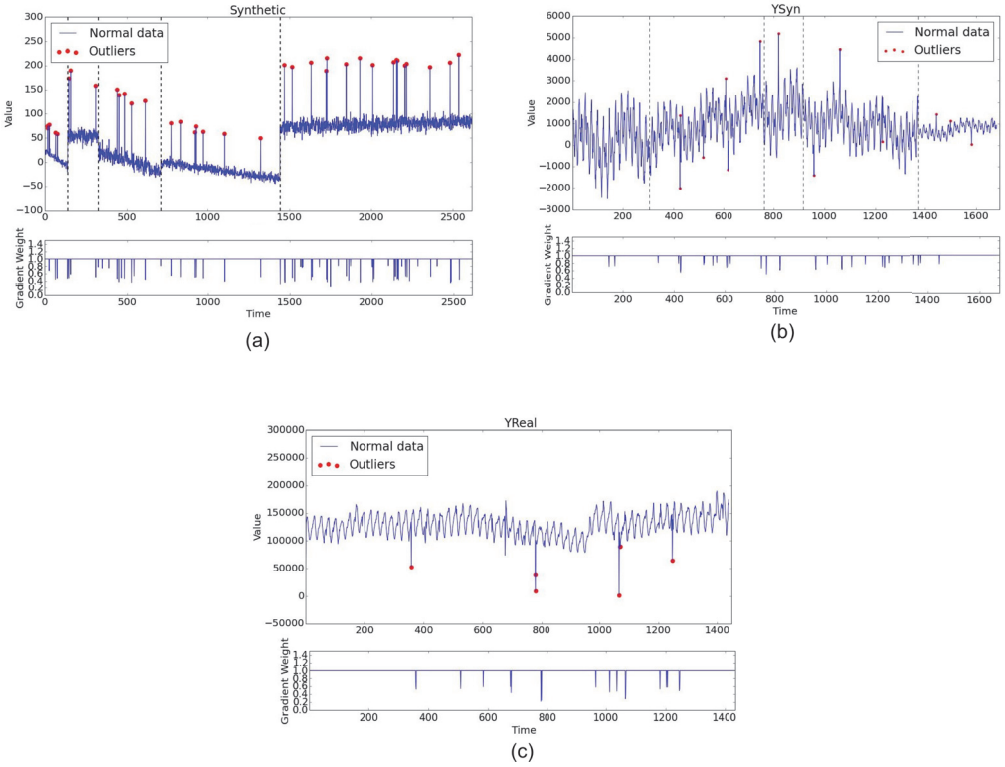


Figure 5.5 – Gradient weights of WG-Learning on each dataset(best viewed in colour): the observed time series (top) and the corresponding weights (bottom).

From Figure 5.4, we can observe that in dataset Synthetic, our WG-Learning is able to catch the details of time series as well as promptly adapting to the data after change points. Meanwhile, WG-Learning can circumvent the adverse effect of outliers by sticking to the underlying patterns of the time series. It indicates that the weighted gradients of WG-Learning are indeed effective. On the contrary, RLSTM suffers from the outliers by presenting predictions having oscillations around outliers as is shown in the second row of Figure 5.4. Compared with WG-Learning, the prediction of SR-LSTM and ELSTM resembles a sliding average and

may loss details of the time series, though they present robust prediction compared with RLSTM. Meanwhile, SR-LSTM and ELSTM are lagged on change points. In ELSTM, a change point is detected when the data points following it incur sufficient change in prediction errors [13, 17, 47]. This means in the online learning, there is a delay in change point detection, and thus ELSTM may miss the update of changing data points. With similar tendency, our WG-Learning provides consistent robustness and adaptiveness on datasets YSyn and YReal.

In addition, we investigate the weighted gradients of WG-Learning. We record the gradient weight at each data point (i.e., β_t in Eq. (5.14)) during the online learning process and visualize them associated with the observed time series in Figure 5.5. We can observe that: most of the gradient weights are close to one at the regular points, and in turn the model can be normally updated in an online fashion. However gradient weights are much less than one at the detected outliers, which avoid deviating the model from the underlying patterns of the time series. On the other hand, gradient weights at the detected change points are mostly retained close to one, such that WG-Learning is able to adapt to the new patterns in real time.

5.6 Conclusion

In this paper we propose an adaptive gradient learning method for recurrent neural networks (RNN) in the context of online learning problem. The WG-Learning aims to incrementally learn the streaming time series and provide robust predictions adapting to the changing patterns as well as resisting to outliers. In the WG-Learning, we introduce the weighted gradient to the online SGD for the RNN models, based on the local features of time series. The method enables to update the RNN models with downweighted gradients for outliers while full gradients for change points. The predictive performance of the WG-Learning in the extensive experiments on both synthetic and real datasets verifies the superiority of the WG-Learning over the baselines.

6 Conclusion and Future Directions

6.1 Conclusion

In this thesis we focus on distributed time series analytics. Particularly, we propose in-depth solutions for three topics of distributed time series analytics, i.e., scalable management and querying of model-view time series, distributed mining of correlation in massive time series streams and robust learning over noisy and non-stationary data.

For the first time, we explore the key-value representation of an interval index for model-view time series data management. Different from conventional external-memory index structure with complex node merging and split mechanisms, our KVI-index, resident partially in memory and partially materialized in the key-value store, is easy to maintain in the dynamic sensor data generation environment. Moreover, we proposed a hybrid query processing approach, namely KVI-Scan-MapReduce, integrating the KVI-index, range scan and MapReduce for model-view data in key-value stores. Extensive experiments in a real testbed showed that our approach outperforms in terms of query response time and index updating efficiency not only query processing methods based on raw sensor data, but also all other approaches considered based on model-view sensor data for time/value range and point queries. As a future work, we plan to explore how to process time and value composite queries and join queries based on the KVI-index.

For the topic of time series data mining, we investigated the problem of mining correlations in time series streams. Due to the lack of efficient solution to this problem in contemporary distributed stream processing environment, we put forward the P2H framework. It harnesses P^2-data-shuffling and δ-hypercube to optimize communication and computation efficiency. P^2-data-shuffling is able to adaptively replicate and shuffle the sliding window of a time series only to the computing nodes containing its correlation partners, thereby dramatically decreasing the communication cost. We evaluate the proposed P2H framework against several

distributed baselines adapted from conventional correlation mining approaches. The results demonstrate that our P2H approach scales better than baselines in the sense that it can simultaneously process more time series streams without bottlenecks due to the communication and computation efficiency.

Regarding robust learning on noisy and non-stationary data, we proposed two solutions respectively for the online and offline data analytics. On one hand, we propose an efficient distributed robust regression tree for handling large and noisy datasets. Our DR2-Tree employs a novel space-bounded data summarization technique, which is able to support both distributed information extraction and robust error criterion estimation for growing the robust regression tree. Extensive experiments reveal that: (1) Our proposed DR2-Tree is robust to datasets with various outlier percentages and magnitudes. (2) DR2-Tree exhibits comparable accuracy as the conventional distributed regression tree for relatively clean datasets with rare outliers. (3) DR2-Tree is much more efficient than exact robust regression and the dynamic histogram based regression tree [14, 133]. Such results verifies the broad applicability of our DR2-Tree framework.

On the other and, for streaming time series bearing both outliers and change points, which is quite common in reality, we propose an adaptive gradient learning method for recurrent neural networks (RNN), referred to as WG-Learning, in the context of online learning problem. The WG-Learning aims to incrementally learn the streaming time series and provide robust predictions adapting to the changing patterns as well as resisting to outliers. In the WG-Learning, we introduce the weighted gradient to the online SGD for the RNN models, based on the local features of time series. The method enables to update the RNN models with downweighted gradients for outliers while full gradients for change points. The predictive performance of the WG-Learning in the extensive experiments on both synthetic and real datasets verifies the superiority of the WG-Learning over the baselines in terms of prediction accuracy.

6.2 Future Directions

We identify that our work presented in this thesis can be further extended in the ways as follows:

Model-View Time Series: As a future work, we plan to explore how to process time and value composite queries and join queries based on the KVI-index. Meanwhile, our current KVI-index has to replicate the data for indexing both time and value dimensions and thus we expect to improve the space efficiency by proposing some ways to avoid this data duplication.

Distribution Mining of Time Series Streams: The correlations in the time series of evolv-

ing entities is often temporally dependent [125, 143], and may also exhibit quite volatile behaviours during abnormal periods or events [28, 143]. Therefore, in addition to the mining of correlations, our current approach entails a probabilistic model responsible for representing the dynamic nature of correlations, such that it can be used for inferring correlations occurrences [7, 15, 28]. One way is to exploit the Markov Random Field to model both the co-occurrence and temporal dependency of correlation occurrences in time series streams. However, such a modeling method cannot capture the strength of correlations and thus detect events relevant to the correlation strength. We can further extend the correlation modeling via stochastic neighbour em- bedding [59, 61]. Then, such a model can be used for correlation change or anomaly analysis.

Robust Learning: The proposed distributed robust regression tree in this thesis can be strengthened from two perspectives. First, since the bins in the histograms of data summarization is dynamically built based on the data, they approximate the data distribution. During the tree growing phase, it is possible that not all the internal tree nodes have corrupted data. Therefore, we can exploit the distribution of the bins in the domain of the target variable to evaluate the skewness of data and then to adaptively choose between conventional and robust statistics. Such a mechanism enables to avoid missing useful information while retaining the robustness in the tree growing phase. Second, the current space bounded histogram in the data summarization lacks deterministic error bound on the robust statistic estimation. We can resort to quality-guaranteed data structures to enhance the data summarization.

Regarding the RNN based online learning of time series streams, we figure out two directions to strengthen our current approach as well. On one hand, current error reference distribution is maintained by the exponential weighted moving average, which takes time to gradually erase the effect of old data in face of change points. We could extend this by applying the idea in concept drift detection to segment the error series and to only keep the up-to-date segment for estimating the latest reference distribution, such that a set of recent forecasting errors representing the current performance of RNN is obtained. On the other hand, for the value difference drift used in weighting the gradient, our current approach requires a threshold parameter to determine the normal drift or not. We could make this step more data-adaptive and avoid parameter tuning by online maintaining a value difference distribution and then utilize the probability of value difference to weight the gradient.

Bibliography

[1] Apache Samza. http://samza.apache.org/.

[2] Apache Storm. https://storm.apache.org/.

[3] Opentsdb. In *http://opentsdb.net/*, 2011.

[4] e. a. Abadi, Daniel. Aurora : A new model and architecture for data stream management. *VLDB Journal*, pages 120–139, 2003.

[5] D. Achlioptas. Database-friendly random projections. In *20th ACM PODS*, pages 274–281. ACM, 2001.

[6] R. P. Adams and D. J. MacKay. Bayesian online changepoint detection. *arXiv preprint arXiv:0710.3742*, 2007.

[7] S. Aman and et al. Influence-driven model for time series prediction from partial observations. In *AAAI*, pages 601–607, 2015.

[8] O. Anava, E. Hazan, S. Mannor, and O. Shamir. Online learning for time series prediction. In *Proceedings of the Conference on Learning Theory*, pages 172–184, 2013.

[9] O. Anava, E. Hazan, and A. Zeevi. Online time series prediction with missing data. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2191–2199, 2015.

[10] T. W. Anderson. *The Statistical Analysis of Time Series*. John Wiley & Sons, 2011.

[11] L. Arge and J. Vitter. Optimal dynamic interval management in external memory. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 560–569. IEEE, 1996.

[12] J. D. Basilico, M. A. Munson, T. G. Kolda, K. R. Dixon, and W. P. Kegelmeyer. Comet: A recipe for learning and using large ensembles on massive data. In *2011 IEEE ICDM*, pages 41–50. IEEE, 2011.

[13] M. Basseville, I. V. Nikiforov, et al. *Detection of abrupt changes: theory and application*, volume 104. Prentice Hall Englewood Cliffs, 1993.

[14] Y. Ben-Haim and E. Tom-Tov. A streaming parallel decision tree algorithm. *Journal of Machine Learning Research*, 11(Feb):849–872, 2010.

[15] Y. Benezeth and et.al. Abnormal events detection based on spatio-temporal co-occurences. In *IEEE CVPR*, pages 2458–2465. IEEE, 2009.

[16] A. Bhattacharya, A. Meka, and A. K. Singh. Mist: Distributed indexing and querying in sensor networks using statistical models. In *Proceedings of the 33rd international conference on Very large data bases*, pages 854–865. VLDB Endowment, 2007.

[17] A. Bifet and R. Gavalda. Learning from time-changing data with adaptive windowing. In *SDM*, volume 7, page 2007. SIAM, 2007.

[18] E. Bingham and H. Mannila. Random projection in dimensionality reduction: applications to image and text data. In *7th ACM SIGKDD conference*, pages 245–250. ACM, 2001.

[19] T. Bozkaya and Z. M. Özsoyoglu. Indexing valid time intervals. In *Proceedings of the 9th International Conference on Database and Expert Systems Applications*, DEXA '98, pages 541–550, London, UK, UK, 1998. Springer-Verlag.

[20] E. Busseti, I. Osband, and S. Wong. Deep learning for time series modeling. Technical report, Stanford University, 2012.

[21] Y. Cai and R. Ng. Indexing spatio-temporal trajectories with chebyshev polynomials. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 599–610. ACM, 2004.

[22] K. Chan. A further analysis of the lead-lag relationship between the cash market and stock index futures market. *Review of financial studies*, 5(1):123–152, 1992.

[23] V. Chandola and R. R. Vatsavai. A gaussian process based online change detection algorithm for monitoring periodic time series. In *SDM*, pages 95–106. SIAM, 2011.

[24] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM TOCS*, 26, 2008.

[25] C. Chatfield. *The Analysis of Time Series: An Introduction*. CRC Press, 2013.

[26] S. Chauhan and L. Vig. Anomaly detection in ecg time signals via deep long short-term memory networks. In *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*, pages 1–7. IEEE, 2015.

[27] Q. Chen, L. Chen, X. Lian, Y. Liu, and J. X. Yu. Indexable pla for efficient similarity search. In *Proceedings of the 33rd international conference on Very large data bases*, pages 435–446. VLDB Endowment, 2007.

[28] X. C. Chen and et al. Online discovery of group level events in time series. In *SDM*, pages 632–640. SIAM, 2014.

[29] Y. Chen, W. Wang, X. Du, and X. Zhou. Continuously monitoring the correlations of massive discrete streams. In *CIKM*, CIKM '11, pages 1571–1576, New York, NY, USA, 2011. ACM.

[30] L. C. Chm-les X, O. CA, and R. J. Yan. Decision tree with better ranking. 2003.

[31] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[32] R. Cole, D. Shasha, and X. Zhao. Fast window correlations over uncooperative time series. In *ACM SIGKDD*, pages 743–749. ACM, 2005.

[33] J. T. Connor, R. D. Martin, and L. E. Atlas. Recurrent neural networks and robust time series prediction. *Neural Networks, IEEE Transactions on*, 5(2):240–254, 1994.

[34] J. Darlington, Y.-k. Guo, J. Sutiwaraphun, and H. W. To. Parallel induction algorithms for data mining. In *International Symposium on Intelligent Data Analysis*, pages 437–445. Springer, 1997.

[35] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[36] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 588–599. VLDB Endowment, 2004.

[37] A. Deshpande and S. Madden. Mauvedb: supporting model-based user views in database systems. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD, pages 73–84, 2006.

[38] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *VLDB Endowment*, 1, 2008.

[39] J. Dittrich, J.-A. Quiané-Ruiz, A. Jindal, Y. Kargin, V. Setty, and J. Schad. Hadoop++: making a yellow elephant run like a cheetah (without it even noticing). *Proc. VLDB Endow.*, 3(1-2):515–529, 2010.

[40] J. Dittrich, J.-A. Quiané-Ruiz, S. Richter, S. Schuh, A. Jindal, and J. Schad. Only aggressive elephants are fast elephants. *Proc. VLDB Endow.*, 5(11):1591–1602, 2012.

[41] R. Elmasri, G. T. J. Wuu, and Y.-J. Kim. The time index: An access structure for temporal data. In *Proceedings of the 16th International Conference on Very Large Data Bases*, VLDB '90, pages 1–12, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[42] M. Y. Eltabakh, F. Özcan, Y. Sismanis, P. J. Haas, H. Pirahesh, and J. Vondrak. Eagle-eyed elephant: split-oriented indexing in hadoop. In *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, pages 89–100, New York, NY, USA, 2013.

[43] F. Esposito, D. Malerba, G. Semeraro, and J. Kay. A comparative analysis of methods for pruning decision trees. *IEEE TPAMI*, 19(5):476–491, 1997.

[44] A. A. Freitas and S. H. Lavington. *Mining very large databases with parallel processing*, volume 9. Springer Science & Business Media, 1997.

[45] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

[46] S. Fries, B. Boden, G. Stepien, and T. Seidl. Phidj: Parallel similarity self-join for high-dimensional vector data with mapreduce. In *ICDE*, pages 796–807. IEEE, 2014.

[47] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46(4):44, 2014.

[48] R. Garnett, M. A. Osborne, and S. J. Roberts. Sequential bayesian prediction in the presence of changepoints. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 345–352. ACM, 2009.

[49] L. George. *HBase: the definitive guide*. O'Reilly Media, Incorporated, 2011.

[50] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12(10):2451–2471, 2000.

[51] A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Studies in Computational Intelligence. Springer, 2012.

[52] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[53] T. Guo, J.-P. Calbimonte, H. Zhuang, and K. Aberer. Sigco: Mining significant correlations via a distributed real-time computation engine. In *IEEE BigData Conference*, pages 747–756. IEEE, 2015.

[54] T. Guo, T. G. Papaioannou, and K. Aberer. Model-view sensor data management in the cloud. In *Big Data, 2013 IEEE International Conference on*, pages 282–290. IEEE, 2013.

[55] T. Guo, T. G. Papaioannou, H. Zhuang, and K. Aberer. Online indexing and distributed querying model-view sensor data in the cloud. In *The 19th International Conference on Database Systems for Advanced Applications*, 2014.

[56] T. Guo, S. Sathe, and K. Aberer. Fast distributed correlation discovery over streaming time series data. In *ACM CIKM*, pages 1161–1170. ACM, 2015.

[57] T. Guo, Z. Yan, and K. Aberer. An adaptive approach for online segmentation of multi-dimensional mobile data. In *Proc. of MobiDE, SIGMOD Workshop*, 2012.

[58] M. Gupta, J. Gao, C. Aggarwal, and J. Han. Outlier detection for temporal data. *Synthesis Lectures on Data Mining and Knowledge Discovery*, 5(1):1–129, 2014.

[59] G. E. Hinton and S. T. Roweis. Stochastic neighbor embedding. In *Advances in neural information processing systems*, pages 833–840, 2002.

[60] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[61] T. Idé, S. Papadimitriou, and M. Vlachos. Computing correlation anomaly scores using stochastic nearest neighbors. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 523–528. IEEE, 2007.

[62] M. Iu and W. Zwaenepoel. Hadooptosql: a mapreduce query optimizer. In *Proceedings of the 5th European conference on Computer systems*, pages 251–264. ACM, 2010.

[63] L. C. Jain, M. Seera, C. P. Lim, and P. Balasubramaniam. A review of online learning in supervised neural networks. *Neural Computing and Applications*, 25(3-4):491–509, 2014.

[64] C. Z. Janikow. Fuzzy decision trees: issues and methods. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 28(1):1–14, 1998.

[65] G. H. John. Robust decision trees: Removing outliers from databases. In *KDD*, pages 174–179, 1995.

[66] M. V. Joshi, G. Karypis, and V. Kumar. Scalparc: A new scalable and efficient parallel classification algorithm for mining large datasets. In *IPPS/SPDP 1998*, pages 573–579. IEEE, 1998.

[67] G. Katz, A. Shabtai, L. Rokach, and N. Ofek. Confdtree: Improving decision trees using confidence intervals. In *2012 IEEE ICDM*, pages 339–348. IEEE, 2012.

[68] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM SIGMOD Record*, 30(2):151–162, 2001.

[69] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 289–296. IEEE, 2001.

[70] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[71] C. P. Kolovson and M. Stonebraker. Segment indexes: dynamic indexing techniques for multi-dimensional interval data. *SIGMOD Rec.*, 20(2):138–147, 1991.

[72] A. Koufakou and M. Georgiopoulos. A fast outlier detection strategy for distributed high-dimensional data sets with mixed attributes. *Data Mining and Knowledge Discovery*, 20(2):259–289, 2010.

[73] H.-P. Kriegel, M. Pötke, and T. Seidl. Managing intervals efficiently in object-relational databases. In *Proc. 26th Int. Conf. on Very Large Databases (VLDB)*, pages 407–418, 2000.

[74] H.-P. Kriegel, M. Pötke, and T. Seidl. Object-relational indexing for general interval relationships. In *International Symposium on Spatial and Temporal Databases*, pages 522–542. Springer, 2001.

[75] M. Kutare, G. Eisenhauer, C. Wang, K. Schwan, V. Talwar, and M. Wolf. Monalytics: online monitoring and analytics for managing large scale data centers. In *Proceedings of the 7th international conference on Autonomic computing*, pages 141–150. ACM, 2010.

[76] N. Laptev, S. Amizadeh, and I. Flint. Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1939–1947. ACM, 2015.

[77] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, 2015.

[78] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon. Parallel data processing with mapreduce: a survey. *AcM SIGMOD Record*, 2012.

[79] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon. Parallel data processing with mapreduce: a survey. *ACM SIGMOD Record*, 40(4):11–20, 2012.

[80] P. Li, T. J. Hastie, and K. W. Church. Very sparse random projections. In *12th ACM SIGKDD conference*, pages 287–296. ACM, 2006.

[81] Y. Li, M. L. Yiu, Z. Gong, et al. Discovering longest-lasting correlation in sequence databases. *PVLDB Endowment*, 6(14):1666–1677, 2013.

[82] C. Liao and et al. Mining influence in evolving entities: A study on stock market. In *Data Science and Advanced Analytics (DSAA), 2014 International Conference on*, pages 244–250. IEEE, 2014.

[83] A. Liaw and M. Wiener. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[84] Z. C. Lipton, D. C. Kale, C. Elkan, and R. Wetzell. Learning to diagnose with lstm recurrent neural networks. *arXiv preprint arXiv:1511.03677*, 2015.

[85] C. Liu, S. C. H. Hoi, P. Zhao, and J. Sun. Online arima algorithms for time series prediction. In *Proceedings of the 13th AAAI Conference on Artificial Intelligence*, 2016.

[86] J. Liu, K. Zhao, B. Kusy, J.-r. Wen, and R. Jurdak. Temporal embedding in convolutional neural networks for robust learning of abstract snippets. *arXiv preprint arXiv:1502.05113*, 2015.

[87] S. Liu, M. Yamada, N. Collier, and M. Sugiyama. Change-point detection in time-series data by relative density-ratio estimation. *Neural Networks*, 43:72–83, 2013.

[88] C. Luo, J.-G. Lou, Q. Lin, Q. Fu, R. Ding, D. Zhang, and Z. Wang. Correlating events with time series for incident diagnosis. In *20th ACM SIGKDD conference*, pages 1583–1592. ACM, 2014.

[89] Q. Lyu and J. Zhu. Revisit long short-term memory: An optimization perspective. In *Advances in neural information processing systems workshop on deep Learning and representation Learning*, 2014.

[90] F. Machado. Communication and memory efficient parallel decision tree construction. 2003.

[91] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal. Long short term memory networks for anomaly detection in time series. In *European Symposium on Artificial Neural Networks*, volume 23.

[92] J. Martens and I. Sutskever. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

[93] L. L. Minku and X. Yao. Ddd: A new ensemble approach for dealing with concept drift. *IEEE Transactions on Knowledge and Data Engineering*, 24(4):619–633, 2012.

[94] D. C. Montgomery, C. L. Jennings, and M. Kulahci. *Introduction to Time Series Analysis and Forecasting*. John Wiley & Sons, 2011.

[95] A. Mueen, S. Nath, and J. Liu. Fast approximate correlation for massive time-series data. In *SIGMOD*, pages 171–182, 2010.

[96] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[97] S. Muthukrishnan. *Data streams: Algorithms and applications*. Now Publishers Inc, 2005.

[98] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari. S4: Distributed stream computing platform. In *ICDM Workshops*, pages 170–177, Dec. 2010.

[99] S. Nishimura, S. Das, D. Agrawal, and A. E. Abbadi. Md-hbase: A scalable multi-dimensional data infrastructure for location aware services. In *Proceedings of the 2011 IEEE 12th International Conference on Mobile Data Management - Volume 01*, MDM '11, pages 7–16, Washington, DC, USA, 2011. IEEE Computer Society.

[100] C. Olaru and L. Wehenkel. A complete fuzzy decision tree technique. *Fuzzy sets and systems*, 138(2):221–254, 2003.

[101] B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo. Planet: massively parallel learning of tree ensembles with mapreduce. *PVLDB*, 2(2):1426–1437, 2009.

[102] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. In *VLDB*, pages 697–708. VLDB Endowment, 2005.

[103] T. G. Papaioannou, M. Riahi, and K. Aberer. Towards online multi-model approximation of time series. In *Mobile Data Management (MDM), 2011 12th IEEE International Conference on*, MDM, page 33, 2011.

[104] T. Preis and et al. Quantifying the behavior of stock correlations under market stress. *Scientific reports*, 2, 2012.

[105] F. Provost and P. Domingos. Well-trained pets: Improving probability estimation trees. 2000.

[106] J. R. Quinlan. Bagging, boosting, and c4. 5. In *AAAI/IAAI, Vol. 1*, pages 725–730, 1996.

[107] e. a. Rajaraman. *Mining of massive datasets*, volume 77. Cambridge University Press Cambridge, 2012.

[108] C. Reiss, J. Wilkes, and J. L. Hellerstein. Google cluster-usage traces: format+ schema. *Google Inc., White Paper*, 2011.

[109] C. Richard, J. C. M. Bermudez, and P. Honeine. Online prediction of time series data with kernels. *IEEE Transactions on Signal Processing*, 57(3), 2008.

[110] G. Ristanoski, W. Liu, and J. Bailey. A time-dependent enhanced support vector machine for time series regression. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 946–954. ACM, 2013.

[111] J. J. Rodriguez, L. I. Kuncheva, and C. J. Alonso. Rotation forest: A new classifier ensemble method. *IEEE TPAMI*, 28(10):1619–1630, 2006.

[112] E. J. Ruiz, V. Hristidis, C. Castillo, A. Gionis, and A. Jaimes. Correlating financial time series with micro-blogging activity. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 513–522. ACM, 2012.

[113] Y. Saatçi, R. D. Turner, and C. E. Rasmussen. Gaussian process change point models. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 927–934, 2010.

[114] Y. Sakurai, S. Papadimitriou, and C. Faloutsos. Autolag: Automatic discovery of lag correlations in stream data. In *ICDE 2005*, pages 159–160. IEEE, 2005.

[115] Y. Sakurai, S. Papadimitriou, and C. Faloutsos. Braid: Stream mining through group lag correlations. In *2005 ACM SIGMOD conference*, pages 599–610. ACM, 2005.

[116] A. D. Sarma, Y. He, and S. Chaudhuri. Clusterjoin: A similarity joins framework using map-reduce. In *PVLDB*, 2014.

[117] S. Sathe and K. Aberer. AFFINITY: Efficiently querying statistical measures on time-series data. In *ICDE*, pages 841–852, 2013.

[118] S. Sathe, T. G. Papaioannou, H. Jeung, and K. Aberer. A survey of model-based sensor data acquisition and management. In *Managing and Mining Sensor Data*. Springer, 2013.

[119] S. Sathe, T. G. Papaioannou, H. Jeung, and K. Aberer. A survey of model-based sensor data acquisition and management. In *Managing and Mining Sensor Data*, pages 9–50. Springer, 2013.

[120] A. Senior, G. Heigold, M. Ranzato, and K. Yang. An empirical study of learning rates in deep neural networks for speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6724–6728. IEEE, 2013.

[121] G. Sfakianakis, I. Patlakas, N. Ntarmos, and P. Triantafillou. Interval indexing and querying on key-value cloud stores. In *In Proc. 29th IEEE International Conference on Data Engineering (ICDE13).*, 2013.

[122] J. Shafer, R. Agrawal, and M. Mehta. Sprint: A scalable parallel classi er for data mining. In *PVLDB*, pages 544–555. Citeseer, 1996.

[123] J. G. Shanahan and L. Dai. Large scale distributed data science using apache spark. In *ACM SIGKDD*, pages 2323–2324. ACM, 2015.

[124] I. Sharfman, A. Schuster, and D. Keren. A geometric approach to monitoring threshold functions over distributed data streams. *TODS*, 2007.

[125] X. Shi, W. Fan, and S. Y. Philip. Dynamic shaker detection from evolving entities. In *SDM*, pages 350–358. SIAM, 2013.

[126] J. Shieh and E. Keogh. i sax: indexing and mining terabyte sized time series. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 623–631. ACM, 2008.

[127] A. Srivastava, E.-H. Han, V. Kumar, and V. Singh. Parallel formulations of decision-tree classification algorithms. In *High Performance Data Mining*, pages 237–261. Springer, 1999.

[128] C. Stuart. Robust regression. 2011.

[129] I. Sutskever. *Training Recurrent Neural Networks*. PhD thesis, University of Toronto, 2013.

[130] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[131] A. Thiagarajan and S. Madden. Querying continuous functions in a database system. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD, pages 791–804, 2008.

[132] L. F. R. A. Torgo. Inductive learning of tree-based regression models. 1999.

[133] S. Tyree, K. Q. Weinberger, K. Agrawal, and J. Paykin. Parallel boosted regression trees for web search ranking. In *WWW*, pages 387–396. ACM, 2011.

[134] O. Vallis, J. Hochenbaum, and A. Kejariwal. A novel technique for long-term anomaly detection in the cloud. In *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14)*, 2014.

[135] J. Wang, S. Wu, H. Gao, J. Li, and B. C. Ooi. Indexing multi-dimensional data in a cloud system. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 591–602, New York, NY, USA, 2010. ACM.

[136] P. Wang, W. Sun, D. Yin, J. Yang, and Y. Chang. Robust tree-based causal inference for complex ad effectiveness analysis. In *ACM WSDM*, pages 67–76. ACM, 2015.

[137] X. Wang, Y. Yao, X. Wang, K. Lu, and Q. Cao. Carpo: Correlation-aware power optimization in data center networks. In *INFOCOM 2012*, pages 1125–1133. IEEE, 2012.

[138] Y. Wang, A. Metwally, and S. Parthasarathy. Scalable all-pairs similarity search in metric spaces. In *19th ACM SIGKDD*, pages 829–837. ACM, 2013.

[139] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, volume 98, pages 194–205, 1998.

[140] W. W.-S. Wei. *Time series analysis*. Addison-Wesley publ Reading, 1994.

[141] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[142] W. K. Wong, M. Xia, and W. C. Chu. Adaptive neural network model for time-series forecasting. *European Journal of Operational Research*, 207(2):807–816, 2010.

[143] D. Wu, Y. Ke, J. X. Yu, S. Y. Philip, and L. Chen. Leadership discovery when data correlatively evolve. *World Wide Web Journal*, 14(1):1–25, 2011.

[144] Y. Wu, J. M. H. Lobato, and Z. Ghahramani. Gaussian process volatility model. In *Advances in Neural Information Processing Systems 27*, 2014.

[145] Q. Xie, S. Shang, B. Yuan, C. Pang, and X. Zhang. Local correlation detection with linearity enhancement in streaming data. In *ACM CIKM*, pages 309–318. ACM, 2013.

[146] K. Yamanishi and J.-i. Takeuchi. A unifying framework for detecting outliers and change points from non-stationary time series data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 676–681. ACM, 2002.

[147] D. Yang and et al. Cands: continuous optimal navigation via distributed stream processing. *PVLDB*, 8(2):137–148, 2014.

[148] H.-C. Yang and D. S. Parker. Traverse: Simplified indexing on large map-reduce-merge clusters. In *Proceedings of the 14th International Conference on Database Systems for Advanced Applications*, DASFAA, pages 308–322, Berlin, Heidelberg, 2009. Springer-Verlag.

[149] L. Yao, Q. Z. Sheng, B. J. Gao, A. H. Ngu, and X. Li. A model for discovering correlations of ubiquitous things. In *2013 IEEE ICDM*, pages 1253–1258. IEEE, 2013.

[150] J. Ye, J.-H. Chow, J. Chen, and Z. Zheng. Stochastic gradient boosted distributed decision trees. In *ACM CIKM*, pages 2061–2064. ACM, 2009.

[151] B. Zadrozny and C. Elkan. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *ICML*, volume 1, pages 609–616. Citeseer, 2001.

[152] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica. Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. In *4th USENIX HotCloud*, pages 10–10. USENIX Association, 2012.

[153] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[154] C. Zhang, Y. Zheng, X. Ma, and J. Han. Assembler: Efficient discovery of spatial co-evolving patterns in massive geo-sensory data. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1415–1424. ACM, 2015.

[155] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *VLDB*, pages 358–369, 2002.

Tian Guo

CONTACT INFORMATION

Room BC-141
School of Computer and Communication Science
Eć ole Polytechnique Fédérale de Lausanne
EPFL, Station 14
Lausanne, CH-1015, Switzerland

Cell: (+41)078-677-3452
Office: (+41)21-693-7501
E-mail: *tian.guo@epfl.ch*
tian.guo0980@gmail.com

EDUCATION

Swiss Federal Institute of Technology, Lausanne (EPFL), Switzerland

Ph.D. Candidate, Computer Science, Expected Graduation Date: October 2016

- Distributed Information Systems Laboratory
- Advisor: Prof. Karl Aberer
- Research Interest: time series data mining, neural computation, distributed computing.

Shanghai Jiao Tong University (SJTU), Shanghai, China

M.S., Autonomous Robot Lab, April 2011

- Advisor: Prof. Weidong Chen
- Topic: Active sensing system for autonomous mobile robots

East China University of Science and Technology (ECUST), Shanghai, China

B.Eng., School of Information Science and Engineering, July 2008.

PUBLICATION

- T. Guo, et al. Convolutional-Recurrent Neural Network for Semantic Trend Mining. In progress.
- T. Guo, et al. Distributed Quantile Regression Tree using Quality Guaranteed Data Summarization. In progress.
- T. Guo, Z. Xu, X. Yao, H. Chen, K. Aberer and K. Funaya. Robust Online Time Series Prediction with Recurrent Neural Networks. IEEE/ACM SIGKDD International Conference on Data Science and Advanced Analytics (**DSAA**) Montreal, Canada, 2016.
- H. Zhuang, R. Rahman, X. Hu, T. Guo, P. Hui and K. Aberer. Data Summarization with Social Contexts. 25th International Conference on Information and Knowledge Management (**CIKM**), Indianapolis, United States, 2016.
- T. Guo, K. Kutzkov, M. Ahmed, J. Calbimonte and K. Aberer. Efficient Distributed Decision Trees for Robust Regression. European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (**ECML/PKDD**), Riva del Garda, Italy, 2016.
- T. Guo and K. Aberer. A Distributed Mining Framework for Influence in Entities. International Joint **EDBT/ICDT** Conference, Bordeaux, France, 2016.
- T. Guo, J. Calbimonte and K. Aberer. Distributed Mining and Modeling of Dynamic Lead-Lag Relations in Evolving Entities. DMIoT at IEEE International Conference on Data Mining series (**ICDM**), Barcelona, Spain, 2016139
- T. Guo, H. Zhuang and K. Aberer. SigCO: Mining Significant Correlations via a Distributed Real-time Computation Engine. IEEE International Conference on Big Data 2015 (**IEEE BigData**), Santa Clara, California, USA, 2015. [Acceptance rate 16.8%]

- T. Guo, S. Sathe and K. Aberer. Fast Distributed Correlation Discovery Over Streaming Time-Series Data. 24th International Conference on Information and Knowledge Management (**CIKM**). Melbourne, Australia, 2015.
- T. Guo, T. G. Papaioannou and K. Aberer. Efficient Indexing and Query Processing of Model-View Sensor Data in the Cloud. Journal on Big Data Research, Elsevier, 2014.
- T. Guo, T. G. Papaioannou, H. Zhuang and K. Aberer. Online indexing and distributed querying model-view sensor data in the cloud. The 19th International Conference on Database Systems for Advanced Applications (**DASFAA**), Bali, Indonesia, 2014.
- T. Guo, T. G. Papaioannou and K. Aberer. Model-View Sensor Data Management in the Cloud. IEEE International Conference on Big Data 2013 (**IEEE BigData**), Santa Clara, California, USA, 2013. [Acceptance rate 17.0%]
- J. Calbimonte, M. Riahi, R. Gwadera, T. Guo, Z. Podnar, G. Ivana and C. Georgoulis. Self-management and Optimization Framework. OpenIoT, 2013
- T. Guo, Z. Yan and K. Aberer. An Adaptive Approach for On-line Segmentation of Multi-Dimensional Mobile Data. MobiDE at the International Conference on Management of Data **ACM SIGMOD/PODS**, Scottsdale, Arizona, USA, 2012, **(Best Paper Award)**
- T. Guo. Laser Based Mobile Robot Target Localizing and Multi Motion Patterns Tracking, Microcomputer Application, December, 2010.

Research and Professional Experience

Network Data Analytics Group, NEC Laboratories Europe
Research Intern, June 2015 - December 2015.

Online learning of Recurrent Neural Network over time series with change points and outliers

Mentor: Zhao Xu

- Analysed the performance of the Long-Short-Term-Memory Neural Network, a popular variant of RNN on time series with change points and outliers.
- Proposed a weighted-gradient online learning framework for LSTM neural networks to perform robust learning and prediction for non-stationary time series.
- Implemented the proposed approach and baselines based on Keras and TensorFlow and verified the precision of the proposed approach through extensive experiments.

Effiicent distributed robust regression tree

Mentors: Konstantin Kutzkov, Mathias Niepert, Mohamed Ahmed

- Analysed the source code of regression tree and random forest algorithms in the distributed machine learning tool, Apache Spark MLlib and spotted the weak points to improve for efficient robust regression.
- Designed distributed regression trees employing various robust loss functions (e.g., MSE, Trimmed-MSE, LAD, MAD).
- Proposed a data summarization based distributed training framework to improve the learning efficiency of the distributed robust regression tree.
- Developed the proposed approaches and baselines based on Apache Spark.
- Evaluated against a variety of baselines including tree ensembles in Apache Spark MLlib and demonstrated the superiority of the proposed approach in both efficiency and effectiveness.

Distributed Information Systems Laboratory, EPFL
Graduate Research Assistant, November 2011 - Now.

Convolutional-LSTM Neural Network for semantic trend mining in temporal data

- Semantic trend mining aims to discover the meaningful trend associated with the duration in temporal data.
- Proposed a hybrid Convolutional-LSTM neural network, which comprises a convolution neural network based segmentation and feature learning module and a LSTM based trend dependency model.
- Implementing the proposed approach and baselines based on TensorFlow.

Distributed mining of statistical information for streaming time series

- Extended state-of-the-art stream statistics (i.e., single stream statistic, cross-correlation) query algorithms to the distributed stream processing environment.
- Designed a hypercube partitioning and affine transformation based continuously statistical correlation mining algorithm for distributed time series.
- Proposed a Markov Random Field based approach to model the dynamic cross-correlations for prediction and event detection.
- Implemented the proposed approach and baselines in Apache Storm, a distributed real-time computation system.
- Achieved 1.5× improvements over baselines in terms of the peak capacity of the system.

OpenIoT - Open source cloud solution for the Internet of Things

- Designed a hybrid in-memory and key-value store index structure (KVI-index) for managing model-view sensor data in the cloud.
- Developed KVI-index based sensor value and time query processing algorithms.
- Implemented the proposed query processing approach using Hadoop MapReduce and HBase.
- Demonstrated order of magnitude improvements over querying distributed raw data in terms of response time.

OpenSense - Community-based sensing using wireless sensor network technology

- Proposed a PCA (principle component analysis) and dynamic clustering based feature selection approach.
- Designed an on-line and adaptive time series segmentation algorithm using the dynamic selected features.
- Evaluated the proposed approach using a real dataset from mobile sensors and achieved .

Autonomous Robot Laboratory, SJTU
Graduate Research Assistant, November 2008 - February 2011.

Perceptive information processing for the active sensing system in mobile robots

- This project aimed at on-line processing various sensor data (vision, laser, audio) for action decision of the autonomous robot in less-structured environments.
- Designed a subjective Bayesian decision based method to assess and predict information uncertainty.
- Proposed a heuristic search algorithm based on the extracted information for action decision.

141

Service robots for the elderly and disabled people
Supported by the National High Technology Research and Development Program of China *11.2008-7.2009*
Sensor data processing algorithm for human detection and localization.

- Utilized computation geometry to extract features from real-time laser data on an auto-mobile robot.
- Developed an Adaboost based human-detection algorithm by using the extracted features.
- Proposed Kalman Filter based real-time human tracking approach.

Awards

- Travel grant in ACM CIKM 2015
- Travel grant in IEEE International Conference on Big Data 2015
- Travel grant in IEEE International Conference on Big Data 2013
- 9th place in Swiss Helvetic Coding Contest, EPFL, Switzerland, 2012
- Three Successive Championships in RoboCup China Open Tournament, 2008, 2009, 2010
  The RoboCup is the world's largest robotics and artificial intelligence competition.
- Silver Award in the 33th ACM International Collegiate Programming Contest, Asia Regional Hangzhou Site, China Open Contest (Top 12 out of 115), 2008
- Honourable Award in the 32th ACM International Collegiate Programming Contest, Asia Regional Nanjing Site, 2007
- Second Prize in Mathematical Contest in Modelling(MCM), the Consortium for Mathematics and Its Application of the United States America, 2007
- National First Prize in China Undergraduate Mathematical Contest in Modelling (Top 5%), 2006

Professional Service

- Reviewer: IEEE Transactions on Services Computing, Journal of Location Based Services, International Journal of Geographical Information Science, Journal of Expert Systems with Applications, Journal of Artificial Intelligence Review, Journal of Robotics and Autonomous Systems, Journal of Web Technologies. International Semantic Web Conference 2013.
- External reviewer: IEEE BigData Congress 2016, ICDM 2015, ICDE 2014, IEEE BigData 2014, IEEE MDM 2014, VLDB 2012, ICDCS 2011.

Teaching Experience

- 2015 Teaching Assistant, Analysis
- 2014 Teaching Assistant, Distributed Information Systems
- 2013 Teaching Assistant, Dynamic Systems
- 2012 Teaching Assistant, Programming

Supervised Master Project

- 2016 Master Thesis Project, TreNet: Learning the Trend in Temporal Data via Hybrid Neural Networks
- 2016 Master Thesis Project, Activity-based Intelligence
- 2016 Master Semester Project, Online Learning of LSTM Neural Network
- 2014 Master Semester Project, Distributed Intersection Join for Multivariate Model-view Sensor Time Series
- 2013 Master Thesis Project, Performance Prediction for Clusters in Data Centers

Technical Skills

- Distributed Computing Platforms: Hadoop MapReduce, HBase, Apache Storm, Apache Spark.
- Machine Learning Tool-sets: Apache Spark MLlib, Scikit-learn.
- Deep Learning Libraries: Keras, TensorFlow, Theano.
- Programming Languages: C/C++, Java, Python, Visual C++, Matlab, Scala.

142

Languages

English: Fluent. Chinese: Native language

References

Karl Aberer, Professor, EPFL
Saket Sathe, Research Staff Member, IBM Research
Mohamed Ahmed, Senior Research Scientist, NEC Laboratories Europe
Zhao Xu, Senior Research Scientist, NEC Laboratories Europe