Thesis for The Degree of Licentiate of Engineering

Parallel Data Streaming Analytics in the Context of Internet of Things

Hannaneh Najdataei

Division of Networks and Systems
Department of Computer Science & Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden, 2019

Parallel Data Streaming Analytics in the Context of Internet of Things

Hannaneh Najdataei

Technical Report No 195L
ISSN 1652-876X
Department of Computer Science & Engineering
Division of Networks and Systems
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden

Printed by Chalmers Reproservice,
Gothenburg, Sweden 2019.

Parallel Data Streaming Analytics in the Context of Internet of Things

Hannaneh Najdataei
Department of Computer Science & Engineering, Chalmers University of Technology

Abstract

We are living in an increasingly connected world, where the ubiquitously sensing technologies enable inter-connection of physical objects, as part of *Internet of Things (IoT)*, and provide continuous massive amount of data. As this growth soars, benefits and challenges come together, which requires development of right tools in order to extract valuable information from data. For that, new techniques (e.g. data stream processing) have emerged to perform continuous single pass analysis and enhance parallelism. However, employing such techniques is not a trivial task due to its challenges such as partial knowledge of the data and the trade-off between parallelism and consistency. Moreover, depending on the source, data volumes may fluctuate over time which requires the degree of parallelism to be adapted in runtime.

In this work, we contribute to the design of computational infrastructures and development of tools to address these challenges. In this regard, we focus on two problem domains. First, we target continuous data analysis and particularly focus on data clustering, as a significant representative problem, to extract information from massive data, generated by high-rate sensors. We propose Lisco, a single-pass continuous Euclidean distance-based clustering which exploits the inherent ordering of the spatial and temporal data, and its parallel counterpart, P-Lisco, to enhance pipeline- and data-parallelism. These algorithms provide high throughput of results with low latency, through pushing the processing closer to the data sources. Moreover we provide a framework, *DRIVEN*, that performs a continuous bounded error approximation to compress the volumes of data, and then transmits the compressed data to next layers of the IoT architecture to perform clustering on it, in a continuous fashion, using generalized form of Lisco. The compression in data retrieval speeds up the transmission of the data while preserving very similar clustering quality as raw data transmission. In the second domain, we target the elasticity in data streaming to utilize computational resources in runtime regarding the data rate fluctuations. For that, we provide a stream processing framework, *STRETCH*, and introduce the concept of *virtual shared-nothing parallelization* that is able to adapt the resources, maximize the throughput and latency, and preserve determinism. Thorough experimental evaluations on architectures representative of high-end servers and of resource-constrained embedded devices indicate the scalability benefits of all proposed algorithms.

Keywords: Internet of Things, data analysis, clustering, edge computing, fog computing, stream/continuous data processing, parallelism, scalability, elasticity

Acknowledgment

I would like to express my sincere gratitude to my supervisors Marina Papatriantafilou, Vincenzo Gulisano and Philippas Tsigas. The challenging journey of PhD has become precious and enjoyable with your uplifting discussions, continuous support and generous guidance. I consider myself lucky to have three great mentors and thoroughly look forward to continue working with all of you.

I specially would like to thank Yiannis Nikolakopoulos for his support and advice to make first years of my PhD studies a rewarding experience. I am also grateful to my co-authors, past and current colleagues in the Division of Networks and Systems for their direct or indirect contribution to this work and providing a friendly environments. Thank you, Adones, Ali, Aljoscha, Amir, Aras, Bapi, Bastian, Bei, Beshr, Carlo, Charalampos, Christos, Dimitris, Elad, Elena, Erland, Fazeleh, Georgia, Iosif, Ivan, Joris, Karl, Magnus, Nasser, Olaf, Oliver, Romaric, Thomas R., Thomas P., Tomas, Valentin T., Valentin P., Vladimir and Wissam.

I wish to acknowledge the financial support by the Swedish Foundation for Strategic Research, project Factories in the Cloud (FiC), with grant number GMT14-0032.

Finally, I am deeply grateful to my parents for their endless love, support, and for confidence in me. To my lovely sister and wonderful brother to be always there for me. To my friends in Gothenburg for providing happy distractions. And, to my dear husband and my better half, Alireza, for his love, sympathetic ear and for believing in me.

Hannaneh Najdataei
Göteborg, April 2019

List of Publications

Appended publications

This thesis is based on the following publications:

[A] **H. Najdataei**, Y. Nikolakopoulos, V. Gulisano, M. Papatriantafilou "Continuous and Parallel LiDAR Point-cloud Clustering"
The 38th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2018.

[B] B. Havers, R. Duvignau, **H. Najdataei**, V. Gulisano, A. Chaitanya Koppisetty, M. Papatriantafilou "DRIVEN: a framework for efficient Data Retrieval and clustering in Vehicular Networks"
The 35th International Conference on Data Engineering (ICDE). IEEE, 2019.

[C] **H. Najdataei**, Y. Nikolakopoulos, M. Papatriantafilou, P. Tsigas, V. Gulisano "STRETCH: Scalable and Elastic Deterministic Streaming Analysis with Virtual Shared-Nothing Parallelism"
under submission, 2019.

Other publications

The following publication was published during my PhD studies. However, it is not appended to this thesis, due to contents overlapping that of appended publications.

[a] **H. Najdataei**, M. Subramaniyan, V. Gulisano, A. Skoogh, M. Papatriantafilou "Stream-IT: Continuous and dynamic processing of production systems data - throughput bottlenecks as a case-study"
The 28th International Symposium on Industrial Engineering (ISIE). IEEE, 2019

Contents

List of Figures

Chapter 1

Introduction

"The world's most valuable resource is no longer oil, but data". According to the economist article [1] in 2017, data is the oil of the digital era. Nowadays, it is expected to see everything of material significance on the planet to be sensor tagged and connected to the internet in order to report its state in real-time [2]. These information-sensing Internet of Things (IoT) devices make data ubiquitous and massive. The raw data itself, nonetheless, is not lucrative but the analytics offers value [3]. Moreover, the characteristics of this massive data (often referred as Big Data) such as high volume and high rate, make the traditional state-of-the-art analysis approaches inefficient and impractical. This inefficiency of traditional approaches, has begun a new wave of scientific revolution and led to innovative analytics.

It is right to say that Big Data has changed the direction of the development of the hardware architecture as well as software [4]. Motivated by scalability demands of analytical methods, the number of cores in processors is being increased to support scalable parallel computing. Moreover, the *cloud computing* paradigm has emerged as one step towards revolutionizing distributed computing to enable convenient and on demand network access to a shared pool of resources (e.g. servers, storage). In IoT applications, the main motivation behind employing *virtual clouds* consists of high-end servers is to carry out heavy data analysis. The reason is that IoT devices are often equipped with reduced computational power, i.e. resource-constrained devices, and hence likely to be less efficient in performing heavy analysis. The benefits of cloud computing are nonetheless followed by challenges. The first challenge is that, when dealing with Big Data, it might be impossible to send all data to the cloud without exhausting the available bandwidth. Also, sending data to the cloud for analysis and getting back the results to make further decisions, could cause significant high latency which is not tolerable for certain applications requiring real-time processing.

The difficulties of shifting all data to the cloud for analysis led to the emergence of the paradigms such as *fog computing* and *edge computing* in which the processing procedures are being pushed down closer to where data originates. While edge computing is not a new concept, it mostly had been used to filter and route data to the cloud. Today, however, edge computing is offering more compute and analytic powers on devices. Similar to the edge

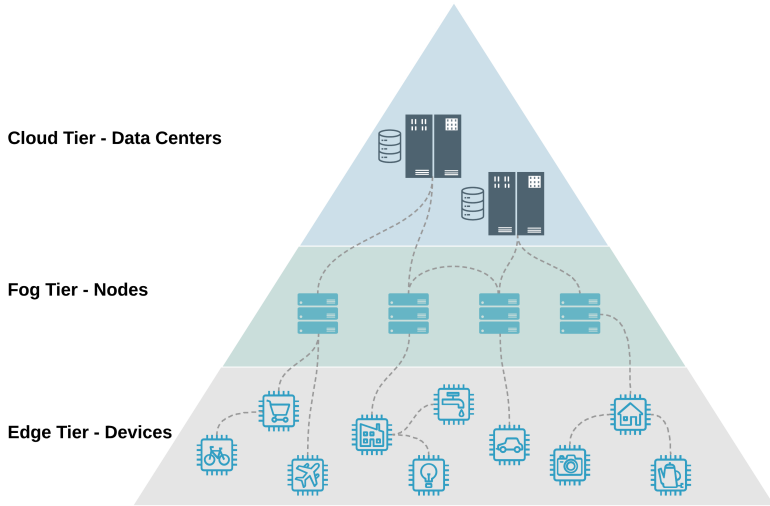


Figure 1.1: The 3-tier architecture including cloud, fog, and edge layers.

computing, fog computing is pushing the analysis from cloud servers down, closer to the source of data. In fog computing, the analysis is performed on intermediate levels, i.e. fog nodes or IoT gateways. Bringing cloud, fog, and edge computing together results in a 3-tier architecture, as shown in Figure 1.1, where the distribution of data processing is enabled along the *Things-to-Cloud* continuum.

Where to process the data? The edge/fog/cloud architecture provides suitable platforms for various processing applications with different requirements. For instance, applications requiring instantaneous decision making and real-time processing are better matched on the edge layer while the cloud computing is more suitable for those that are related to Big Data analytics or less sensitive to the response time. Furthermore, different tiers of the architecture offer platforms with specific capabilities. For example, edge devices often have limited computational power with limited memory capacity while cloud consists of high-end servers.

When to process the data? The increasing speed of the data generation demands some form of analytics to process data before it becomes too big. The accumulated data can cause several challenges from latency in producing the processing results to the difficulties in memory usages. To this end, regardless of which layer the analysis is running on, there is a need to process the constant flow of data continuously. The *continuous processing* improves memory access patterns as well as real-time processing, which is a requirement for many of the applications (e.g. e-commerce order processing, failure detection, air traffic control). An example of continuous processing is *data stream processing*, or shortly data streaming [5].

How to process the data? Data streaming is a processing paradigm, particularly suited for applications with small delay constraint to process massive amount of data. It performs analysis on data in motion, as the latter is received,

without need to be stored. Prior to the stream processing, the data needed to be stored in a database or other forms of storage and only after that, the analytical queries were performed over batch of the stored data. Nowadays, the conventional store-then-process techniques are not enough for many application due to the performance and storage constraints.

Stream processing introduces a new paradigm in which the queries exist continuously while the data flows through them. In another words, it provides continuous analysis in which the data is modeled as transient data streams rather than persistent relations. For example, by using stream processing and querying the constant receiving data from a temperature sensor, it is possible to raise an alarm once the temperature reaches a certain threshold. The benefits of stream processing, nonetheless, come with a price: the need for designing efficient streaming analysis while dealing with the continuous arrival of data in rapid, multiple and unbounded streams with various rates over time.

In this thesis, we contribute to the body of research on efficient and practical data analytics to achieve higher throughput of results with lower processing response time. We design algorithms for continuous data analysis that do not rely on platform-specific constructs, so to be suitable for various architectures deployed on different layers of the 3-tier architecture. We present efficient data structures to leverage parallelism and contribute to the online analytics. In addition, we explore the adaptive reconfiguration of processing units in order to fully utilize the resources and prevent the unnecessary latency for over-provisioning as well as prevent saturation of the system for under-provisioning.

The rest of this chapter is organized as follows. Section 1.1 reviews the introductory background about IoT data characteristics, multicore systems with the consequent parallelism, and data stream processing. Section 1.2 discusses the detailed thesis scope by describing problem domains and their corresponding challenges. Section 1.3 presents the highlights of contributions in the thesis to solve the aforementioned challenges. Finally, Section 1.4 concludes the discussion of this chapter.

1.1 Background

In this section we overview the background topics which are related to the scope of this thesis. First, we discuss the characteristics of data in IoT applications and the reasons that make the processing challenging for it. Then, we provide background on multicore systems and how they support parallelism as tools to enable efficient data processing. Finally, we review data stream processing and how it differs from conventional data processing so to be suitable for IoT applications.

1.1.1 IoT Data Features

Since 2011, the emergence of Cyber-Physical Systems and Industry 4.0 [6] have rapidly accelerated the growth of data and made Big Data a trend in industry. The main intention was based on the desire to enhance the productivity and make evolutionary progresses [7]. This gives immense opportunities to the research community to introduce innovative, practical, and efficient analytics in order to help the industry achieve its goals. Big data analytics is a form

of advanced data mining (i.e. Knowledge Discovery from Data [8]) that has progressed gradually to turn a large collection of data into knowledge. To distinguish the Big Data analytics from the conventional ones, it is necessary to define Big Data and its features.

By hearing the term Big Data, normally the first impression is about its size. However, it involves several dimensions while size is only one. Although there has been a divergent discourse on the exact definitions for Big Data during past years, all share a few dimensions based on the *"three Vs"* suggested by Doug Laney [9]. Laney suggested the Volume, Velocity, and Variety, as three dimensions of challenges in data management. Later, Gartner [10] gave a more detailed definition as:

> "Big Data is high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation."

In the following, each dimension is described briefly.

Volume refers to the size of the data. As also appeared in the name, Big Data is enormous amount of data. It is, nevertheless, impractical to define a threshold for the size of a dataset to be considered as Big Data. The reason is that the volume dimension along the other dimensions, i.e. velocity and variety, categorizes a dataset as Big Data. In some cases (e.g. sensor readings), data is in form of unbounded streams with infinite size which can be also categorized as Big Data.

Velocity refers to the speed of generating data. In recent years, using the well developed and ubiquitous technological tools, the generating rate of data is unprecedented. However, the term of velocity refers to not only the speed of incoming data, but also the speed that the data should be processed. Therefore, due to the increasing rate at which data is generated, there is a growing demand for real-time processing [11].

Variety refers to the range of the data types. Big Data covers the various data types of the spectrum from fully structured (e.g. tabular data) that can be easily sorted and stored, to unstructured (e.g. video, audio) that is random and difficult for the machine to analyze. The high variety of the data is what makes data "really big" [12].

In the literature, various big data analytics are introduced for different IoT applications [4], among all, the online near real-time processing is the interest of this thesis. Such analytics are for applications in which timeliness of the response is at the top priority. In these applications typically the input data is changing constantly and rapidly (e.g. sensor data) where the stream flow rate may vary considerably. To achieve the goals, online analytics need to utilize all the available computational resources, and consequently leverage techniques such as parallel computing and stream processing [13].

1.1.2 Parallelism and Multicore Processing

The demand for more computing power, encouraged processor designers to improve the performance of a processor by increasing the clock rate. This strategy worked fine until it hit the physical limit (i.e. power wall) which showed

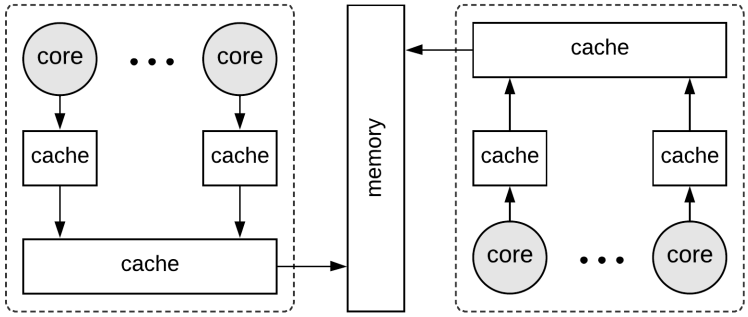


Figure 1.2: Classical architecture for shared-memory multicore systems

that the power dissipation does not scale with the size of a transistor. After crashing into the power wall, the designing approach shifted from employing a power-inefficient processor to usage of many efficient ones on the same chip, and hence promoting parallel computing rather than sequential programming [14]. While parallel computing is not new, during the recent years the interest in it has increased due to multicores becoming the norm in computing systems, ranging from smaller devices such as phones, to high-end servers. Therefore, it is true to say that the interest in parallel computing nowadays, is not the result of an innovation in programming but actual limitations in building power-efficient, high clock-rate, single core chips [15].

A processor on a chip hosting multiple computational units (i.e. *cores*), is referred as multicore processor. A system containing one or more multicore processors is referred as multicore system. Such systems enable parallelism (i.e. multi-threaded executions) by running tasks on different cores. Ideally, a large task is divided into several small independent sub-tasks, each is assigned to a core to be completed. However, distributing sub-tasks over cores is not always trivial. In many cases, sub-tasks have correlations that require the overlapping execution of *processes/threads* to be synchronized through the shared memory. In the rest of this thesis, we use the terms process and thread interchangeably to refer to an execution entity that is running on the processor.

Figure 1.2 illustrates a classical architecture for shared-memory multicore systems including multiple processors with several cores, connected via a shared memory, where each core may have several private and shared caches. The cores execute tasks independently and coordinate via a shared address space to which each core has access with different layer of memory hierarchy.

In parallel computing, concurrent access of cores to a data object shared in the memory, needs to be synchronized in order to guarantee data consistency. This is provided by several *synchronization* mechanisms (e.g. locks, semaphores) and hardware primitives (e.g. compare-and-swap, test-and-set) using which, the notion of *atomic operations* is supported. Atomic operations are the ones that appear to be completed by a thread without any interference [16].

Shared data objects can vary, from basic ones, such as registers, to higher level objects described through *Abstract Data Types* (ADTs). ADT is an interface definition of operations that can be executed on the data structure.

An algorithmic implementation of ADT in the shared-memory system is a *concurrent data structure* which organizes data for efficient concurrent access while hiding details on the interaction of the processes. An efficient concurrent data structure is a key to harness the available parallelism in multicore systems by providing correct synchronization with high-level of interface operations.

Design and implementation of concurrent data structures, nonetheless, are challenging as they are required to be correct and scalable [16]. The correctness of such data structures is proved through the *safety* and *liveness* properties [17]. The safety property describes the consistency guarantees by showing that "something bad will not happen", while the liveness property describes the progress guarantees by showing "something good will eventually happen".

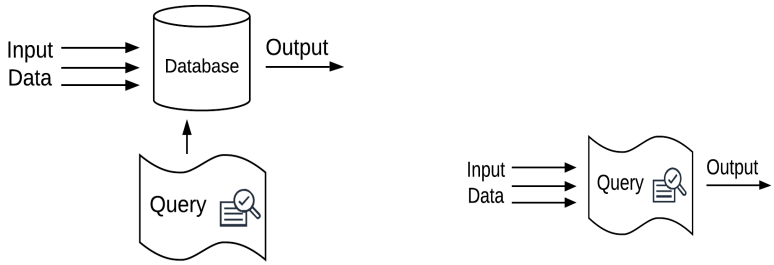
Various formalizations are presented in the literature for the safety property such as *linearizability* [18] and *sequential consistency* [19]. Linearizability preserves real-time occurrence of the operations and requires that each operation takes effect instantaneously at some point (i.e. linearization point) between its invocation and response. An execution is linearizable if there exists an ordered sequence of invocation and response events (i.e. *history*) that observes real-time ordering of the latter at all processes. However, in some cases, the real-time order of events at different processes may not be significant. In such situations, sequential consistency is used as the safety condition. Instead of real-time order of events, sequential consistency preserves the program order of operations issued by the same process. Sequential consistency is a weaker condition compared to the linearizability since every linearizable execution also provide sequential consistency but the reverse is not necessarily true.

Similarly, the liveness property is defined through various formalizations such as *wait-freedom* [20] and *lock-freedom* [16]. Wait-freedom is the strongest progress guarantee which concerns individual progress. It guarantees that *every* process has a bound on the number of steps to take before its operation completes regardless of delays or failures of other processes. Lock-freedom guarantees *some* process complete it operation after a bounded number of steps, and hence ensures the system-wide progress. During the past years, extensive effort has been made to construct more efficient and practical data structures [16, 21–24].

The new developments in hardware are not limited to high-end servers (e.g. Intel Xeon Phi [25]), but affect all computational devices, even the embedded resource-constraint ones (e.g. Odroid XU4 [26]). For instance, the embedded devices used for edge computing are now supporting parallel programming [26] by employing multicore architecture. This motivates the design and implementations of high-level parallel and heterogeneous programming models which can be adapted to any platform in the IoT architecture [27–29].

1.1.3 Data Stream Processing

Traditional database management systems (DBMSs) have been used for decades to manage data. The primary goal of DBMSs is to store data in a form of persistent dataset and then run *one-time* queries over it. As an example, consider using a DBMS in a university to maintain information about courses, students, and grades. Upon receiving a new data record (referred as tuple in the remaining), containing *CourseID*, *StudentID*, and *Grade* fields, a DBMS



(a) One-time query in DBMSs

(b) Continuous query in stream processing

Figure 1.3: Information processing

stores it in the database. Then, once a request made by a user of a DBMS to process a query such as *listing all the courses that a specific student has failed*, DBMS retrieves information from the storage and run the query to produce the results.

Although DBMSs are useful in applications where the updates of the database are relatively infrequent, they are inefficient for modern high-volume and high-velocity data-driven applications where Big Data is being generated. In such applications, data is changing constantly which makes the frequent access to the storage costly. Furthermore, due to the need for storing tuples before processing, DBMS is impractical for applications requiring real-time analysis. In this case, considering the architecture of DBMSs, one option is to remove the requirement of storing tuples before processing, and therefore analyse tuples upon receiving them. This modification has emerged a new paradigm for continuous processing, which is referred as *data stream processing*.

Data stream processing is one pass analysis over the data on the fly. In contrast to DBMSs, stream processing employs *continuous queries* [30] which are queries that are issued once and continuously run over the flow of tuples. As an example, consider a scenario where a sensor is used to monitor speed of a vehicle and raise an alarm if it exceeds a certain threshold. Using stream processing, the query is issued once and run over constantly arriving tuples reporting speed of the vehicle. In this way, there is no need to store all the tuples for later processing. Figure 1.3 illustrates high level overview of information processing procedure using DBMSs and stream processing. As shown in the figure, stream processing can run the query immediately after a new tuple arrives which in turn enables online analysis. However, it faces issues in the way that tuples need to be processed.

A challenging issue regarding data stream processing lies in the fact that naturally, streaming data is unbounded. In addition, since stream processing does not store data, there is a requirement to keep a portion of it to run some queries that need past data. In the vehicle example, assume we are interested in finding the average speed during the past hour and raise an alarm if the average is above a threshold. Various models have been proposed to keep a portion of a stream data with differences in downgrading the importance of the older tuples [31]. Among all, *sliding window* [32] is one of the prominent

models in which only the most recent tuples that fit in a window are kept. The window itself can be either *time-based* or *tuple-based* and is defined by two parameters *size* and *advance*. In time-based windows, size is indicating the length of a window in time units and advance shows how much in time the window should go forward (e.g. to group latest tuples within a window of size 10 seconds every 3 seconds). In tuple-based windows, size is the length of a window in the order of number of tuples and advance indicates how many tuples the window should go forward (e.g. to group last 10 tuples every 3 receiving tuples).

To support the stream processing paradigm, *Stream Processing Engines* (SPEs) have emerged as a new class of system software. SPEs are generally providing high level programming interfaces to run continuous queries. They are modeled as Directed Acyclic Graphs (DAGs) where vertices are processing operators and directed edges are continuous streams of data. Examples of such systems are STREAM [33], Apache Storm [34], Apache Flink [35], StreamCloud [36]. The processing operators are either *stateless*, i.e. do not keep state as the result of previous tuples and perform actions based on each tuple individually (e.g. filter out the tuples using their attributes), or *stateful*, i.e. use state affected by the previous tuples to produce results (e.g. aggregate to compute average). Due to the unbounded nature of the streams, stateful operators are computed over windows. Data streaming can be used to implement analysis tools for a wide range of applications requiring simple analytics (e.g. filter, average) or more complicated ones (e.g. clustering, regression).

SPEs provide pipeline and task parallelism, naturally, by assigning tasks and operators to different processing units. However, in most cases, distributing operators on processing units causes imbalanced workload that affects system performance. Typically, the performance of SPEs is measured by two metrics; *throughput* and *latency*. Throughput indicates number of tuples processed per time unit while latency is the time difference between receiving a tuple and producing the corresponding results.

In order to achieve higher throughput and lower latency, SPEs enable data parallelism by replicating instances of operators and *partitioning* data over them. For stateless operators, the partitioning can be accomplished in an arbitrary fashion (e.g. random, round-robin) while stateful operators require techniques that are aware of states and route tuples, that affect the same state, to the same partition.

Partitioned operators leverage data parallelism by creating sub-streams and assigning the abnalysis of each sub-stream to a computational process. Therefore, the processing latency of each sub-stream is affected by its tuples and the specifications of the corresponding process. This might cause inconsistency in data received by the downstream operator if behaviour can be affected by the order. Similar to the correctness properties for concurrent data structures, discussed in Section 1.1.2, streaming consistency needs to be discussed for data stream processing. One way to formalize the streaming consistency is through the *determinism* property.

Determinism is required to ensure the consistent results for different executions with the same sequence of input tuples. To support determinism, one option is to merge the input streams into one timestamp-sorted stream and then, process tuples from the latter in the timestamp order. For that, concur-

rent data structures can be used to coordinate all processes involved in the procedure [37, 38]. ScaleGate [24, 39, 40] and W-Hive [24, 40] are examples of such data structures that have been proposed to enable communication among operators in SPEs and support determinism. ScaleGate efficiently merges several timestamp-sorted streams into one and allows the operator instance to process tuples in timestamp order once they are "ready". A tuple is defined as ready to be processed, if its timestamp is less than or equal to the latest tuple timestamps received from all input streams [39].

In a nutshell, ScaleGate enhances data parallelism while supporting consistency. It provides an encapsulated object, allowing for an arbitrary number of sources to add their input tuples while arbitrary number of reader entities consume the ready tuples. Furthermore, it supports determinism since all the reader entities read the tuples in the timestamp order regardless of the order of input tuples.

1.2 Research Questions and Challenges

This thesis investigates aspects of computational infrastructures and development of tools for IoT data analytics in the edge/fog/cloud architecture to enhance parallelism while preserving determinism, with the focus on two problem domains: (i) continuous processing, and (ii) elasticity in stream processing.

1.2.1 Continuous Processing and Analysis

When focusing on data analysis for IoT applications, the major challenge is given by the volume and velocity of the data that is produced using devices (e.g. sensors) at the edge. This requires efficient analytical solutions to distribute the processing procedure over physical computational resources deployed at different tiers. In this way, the system has the opportunity to sustain the data rate and prevent the communication bandwidth exhaust. Another challenge is given by the computational and memory limitations of devices at the edge, which requires fine-grained analysis to enhance the usage of resources.

As discussed in Section 1.1.3, continuous analysis is proposed as a general approach to address these challenges. Designing algorithms for continuous clustering is nonetheless not a trivial task [31]. Continuous processing requires efficient data structures shared by threads for concurrent and frequent accesses in order to support parallelism without adding an extra overhead to the processing time. Moreover, it provides online single pass analysis of data which means at any moment in time, the analysis might have incomplete information.

In this problem domain, we focus on *data clustering*, as an important representative example in data analysis context and a key-component for other problems (e.g. machine learning, pattern recognition, information retrieval). Data clustering is the process of grouping data into sets (i.e. clusters) using similarity metrics, in a way that the intra-cluster similarity is maximized. An extensive literature exists about clustering, which can be classified based on the semantics used to define a cluster [8] (e.g. k-means approach to produce balanced spherical-shape clusters, density-based clustering [41] to partition sets into unknown number of arbitrarily shaped clusters).

During the past decade, the problem of clustering for high-rate flow of data for different applications has been widely studied [42]. Generally, the studies indicate the trade-off between accuracy and performance for continuous clustering. For some applications that the results need to be accurate, typically data is collected in batches and then the clustering algorithm is performed [43]. However, this introduces latency into the processing. Therefore, if the accuracy is not top priority, approximation algorithms [31, 44] are run over data continuously to achieve higher performance with an accuracy loss on the clustering.

Based on the above mentioned challenges, the general research questions in the domain, which this thesis investigates, are:

Can we design algorithms for continuous clustering (i) to be scalable and therefore suitable for IoT systems, (ii) to be hardware independent thus to be deployed on various platforms, while (iii) producing accurate results with high performance? Can we provide a solution, in case of limited communication bandwidth compared to the volume of sensed data, to (iv) integrate approximation techniques with continuous clustering, while (v) having a small accuracy loss?

Section 1.3.1 gives a brief overview of the proposed solutions in the thesis that aim at addressing these issues. Later, Chapters 2 and 3 describe the contributed results in detail.

1.2.2 Elasticity in Stream Processing

In the concept of stream processing, there is a rich scientific literature on leveraging the modern multicore systems' computational power in terms of parallelism where the number of parallel instances is fixed. Such techniques focus on a specific operator parallelization [39, 45, 46] or determinism [38, 39], among other aspects. These fixed resources, nonetheless, are not able to be adjusted at runtime and therefore render over- or under-provisioning scenarios in the case of changing stream rates [47]. In the over-provisioning scenario, the number of allocated resources is set regarding the load peaks, which causes the under-utilization of the system and high cost during non-peak times. In the under-provisioning scenario, there are not enough resources, which causes higher analysis latency and, in the case of bursty workload, saturation in the system [48]. Therefore, in addition to scalability, *elasticity* has to be provided in order to enhance resource utilization in the system.

The elasticity mechanism enables runtime resource adjustments to adapt the number of parallel computational instances regarding the stream rate. It introduces challenges especially for the stateful analysis where redistribution of work among new number of resources is often followed by state migration [48, 49]. *State migration* is the procedure of transferring the current state from one thread to the new one, as the latter has been running the analysis. While state migration is particularly needed in distributed shared-nothing systems, it hampers the performance unnecessarily, if parallel instances have the option of sharing memory. On the other hand, to take advantage of intra-node resources (e.g. shared-memory), it is required to efficiently manage and scale computation on a node level first (i.e. scaling up) before scaling out to the distributed nodes [50].

Another challenge in programming elastic stateful analysis, relies on pre-

serving determinism even with varying degree of parallelism. Preserving determinism in elastic analysis, in turn, might require elastic and re-configurable data structures.

In the second problem domain, we focus on elasticity in stream processing to address its discussed challenges, which yield new and interesting research questions as follows:

Can we support elastic stateful analysis and prioritizes scaling up to utilize intra-node resources, rather than scaling out, while (i) enabling deterministic execution, (ii) safely sharing state with all parallel instances to enhance performance, and accordingly, (iii) maximizing efficiency in terms of throughput, latency and reconfiguration times?

This thesis investigates the solutions to address the above mentioned issues. Section 1.3.2 overviews the proposed solution while more details are presented in Chapter 4.

1.3 Thesis Contributions

The contributions of this thesis are solutions for a number of data analytical challenges in the problem domains mentioned in the previous section. In this section, we outline the results and more details are presented in Chapter 2 to Chapter 4.

1.3.1 Efficient Continuous Data Clustering

An inspiring yet challenging application for continuous data processing in the context of IoT is data clustering for LiDAR (Light Detection And Ranging) sensor. LiDAR is a high rate sensor that generates massive amount of data in form of large streams of readings. A LiDAR commonly mounts several lasers on a rotating column. At each rotation step, these lasers shoot light rays and, based on the time the reflected rays take to reach back to the sensor, produce distance readings. Therefore, each laser in the LiDAR generates stream of data. Combining all lasers of the LiDAR, produces a stream of distance readings as 3D points at high rates (known as point clouds), in the realm of millions of readings per second.

Due to the high data rates and sensitivity of the applications, the LiDAR point cloud clustering algorithms are required to be fast and accurate. For that, in Chapter 2, we introduce *Lisco*, a single-pass continuous Euclidean distance-based clustering, to process LiDAR point cloud in near real-time. Lisco exploits the inherent ordering of LiDAR data points to achieve the same accuracy as state-of-the-art batch-based clustering algorithm while decreasing the latency. Furthermore, the parallel version of Lisco, *P-Lisco*, is proposed to exploit the internal pipeline of the data analysis steps. Both Lisco and P-Lisco algorithms are architecture independent and thus, suitable to run on any platform.

Later, in Chapter 3, we present a framework, called *DRIVEN* (Data Retrieval and clusterIng in VEhicular Networks), to utilize computational resources and communication bandwidth of different tiers in the edge/fog/cloud architecture in the context of vehicular networks. The framework applies a continuous bounded error approximation through Piecewise Linear Approximation [51], to

compress volumes of data before forwarding to the next tier. It also performs the generalized form of Lisco leveraging the inherent ordering of the spatial and temporal data being collected in a continuous fashion, as the analysis tool. The framework is tested using real-world LiDAR and GPS data to study the trade-offs in compression between the speed up, both for data processing and data transferring, and accuracy of the clustering results.

1.3.2 Scalable and Elastic Deterministic Data Stream Analysis

The challenges discussed in Section 1.2.2, address the trade-off between data sharing and shared-nothing scenarios among threads to enhance parallelism as well as synchronization. In case of shared-nothing, the threads work independently and parallelism is maximized, but the reconfiguration and state migration are costly. In the case of data sharing, there is no need for state migration and the workload shifting among threads can be done efficiently, but contention, which is caused by concurrent access of threads, might affect the scalability. Therefore, there is a need to find the right amount of sharing to enable efficient reconfiguration and workload redistribution as well as enhancing parallelism and performance while preserving determinism.

Towards this goal, in Chapter 4, we introduce the concept of "virtual shared-nothing" parallelization and leverage it in the proposed framework, called *STRETCH*. The framework allows users to exploit parallelization techniques and take advantage of shared-memory synchronization to boost the scaling up of streaming applications. We also extend the API of ScaleGate to allow for live provisioning and decommissioning of both source and readers entities while preserving determinism. Furthermore, we provide a fully-implemented prototype and empirical evidence that the *STRETCH*'s virtual shared-nothing parallelism allows for better performance than existing shared-nothing ones. It is also shown that *STRETCH* framework enables fast elastic reconfiguration in case of requiring thread provisioning or decommissioning.

1.4 Conclusions and Future Work

This thesis targets the challenges of data analysis and computational infrastructures in the context of Internet of Things by focusing on two problem domains; continuous analysis and elasticity in stream processing.

For the first domain, we focus on continuous clustering algorithms to rapidly extract valuable information from raw data that come in high-rate streams. We have presented the Lisco algorithm to process LiDAR point clouds while the latter are being collected. Then, we present its parallel counterpart, P-Lisco, to leverage the parallelism of processing pipeline in an architecture-independent fashion. As the next step in this domain, we presented the *DRIVEN* framework for data retrieval and clustering in vehicular networks. The framework provides efficient information retrieval using Piecewise Linear Approximation to compress the data stream in a continuous fashion. It also utilizes Lisco's properties to generalize its use and perform appropriate clustering as the analysis tool.

For the second domain, we focus on elasticity capability in stream processing to handle the bursty data streams. For that, we present the *STRETCH*

framework which enables virtual shared-nothing parallelism and provides extra efficient reconfiguration when scaling is required. Moreover, it maximizes throughput and latency of stateful analysis while supporting determinism even with varying degree of parallelism in the runtime.

Based on insights from work in elasticity characteristic of data streaming, there are interesting aspects worth to be investigated as followup work. As an example, would it be possible to combine scale-out capabilities with *STRETCH*'s scale-up mechanism? This would provide fast reconfiguration when needed as well as preserving determinism. Another interesting direction concerns the scaling control policy. Currently, *STRETCH* uses a manual controller which can be substituted with an intelligent automatic one, to trigger reconfiguration for provisioning or decommissioning threads based on the data rate and available resources.

Data streaming with elasticity capability can be applied for many applications requiring varying amount of resources over time to cope with fluctuations. Examples of such applications include analysis regarding social media (e.g. twitter), where an event or occurrence may cause a huge peak in data generation.

Furthermore, the achievements of Lisco algorithm for LiDAR motivate investigating the properties of the data in general, that can enable continuous and parallel processing. It also inspires employing continuous processing for several applications dealing with challenges related big IoT data. For instance, new and upcoming production systems are empowered with sensing and communication devices, which generate massive data. Efficiently processing this massive data, may improve the functionality of the system and increase the productivity. A specific example in this regard is monitoring the throughput and detect the bottlenecks (i.e. limiting factors) in a production line. The reason is that an improvement in bottleneck detection procedure results in an improvement in real-time production control which in turn facilitates the implementation of improvement actions in the production system.

Another application that can benefit from data streaming is online object tracking. This application is particularly interesting for security and safety reasons. For example, using data streaming, a moving object in a prohibited area can be detected online so to react properly. Moreover, continuous processing can facilitate the detection of an object for moving vehicles and hence allows them to plan better for avoiding the obstacle.

Data streaming can also be applied for validation and quality improvement on data before performing analysis. This pre-processing approach, can improve the quality of analysis, as well decrease the size of data before transferring via a communication link to other tiers or storing in the database.

Bibliography

[1] "The world's most valuable resource is no longer oil, but data," 2017, https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data.

[2] M. Stonebraker, U. Çetintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," *ACM Sigmod Record*, vol. 34, no. 4, pp. 42–47, 2005.

[3] S. LaValle, E. Lesser, R. Shockley, M. S. Hopkins, and N. Kruschwitz, "Big data, analytics and the path from insights to value," *MIT sloan management review*, vol. 52, no. 2, p. 21, 2011.

[4] C. P. Chen and C.-Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on big data," *Information sciences*, vol. 275, pp. 314–347, 2014.

[5] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2002, pp. 1–16.

[6] N. Jazdi, "Cyber physical systems in the context of industry 4.0," in *2014 IEEE international conference on automation, quality and testing, robotics*. IEEE, 2014, pp. 1–4.

[7] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers, "Big data: The next frontier for innovation, competition, and productivity," 2011.

[8] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[9] D. Laney, "3-d data management: Controlling data volume, velocity and variety," 2001, http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf.

[10] G. I. Glossary, 2019, https://www.gartner.com/it-glossary/big-data/.

[11] A. Gandomi and M. Haider, "Beyond the hype: Big data concepts, methods, and analytics," *International Journal of Information Management*, vol. 35, no. 2, pp. 137–144, 2015.

[12] S. Sagiroglu and D. Sinanc, "Big data: A review," in *2013 International Conference on Collaboration Technologies and Systems (CTS)*. IEEE, 2013, pp. 42–47.

[13] M. Marjani, F. Nasaruddin, A. Gani, A. Karim, I. A. T. Hashem, A. Siddiqa, and I. Yaqoob, "Big iot data analytics: architecture, opportunities, and open research challenges," *IEEE Access*, vol. 5, pp. 5247–5261, 2017.

[14] S. Akhter and J. Roberts, *Multi-core programming*. Intel press Hillsboro, 2006, vol. 33.

[15] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek *et al.*, "A view of the parallel computing landscape," *Communications of the ACM*, vol. 52, no. 10, pp. 56–67, 2009.

[16] M. Herlihy and N. Shavit, *The art of multiprocessor programming.* Morgan Kaufmann, 2011.

[17] L. Lamport, "Proving the correctness of multiprocess programs," *IEEE transactions on software engineering*, no. 2, pp. 125–143, 1977.

[18] M. P. Herlihy and J. M. Wing, "Linearizability: A correctness condition for concurrent objects," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 12, no. 3, pp. 463–492, 1990.

[19] L. Lamport, "How to make a correct multiprocess program execute correctly on a multiprocessor," *IEEE Transactions on Computers*, vol. 46, no. 7, pp. 779–782, 1997.

[20] M. Herlihy, "Wait-free synchronization," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 13, no. 1, pp. 124–149, 1991.

[21] D. Cederman, A. Gidenstam, P. Ha, H. Sundell, M. Papatriantafilou, and P. Tsigas, "Lock-free concurrent data structures," *Programming Multicore and Many-core Computing Systems*, vol. 86, p. 59, 2017.

[22] D. Cederman, B. Chatterjee, N. Nguyen, Y. Nikolakopoulos, M. Papatriantafilou, and P. Tsigas, "A study of the behavior of synchronization methods in commonly used languages and systems," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing.* IEEE, 2013, pp. 1309–1320.

[23] Y. Nikolakopoulos, A. Gidenstam, M. Papatriantafilou, and P. Tsigas, "A consistency framework for iteration operations in concurrent data structures," in *2015 IEEE International Parallel and Distributed Processing Symposium.* IEEE, 2015, pp. 239–248.

[24] V. Gulisano, Y. Nikolakopoulos, D. Cederman, M. Papatriantafilou, and P. Tsigas, "Efficient data streaming multiway aggregation through concurrent algorithmic designs and new abstract data types," *ACM Transactions on Parallel Computing (TOPC)*, vol. 4, no. 2, p. 11, 2017.

[25] Intel. (2019) Xeon-phi. https://www.intel.com/.

[26] Hardkernel. (2019) Odroid-xu4. https://www.hardkernel.com.

[27] A. Ernstsson, L. Li, and C. Kessler, "Skepu 2: Flexible and type-safe skeleton programming for heterogeneous parallel systems," *International Journal of Parallel Programming*, vol. 46, no. 1, pp. 62–80, 2018.

[28] Y. Nikolakopoulos, M. Papatriantafilou, P. Brauer, M. Lundqvist, V. Gulisano, and P. Tsigas, "Highly concurrent stream synchronization in many-core embedded systems," in *Proceedings of the Third ACM International Workshop on Many-core Embedded Systems.* ACM, 2016, pp. 2–9.

[29] I. Walulya, Y. Nikolakopoulos, M. Papatriantafilou, and P. Tsigas, "Concurrent data structures in architectures with limited shared memory support," in *European Conference on Parallel Processing*. Springer, 2014, pp. 189–200.

[30] S. Babu and J. Widom, "Continuous queries over data streams," *ACM Sigmod Record*, vol. 30, no. 3, pp. 109–120, 2001.

[31] M. Garofalakis, J. Gehrke, and R. Rastogi, *Data Stream Management: Processing High-Speed Data Streams*. Springer, 2016.

[32] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows," *SIAM journal on computing*, vol. 31, no. 6, pp. 1794–1813, 2002.

[33] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma *et al.*, "Stream: The stanford stream data manager," *IEEE Data Eng. Bull.*, vol. 26, no. 1, pp. 19–26, 2003.

[34] "Apache Storm," http://storm.apache.org, 2019.

[35] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 36, no. 4, 2015.

[36] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, C. Soriente, and P. Valduriez, "Streamcloud: An elastic and scalable data streaming system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 12, pp. 2351–2365, 2012.

[37] V. Gulisano, Y. Nikolakopoulos, M. Papatriantafilou, and P. Tsigas, "Data-streaming and concurrent data-object co-design: Overview and algorithmic challenges," in *Algorithms, Probability, Networks, and Games*. Springer, 2015, pp. 242–260.

[38] I. Walulya, D. Palyvos-Giannas, Y. Nikolakopoulos, V. Gulisano, M. Papatriantafilou, and P. Tsigas, "Viper: A module for communication-layer determinism and scaling in low-latency stream processing," *Future Generation Computer Systems*, vol. 88, pp. 297–308, 2018.

[39] V. Gulisano, Y. Nikolakopoulos, M. Papatriantafilou, and P. Tsigas, "Scalejoin: A deterministic, disjoint-parallel and skew-resilient stream join," *IEEE Transactions on Big Data*, 2016.

[40] D. Cederman, V. Gulisano, Y. Nikolakopoulos, M. Papatriantafilou, and P. Tsigas, "Brief announcement: concurrent data structures for efficient streaming aggregation," in *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures*. ACM, 2014, pp. 76–78.

[41] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, 1996, pp. 226–231.

[42] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. De Carvalho, and J. Gama, "Data stream clustering: A survey," *ACM Computing Surveys (CSUR)*, vol. 46, no. 1, p. 13, 2013.

[43] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz, "Close-range scene segmentation and reconstruction of 3d point cloud maps for mobile manipulation in domestic environments," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on.* IEEE, 2009, pp. 1–6.

[44] G. Cormode, M. Garofalakis, P. J. Haas, C. Jermaine *et al.*, "Synopses for massive data: Samples, histograms, wavelets, sketches," *Foundations and Trends® in Databases*, vol. 4, no. 1–3, pp. 1–294, 2011.

[45] B. Gedik, R. R. Bordawekar, and P. S. Yu, "Celljoin: a parallel stream join operator for the cell processor," *The VLDB Journal—The International Journal on Very Large Data Bases*, vol. 18, no. 2, pp. 501–519, 2009.

[46] P. Roy, J. Teubner, and R. Gemulla, "Low-latency handshake join," *Proceedings of the VLDB Endowment*, vol. 7, no. 9, pp. 709–720, 2014.

[47] C. Hochreiner, M. Vögler, S. Schulte, and S. Dustdar, "Elastic stream processing for the internet of things," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD).* IEEE, 2016, pp. 100–107.

[48] V. Cardellini, M. Nardelli, and D. Luzi, "Elastic stateful stream processing in storm," in *2016 International Conference on High Performance Computing & Simulation (HPCS).* IEEE, 2016, pp. 583–590.

[49] B. Gedik, "Partitioning functions for stateful data parallelism in stream processing," *The VLDB Journal—The International Journal on Very Large Data Bases*, vol. 23, no. 4, pp. 517–539, 2014.

[50] P. B. Gibbons, "Big data: Scale down, scale up, scale out." in *IPDPS*, 2015, p. 3.

[51] R. Duvignau, V. Gulisano, M. Papatriantafilou, and V. Savic, "Piecewise linear approximation in data streaming: Algorithmic implementations and experimental analysis," *arXiv preprint arXiv:1808.08877*, 2018.