Mining Big and Fast Data: Algorithms and Optimizations for Real-Time Data Processing

MUHAMMAD ANIS UDDIN NASIR

Doctoral Thesis in Information and Communication Technology
School of Electrical Engineering
and Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden 2018

KTH School of Electrical Engineering
and Computer Science
SE-164 40 Kista
SWEDEN

TRITA-EECS-AVL-2018:27
ISBN 978-91-7729-729-1

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framläggges till offentlig granskning för avläggande av doktorsexamen i informations och kommunikationsteknik måndagen den 7 maj 2018 klockan 09:00 i Sal B, Electrum, Kungl Tekniska högskolan, Kistagången 16, Kista.

Tryck: Universitetsservice US AB

Abstract

In the last decade, real-time data processing has attracted much attention from both academic community and industry, as the meaning of big data has evolved to incorporate as well the speed of data. The massive and rapid production of data comes via numerous services, i.e., Web, social networks, Internet of Things (IoT) and mobile devices. For instance, global positioning systems are producing continuous data points using various location-based services. IoT devices are continuously monitoring variety of parameters, like temperature, heart beats, and others, and sending the data over the network. Moreover, part of the data produced by these real-time services is linked-data that requires tools for streaming graph analytics. Real-time graphs are ubiquitous in many fields, from the web advertising to bio-analytics. Developing analytical tools to process this amount of information at a real-time is challenging, yet extremely essential, for developing new services in areas such as web analytics, e-health and marketing.

Distributed stream processing engines (DSPEs) are often employed for real-time data processing, as they distribute work to many machines to achieve the required performance guarantees, i.e., low latency and high throughput. However, the scalability of DSPEs is often questioned when the input streams are skewed or the underlying resources are heterogeneous. In this thesis, we perform a scalability study for DSPEs. In particular, we study the load-balancing problem for DSPEs, which is caused by the skewness in the workload and heterogeneity in the cluster. In doing so, we develop several efficient and accurate algorithms to reduce the load imbalance in a distributed system. Moreover, our algorithms are integrated into Apache Storm, which is an open source stream processing framework.

Another dimension of real-time data processing involves developing novel algorithms for graph-related problems. The later part of the thesis presents several algorithms for evolving graphs. One of the most interesting features of real-world networks is the presence of community structure, which divides a network into groups of nodes with dense connections internally and sparse connections between groups. We study the community detection problem in the fully dynamic settings by formulating it as a top-k densest subgraph problem. In doing so, we achieve an extremely efficient approximation algorithm that scales to graphs with billions of edges. Further, we study the top-k graph pattern-mining problem in fully dynamic settings and develop a probabilistic algorithm using reservoir sampling. We provide the theoretical analysis for the proposed algorithms and show via empirical evaluation that our algorithms achieve up to several orders of magnitude improvement compared to the state-of-the-art algorithm.

Keywords: Stream Processing, Load Balancing, Fully Dynamic Graphs, Real-Time Data Processing, Top-k Densest Subgraph, Frequent Subgraph Mining.

Sammanfattning

Under det senaste decenniet har databehandling i realtid dragit stor uppmärksamhet från såväl akademiker som från näringslivet, eftersom den storskaliga databehandlingen har utvecklats för att införliva också bearbetningshastigheten. Den massiva och snabba produktionen av data kommer via många tjänster, dvs webb, sociala nätverk, trådlösa sensorer och mobila enheter. Till exempel skapar Facebook-användare kontinuerligt nya inlägg och trender, vilket förändrar nätverksegenskaperna. Twitter-användare producerar inlägg med hög hastighet, vilket gör gamla inlägg mindre relevanta och ändrar nätverken av retweet och omnämnanden. Dessutom är en del av data som produceras av dessa dynamiska tjänster länkade data som kräver verktyg för strömmande grafanalys. Sådana högfrekventa länkade data är allestädes närvarande inom många områden, från webbannonsering och bioanalys. Att utveckla analysverktyg för att bearbeta denna mängd information med snabb hastighet är utmanande men ändå ytterst viktigt för att utveckla nya tjänster inom områden som webbanalys, e-hälsa och marknadsföring.

Distribuerade strömbehandlingsmotorer (DSPEs) används ofta för databehandling i realtid, eftersom de är kapabla att utnyttja en uppsättning maskiner för att uppnå nödvändiga prestandagarantier, dvs låg latens och hög genomströmning. Skalbarheten hos DSPEsär emellertid ofta ifrågasatt när ingångsströmmarna är obalanserade eller de underliggande resurserna är heterogena. I denna avhandling utför vi en skalbarhetsstudie för DSPEs. I synnerhet studerar vi lastbalanseringsproblemet för distribuerade strömbehandlingsmotorer, vilket orsakas av skevheten i arbetsbelastningen och heterogeniteten i klustret. Därigenom utvecklar vi flera effektiva och korrekta algoritmer för att minska belastningsobalansen i ett distribuerat system. Dessutom integreras våra algoritmer i Apache Storm, vilket är ett strömbehandlingsramverk implementerat med öppen källkod.

En annan dimension av realtidsdatabehandling innebär att man utvecklar nya algoritmer grafrelaterade problem. Den senare delen av avhandlingen presenterar flera algoritmer för utvecklande grafer. En av de mest intressanta funktionerna i verkliga nätverk är närvaron av gemenskapsstruktur, som delar ett nätverk i grupper av noder med täta anslutningar internt och sparsamma kopplingar mellan grupper. Vi studerar gemenskapsdetektionsproblemet i fullt dynamiska kontexter genom att formulera det som problemet att finna den topp-k-tätaste delgrafen. Därmed föreslår vi en extremt effektiv approximationsalgoritm för det problem som är skalbar till grafer med miljarder kanter. Vidare studerar vi detekteringsproblemet för topp-k-tätaste grafmönster i fullt dynamiska kontexter och utvecklar en probabilistisk algoritm med hjälp av reservoarsampling. Vi tillhandahåller den teoretiska analysen av de föreslagna algoritmerna och visar via empirisk utvärdering att våra algoritmer uppnår upp till flera storleksordningars förbättring jämfört med den senast föreslagna algoritmen.

To my parents

Acknowledgements

Firstly, I would like to express great appreciation and gratitude to my PhD advisor, Sarunas Girdzijauskas. I am profoundly thankful for his guidance and continuous support. I could not imagine being able to finish this PhD without his support. Further, I would like to offer my special thanks to my second advisor, Seif Haridi. His constructive discussions guided me towards practical research.

I consider myself extremely lucky to meet Gianmarco de Francisci Morales when I started my PhD. He has played an extremely important role in my technical development through his broad knowledge and expertise. I am sincerely grateful for his advice, guidance, patience and devoted time. I would like to express great indebtedness and gratitude for the guidance provided by Aristides Gionis. Moreover, I would like to express my gratitude towards Nicolas Kourtellis and all the other co-authors of the articles included in the thesis. Special thanks to Marco Serafini, David García-Soriano, Hiroshi Horii, Cigdem Aslay, Fatemeh Rahimian, Rudy Raymond, and Takayuki Osogami for all the constructive discussion, help and guidance.

Besides, I would like to express my gratitude towards my parents for their patience, and immeasurable understanding. Further, I would not be able to complete this long journey without the support of my brothers: Sufian Nasir and Subhan Nasir. Moreover, I would like to thank my good friends Aini Laine, Amira Soliman, Ghazanfar Salari, Mansoor Khurshid, Muhammad Ali Orakzai, Samia Khalid, Sofia Wessén, Taheer Ahmed, and Zeeshan Hyder. I would like also to thank my colleagues and friends at EECS, especially Ahsan Javed Awan, Alexandru A Ormenisan, Hooman Piero Sajjad, Kamal Hakimzadeh, Kambiz Ghoorchian, Lars Kroll, Leila Bahri, Mahmoud Ismail, Paris Carbone, Salman Niazi, Shatha Jaradat, Theodore Vasiloudis, Vasiliki Kalavri, and Zainab Abbass for all the scientific discussions. Further, I would like to thank Jim Dowling, Vladimir Vlassov, Christian Schulte, and Alf Thomas Sjöland. Also, I would like to thank the PhD office at EECS, particularly Sandra Gustavsson Nylén, Susy Mathew, Madeleine Printzsköld, and Emanuel Borg for their support during with administrative matters. .

Moreover, I would also like to take the opportunity to acknowledge the support I received during this work from the iSocial EU Marie Curie ITN project (FP7-PEOPLE-2012-ITN). Besides, I would like to thank Yahoo Labs Barcelona, Aalto University, and IBM Research for hosting me for internships during the PhD. Special thanks for all my EMDC friends including Mario, Manos, Ioanna, Andreia, and Zygimantas. Also, I would like to thank my friend Madoka Sahara for helping in moving to Tokyo. All anonymous reviewers of accepted and rejected papers, for providing observant and constructive feedback. The welcoming cities of Barcelona, Stockholm, Helsinki, and Tokyo, which hosted me during the past four years and all the wonderful people I met there.

Anis Nasir,
April 9, 2018

Contents

List of Acronyms

ACID	Atomicity, Consistency, Isolation, Durability
CG	Consistent Grouping
DAG	Directed Acyclic Graph
DSD	Densest Subgraph Discovery
DSPE	Distributed Stream Processing Engines
FD	Fully Dynamic
FSM	Frequent Subgraph Mining
HDFS	Hadoop Distributed File System
ICDE	International Conference in Data Engineering
IoT	Internet of Things
KDD	Knowledge Discovery and Data Mining
KG	Key Grouping
PEI	Processing Element Instance
PKG	Partial Key Grouping
PM	Pattern Mining
PoBC	Power of Both Choices
PoDC	Power of D Choices
PoRC	Power of Random Choices
PoTC	Power of Two Choices
PoWC	Power of W Choices
SG	Shuffle Grouping

Part I

Thesis Overview

Chapter 1

Introduction

Finding the problem is the hard part.

Kevin Systrom

Big Data–a buzzword from the last decade, which challenged many engineers, scientists, and researchers to invent novel ways of storing and processing the data. Big Data refers to the data that is too big, too fast, or too hard for existing tools to process [1]. It is understood as a multi-dimensional problem, with each dimension bringing a different set of challenges. Three of the most widely considered dimensions are: volume, velocity, and variety. Volume refers to the size of data, velocity refers to the speed of data arrival, and variety refers to the different types of data. While the multi-dimensionality of the data brings several challenges, it is of utter importance to be able to process the data to extract meaningful insights.

You might wonder at this point about the question: "Why are we interested in data?" Data in the raw form is just a batch of appended rows of bytes. Albeit we collect huge amount of data, the raw data does not provide much meaningful information. It is rather the value that is extracted from the analysis of the data that matters. Data contains variety of latent patterns that are of interest for understanding unprecedented data features. These patterns allow gaining deeper insight into data, which increases the knowledge and enables valuable future decisions.

In the last decade, most of the efforts in academic community and industry have been focused on handling and processing large volumes of data. The massive production of data comes via numerous services, i.e., internet search, social media, mobile devices, the internet of things, business transactions, next-generation radio astronomy telescopes, high-energy physics synchrotron, and content distribution. This data deluge requires reviewing the traditional storage and processing tools, as they fall short in providing the required guarantees, i.e., high availablity, fast processing, accruate output, and more. It is due to the fact that the volume and the required processing capabilities exceed the capacity of a single machine. As a result, many enterprises opt for horizontal scaling, which enables processing the large volume of data on set of commodity machines, usually running in data center environments. MapReduce is one of the most prominent data processing paradigms that is an outcome of the efforts in the field of large scale data processing. It is a programming model, which provides both storage and processing capabilities exploiting set of machines in a cluster. A MapReduce program consists of two procedures: *Map* and *Reduce*. A Map function transforms the input and generates output in the form of key/value pairs. Moreover, a Reduce function combines the output of a map function to produce the final output. Several other well-known frameworks include Spark, Dyrad, Giraph and others. Considering the presence of the set of large-scale data processing frameworks, handling large volume of data is becoming more accessible even for small companies and businesses.

Most of the aforementioned frameworks usually operate on bounded or batch data. In this model, the data is first stored on data-stores, e.g., distributed file systems or log-based systems. Later, a snapshot of the data is created and processed to extract useful information. This execution model is effective for large volume of data, e.g., daily reporting, data warehousing, training machine learning models, and more. However, such model is not an intuitive fit for the real-time data processing, which requires low latency between the data ingestion and output generation. Therefore, there is a solid necessity to review existing batch processing tools and develop new frameworks to meet the real-time data processing demands.

Real-time processing requires dealing with *Fast Data*, which is the data that is produced at a rapid rate. It requires low latency processing for applications where output should be generated within short time period. Fast Data is often compared with a firehose, where the data arrives at a fast speed. For instance, global positioning systems are producing continuous data points using various location-based services. Internet of Things (IoT) devices are continuously monitoring variety of parameters, like temperature, heart beats, and others, and sending the data over the network. This continuous generation of data adds a new time dimension, where fresh data points are given more importance. Real-time processing allows systems to monitor and react to events with low latency allowing one to handle many use-cases like fraud detection, personalization, monitoring.

Analyzing real-time data is important with applications in networking, such as monitoring network links for elephant flows, counting the number of distinct flows, estimating the distribution of flow sizes, and so on. Further, machine learning tasks, such as spam detection, personalization, and recommendation are few well-known use-cases for real-time applications [2]. Financial transactions have a very interesting use-case for real-time fraud detection [3]. Real-time trend detection is one of the incentives that come out from the modern online social networks like Twitter [4]. Online social networks are used to identify emerging stories, and events that capture popular attention [5]. Furthermore, there are variety of use-cases in the social mobile gaming industry, which requires monitoring event in real-time for applications, such as real-time personalization/segments, real-time engagement, fraud detection and real-time triggers[1].

Processing data at fast speed presents several novel challenges due to its rapid nature. Firstly, real-time data processing imposes the constraint that each event is only processed once, thus preventing iterative processing. Furthermore, real-time data processing requires low-latency applications, which restricts the use of expensive operations, e.g., disk seeks, sending data over the wide area network, and more. Also, real-time data processing requires providing high throughput guarantees, to be able to process maximum number of concurrent requests. Resource utilization becomes an important factor to provide real-time processing guarantees, while keeping the hardware cost low. Our contribution in this thesis comes to put a stone in the area of fast data processing. The goal is to design novel techniques and propose optimizations to push the state-of-the-art in the domain.

1.1 Preliminaries

In this section, we provide the preliminaries of the thesis and provide an overview of few closely related topics.

Graphs. Real-world data exists in variety of forms, i.e., numeric, text, graphs, images, and others. It is extremely important to understand different properties of data for designing efficient data processing techniques. For instance, graphs[2] are present in many fields, from the Web advertising to bio-analytics. Social networks are one of the common examples representing data in the form of graph. Few other examples of real-world graph include citation networks, road networks, email, mobile networks, molecules interactions and many others.

Representing real-world networks in the form of graphs enables answering various interesting questions. For example, which users in a social network should be chosen to initiate a viral campaign? Which users in a social network should be chosen as a mediator between two communities? Which callers show anomalous behavior in phone call activity in a cell phone call network? Which webpages are more authoritative in web graphs? Which group of users in a social network might have ties with or are part of terrorist organizations? In this thesis, we study several challenging graph related problems and design efficient real-time algorithms to solve them.

Data Skew. Another aspect of data is that most of the natural workloads follow a skewed key distribution [6, 7]. For example, Internet topology follows a simple power-law distribution [8]. Similarly, most of the natural langugage processing workloads can be approximated with a Zipfian distribution [9], which states that given some corpus of natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table. Moreover, Pareto

[1]`https://techblog.king.com/rbea-scalable-real-time-analytics-king/`

[2]Graphs are not plots. Graphs consist of vertices, which are connected with each other via edges.

distribution is used in description of social, scientific, geophysical, actuarial, and many other types of observable phenomena. Furthermore, social networks and graphs follow a power-law degree distribution [10]. These examples show that many of the natural workloads can be modeled as a power-law distribution.

To make the discussion more concrete, we introduce a simple example application to show the problems that arrive due to the presence of skewness in the data. Consider running a parallel version of "word count" on a skewed input, which is processed in a large cluster environment. The input consists of a set of documents, containing large number of words. A trivial way to exploit a large cluster, for counting, is to partition and distribute the words across the cluster, and combine the local aggregates to get the final result. Hashing is often employed for partitioning, where each word is mapped to the same partition. Due to the skeweness in the distribution of word frequencies, some words appear more frequently than others. Partitioning the stream using this approach translates the skewness in the word frequencies into some partitions having large number of words, hence some machines doing more work than others. We refer to this uneven assignment as *load imbalance.* This example elaborates how skewness in data leads to uneven workload distribution, which eventually leads to unexpected or poor performance. In this thesis, we study the load balancing problem in the context of distributed stream processing engines and design several interesting techniques to tackle the problem.

Streaming Engines. Distributed Stream Processing Engines (DSPEs) are often employed for real-time data processing, as they are capable of achieving required performance guarantees, i.e., low latency and high throughput. DSPEs differ from the batch processing frameworks, as they process the data in-memory by applying light weight computations. They leverage large clusters of commodity machines and distribute the processing functionality across the cluster to enable high throughput. Variety of DSPEs have evolved in the last two decades due to their added value in various domains, i.e., data mining, web analytics, banking, and others. These frameworks are competitive for metrics like, high availability, scalability, fault-tolerance, ease-of-use, manageability, debug-ability, compatability, state-management, windowing, self-healing, back pressure, processing guarantees and simplicity [3].

Applications of DSPEs, especially in data mining and machine learning, typically require accumulating state across the stream by grouping the data on common fields [11, 12]. Akin to MapReduce, this grouping in DSPEs is usually implemented by partitioning the stream on a *key* and ensuring that messages with the same key are processed by the same processing element instance (PEI). This partitioning scheme is called *key grouping.* Typically, it maps keys to sub-streams by using a hash function. Hash-based routing allows each source PEI to route each message solely via its key, without needing to keep any state or to coordinate among PEIs. Each PEI[3] is associated to one or more keys, and keys associated to the same PEI are said to be *grouped* together by a *grouping scheme.* Another partitioning scheme called shuffle grouping achieves excellent load balancing by using a round-robin

[3]We refer to a PEI as a *worker* throughout the thesis.

routing, i.e., by sending a message to a new PEI in cyclic order, irrespective of its key. However, this scheme is mostly suited for stateless computations. Shuffle grouping may require an additional aggregation phase and more memory to express stateful computations. In this thesis, we design several novel grouping schemes that are more scalable than key grouping and are more suitable for stateful computations than shuffle grouping. Our scheme distributes the data items equally among the machines by treating the head of the key distribution differently than the tail, which enables them to achieve load balance for skewed streams in both homogenous and heterogeneous environments.

Streaming Models. There is a wide range of stream processing models, which differ based on the allowed processing time and space complexity. Since stream processing deals with continuous data, it is preferred to look at each event only once. Streaming (insertion-only) is a one of the fundamental models for computations on massive datasets [13, 14]. A streaming algorithm processes the events in incremental fashion. This model requires large amount of memory as the data stream grows. Feigenbaum et al. [15] proposed semi-streaming algorithms for graphs. In this model, the algorithm is allowed to utilize memory proportional to the number of vertices in the graph, rather than the number of edges [16]. Datar et al. [17] proposed the sliding window model, in which the function of interest is computed over a fixed-size window in the stream. The window slides at each timestamp adding a new event and discarding the last event in the window. Sliding windows are used in variety of applications and are often considered in various contexts, i.e., time, count, or sessions in addition to data-driven windows. Lastly, fully dynamic algorithms [18] allow items to be inserted and deleted during the execution. In this thesis, we study several graph-related problems in fully dynamic settings, as it captures the dynamic nature of the graphs.

1.2 Reserach Motivation

There is a large body of literature related to large-scale distributed stream processing engines. In the last decade, stream processing attracted both enterprises and open source community. The main focus of these engines is to process the incoming streams in distributed fashion to meet the high scalability requirement. S4 [19], Storm [20, 21], MillWheel [22], DataFlow [23], Apache Samza [24], and Flink [25] are few well-known streaming frameworks. Alongside, Apache Kafka [26] and RabbitMQ [27] are distributed message queue, which serves as a data source for the streaming engines. The generations of DSPEs are a result of unprecedented growth in the volume and velocity of the data. One of the major characteristics of any DSPE is the scalability, which is the capability to handle a growing amount of work. To cope with data at such a scale requires DSPEs that scale with the data growth and provide necessary processing guarantees.

DSPEs expose a transparent interface that enables writing streaming applications and executes them in a distributed manner. Streaming applications are represented

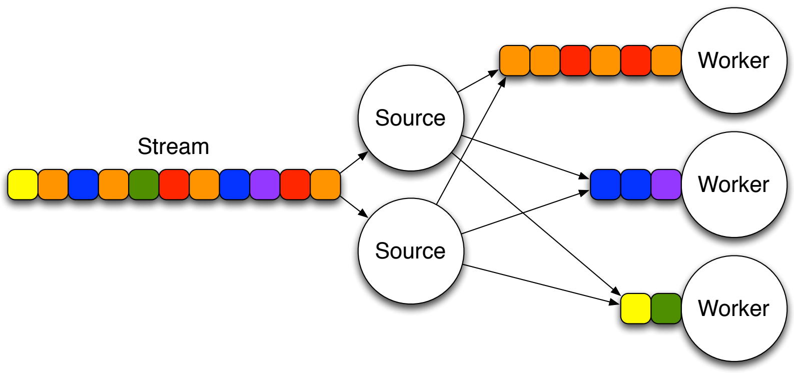

Figure 1.1: Load imbalance generated by skew in the key distribution when using key grouping. The color of each message represents its key.

by directed acyclic graphs (DAG) where vertices, called *processing elements* (PEs), represent operators, and edges, called *streams*, represent the data flow from one PE to the next. Figure 1.1 provides an example of a simple DAG, which has two set of operators: *source* and *workers*. Morever, each operator applies various transformation of the incoming stream and produces one or more output streams, which serves as an input for the downstream operators. These applications cover wide range of business use-cases, covering various streaming applications. However, there exist a large set of streaming applications, which are hard to parallelize and execute in the distributed manner due to their innate complexity. A subset of these streaming applications are related to graph theory.

Developing analytical tools to process large graphs is essential for developing new services in the areas, such as social networks, web analytics, e-health and marketing. In most of the cases, the size of the graphs along with metadata is too huge to fit in a single machine. Therefore, bunch of distributed graph processing engines have been developed in the last decade to enable processing huge graphs. Google came with a scalable and fault-tolerant graph processing platform called Pregel [28], with an open source implementation called Giraph[4]. Further, the pregel model has been adapted in various modern graph processing systems, like GraphX [29], GraphLab [30, 31], GPS [32], PowerGraph [10], and others [33]. These frameworks provide implementations of various simple graph algorithms, e.g., PageRank, connected components, label propagation, community detection and others.

Most real-world graphs are dynamic and rapidly changing. For instance, Facebook users are continuously creating new connections and removing old ones, thus changing the network structure. Twitter users produce posts at a high rate, which makes old

[4]`http://giraph.apache.org/`

posts less relevant and changes the retweet and mention networks. Processing graph events in real-time is essential for variety of applications, e.g., community tracking, event detection, story identification, fraud detection, and more. In the recent years processing dynamic graphs attracted attention from various communities including, theoretical computer science, data mining, and data management. We refer the reader for a detailed survey on streaming algorithm by McGregor [34]. Advancing real-time data processing requires developing analytical tools to process large dynamic graphs.

1.3 Research Objectives

Key-based workload partitioning is a common strategy used in all generations of parallel stream processing engines, which enables distribution of keys over worker threads in a logical operator. Figure 1.1 illustrates how such a randomized scheme results in poor balancing performance when the workload distribution is skewed. This creates a straggler among the set of machines in a distributing environment, as it assigns more work to one of PE than the others, which leads to low-performance for streaming applications. We envision a novel workload paritioning strategy that is being able to distribute the skewed workload in an efficient way that minimizes the load imbalance among PEs, while keeping the semantics of the key-based workload partitioning. The goal of the thesis is to be able to propose a scheme that is capable of handling highly skewed streams and scales to large number of machines

For various commercial enterprises, the resources available for running a DSPE consist of dedicated machines, private clouds, bare metal, virtualized data centers and commodity hardware. Resource utilization in a cluster with heterogeneous resources is challenging, as underlying resources are often unknown. For streaming applications, the heterogeneity is invisible to the upstream PEIs and requires inferring the resource capacities in order to generate a fair assignment of the tasks to the downstream PEIs. However, gathering statistics and finding optimal placement often leads to bottlenecks, while at the same time microsecond latencies are desired. We envision a workload partitioning strategy that enables distributing the workload in a heterogenous environment in an efficient manner and improves the resource utilization.

One category of fast stream data is linked-data that can mostly be represented by graphs. Graphs have been extensively studied in the literature, and many of their fundamental problems have been addressed under different traditional settings, where data is assumed to be static. However, with the latest proliferation of dynamic evolving graph and the need for their real-time processing, there is much interest in developing novel algorithms that can solve the traditional problems from graph theory under this dynamic setting. We focus in this thesis on two interesting and challenging graph-related problems: *top-k densest-subgraph* and *frequent subgraph mining*.

Finding a subgraph with maximal density in a given graph is a fundamental graph-mining problem, known as the *densest-subgraph* problem. Density is commonly defined as the ratio between number of edges and vertices [35]. The densest-subgraph problem has many applications, for example, in community detetion [36, 37], event detection [5], link-spam detection [38], and distance query indexing [39]. The densest subgraph problem has been extensively studied in various settings [40, 41, 42, 43]. In applications, we are often interested not only in one densest subgraph, but in the *top-k*. [44, 45, 17]. Finding the top-k densest subgraphs in a fully dynamic settings is of interest to several real-time applications, e.g., community tracking [46], event detection [47], story identification [5], fraud detection [48], and more. Our goal is to be able to provide accurate and efficient approximation algorithm for the top-k densest subgraph problem in fully dynamic model with provable theoretical guarantees.

Frequent-subgraph mining (FSM) is a fundamental graph-mining task with applications in various disciplines, including bioinformatics, security, and social sciences. The goal of FSM is to find subgraph patterns of interest that are frequent in a given graph. Such subgraphs might be indicative of an important protein interaction, a possible intrusion, or a common social norm. FSM also finds applications in graph classification and indexing. Existing algorithms for subgraph mining are not scalable to large graphs that arise, for instance, in social domains. In addition, these graphs are usually produced as a result of a dynamic process, hence are subject to continuous changes. For example, in social networks new edges are added as a result of the interactions of their users, and the graph structure is in continuous flux. Whenever the graph changes, e.g., by adding or removing an edge, a large number of new subgraphs can be created, and existing subgraphs can be modified or destroyed. Keeping track of all the possible changes in the graph is subject to combinatorial explosion, thus, is highly challenging. Only a few existing works consider a similar setting [49, 50]. Our goal is to be able to provide accurate and efficient algorithm for the frequent subgraph mining problem in fully dynamic settings with provable theoretical guarantees.

1.4 Research Statement

This thesis aims to develop methods and propose optimizations to enable real-time data processing at large-scale. Our first goal is to identify challenges that appear in distributed stream processing engines due to the presence of skewness in the data stream and heterogeniety among the resources. Further, we aim to mitigate these challenges by proposing novel algorithms that are capable of operating at large scale. Our second goal is to be able to address several graph-related problems for evolving graphs. Moreover, we aim to propoose efficient and accurate algorithms for these problems. The main thesis statement is:

We identify that skewness in the stream and heterogeneity in the cluster are among the main hurdles towards scalability of distributed stream processing *engines. We address these challenges by proposing novel techniques that are capable of mitigating these obstacles. Our techniques ensure that data items in the head of the key distribution are treated differently than in the tail, which enables them to achieve load balance for skewed streams in both homogenous and heterogeneous environments. Further, we identify the need for developing novel real-time algorithms for graph-related problems. In doing so, we address two problems in the domain of graph theory and propose efficient approximation algorithms in fully dynamic settings that operate at very large scale Our algorithms rely on several fundamental techniques, like kernelization, reservoir sampling, random pairing, and others.*

1.5 Thesis Contributions

In this thesis, we present work that builds on previous research in Distributed Systems, Databases, Data Mining, and Theoretical Computer Science. In doing so, we develop algorithms that are extremely efficient and simple.

We first perform the scalability study for distributed stream processing engines. We study the load balancing problem that appears due to the presence of skewness in the data stream and heterogeneity in the computing elements in any distributed system. Moreover, our algorithms are integrated into Apache Storm, which is an open source stream processing framework. Further, we provide algorithm for two interesting graph problems in fully dynamic settings, i.e., finding top-k densest subgraph and mining frequent graph patterns. In summary, we make following contributions:

- We study the load-balancing problem, which becomes prominent in the presence of skewness in the data stream, in the context of distributed stream processing engines. In doing so, we introduce PARTIAL KEY GROUPING (PKG) [51, 52], a new stream partitioning scheme that adapts the classical "power of two choices" to a distributed streaming setting by leveraging two novel techniques: *key splitting* and *local load estimation*. Key splitting leverages *both* choices by relaxing the atomicity constraints of key grouping. Local load estimation solves the problem of gauging the load of downstream servers without any communication overhead. PKG achieves better load balancing than key grouping while being more scalable than shuffle grouping.
- Expanding on the previous work, we study the load balancing problem for extremely skewed streams and propose two novel strategies: D-Choices and W-Choices [53]. Both the schemes leverage heavy hitter algorithm to efficiently identify the hottest keys in the stream. These hot keys are assigned to d$\geq$ 2 choices to ensure a balanced load, where d is tuned automatically to minimize the memory and computation cost of operator replication for D-Choices algorithm. W-Choices assigns the heavy keys on all the set of workers. The technique works online and does not require the use of routing tables.

- We study the load balancing problem in the presence of skewness and heterogeniety in the cluster [54]. In doing so, we propose a novel partitioning strategy called Consistent Grouping (CG), which enables each processing element instance (PEI) to process the workload according to its capacity. The main idea behind CG is the notion of small, equal-sized virtual workers at the sources, which are assigned to workers based on their capacities. We provide a theoretical analysis of the proposed algorithm and show via extensive empirical evaluation that our proposed scheme outperforms the state-of-the-art approaches, like key grouping.
- We study the top-k densest-subgraph problem in the sliding-window model and propose an efficient fully-dynamic algorithm [55]. In contrast to existing state-of-the-art solutions that require iterating over the entire graph upon any update, our algorithm profits from the observation that updates only affect a limited region of the graph. Therefore, the top-k densest subgraphs are maintained by only applying local updates. We provide a theoretical analysis of the proposed algorithm and show empirically that the algorithm often generates denser subgraphs than state-of-the-art competitors.
- We initiate the study of approximate frequent subgraph mining (FSM) problem in both incremental and fully-dynamic streaming settings, where arbitrary edges can be added or removed from the graph [56]. For each streaming setting, we propose algorithms that can extract a high-quality approximation of the frequent k-vertex subgraphs for a given threshold, at any given time instance, with high probability. Our algorithm leverages reservoir sampling [57] and random pairing [58]. We provide theoretical analysis of the proposed algorithms and emprically demonstrate that the proposed algorithms generate high-quality results compared to baselines.

1.6 List of Publications

1. Nasir, Muhammad Anis Uddin, Gianmarco De Francisci Morales, David Garcia-Soriano, Nicolas Kourtellis, and Marco Serafini. "The power of both choices: Practical load balancing for distributed stream processing engines." In Data Engineering (ICDE), 2015 IEEE 31st International Conference on, pp. 137-148. IEEE, 2015. [51].

   Contributions. The author of this thesis was actively involved in brainstorming and coming up with idea to address the problem. Moreover, he was continuously involved in the design and implementation of the proposed algorithm. Further, the author of this thesis performed most of the experiments and participated in writing of the paper.

2. Nasir, Muhammad Anis Uddin, Gianmarco De Francisci Morales, David Garcia-Soriano, Nicolas Kourtellis, and Marco Serafini. "Partial key grouping: Load-balanced partitioning of distributed streams." arXiv preprint arXiv:1510.07623 (2015). [52].

Contributions. The author of this thesis was actively involved in brainstorming and coming up with idea to address the problem. Moreover, he was continuously involved in the design and implementation of the proposed algorithm. Further, the author of this thesis performed most of the experiments and participated in writing of the paper.

3. Nasir, Muhammad Anis Uddin, Gianmarco De Francisci Morales, Nicolas Kourtellis, and Marco Serafini. "When two choices are not enough: Balancing at scale in distributed stream processing." In Data Engineering (ICDE), 2016 IEEE 32nd International Conference on, pp. 589-600. IEEE, 2016. [53].

   Contributions. The author of this thesis came up with the idea to address this problem. Moreover, he was continuously involved in the design and implementation of the proposed algorithm. Further, the author of this thesis performed most of the experiments and participated in writing of the paper.

4. Nasir, Muhammad Anis Uddin, Hiroshi Horii, Marco Serafini, Nicolas Kourtellis, Rudy Raymond, Sarunas Girdzijauskas, and Takayuki Osogami. "Load Balancing for Skewed Streams on Heterogeneous Cluster." arXiv preprint arXiv:1705.09073 (2017). [54] (under submission).

   Contributions. The author of this thesis came up with the idea to address this problem. Moreover, he was completely involved in the design and implementation of the proposed algorithm. Further, the author of this thesis performed most of the experiments and wrote most of the paper.

5. Muhammad Anis Uddin Nasir, Aristides Gionis, Gianmarco De Francisci Morales, and Sarunas Girdzijauskas. 2017. Fully Dynamic Algorithm for Top-k Densest Subgraphs. In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM '17). ACM, New York, NY, USA, 1817-1826 [55].

   Contributions. The author of this thesis was actively involved in brainstorming and coming up with idea to address the problem. Moreover, he was continuously involved in the design and implementation of the proposed algorithm. Further, the author of this thesis performed most of the experiments and participated in writing of the paper.

6. Cigdem Aslay, Muhammad Anis Uddin Nasir, Gianmarco De Francisci Morales, and Aristides Gionis. 2018. Mining Frequent Patterns in Evolving Graphs [56] (under submission).

   Contributions. The author of this thesis was actively involved in brainstorming and coming up with idea to address the problem. Moreover, he was continuously involved in the design and implementation of the proposed algorithm. Further, the author of this thesis performed most of the experiments and participated in writing of the paper.

Other Publications

7. Nasir, Muhammad Anis Uddin, Sarunas Girdzijauskas, and Nicolas Kourtellis. "Socially-aware distributed hash tables for decentralized online social networks." In Peer-to-Peer Computing (P2P), 2015 IEEE International Conference on, pp. 1-10. IEEE, 2015. [59].

   Contributions. The author of this thesis came up with the idea to address this problem. Moreover, he was completely involved in the design and implementation of the proposed algorithm. Further, the author of this thesis performed most of the experiments and wrote most of the paper.

8. Nasir, Muhammad Anis Uddin, Fatemeh Rahimian, and Sarunas Girdzijauskas. "Gossip-based partitioning and replication for online social networks." In Advances in Social Networks Analysis and Mining (ASONAM), 2014 IEEE/ACM International Conference on, pp. 33-42. IEEE, 2014. [60].

   Contributions. The author of this thesis came up with the idea to address this problem. Moreover, he was completely involved in the design and implementation of the proposed algorithm. Further, the author of this thesis performed most of the experiments and wrote most of the paper.

9. Nasir, Muhammad Anis Uddin. "Fault Tolerance for Stream Processing Engines." arXiv preprint arXiv:1605.00928 (2016) [61].

   Contributions. The author of this thesis came up with the idea to address this problem. Further, the author of this thesis performed the experiments and wrote the paper.

1.7 Outline

This thesis is organized in four chapters as follows. Chapter 2 covers the background and related work to the proposed algorithms in the thesis. Chapter 3 provides a summary of the papers included in the thesis. Chapter 4 concludes the contribution of the performed research. The complete publications are presented at the end of thesis.

Chapter 2

Background

A list is as strong as its weakest link.

Donald Knuth

Real-time data processing intersects several fields, i.e., databases, data mining, machine learning, and theoretical computer science. Consequently, the literature in the field is extremely wide, which makes it difficult to provide a comprehensive survey of the related fields. In this chapter, we provide a brief overview of the related fields covering various systems, algorithms, applications, and fundamental techniques. Our basic goal is to give highlights on both systems and theoretical research related to real-time data processing.

2.1 Traditional Data Processing

Relational databases (DBMS) arrived on the scene as research prototypes in the 1970's, in the form of System R [62] and INGRES [63]. Since then, it was believed that DBMS is the answer to most of the business needs and majority of the research was focused on optimizing such systems [64, 65, 66, 67]. DBMS stores relational tables in disk in the form of rows. Further, they leverage B-trees for indexing, use a cost-based optimizer, and provide ACID[1] transaction properties. Such data stores are optimized for fast writes and are used for online transaction processing (OLTP). In 2005 Stonebraker and Cetintemel [68] highlighted that DBMS cannot serve for many modern business use-cases, e.g., stream processing, data warehousing, natrual language processing, and scientific applications [69]. Further, there have been several efforts in developing relational in-memory databases to meet the streaming requirements [70, 71, 72]. However, relational DBMS are still not the standards for processing data streams.

[1]Atomicity, Consistency, Isolation, Durability

Data warehouse (DWH) is a multi-dimensional data store used for reporting and data analysis. DWH are optimized for online analytical processing (OLAP). OLAP works by gathering historical data from multiple operational databases into a DWH, where complex business intelligence queries are executed to extract interesting information. OLAP is a database technology that has been optimized for querying and reporting, instead of processing transactions. The source of data for OLAP is OLTP, however these two data stores are separated so that the business requirements are met in a sophisticated manner. OLAP systems are implemented often using column stores [73], which are optimized for analytical workloads [74]. The main reason behind the usage of column stores is their I/O efficiency for read-only queries for attributes accessed by the query. OLAP operates by re-computing summaries and aggregates as more information is added into the data warehouse on periodic basis, i.e., hourly or daily basis. Both OLTP and OLAP systems have been providing for various business intelligence needs over the decades and have been evolved to provide required data management capabilities. However, DWH operates on snapshots of data, which is periodically copied from OLTP system, to generate summaries and aggregates. This makes OLAP systems an inappropriate choice for real-time data processing.

On the other hand, statistical software like, SAS, R, SPSS, WEKA and Matlab are capable of providing the tools for various complex, real-time data mining and analysis [75]. These tools and packages provide implementations of wide range of data mining and machine learning algorithms, which serves for various complex business needs. These systems lack in two dimensions: First, these systems are not capable of providing support for real-time data analysis. Second, they lack data management support, similar to OLTP and OLAP systems. There have been several efforts in designing array databases to allow complex data mining and analysis with sophisticated data management [76, 77]. Moreover, there have been much efforts in developing machine learning libraries for streaming data [2]. However, sufficient efforts are still required to develop novel algorithms and to merge sophisticated data management and real-time complex data analysis for commercial purposes.

2.2 Batch Processing

Batch data processing is an efficient way of processing high volumes of data, where the data is grouped together in the form of batches and processed to generate the desired output. One of the properties on batch processing is that it operates on the bounded and finite data, which is the opposite of real-time, infinite and unbounded data processing. However, it is extremely important to understand batch processing systems, as they have a strong influence on architecture of streaming systems and it is often the case that streaming pipelines operate concurrently with the batch jobs. Batch processing has been one of the most common ways for data processing. For instance, traditional enterprises execute jobs periodically to generate reports and identify business trends. Due to the data deluge in the recent decades, many new

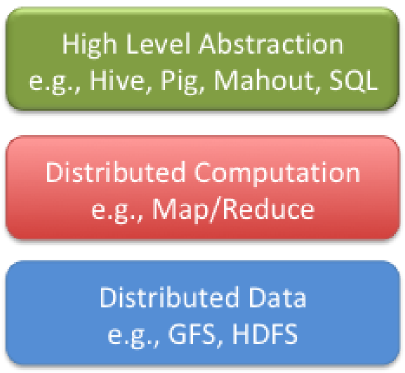

Figure 2.1: Batch processing architecture.

storage and processing models have emerged to facilitate batch processing. In this section, we provide a brief survey of these frameworks.

2.2.1 MapReduce

MapReduce [78] is a distributed computing engine developed by Google to support processing on their crawled database. Hadoop [79] is an open source clone of MapReduce. Essentially, MapReduce is a combination of two things: distributed file system (DFS) and MapReduce (MR). Hadoop comes with Hadoop Distributed Files System (HDFS), which is an open source implementation of the Google File System (GFS) [80]. HDFS is based on shared-nothing architecture and provides the storage capabilities for the data leveraging large clusters, while being both scalable and fault tolerant. MR is programming framework that runs on top of commodity hardware and provides the data processing functionalities for the data that resides on HDFS. Each MR job consists of two phases: Map and Reduce. Additionally, there has been bunch of high level abstractions on top of Hadoop to provide querying and data access functionalities, e.g., Hive [81], PIG [82], Mahout [83], SQL, and others. Figure 2.1 provide a basic batch processing model segregating different components of MapReduce.

MapReduce might currently not being the most interesting data processing engine. However, it was among one of the pioneer systems that showed the route towards parallel data intensive processing. Engineers at Google further came up with more interesting and interactive solution for batch processing like, BigTable [84], Dremel [85] and F1 [86]. Dryad [87] is Microsoft's alternative to MapReduce. Spark [88] is a faster, in-memory, more functional version of Map-Reduce, which is currently one of the major open source projects that has been adopted by several enterprises. Apache Flink [25] is an open source data processing framework that provides batch and stream processing under a single distributed system. All these frameworks enable processing large datasets in a large cluster configuration, while providing desired guarantees, i.e., scalability, fault tolerance, consistency, high availability, and others.

2.2.2 Graph Processing

Many practical computing problems concern large graphs. Graph algorithms often exhibit poor locality of memory access, very little work per vertex, and a changing degree of parallelism over the course of execution [89, 90]. To cope with the ubiquity of large graphs and complexity of graph algorithms, Google came with a scalable and fault-tolerant graph processing platform called Pregel [28], with an open source implementation Giraph[2]. Pregel implements the known the Bulk Synchronous Parallel (BSP) model [91], where computations consist of a sequence of iterations, called supersteps. During a superstep the framework invokes a user-defined function for each vertex, conceptually in parallel. This model of data processing is known as vertex-centric model, which has been adapted in various modern graph processing systems, like GraphX [29], GraphLab [30, 31], GPS [32] , PowerGraph [10], and others [33]. Moreover, there have been efforts enabling large-scale graph processing using GPUs [92, 93, 94].

2.3 Stream Processing

In the previous section, we discussed batch processing where data is stored in storage system and is processed to generate the desired output. One strong assumption remained throughout previous section is that the data is bounded. However, most of the real-world workloads are unbounded and arrive continuously in the form of events, e.g., call logs, web traffic, sensor network data, and many others. So practically, batch processing is an artificial way of processing unbounded data, where the data is divided into chunks and processed on periodic basis. However, this processing model enforces that the change in input is only reflected in the output, once the job is executed. A way out through this bottleneck is to run the job on shorter periods, and this is the idea behind stream processing. Stream processing involves a continual input, process and output of data. The critical question in terms of stream processing is: “How fast do you want your results?” What are the customer or user needs? Is it necessary to get results every minute, every hour or every day? This time dimension adds a new complexity to data processing.

Real-time data processing gives an organization the ability to take immediate action for those times when acting within seconds or minutes is significant. The goal is to obtain the insight required to act prudently at the right time - which increasingly means immediately. For instance, real-time data processing covers many

[2]http://giraph.apache.org/

use-cases at Facebook, including real-time reporting of the aggregates, anonymised voice of Facebook users, analytics for mobile applications, and insights for Facebook page administrators [95]. Two main performance characteristics of distributed stream processing systems are low latency and high throughput.

Real-time applications have variety of time requirements depending on their complexities. For example, a simple application might only require constant amount of operations per data item, e.g., filters, aggregates, and others. Surprisingly, large number of machine learning and data mining applications are implemented using aggregate-like operations, e.g., naive bayes classifier, decision tree, heavy hitters, and many others [2, 96]. Few other use-cases for simple streaming applications are: What are the call time for each mobile user? How many worth of purchases are performed for each item? How many tweets are posted with hashtag “Trump”? How many game players have started and ended a level of candy crush on the first attempt?

Further, there exist many streaming applications, which require more than constant amount of operation per data item, i.e., polylograthmic, linear or polynomial time operations. For example, finding most influential users in a social network, finding communities in a network, finding the bridges in a graph, and finding dense components in a graph. Many of these applications exist in the domain of streaming graph analytics. There has been considerable interest in designing algorithms for processing massive graphs in the data stream model where the input is defined by a stream of data [41, 97, 98, 99]. Algorithms in this model must process the input stream in the order it arrives while using only a limited amount memory.

Complex event processing (CEP) is another domain of stream processing in which a user is interested in identification of certain patterns in the data, e.g., Is there a user purchasing beer along with diapers? Is there a user accessing more than ten similar web pages in an hour? Is there a group of politician tweeting about the same topic? Is there a correlation with price of bitcoin and other crypto currencies? CEP deals with mining complex patterns in the data streams, rather than maintaining a state [100, 101]. Therefore, CEP engines often opt to throw away some events when failure happens and continue, while streaming engines focus on reliable message processing. CEP engines let users write queries using a higher level language, such as a SQL like query. Also, CEP has build in operators, such as time windows, session windows, temporal event sequences and others.

2.4 Distributed Stream Processing Engines

There has been an unmanageable development in the domain of distributed stream processing engines in last three decades, where various frameworks are developed and claimed to be better than one another. The first generation systems were either main-memory database systems or rules engines. Few of the first generation streaming engines include HiPAC [102], Starburst [103], Postgres [104], Ode [105], and NiagaraCQ [106]. These systems are limited in functionality and are not able to scale well for large volume of data streams.

Second-generation DSPEs are focused on extending SQL for processing streams. TelegraphCQ [107] and STREAM [108] proposed a data model integrating streams into SQL. Aurora [109] and Medusa [110] used operator definitions to form a directed acyclic graph (DAG) for processing stream data in a single node system. Borealis [111] extended Aurora for distributed stream processing with a focus on fault tolerance and distribution.

The third generation of stream processing engines attracted both enterprises and open source community. The main focus of these engines is to process the incoming streams in distributed fashion to fulfil the high volume and velocity needs. S4 [19] is developed at Yahoo following the actor model [112]. Twitter contributes in the forth generation of streaming frameworks with Storm [20] and Heron [21]. Dhalion [113] is a self regulating API that executes on top of Heron. Google contributes in the form of MillWheel [22] and Cloud Dataflow [23]. LinkedIn contributes with Apache Kafka [26], a distributed message queue, which serves as a data source for the streaming engines. Also, Apache Samza [24] is a streaming engine that was initially developed at LinkedIn. Spark Streaming [114] came along with Spark [88], which is an in-memory data processing platform. Apache Flink [25] is well-known with its capabilities for state-management. Facebook executes inhouse developed pipeline for streaming application [95]. Morevoer, SummingBird [115] is one of the efforts for integrating batch and stream processing frameworks. Recently, Huang and Lee [116] proposed a framework that provides bounded accuracy guarantees in case of failures.

The generations of DSPEs are a result of unprecedented growth in the volume and velocity of the data. However, most of the DSPEs are incapable to scale when exposed to skewed streams. To complicate the matter, scalability becomes even more challenging when the underlying resources are heterogeneous. This thesis contributes by designing several algorithms and by proposing optimizations that enable DSPEs to operate under such adverse conditions. In the next sections, we provide details about how streaming applications are expressed in DSPEs.

2.4.1 Directed Acyclic Graphs

Most of these frameworks represent a streaming application in the form of a directed acyclic graph (DAG), a.k.a topologies. A topology consists of operators, which are responsible for applying various transformations on the data that flow through it. Moreover, operators are connected through edges, which represent the data flow between operators. In case of Storm and Heron, these operators are further divided into two disjoint sets- spouts and bolts. Spouts are tuple sources for the topology. Typical spouts pull data from queues, such as Kafka [26] or Kestrel [3]. On the other hand, bolts process the incoming tuples and pass them to the next set of bolts downstream. Flink follows the same paradigm as Storm and Heron and represents a

[3]`https://github.com/twitter-archive/kestrel`

streaming application as a series of transformations. Note that Flink, Heron and Storm also allow a streaming application to be represented as a directed cyclic graph, which can have cycles. In case of Spark Streaming, these operators are a continuous series of RDDs, which is Spark's abstraction of an immutable, distributed dataset.

2.4.2 Streaming Operators

The main logic of a streaming application is written in the form of set of operators, which are responsible for applying different transformation on the incoming stream. These transformations are similar to database operations, like filter, select, aggregate and join. Scalability is achieved by running various instances of the operator, which run in parallel and process the sub-stream. For understanding streaming operators, consider an example of running streaming top-k word count. This application is an adaptation of the classical MapReduce word count to the streaming paradigm where we want to generate a list of top-k words by frequency at periodic intervals (e.g., each T seconds). It is also a common application in many domains, for example to identify trending topics in a stream of tweets. The streaming word count is implemented using three set of operators: splitter, counter and aggregator. The splitter is responsible for splitting each sentence into words. The counter keeps a running counter for each word. At periodic intervals, the counters send their top-k counters to a single downstream aggregator to compute the top-k words. While this application is clearly simplistic, it models quite well a general class of applications common in data mining and machine learning whose goal is to create a model by tracking aggregated statistics of the data.

2.4.3 Data Streams

The input in case of batch processing model consists of files or set of records. What would be a streaming equivalent for it? In stream context, data is modeled best as transient data streams rather than persistent relations, i.e., a record is more commonly known as an event, but it is essentially the same thing: a small, self-contained, immutable object. An event usually contains a key, a timestamp and a value. The key is a unique identifier for the event. The timestamp indicates the time at which the event occurred and the value contains the data points representing the information, e.g., temperature, call records, bank transactions, and others.

Graph streams are often considered differently from the data streams. In particular, a graph is fed into the system as either a vertex or an edge stream. A vertex stream is identical to an adjacency list, where each vertex is presented along with its neighbors. On the other hand, an edge stream is equivalent to an edge list, where edges are fed into the system as a pairs of vertices. Edge streams are considered more natural as they capture the evolution of the network.

2.4.4 Stream Partitioning

Sub-streams are created by partitioning the input stream via a *stream partitioning* function, which maps each key in the key space to a natural number. This number identifies the worker responsible for processing the message. Each worker is associated to one or more keys, and keys associated to the same worker are said to be *grouped* together by a *grouping scheme*. Henceforth, we consider two of the well-known grouping schemes:

- ***Key Grouping (*kg*)***: This scheme ensures that messages with the same key are handled by the same downstream worker. KG is usually implemented via hashing. It is effective for stateful operators, where the goal is to group by events based on the same key.
- ***Shuffle Grouping (*sg*)***: This scheme evenly distributes messages across the available downstream workers, thus ensuring ideal load balance. SG is usually implemented via round-robin selection of workers. This scheme is effective for stateless operators, where each event can be handled independently.

2.4.4.1 Streaming Sources

Streaming Operators follow a push-based mechanism rather a pull-based mechanism. An event in generated once by a producer (a.k.a publisher or sender) and then potentially processed by multiple consumers (subscribers or recipients). A source that produces a possibly infinite amount of data elements, which can not be loaded into the memory, thus making batch processing unfeasible. Kafka, Scribe[4], RabbitMQ are few persistent, distributed messaging system for collecting, aggregating and delivering high volumes of log data with low latency and high throughput. For instance, Apache Kafka provides three key capabilities: publish and subscribe to streams of records, store stream of records in a distributed and fault-tolerant durable way, and process streams of records as they occur. Kafka runs as a cluster on one or more machines, and stores the stream of records in the categories called topics. Further, each topic is divided into partitions, where each partition provides total ordering guarantees for the stored events. Further, consumers (streaming operators) process the events from each partition in parallel. This allows Kafka to scale with the input volume and serve for high velocity data processing needs.

2.4.5 Stream Processing Model

Stream processing models are divided based on the restrictions on the allowed computations and available memory. Since the stream processing deals with unbounded data, it is preferred to look at each event only once. Such algorithms are referred as one pass algorithm, where each item is discarded after processing. Moreover, the amount of space required to compute an exact answer may also grow without

[4] `https://github.com/facebookarchive/scribe`

bound. However, most of the machines are restricted with limited memory. In this section, we discuss several data processing models and their requirements.

Streaming (Insertion-only). Streaming is a fundamental model for computations on massive datasets [13, 14]. A streaming algorithm processes the events from the beginning time and only involves the insertion of data items. This model requires large amount of memory as the data stream grows.

Semi-Streaming. Feigenbaum et al. [15] proposed semi-streaming algorithms for graphs. In this model, the algorithm is allowed to utilize memory proportional to the number of vertices in the graph, rather than the number of edges [16]. This is because most problems are provably intractable if the available space is sublinear in n, whereas many problems become feasible once there is memory roughly proportional to the number of nodes in the graph.

Sliding Windows. Datar et al. [17] proposed the sliding window model, in which the function of interest is computed over a fixed-size window in the stream. The window slides at each timestamp adding a new event and discarding the last event in the window. Datar et al. [17] proposed exponential histograms for maintaining vasrious statistics in the data strem. Further, Braverman and Ostrovsky [117] proposed smooth histograms for sliding windows, which extended exponential histograms to various other functions that are *smooth.* [45] proposed various graph algorithms for sliding window model. We refer to [34] for the detailed set of algorithm for graph processing in sliding windows. Sliding windows are used in variety of applications and are often considered in various contexts, i.e., time, count, or sessions in addition to data-driven windows.

Fully-Dynamic Streams. In this model, items are inserted and deleted during the execution of the algorithm [18]. Note that the sliding-window model is different from the fully-dynamic model in the sense that in the sliding-window model the item deletions are implicit, whereas in case of fully-dynamic the identity of the deleted item is explicit, i.e., when an item leaves the active window it is effectively deleted but we may not know the identity of the deleted event unless we store the entire window. In the case of fully-dynamic graph streams, but the edge could correspond to any of the edges already inserted but not deleted. Note that any fully-dynamic algorithm is also applicable for fixed-size sliding windows. However, sliding window algorithms cannot be directly used as fully-dynamic algorithms.

2.5 Approximation Algorithms

Due to the tight memory requirement for unbounded data streams, high-quality approximations are often acceptable in place of exact answers. An algorithm computes an α-*approximation* of a function $f(x)$ if algorithm computes a solution $f(\tilde{x})$ such that $f(x) \geq \frac{1}{\alpha} f(\tilde{x})$. If approximate results are acceptable, there is a class of specialized algorithms, which can produce results orders-of magnitude faster and with mathematically proven error bounds. In this section, we discuss few approximation techniques for data streams, which allows to produce answer to queries while keeping the required memory bounded.

2.5.1 Sampling

The most prominent scheme to generate an approximate answer is sampling, which is the selection of a subset of individuals from within a statistical population to estimate characteristics of the whole population. Results from probability theory and statistical theory are employed to ensure that the estimates generated through samples are representative of the whole population. Two extremely simplistic algorithms for sampling in data streams are: simple random sampling (SRS) [118] and reservoir sampling (RS) [57]. In SRS, a sample is chosen by picking each item in the data stream with an equal probability of being chosen. The inclusion probability is provided as an input parameter to the sampling algorithm. Consider a $p = 0.5$, then each item in the data stream has half the chance to be included in the sample. SRS is extremely simple, however one downside of SRS is that the size of the sample grows along with size of the data stream. On the other hand, RS is a fixed-size randomized sampling scheme, which maintains a fixed-size uniform sample of the data stream. The size of the sample is provided as the input parameter. The algorithm initializes with a fixed-size input array, which initially gets filled by the items in the data stream. Once the array is filled, each i-th item is added to the sample with probability $\frac{1}{i}$ by replacing it with a randomly selected item from the sample. Random pairing [58] is a fully dynamic algorithm for reservoir sampling, that compensate for item deletions using the future addition.

2.5.2 Sketching

Sketching is a scheme in which concise summaries of the data stream are built and used to answer queries related to the stream. Such sketches are much shorter (often exponentially) than the original stream, but nevertheless retain important and useful information. For instance, Flajolet-Martin algorithm [119] is one famous algorithm that provides the approximate answer to the *count-distinct problem* for data streams with logarthmic space-consumption. Similarly, Bloom filter provides a memory-efficient implementation of the *contains* operations for the set of unique items. HyperLogLog is a streaming algorithm used for estimating the number of distinct elements (the cardinality) of very large data sets. Count-Min sketch is a probabilistic sub-linear space streaming algorithm for finding the heavy hitters in the data streams. Most of these data structure relies on the hash functions, which is one of the most important and simple approach for dimensionality reduction. All these algorithms are approximate and are optimized for huge datasets, which do not fit in memory. Stream-lib[5] provides an efficient Java library with implementations

[5] `https://github.com/addthis/stream-lib`

of these data structures. Moreover, Google Guava[6] has Bloom filter implementation using murmur hash.

[6]`https://github.com/google/guava`

Chapter 3

Summary of Papers

In all summaries, the problems seem simpler than they actually are.

Rollo May

This thesis is comprised of five conference papers and one journal paper. In this chapter, we summarize the content and list the contribution of each paper.

3.1 Load Balancing for Distributed Stream Processing Engines

DSPEs are often employed for real-time data processing, which exploit a set of machines to achieve required performance guarantees, i.e., low latency and high throughput. Applications of DSPEs, especially in data mining and machine learning, typically require accumulating state across the stream by grouping the data on common fields. Akin to MapReduce, this grouping in DSPEs is usually implemented by partitioning the stream on a key and ensuring that messages with the same key are processed by the same PEI. This partitioning scheme is called key grouping. Typically, it maps keys to sub-streams by using a hash function. Hash-based routing allows each source PEI to route each message solely via its key, without needing to keep any state or to coordinate among PEIs. Alas, it results in load imbalance due to the presence of skewness in the data stream.

3.1.1 The Power of *Both* Choices

We study the problem of load balancing in distributed stream processing engines, which is exacerbated in the presence of skew. We introduce PARTIAL KEY GROUPING (PKG), a new stream partitioning scheme that adapts the classical "power of two choices" to a distributed streaming setting by leveraging two novel techniques: *key splitting* and *local load estimation*. Key splitting leverages *both* choices by relaxing the atomicity constraints of key grouping. Local load estimation solves the problem of gauging the load of downstream servers without any communication overhead. Both techniques are extremely simple to implement and can be deployed on top of existing DSPEs without requiring changes to their core. In so doing, it achieves better load balancing than key grouping while being more scalable than shuffle grouping. We test PKG on several large datasets, both real-world and synthetic. Compared to standard hashing, PKG reduces the load imbalance by up to several orders of magnitude, and often achieves nearly-perfect load balance. This result translates into an improvement of up to 60% in throughput and up to 45% in latency when deployed on a real Storm cluster.

Contributions. In summary, we make the following contributions.

- We study the problem of load balancing in modern distributed stream processing engines.
- We show how to apply power of two choices to DSPEs in a principled and practical way, and propose two novel techniques to do so: key splitting and local load estimation.
- We propose PKG, a novel and simple stream partitioning scheme that applies to any DSPE. When implemented on top of Apache Storm, it requires a single function and less than 20 lines of code.[1]
- We measure the impact of PKG on a real deployment on Apache Storm. Compared to key grouping, it improves the throughput of an example application on real-world datasets by up to 60%, and the latency by up to 45%.

Related Publications.

1. Nasir, Muhammad Anis Uddin, Gianmarco De Francisci Morales, David Garcia-Soriano, Nicolas Kourtellis, and Marco Serafini. "The power of both choices: Practical load balancing for distributed stream processing engines." In Data Engineering (ICDE), 2015 IEEE 31st International Conference on, pp. 137-148. IEEE, 2015. [51].

2. Nasir, Muhammad Anis Uddin, Gianmarco De Francisci Morales, David Garcia-Soriano, Nicolas Kourtellis, and Marco Serafini. "Partial key grouping: Load-balanced partitioning of distributed streams." arXiv preprint arXiv:1510.07623 (2015). [52].

3.1.2 The Power of *D*-Choices and *W*-Choices

Carefully balancing load in distributed stream processing systems has a fundamental impact on execution latency and throughput. Load balancing is challenging because

[1] Available at `https://github.com/gdfm/partial-key-grouping`

real-world workloads are skewed: some tuples in the stream are associated to keys which are significantly more frequent than others. Skew is remarkably more problematic in large deployments: having more workers implies fewer keys per worker, so it becomes harder to "average out" the cost of hot keys with cold keys.

We propose a novel load balancing technique that uses a heavy hitter algorithm to efficiently identify the hottest keys in the stream. These hot keys are assigned to $d \geq 2$ choices to ensure a balanced load, where d is tuned automatically to minimize the memory and computation cost of operator replication. The technique works online and does not require the use of routing tables. Our extensive evaluation shows that our technique can balance real-world workloads on large deployments, and improve throughput and latency by **150**% and **60**% respectively over the previous state-of-the-art when deployed on Apache Storm.

Contributions. In summary, this paper makes the following contributions:

- We propose two new algorithms for load balancing of distributed stream processing systems; the algorithms are tailored for large-scale deployments and suited to handle the skew in real-world datasets;
- We provide an accurate analysis of the parameters that determine the cost and effectiveness of the algorithms;
- An extensive empirical evaluation[2] shows that our proposal is able to balance the load at large scale and in the presence of extreme skew;
- The reduced imbalance translates into significant improvements in throughput and latency over the state-of-the-art when deployed on Apache Storm.

Related Publications.

1. Nasir, Muhammad Anis Uddin, Gianmarco De Francisci Morales, Nicolas Kourtellis, and Marco Serafini. "When two choices are not enough: Balancing at scale in distributed stream processing." In Data Engineering (ICDE), 2016 IEEE 32nd International Conference on, pp. 589-600. IEEE, 2016. [53].

3.1.3 The Power of *Random* Choices

Streaming applications frequently encounter skewed workloads and execute on heterogeneous clusters. Optimal resource utilization in such adverse conditions becomes a challenge, as it requires inferring the resource capacities and input distribution at run time. In this paper, we tackle the aforementioned challenges by modeling them as a load balancing problem. We propose a novel partitioning strategy called *Consistent Grouping* (CG), which enables each processing element instance (PEI) to process the workload according to its capacity. The main idea behind CG is the notion of small, equal-sized "virtual workers" at the sources, which

[2]Code available at `https://github.com/anisnasir/SLBSimulator` and `https://github.com/anisnasir/SLBStorm`

are assigned to physical workers based on their capacities. In particular, CG achieves 3.44**x** better performance in terms of latency compared to key grouping.

Contributions. In summary, our work makes the following contributions:

- We propose a novel grouping scheme called Consistent Grouping to improve the scalability for DSPEs running on heterogeneous clusters and processing skewed workload.
- We provide a theoretical analysis of the proposed scheme and show the effectiveness of the proposed scheme via extensive empirical evaluation on synthetic and real-world datasets. In particular, CG achieves bounded imbalance and generates almost optimal memory footprint.
- We measure the impact of CG on a real deployment on Apache Storm. Compared to key grouping, it improves the throughput of an example application on real-world datasets by up to 2**x**, reduces the latency by 3.44**x**.

Related Publications.

1. Nasir, Muhammad Anis Uddin, Hiroshi Horii, Marco Serafini, Nicolas Kourtellis, Rudy Raymond, Sarunas Girdzijauskas, and Takayuki Osogami. "Load Balancing for Skewed Streams on Heterogeneous Cluster." arXiv preprint arXiv:1705.09073 (2017). [54] (under submission).

3.2 Fully Dynamic Graph Algorithms

Next, we move our attention towards studying graph related problems in fully dynamic settings. In particular, we study two problems that are: top-k densest subgraph and mining frequent graph patterns. Finding the top-k densest subgraphs in a sliding window is of interest to several real-time applications, e.g., community tracking [46], event detection [47], story identification [5], fraud detection [48], and more. Similarly, frequent subgraph mining (FSM) has numerous applications ranging from biology to network science, as it provides a compact summary of the characteristics of the graph.

3.2.1 Top-k Densest Subgraph for Sliding Windows

Given a large graph, the densest-subgraph problem asks to find a subgraph with maximum average degree. When considering the top-k version of this problem, a naïve solution is to iteratively find the densest subgraph and remove it in each iteration. However, such a solution is impractical due to high processing cost. The problem is further complicated when dealing with dynamic graphs, since adding or removing an edge requires re-running the algorithm. In this paper, we study the top-k densest-subgraph problem in the sliding-window model and propose an efficient fully-dynamic algorithm. The input of our algorithm consists of an edge stream, and the goal is to find the node-disjoint subgraphs that maximize the sum of their densities. In contrast to existing state-of-the-art solutions that require iterating over the entire graph upon any update, our algorithm profits from the observation that updates only affect a limited region of the graph. Therefore, the top-k densest subgraphs are maintained by only applying local updates. We provide a theoretical analysis of the proposed algorithm and show empirically that the algorithm often generates denser subgraphs than state-of-the-art competitors. Experiments show an improvement in efficiency of up to five orders of magnitude compared to state-of-the-art solutions.

Contributions. In summary, we make the following contributions:

- We study the top-k densest vertex-disjoint subgraphs problem in the sliding-window model.
- We provide a brief survey on adapting several algorithms for densest subgraph problem for the top-k case.
- We propose a scalable fully-dynamic algorithm for the problem, and provide a detailed analysis of it.
- The algorithm is open source and available online, together with the implementations of all the baselines.[3]
- We report a comprehensive empirical evaluation of the algorithm in which it significantly outperforms previous state-of-the-art solutions by several orders of magnitude, while producing comparable or better quality solutions.

Related Publications.

1. Muhammad Anis Uddin Nasir, Aristides Gionis, Gianmarco De Francisci Morales, and Sarunas Girdzijauskas. 2017. Fully Dynamic Algorithm for Top-k Densest Subgraphs. In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM '17). ACM, New York, NY, USA, 1817-1826 [55].

3.2.2 Top-k Frequent Graph Patterns in Evolving Graphs

Given a labeled graph, the frequent-subgraph mining (FSM) problem asks to find all the k-vertex subgraphs that appear with frequency greater than a given threshold. FSM has numerous applications ranging from biology to network science, as it provides a compact summary of the characteristics of the graph. However, the task is challenging, even more so for evolving graphs due to the streaming nature of the input and the exponential time complexity of the problem. In this paper, we initiate the study of approximate FSM problem in both incremental and fully-dynamic streaming settings, where arbitrary edges can be added or removed from the graph. For each streaming setting, we propose algorithms that can extract a high-quality approximation of the frequent k-vertex subgraphs for a given threshold, at any given time instance, with high probability. In contrast to the existing state-of-the-art

[3]`https://github.com/anisnasir/TopKDensestSubgraph`

solutions that require iterating over the entire set of subgraphs for any update, our algorithms operate by maintaining a uniform sample of k-vertex subgraphs with optimized neighborhood-exploration procedures local to the updates. We provide theoretical analysis of the proposed algorithms and emprically demonstrate that the proposed algorithms generate high-quality results compared to baselines.

Contributions. Concretely, our main contributions are the following:

- We are the first to study the problem of frequent-subgraph mining on evolving labeled graphs.
- We propose a new subgraph-based sampling scheme.
- We show how to use random pairing to handle deletions.
- We describe how to implement neighborhood exploration efficiently via "skip optimization."
- We provide theoretical analysis and guarantees on the accuracy of the algorithm.

Related Publications.

1. Cigdem Aslay, Muhammad Anis Uddin Nasir, Gianmarco De Francisci Morales, and Aristides Gionis. 2018. Mining Frequent Patterns in Evolving Graphs [56] (under submission).

Chapter 4

Conclusion and Future Work

Someone is sitting in shade today
because someone planted a tree
long time ago.

Warren Buffett

In this thesis, we studied several interesting problems in the domain of real-time data processing. We proposed several algorithms and optimizations for the problems from infrastructure to algorithm level. We pushed the state-of-the-art solutions in two unique dimensions and opened several other interesting questions in the domain.

First, we designed optimization techniques for large-scale distributed systems, which enable operating under adverse situations, i.e., skewed input and heterogeneous environments. In particular, we studied the load-balancing problem for distributed stream processing engines, which is caused by the skewness in the workload and heterogeneity in the cluster. In doing so, we developed several algorithms to reduce the load imbalance across a distributed system. We empirically showed that our algorithms achieve superior performance compared to the state-of-the-art approaches. Moreover, our algorithms are integrated into Apache Storm, which is an open source stream processing framework.

Our work gives rise to further interesting research questions in the domain of load balancing for distributed stream processing engines. Is it possible to achieve good load balance without foregoing atomicity of processing of keys? What are the necessary conditions, and how can it be achieved? In particular, can a solution based on rebalancing be practical? And in a larger perspective, which other primitives can a DSPE offer to express algorithms effectively while making them run efficiently? While most DSPEs have settled on just a small set, the design space still remains largely unexplored.

Second, we designed extremely efficient and accurate approximation algorithms for challenging and interesting graph-related problems. First, we studied the top-k densest subgraphs problem for graph streams, and proposed an efficient one-pass fully dynamic algorithm. In contrast to the existing state-of-the-art solutions that require iterating over the entire graph upon update, our algorithm maintains the solution in one-pass. Additionally, the memory requirement of the algorithm is independent of k. The algorithm is designed by leveraging the observation that graph updates only affect a limited region. Therefore, the top-k densest subgraphs are maintained by simply applying local updates to small subgraphs, rather than the complete graph. We provided a theoretical analysis of the proposed algorithm and showed empirically that the algorithm often generates denser subgraphs than the state-of-the-art solutions. Further, we observed an improvement in performance of up to five orders of magnitude when compared to the baselines.

This work gives rise to further interesting research questions: Is it necessary to leverage k-core decomposition algorithm as a backbone? Is it possible to achieve stronger bounds on the threshold for high-degree vertices? Can we design an algorithm with a space bound on the size of the bag? Is it possible to achieve stronger approximation guarantees for the problem? We believe that solving these questions will further enhance the proposed algorithm, making it a useful tool for numerous practical applications.

Lastly, we initiated the study of approximate frequent-subgraph mining (FSM) in both incremental and fully-dynamic streaming settings, where the edges can be arbitrarily added or removed from the graph. For each streaming setting, we proposed algorithms that can extract a high-quality approximation of the frequent k-vertex subgraph patterns, for a given threshold, at any given time instance, with high probability. Our algorithms operate by maintaining a uniform sample of k-vertex subgraphs at any time instance, for which we provide theoretical guarantees. We also proposed several optimizations to our algorithms that allow achieving high accuracy with improved execution time. We showed empirically that the proposed algorithms generate high-quality results compared to natural baselines.

This work gives rise to further interesting research questions: Is it necessary to leverage reservoir sampling along with random pairing? Is it possible to leverage ℓ_0-sampler for the dynamic case? Can we incorporate random walk like approaches into the solution? Is it possible to achieve stronger approximation guarantees for the problem? We believe that our work has opened an interesting domain of research with many unanswered questions.

Bibliography

[1] S. Madden, "From databases to big data," *IEEE Internet Computing*, vol. 16, no. 3, pp. 4–6, 2012.

[2] G. D. F. Morales and A. Bifet, "Samoa: scalable advanced massive online analysis." *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 149–153, 2015.

[3] 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.

[4] M. Mathioudakis and N. Koudas, "Twittermonitor: trend detection over the twitter stream," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 1155–1158.

[5] A. Angel, N. Sarkas, N. Koudas, and D. Srivastava, "Dense subgraph maintenance under streaming edge weight updates for real-time story identification," *VLDB*, vol. 5, no. 6, pp. 574–585, 2012.

[6] A. Clauset, C. R. Shalizi, and M. E. Newman, "Power-law distributions in empirical data," *SIAM review*, vol. 51, no. 4, pp. 661–703, 2009.

[7] M. E. Newman, "The structure and function of complex networks," *SIAM review*, vol. 45, no. 2, pp. 167–256, 2003.

[8] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *ACM SIGCOMM computer communication review*, vol. 29, no. 4. ACM, 1999, pp. 251–262.

[9] J. Lin *et al.*, "The curse of zipf and limits to parallelization: A look at the stragglers problem in mapreduce," in *7th Workshop on Large-Scale Distributed Systems for Information Retrieval*, vol. 1. ACM Boston, MA, USA, 2009, pp. 57–62.

[10] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in *OSDI 12*, 2012, pp. 17–30.

[11] Y. Ben-Haim and E. Tom-Tov, "A Streaming Parallel Decision Tree Algorithm," *JMLR*, vol. 11, pp. 849–872, 2010.

[12] R. Berinde, P. Indyk, G. Cormode, and M. J. Strauss, "Space-optimal heavy hitters with strong error bounds," *ACM Trans. Database Syst.*, vol. 35, no. 4, pp. 1–28, 2010.

[13] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. ACM, 1996, pp. 20–29.

[14] M. R. Henzinger, P. Raghavan, and S. Rajagopalan, "Computing on data streams." *External memory algorithms*, vol. 50, pp. 107–118, 1998.

[15] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang, “On graph problems in a semi-streaming model,” *Theoretical Computer Science*, vol. 348, no. 2-3, pp. 207–216, 2005.

[16] S. Muthukrishnan *et al.*, “Data streams: Algorithms and applications,” *Foundations and Trends® in Theoretical Computer Science*, vol. 1, no. 2, pp. 117–236, 2005.

[17] 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.

[18] J. M. Utterback and W. J. Abernathy, “A dynamic model of process and product innovation,” *Omega*, vol. 3, no. 6, pp. 639–656, 1975.

[19] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, “S4: Distributed stream computing platform,” in *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on.* IEEE, 2010, pp. 170–177.

[20] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham *et al.*, “Storm@ twitter,” in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data.* ACM, 2014, pp. 147–156.

[21] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja, “Twitter heron: Stream processing at scale,” in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data.* ACM, 2015, pp. 239–250.

[22] T. Akidau, A. Balikov, K. Bekiroğlu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle, “Millwheel: fault-tolerant stream processing at internet scale,” *Proceedings of the VLDB Endowment*, vol. 6, no. 11, pp. 1033–1044, 2013.

[23] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt *et al.*, “The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing,” *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1792–1803, 2015.

[24] M. Kleppmann and J. Kreps, “Kafka, samza and the unix philosophy of distributed data.” *IEEE Data Eng. Bull.*, vol. 38, no. 4, pp. 4–14, 2015.

[25] 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.

[26] J. Kreps, N. Narkhede, J. Rao *et al.*, “Kafka: A distributed messaging system for log processing,” in *Proceedings of the NetDB*, 2011, pp. 1–7.

[27] A. Videla and J. J. Williams, *RabbitMQ in action: distributed messaging for everyone.* Manning, 2012.

[28] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, “Pregel: a system for large-scale graph processing,” in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data.* ACM, 2010, pp. 135–146.

[29] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, “Graphx: A resilient distributed graph system on spark,” in *First International Workshop on Graph Data Management Experiences and Systems.* ACM, 2013, p. 2.

[30] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein,

"Distributed graphlab: a framework for machine learning and data mining in the cloud," *Proceedings of the VLDB Endowment*, vol. 5, no. 8, pp. 716–727, 2012.

[31] Y. Low, J. E. Gonzalez, A. Kyrola, D. Bickson, C. E. Guestrin, and J. Hellerstein, "Graphlab: A new framework for parallel machine learning," *arXiv preprint arXiv:1408.2041*, 2014.

[32] S. Salihoglu and J. Widom, "Gps: A graph processing system," in *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*. ACM, 2013, p. 22.

[33] R. R. McCune, T. Weninger, and G. Madey, "Thinking like a vertex: a survey of vertex-centric frameworks for large-scale distributed graph processing," *ACM Computing Surveys (CSUR)*, vol. 48, no. 2, p. 25, 2015.

[34] A. McGregor, "Graph stream algorithms: a survey," *SIGMOD*, vol. 43, no. 1, pp. 9–20, 2014.

[35] V. Batagelj and M. Zaversnik, "An o (m) algorithm for cores decomposition of networks," *arXiv preprint cs/0310049*, 2003.

[36] J. Chen and Y. Saad, "Dense subgraph extraction with application to community detection," *TKDE*, vol. 24, no. 7, pp. 1216–1230, 2012.

[37] Y. Dourisboure, F. Geraci, and M. Pellegrini, "Extraction and classification of dense communities in the web," in *WWW*. ACM, 2007, pp. 461–470.

[38] D. Gibson, R. Kumar, and A. Tomkins, "Discovering large dense subgraphs in massive graphs," in *VLDB*. VLDB Endowment, 2005, pp. 721–732.

[39] T. Akiba, Y. Iwata, and Y. Yoshida, "Fast exact shortest-path distance queries on large networks by pruned landmark labeling," in *SIGMOD*. ACM, 2013, pp. 349–360.

[40] B. Bahmani, R. Kumar, and S. Vassilvitskii, "Densest subgraph in streaming and mapreduce," *VLDB*, vol. 5, no. 5, pp. 454–465, 2012.

[41] S. Bhattacharya, M. Henzinger, D. Nanongkai, and C. Tsourakakis, "Space-and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams," in *STOC*. ACM, 2015, pp. 173–182.

[42] A. Epasto, S. Lattanzi, and M. Sozio, "Efficient densest subgraph computation in evolving graphs," in *WWW*, 2015, pp. 300–310.

[43] A. Gionis and C. E. Tsourakakis, "Dense subgraph discovery: Kdd 2015 tutorial," in *SIGKDD*. ACM, 2015, pp. 2313–2314.

[44] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *PODS*. ACM, 2002, pp. 1–16.

[45] M. S. Crouch, A. McGregor, and D. Stubbs, "Dynamic graphs in the sliding-window model," in *ESA*. Springer, 2013, pp. 337–348.

[46] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," *ICML*, vol. 42, no. 1, pp. 181–213, 2015.

[47] P. Rozenshtein, A. Anagnostopoulos, A. Gionis, and N. Tatti, "Event detection in activity networks," in *SIGKDD*, 2014, pp. 1176–1185.

[48] A. Beutel, W. Xu, V. Guruswami, C. Palow, and C. Faloutsos, "Copycatch: stopping group attacks by spotting lockstep behavior in social networks," in *WWW*, 2013, pp. 119–130.

[49] A. Bifet, G. Holmes, B. Pfahringer, and R. Gavaldà, "Mining frequent closed graphs on evolving data streams," in *KDD '11*, 2011, pp. 591–599.

[50] A. Ray, L. Holder, and S. Choudhury, "Frequent subgraph discovery in large attributed streaming graphs," in *BigMine*, 2014, pp. 166–181.

[51] M. A. U. Nasir, G. D. F. Morales, D. GarcÃa-Soriano, N. Kourtellis, and M. Serafini, "The power of both choices: Practical load balancing for distributed stream processing engines," in *2015 IEEE 31st International Conference on Data Engineering*, April 2015, pp. 137–148.

[52] M. A. U. Nasir, G. D. F. Morales, D. Garcia-Soriano, N. Kourtellis, and M. Serafini, "Partial key grouping: Load-balanced partitioning of distributed streams," *arXiv preprint arXiv:1510.07623*, 2015.

[53] M. A. U. Nasir, G. D. F. Morales, N. Kourtellis, and M. Serafini, "When two choices are not enough: Balancing at scale in distributed stream processing," in *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, May 2016, pp. 589–600.

[54] M. A. U. Nasir, H. Horii, M. Serafini, N. Kourtellis, R. Raymond, S. Girdzijauskas, and T. Osogami, "Load balancing for skewed streams on heterogeneous cluster," *arXiv preprint arXiv:1705.09073*, 2017.

[55] M. A. U. Nasir, A. Gionis, G. D. F. Morales, and S. Girdzijauskas, "Fully dynamic algorithm for top-k densest subgraphs," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, ser. CIKM '17. New York, NY, USA: ACM, 2017, pp. 1817–1826. [Online]. Available: http://doi.acm.org/10.1145/3132847.3132966

[56] C. Aslay, M. A. U. Nasir, G. D. F. Morales, and A. Gionis, "Mining frequent patterns in evolving graphs," 2018.

[57] J. S. Vitter, "Random sampling with a reservoir," *ACM Transactions on Mathematical Software (TOMS)*, vol. 11, no. 1, pp. 37–57, 1985.

[58] R. Gemulla, W. Lehner, and P. J. Haas, "A dip in the reservoir: Maintaining sample synopses of evolving datasets," in *Proceedings of the 32nd international conference on Very large data bases.* VLDB Endowment, 2006, pp. 595–606.

[59] M. A. U. Nasir, S. Girdzijauskas, and N. Kourtellis, "Socially-aware distributed hash tables for decentralized online social networks," in *2015 IEEE International Conference on Peer-to-Peer Computing (P2P)*, Sept 2015, pp. 1–10.

[60] M. A. U. Nasir, F. Rahimian, and S. Girdzijauskas, "Gossip-based partitioning and replication for online social networks," in *2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014)*, Aug 2014, pp. 33–42.

[61] M. A. U. Nasir, "Fault tolerance for stream processing engines," *arXiv preprint arXiv:1605.00928*, 2016.

[62] M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. N. Gray, P. P. Griffiths, W. F. King, R. A. Lorie, P. R. McJones, J. W. Mehl *et al.*, "System r: Relational approach to database management," *ACM Transactions on Database Systems (TODS)*, vol. 1, no. 2, pp. 97–137, 1976.

[63] M. Stonebraker, G. Held, E. Wong, and P. Kreps, "The design and implementation of ingres," *ACM Transactions on Database Systems (TODS)*, vol. 1, no. 3, pp. 189–222, 1976.

[64] J. D. Ullman, *Principles of database systems.* Galgotia publications, 1984.

[65] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of databases: the logical level.* Addison-Wesley Longman Publishing Co., Inc., 1995.

[66] R. Ramakrishnan and J. Gehrke, *Database management systems.* McGraw Hill, 2000.

[67] D. J. DeWitt, R. Gerber, G. Graefe, M. Heytens, K. Kumar, and M. Muralikrishna, *A High Performance Dataflow Database Machine.* Computer Science Department, University of Wisconsin, 1986.

[68] M. Stonebraker and U. Cetintemel, "" one size fits all": an idea whose time has come and gone," in *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on.* IEEE, 2005, pp. 2–11.

[69] M. Stonebraker, C. Bear, U. Çetintemel, M. Cherniack, T. Ge, N. Hachem, S. Harizopoulos, J. Lifter, J. Rogers, and S. Zdonik, "One size fits all? part 2: Benchmarking results," in *Proc. CIDR*, 2007.

[70] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland, "The end of an architectural era:(it's time for a complete rewrite)," in *Proceedings of the 33rd international conference on Very large data bases.* VLDB Endowment, 2007, pp. 1150–1160.

[71] J. Meehan, N. Tatbul, S. Zdonik, C. Aslantas, U. Cetintemel, J. Du, T. Kraska, S. Madden, D. Maier, A. Pavlo *et al.*, "S-store: Streaming meets transaction processing," *Proceedings of the VLDB Endowment*, vol. 8, no. 13, pp. 2134–2145, 2015.

[72] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. P. Jones, S. Madden, M. Stonebraker, Y. Zhang *et al.*, "H-store: a high-performance, distributed main memory transaction processing system," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1496–1499, 2008.

[73] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil *et al.*, "C-store: a column-oriented dbms," in *Proceedings of the 31st international conference on Very large data bases.* VLDB Endowment, 2005, pp. 553–564.

[74] D. J. Abadi, S. R. Madden, and N. Hachem, "Column-stores vs. row-stores: How different are they really?" in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data.* ACM, 2008, pp. 967–980.

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

[76] P. G. Brown, "Overview of scidb: large scale array storage, processing and analysis," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data.* ACM, 2010, pp. 963–968.

[77] M. Stonebraker, P. Brown, D. Zhang, and J. Becla, "Scidb: A database management system for applications with complex analytics," *Computing in Science & Engineering*, vol. 15, no. 3, pp. 54–62, 2013.

[78] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[79] T. White, *Hadoop: The definitive guide.* " O'Reilly Media, Inc.", 2012.

[80] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *ACM SIGOPS operating systems review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[81] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive: a warehousing solution over a map-reduce framework," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1626–1629, 2009.

[82] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, “Pig latin: a not-so-foreign language for data processing,” in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data.* ACM, 2008, pp. 1099–1110.

[83] S. Owen and S. Owen, “Mahout in action,” 2012.

[84] 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 Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[85] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, “Dremel: interactive analysis of web-scale datasets,” *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 330–339, 2010.

[86] J. Shute, R. Vingralek, B. Samwel, B. Handy, C. Whipkey, E. Rollins, M. Oancea, K. Littlefield, D. Menestrina, S. Ellner *et al.*, “F1: A distributed sql database that scales,” *Proceedings of the VLDB Endowment*, vol. 6, no. 11, pp. 1068–1079, 2013.

[87] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, “Dryad: distributed data-parallel programs from sequential building blocks,” in *ACM SIGOPS operating systems review*, vol. 41, no. 3. ACM, 2007, pp. 59–72.

[88] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, “Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing,” in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation.* USENIX Association, 2012, pp. 2–2.

[89] A. Lumsdaine, D. Gregor, B. Hendrickson, and J. Berry, “Challenges in parallel graph processing,” *Parallel Processing Letters*, vol. 17, no. 01, pp. 5–20, 2007.

[90] K. Munagala and A. Ranade, “I/o-complexity of graph algorithms,” in *SODA*, vol. 99, 1999, pp. 687–694.

[91] L. G. Valiant, “A bridging model for parallel computation,” *Communications of the ACM*, vol. 33, no. 8, pp. 103–111, 1990.

[92] J. Zhong and B. He, “Medusa: Simplified graph processing on gpus,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1543–1552, 2014.

[93] F. Khorasani, K. Vora, R. Gupta, and L. N. Bhuyan, “Cusha: vertex-centric graph processing on gpus,” in *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing.* ACM, 2014, pp. 239–252.

[94] Y. Wang, A. Davidson, Y. Pan, Y. Wu, A. Riffel, and J. D. Owens, “Gunrock: A high-performance graph processing library on the gpu,” in *ACM SIGPLAN Notices*, vol. 51, no. 8. ACM, 2016, p. 11.

[95] G. J. Chen, J. L. Wiener, S. Iyer, A. Jaiswal, R. Lei, N. Simha, W. Wang, K. Wilfong, T. Williamson, and S. Yilmaz, “Realtime data processing at facebook,” in *Proceedings of the 2016 International Conference on Management of Data.* ACM, 2016, pp. 1087–1098.

[96] G. De Francisci Morales, “Samoa: A platform for mining big data streams,” in *Proceedings of the 22nd International Conference on World Wide Web.* ACM, 2013, pp. 777–778.

[97] N. Kourtellis, G. D. F. Morales, and F. Bonchi, “Scalable online betweenness

centrality in evolving graphs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 9, pp. 2494–2506, 2015.

[98] N. Ohsaka, T. Akiba, Y. Yoshida, and K.-i. Kawarabayashi, "Dynamic influence analysis in evolving networks," *Proceedings of the VLDB Endowment*, vol. 9, no. 12, pp. 1077–1088, 2016.

[99] L. D. Stefani, A. Epasto, M. Riondato, and E. Upfal, "Triest: Counting local and global triangles in fully dynamic streams with fixed memory size," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 11, no. 4, p. 43, 2017.

[100] E. Wu, Y. Diao, and S. Rizvi, "High-performance complex event processing over streams," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data.* ACM, 2006, pp. 407–418.

[101] D. Anicic, S. Rudolph, P. Fodor, and N. Stojanovic, "Stream reasoning and complex event processing in etalis," *Semantic Web*, vol. 3, no. 4, pp. 397–407, 2012.

[102] U. Dayal, B. Blaustein, A. Buchmann, U. Chakravarthy, M. Hsu, R. Ledin, D. McCarthy, A. Rosenthal, S. Sarin, M. J. Carey *et al.*, "The hipac project: Combining active databases and timing constraints," *ACM Sigmod Record*, vol. 17, no. 1, pp. 51–70, 1988.

[103] J. Widom, "The starburst rule system: Language design, implementation, and applications," *IEEE Data Engineering Bulletin*, 1992.

[104] M. Stonebraker and G. Kemnitz, "The postgres next generation database management system," *Communications of the ACM*, vol. 34, no. 10, pp. 78–92, 1991.

[105] N. H. Gehani and H. V. Jagadish, "Ode as an active database: Constraints and triggers." in *VLDB*, vol. 91, 1991, pp. 327–336.

[106] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang, "Niagaracq: A scalable continuous query system for internet databases," in *ACM SIGMOD Record*, vol. 29, no. 2. ACM, 2000, pp. 379–390.

[107] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah, "Telegraphcq: continuous dataflow processing," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data.* ACM, 2003, pp. 668–668.

[108] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J. Widom, "Stream: the stanford stream data manager (demonstration description)," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data.* ACM, 2003, pp. 665–665.

[109] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: a new model and architecture for data stream management," *the VLDB Journal*, vol. 12, no. 2, pp. 120–139, 2003.

[110] U. Cetintemel, "The aurora and medusa projects," *Data Engineering*, vol. 51, no. 3, 2003.

[111] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina *et al.*, "The design of the borealis stream processing engine." in *Cidr*, vol. 5, no. 2005, 2005, pp. 277–289.

[112] G. A. Agha, "Actors: A model of concurrent computation in distributed systems." Massachusetts Institute Of Technology Cambridge Artificial Intelligence Lab, Tech. Rep., 1985.

[113] A. Floratou, A. Agrawal, B. Graham, S. Rao, and K. Ramasamy, "Dhalion:

self-regulating stream processing in heron," *Proceedings of the VLDB Endowment*, vol. 10, no. 12, pp. 1825–1836, 2017.

[114] 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." *HotCloud*, vol. 12, pp. 10–10, 2012.

[115] O. Boykin, S. Ritchie, I. O'Connell, and J. Lin, "Summingbird: A framework for integrating batch and online mapreduce computations," *Proceedings of the VLDB Endowment*, vol. 7, no. 13, pp. 1441–1451, 2014.

[116] Q. Huang and P. P. Lee, "Toward high-performance distributed stream processing via approximate fault tolerance," *Proceedings of the VLDB Endowment*, vol. 10, no. 3, pp. 73–84, 2016.

[117] V. Braverman and R. Ostrovsky, "Smooth histograms for sliding windows," in *FOCS'07*. IEEE, 2007, pp. 283–293.

[118] D. Yates, D. S. Moore, and D. S. Starnes, *The practice of statistics: TI-83/89 graphing calculator enhanced*. Macmillan, 2002.

[119] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *Journal of computer and system sciences*, vol. 31, no. 2, pp. 182–209, 1985.