Thesis for The Degree of Licentiate of Engineering

Data Sketches and Parallelism for Efficient Data Pipelines

Martin Hilgendorf

Department of Computer Science and Engineering
Chalmers University of Technology | University of Gothenburg
Gothenburg, Sweden, 2026

Data Sketches and Parallelism for Efficient Data Pipelines

Martin Hilgendorf

ISSN 1652-876X

Department of Computer Science and Engineering
Division of Computer and Network Systems
Distributed Computing and Systems
Chalmers University of Technology | University of Gothenburg
SE-412 96 Göteborg,
Sweden
Phone: +46(0)31 772 1000

Printed by Chalmers Digitaltryck,
Gothenburg, Sweden 2026.

We don't abandon our pursuits because we despair of ever perfecting them.

—Epictetus, 'Discourses'

Data Sketches and Parallelism for Efficient Data Pipelines

Martin Hilgendorf

Department of Computer Science and Engineering
Chalmers University of Technology | University of Gothenburg

Abstract

Data Summarisation transforms massive datasets into convenient and compact *summaries*, or *synopses*, to approximate the result of queries. Such summaries are orders of magnitude smaller than the original data, with mathematical guarantees on the approximation accuracy, making them an attractive solution to the challenges of Big Data. In recent years, *Data Sketches* have seen widespread adoption for efficiently summarising Big Data in a single pass and in small memory and time. This thesis studies several challenges arising when utilising sketches or compression in Big Data processes and *pipelines*.

Parallelising the construction of sketches becomes essential with high data rates and volumes. Simultaneously, long-running analytics processes in continuous, high-rate pipelines require *concurrent, low-latency querying* of sketches, concurrently with updates. Further, the ability to estimate several query types from a single sketch avoids construction of multiple different sketches, for space, timeliness, and consistency reasons. Integrating all these dimensions for the first time, this thesis proposes LMQ-Sketch and explores the challenging trade-offs in designing parallel data sketches for the classical problem of frequency moment and item frequency estimation. The work sheds light to necessary synchronisation among concurrent operations, carefully balancing consistency, freshness, and accuracy with a low memory footprint and high throughput.

Data-intensive systems that distribute work across threads or nodes by locally building sketches — subsequently shared and merged — but may face workload and resource fluctuations. To this end, the thesis introduces ReSketch as a *resizable*, *partitionable* sketch with *enhanced mergeability*, permitting merging sketches of different sizes to obtain a good balance of memory footprint and accuracy, while enabling novel elastic capabilities.

Beyond studying data summaries as pipeline components, the thesis presents FORTE, a framework for lossless data transfer pipelines where compression, necessary for timely, reliable, and efficient transmission, must be balanced against resource constraints, while ensuring data integrity, confidentiality, and governance. Latency, throughput, and sustainable rates are measured in a real-world industrial use-case managing TBs of data per day, demonstrating benefits of effective pipelining, scheduling, and efficient use of resources. Altogether, this thesis examines how data reduction and pipeline optimisation can enable parallel, efficient, and performant data pipelines, opening new avenues for bridging the gap between algorithms and practical deployment in adaptable, scalable, resource-aware data processing systems.

Keywords

Data Pipelines, Data Summarisation, Data Sketches, Approximate Query Processing, Parallelism and Concurrency, Hashing, Compression, Scheduling

List of Publications

Appended Publications

This thesis is based on the following publications and submitted research works:

[**Chapter A**] **Martin Hilgendorf** and Marina Papatriantafilou, *LMQ-Sketch: Lagom Multi-Query Sketch for High-Rate Online Analytics.*
In 39th International Symposium on Distributed Computing (DISC 2025). Leibniz International Proceedings in Informatics (LIPIcs), Volume 356, pp. 36:1–36:24, Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 2025.

[**Chapter B**] Vinh Quang Ngo, **Martin Hilgendorf**, and Marina Papatriantafilou, *ReSketch: A Mergeable, Partitionable, and Resizable Sketch.*
Under revision. 2026.

[**Chapter C**] **Martin Hilgendorf**, Vincenzo Gulisano, Marina Papatriantafilou, Jan Engström, and Binay Mishra, *FORTE: An Extensible Framework for Robustness and Efficiency in Data Transfer Pipelines.*
In Proceedings of the 17th ACM International Conference on Distributed and Event-based Systems (DEBS '23), Neuchâtel, Switzerland, pp. 139–150. 2023.

Other publications

The following works were also published during my studies. However, they are not appended to this thesis, due to contents overlapping that of appended publications or contents not related to the thesis.

- Carl-Magnus Wall, Måns Josefsson, **Martin Hilgendorf**, Marina Papatriantafilou, and Binay Mishra, *V-Mon: Scalable and Fault-Tolerant Stream Processing Pipeline for Monitoring Vehicular Data Validity.* In Proceedings of the 19th ACM International Conference on Distributed and Event-based Systems (DEBS '25), Gothenburg, Sweden, pp. 249–250. 2025.

Authorship Statement

The following statement summarises my personal contribution to each of the chapters included in this thesis. Roles described are according to the Contributor Roles Taxonomy (CRediT)[1]. Where relevant (Chapter B), co-authors have agreed with it.

[Chapter A] *LMQ-Sketch: Lagom Multi-Query Sketch for High-Rate Online Analytics.*

I lead the conceptualisation, methodology, software implementation, validation, analysis, and manuscript writing and editing, with supervision and support from Marina Papatriantafilou.

[Chapter B] *ReSketch: A Mergeable, Partitionable, and Resizable Sketch.*

Vinh Quang Ngo and I shared the work as follows: both contributed to the conceptualisation, implementation, writing, and editing under the lead of Vinh. I planned the experimental evaluation, and Vinh developed the theoretical analysis.

[Chapter C] *FORTE: An Extensible Framework for Robustness and Efficiency in Data Transfer Pipelines.*

I contributed as the lead responsible for the conceptualisation, methodology, software implementation, and manuscript writing and editing. Jan Engström and Binay Mishra supported the conceptualisation, investigation, methodology, and resources. Marina Papatriantafilou and Vincenzo Gulisano contributed with supervision and supported writing and editing.

[1] `https://credit.niso.org/`

Acknowledgment

Thank you, first and foremost, to my supervisor, Marina Papatriantafilou, for her never-ending patience, the support and guidance she has provided me on this journey so far, the deep and insightful discussions, sharing her experience, and originally accepting the Master's thesis proposal that marked the beginning of our collaboration. I am excitedly looking forward to continuing on the next part of this journey. I also thank my co-supervisors Vincenzo Gulisano and Philippas Tsigas for their advice and support throughout.

Thank you to the discussion leader for my Licentiate seminar, Paris Carbone, for accepting the invitation to this role, and to my PhD examiner Alejandro Russo for his support throughout the process.

Thank you to Yixing (and Pidan), Kåre, Vinh, Jacob, and Jingyu for all the great experiences, trips, card games, discussions, shared advice and office chats along the way together. You have made this adventure so far a truly enjoyable and memorable experience.

Thank you also to the colleagues in my unit and division, including, amongst others, Ahmed, Atmane, Azadeh, Elad, Francisco, Hashim, Ilias, Magnus, Marco, Masoom, Muoi, Rhouma, Romaric, Tomas, Torbjörn, Umer, Yasir, and Yenan, as well as Bastian, Christos, Dimitris, Georgia, Huaifeng, and Kalle.

Thank you to the research school and administrative staff who have simplified day-to-day tasks and supported my journey, particularly Clara, Pamela, Lasse, and Monica.

Thank you to the Data Science Lab team at Volvo: Tomislav, Janne, Binay, Anton, and all others, as well as my student-now-collaborator Carl-Magnus.

Thank you to the Swedish Research Council (Vetenskapsrådet) for funding my PhD education under project *"EPITOME: Summarization and structuring of continuous data in concurrent processing pipelines"* with grant number 2021-05424.

Thank you to my family. To my parents, who never stopped believing in me, and their continuous support that enabled me to get to where I am now. To my brother, for always supporting me and inspiring me through a great journey of his own.

I am grateful to you all.

Martin Hilgendorf
Göteborg, May 2026

Contents

Part I

Thesis Overview

Chapter 1

Introduction

The proliferation of data collection and gathering practices in modern applications, systems, and products — motivated by a need to enable increasingly complex and detailed analytics — to extract well-informed knowledge and conclusions in the age of Big Data has brought a variety of technological challenges. *Data pipelines* are the multi-step processes used for moving, storing, indexing, transforming, and processing these massive datasets. However, as Big Data continues to outgrow the capabilities of traditional data processing approaches, deploying additional resources, both in data centres and at edge systems, is beginning to reach feasibility limits. Applications contend with network bandwidth limitations and latency constraints, while scaling up processing resources comes at increased capital cost and energy consumption. Hence, novel techniques to efficiently capture, store, and analyse ever-growing quantities of data become increasingly important.

1.1 Big Data

Modern applications and systems often include a multitude of sensors and data capturing functionalities which generate a myriad of signals and data points, constituting large and ever-growing datasets. Examples include cyberphysical systems such as Industrial Internet of Things (IIoT), sensor networks on edge devices, or connected autonomous vehicles that sense and communicate detailed data about their environment for path planning, coordination, or detecting hazards.

Datasets from such applications continue to grow, eventually reaching the realm of Big Data, where their sheer sizes effectively prevents timely collection, storage, or processing. Big Data is commonly brings two challenges to systems managing it: its *volume*, with datasets exceeding TB and PB, and its *velocity*, the rate at which it is produced, emitted, sensed, or transmitted.

The rapid development of technology, particularly the growth of computer processor performance, has enabled massive increases in these characteristics of datasets that are captured and processed. As Big Data continue to grow bigger,

however, a critical question arises: How can these never-before-seen quantities of data be turned into something useful — getting value out of the Big Data —, and how to do so, quickly and efficiently? The challenge is twofold:

1. Application constraints: Big Data applications may have vastly different requirements, e.g. on the latency or timeliness of processing results: some require near-real-time results, others aggregate data and analyse it in batches on a daily basis. Needing to transfer bulk data for processing in turn places requirements on the communication network, for example in terms of adequate communication latency and providing sufficient bandwidth.

   Further, the needed accuracy of results may differ: for coordination of vehicles, processing of, e.g., camera, lidar, or geospatial data may need to be performed at very high resolution, while monitoring of performance metrics in a large-scale online platform may tolerate some inaccuracy from operating on lower-resolution data, e.g. rounded numbers, but gain a lot in efficiency through avoided complexity by doing so.

2. System constraints: Available hardware resources in the processing system, including processors, communication networks, and data storage, determine the capacity for processing work that can be performed within an acceptable latency. Applications for Big Data processing hence need to utilise the available hardware and its capabilities in efficient ways.

Traditional processing techniques face scalability challenges in one or both of these; as data scales beyond the capabilities of the processing system, the application's requirements on processing latency or throughput may no longer be met.

Parallelising Processing Typical datasets in Big Data environments have long outgrown the storage and processing capacities of single processors, requiring distributing the data over many processors and nodes working together in clusters. While a massive dataset can be stored in full in a data centre in this way, answering queries about the data may require reading the complete dataset, an expensive operation. Parallelising query operations by distributing the work across nodes, e.g. in database clusters using MapReduce [22], Apache Hadoop and HDFS [5], Apache Spark [8], etc., has enabled great speed-ups. Distributed processing brought a variety of challenges to be addressed, such as the high cost of data movement between storage and processing locations. Hence, a commonly adopted approach instead opts to bring processing capabilities closer to the data, as in MapReduce. More recently, Processing-in-Memory hardware technologies appear, which place processing capabilities near or in the physical data storage chips, with the aim to reduce energy, bandwidth, and latency costs of data movement [44].

While systems advancements aim to increase available processing capabilities, applications of Big Data processing also increase in complexity. Autonomous vehicles or production lines in industrial Internet of Things and Industry 4.0 demand low-latency communication in order to perform the

necessary analytics, scheduling, and planning for safe operation in a timely manner. To enable precise and correct decision-making, such system may include high-resolution sensors to provide sufficiently detailed data in order to complete critical task with the required level of safety; for example, coordinating the journey of two autonomous mining trucks passing each other in narrow tunnels [56, 57]. In these cases, transferring such high-resolution data is no longer be feasible due to its sheer volume and velocity in relation to available bandwidth and associated communication cost; and processing must instead be performed onboard the vehicle for such critical tasks. As computational resources on vehicles are limited, both due to cost and energy demand, this may require moving other tasks off-board by harnessing fog or cloud compute resources instead. While scaling up those resources to cope with ever-growing requirements can be a solution in the short-term, keeping up with Big Data ultimately requires rethinking the processing itself.

Key Idea To address these challenges, we should reconsider the system design and more directly address the root of the problem: *Instead of scaling up the resources, begin by scaling down the data and making more efficient use of existing resources* [28]. The idea is to reduce the amount of data to be processed to smaller, more convenient sizes, thereby reducing the cost of performing the processing.

1.2 Data Summarisation

Reducing Big Data to more manageable volumes can be done by identifying which subset of its properties or aspects are relevant for downstream analysis, and retaining only the needed parts of the data, thus transforming the Big Data into summarised representations [17, 20]. These summaries — significantly smaller and more convenient than the original dataset — can then be used to answer particular queries about the data much more efficiently. To accomplish this increase in efficiency, data summaries trade off some accuracy, an acceptable exchange for a large number of applications. For example, monitoring the number of visitors to a popular online service may not require accuracy to the last digit; a close-enough result can be acceptable, especially if it can be calculated in a significantly cheaper fashion.

Importantly, these data summaries should be lightweight to produce, ideally requiring only a single pass over the original data. Further, continuous data operations such as data streaming pipelines can benefit from summaries that can be constructed and updated incrementally with new data items. For example, distributed monitoring, where a central coordinator needs to track statistics about data streams observed at distributed nodes, can make use of summaries to efficiently reduce communication cost. Instead of nodes forwarding all data items to the coordinator for accurate processing, the nodes can instead locally and incrementally construct compact summaries of their respective data stream, which are then transmitted to and merged at the coordinator. As streaming data continues to grow in prevalence and velocity, becoming too large to store

in its entirety, summaries become an essential component in the data analytics stack.

1.3 Roadmap

The following chapters of this Thesis Overview part introduce further background, preliminaries, and the state of the art in data pipelines, concurrency, and data sketching (Chapter 2). Associated open research problems and their challenges are described, along with the research questions posed by the thesis (Chapter 3), followed by a summary of the research contributions and how they address these questions (Chapter 4). This part concludes with an outlook on interesting directions for future research in this areas (Chapter 5). Part II consists of three chapters which detail the contributions of the thesis.

Chapter 2

Background

This chapter provides background information on several key areas for the works in this thesis. First, data pipelines and their design, characteristics, and key challenges are discussed (Section 2.1), followed by an overview of concepts in concurrency and parallelism for shared-memory and distributed systems (Section 2.2). Data sketches are described in Section 2.3, including foundational sketches for different types of queries about data, and several works from the current state of the art in the domains of parallelism for sketches.

2.1 Data Pipelines

A pipeline describes the sequence of operations that constitute a process. Pipelines consist of several steps, here termed *tasks*, through which data is passed in order: the output of a task becomes the input to the following task. Pipelines are found in many aspects of computing, from instruction pipelining within processors for executing multiple instructions in parallel on a single hardware thread to improve hardware utilisation, network pipelining in protocols such as HTTP, to software pipelines where various programs are executed in parallel and pass data to each other in sequence. In the domain of data analytics and Big Data, pipelines are commonly used to model and implement various such as data transfer, transformations, or analysis workflows over *batches* of data.

2.1.1 Concepts

The decomposition of a larger process into individual steps, that can execute in parallel on different data items, allows the work to be *pipelined*. Pipelining can be seen similar to an assembly line; while a long process will not see its *latency* reduced by parallelising the various tasks, *throughput* improves as all tasks can perform useful work and idleness is avoided. Overall throughput of the pipeline will be limited by one or more *bottlenecks*: the task with lowest throughput relative to other tasks in the pipeline. Thus, to avoid

bottlenecks and throughput penalties, a pipeline should balance the throughput performance of the constituting tasks.

2.1.2 Data Models

In *batch processing*, pipeline tasks operate on a batch of multiple data items (e.g. files or database records) in bulk, on a schedule or triggered by an event [37]. Data is first fully stored in large-scale bulk storage, such as databases or distributed file systems (e.g., the Hadoop Distributed File System [5]), and is then always available to the processing if and when needed [39].

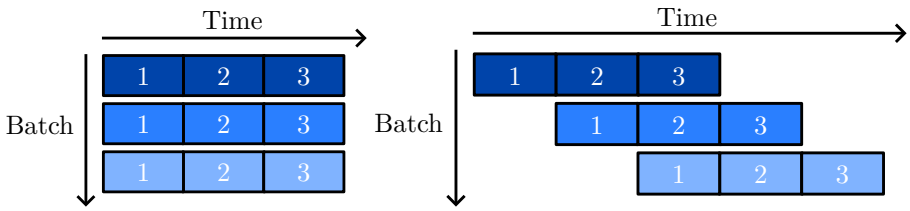
In *streaming* pipelines, data arrives continuously in streams of *tuples*, small units (usually) of similar size and *schema* (the structure of the tuple, including contained fields and their data types) [21, 39]. Due to the high arrival rate of input tuples (cf. the velocity aspect of Big Data) — as is becoming more common in the modern applications featuring near-real time processing of sensors, transactions, or online activities — the volume of data becomes too large to be stored in its entirety (as in batch pipelines), and must instead be processed immediately in an *online* fashion; else, the data is lost [37].

As input data arrives incrementally, stream processing pipelines need to maintain summaries of the data seen so far by applying techniques such as sampling [15], building summaries (as studied in this thesis) [20], or applying fixed-size moving *windows* containing only the most recent elements of the stream to analyse. Notably, a streaming pipeline typically does not control the rate of the input data stream; incremental maintenance of these structures must be therefore be efficient and quick in order to keep pace with the input rate and avoid bottlenecks.

2.1.3 Implementing Pipelines

A common programming model for data pipelines is to consider tasks as the smallest schedulable units, and arranging them in a directed acyclic graph (DAG) to model their communication and dependencies, to enable correct scheduling. *Workflow orchestration* managers are centralised tools for scheduling and coordinating execution of batch pipelines and their tasks, as well as providing a frontend for users to observe the status of the pipeline. Examples include Apache Airflow [6], Argo Workflows [9], and Dagster [43].

Stream processing engines such as Apache Flink [4] provide tools for implementing such online processing pipelines and continuously producing query results — as opposed to the traditional batch data analytics model, where data is first fully stored and indexed in databases or large-scale *data warehouses*, and subsequently queried asynchronously or on-demand by consumers. Streaming pipelines consist of long-running tasks called *operators*, which are again composed into DAGs to model the, now continuous, flow of data through the pipeline. Operators can optionally store a limited local state, such as the aforementioned samples, windows, or summaries, while other operators are *stateless*, for example implementing a filter to discard certain tuples based on predefined predicates.



(a) Data parallelism. **(b)** Pipeline parallelism.

Figure 2.1: Executions of a simple pipeline consisting of three tasks (numbered), processing three input data batches (coloured).

2.1.4 Parallelism in Pipelines

Data pipelines commonly parallelise work in two ways, illustrated in Figure 2.1: *Data parallelism* involves splitting the input data into smaller parts, which are then processed in parallel 'instances' of a pipeline. This permits scaling out the processing by distributing it across multiple processors or nodes, each executing the same task(s) on a different subset of the data. On the other hand, each additional parallel execution demands a full set of the processing resources required — if task 2 in Figure 2.1(a) represents a network transfer from a remote site, all files will contend for the available network bandwidth. Workload can be balanced by adjusting the amount of data assigned to each processor; the figure shows a perfectly balanced workload assignment.

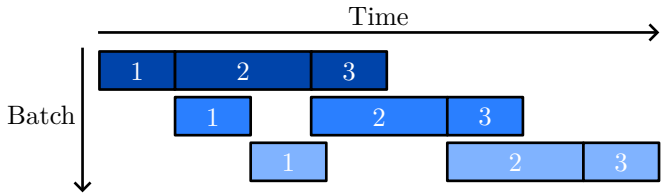
Pipeline parallelism instead exploits the task-sequence nature of pipelines by executing different pipeline tasks in parallel, e.g. on different batches of input data. If pipeline tasks are serviced by dedicated, but limited, resources or execution units, e.g. transfers from remote sites, this approach can improve utilisation of those resources by avoiding idleness or contention — the execution shown in Figure 2.1(b) concurrently executes an instance of each of the three tasks during the middle time-step.

2.1.5 Key Challenges

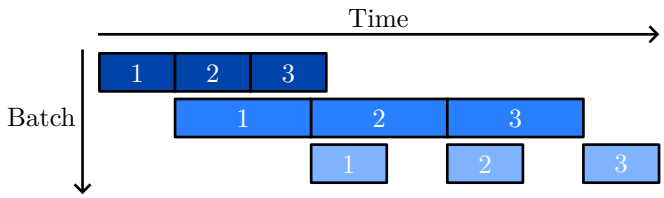
Imbalance, Scheduling, and Pipeline Bubbles

In an ideal setting, a (batch) pipeline is perfectly balanced in terms of the throughput of its tasks, input data batches are of identical size, and each task hence takes an identical amount of time (as illustrated in Figure 2.1). If either of these cases does not hold, one task may have finished its processing, but the subsequent task is still busy with a prior batch, and processing cannot continue.

Consider again the example pipeline in Figure 2.1(b), where task 2 represent a data transfer but relatively limited network bandwidth forms a throughput bottleneck in the pipeline. Now, shown in Figure 2.2(a), task 1 of the second batch completes while the transfer in task 2 of the first batch is still ongoing – processing of the second batch becomes *stalled* and it cannot proceed. This stall introduces a pipeline *bubble* in the execution and leads to idleness, increased



(a) Low throughput in task 2 stalls the processing of subsequent batches.



(b) Larger size of the middle batch stalls all tasks for subsequent batches.

Figure 2.2: Examples of pipeline stalls and bubbles.

latency and reduced throughput. Similarly, if input data batches are of uneven size, larger batches may introduce stalls for subsequent batches, as shown in Figure 2.2(b), leading to a *convoy effect*.

Uneven batch sizes in the first example can be addressed by load balancing to re-batch data into more evenly sized batches, if the application and nature of the data permit this, for example image processing of many small, independent files. Alternatively, batches can be scheduled for processing using different policies; for example, a shortest job first policy could avoid convoy effects by prioritising smaller batches, allowing them to overtake larger ones. However, compared to a strict FIFO policy, this may cause starvation for large batches as they could be queueing for potentially unbounded amounts of time depending on arrival patterns of smaller batches.

Identifying and addressing task bottlenecks (Figure 2.2(a)) can require in-depth understanding of the environment in which the pipeline is executing. Within this environment, pipeline components do not operate in isolation, and configuration adjustments for performance tuning can impact the performance of other tasks. For example, to address the slow transfer, data could be compressed before sending. However, compression and decompression demand additional processing resources, increasing contention for these, which may in turn adversely impact other tasks. This is to say that there may be intricate dependencies and non-obvious impacts, which can make balancing performance across tasks a challenge, motivating detailed performance modelling of the pipeline, understanding of its parameters and efficient harnessing of available resources.

Approximate Query Processing

Unlike batch pipelines, where ample storage and processing resources have been assumed to exist, streaming pipelines cannot keep the entire data for

analysis [21]. Further, the rates of streams often prohibit the use of large but slow bulk data storage systems, requiring all processing to take place in fast, but more orders of magnitude smaller than the input data, main memory. As a consequence, this generally makes it infeasible to compute exact results to queries, excluding trivial ones, over the data. Instead, the processing pipeline must transform the input stream into a more compact representation, a process which inevitably removes information while maintaining certain key features of the input more accurately [20, 39]. Two key questions arise from this insight: selecting what information to preserve, and how accurately to represent it. Answering these questions requires identifying the types of data summarisation algorithms that can efficiently summarise the needed attributes of the input stream, and the resources needed to do so with an acceptable level of accuracy.

2.2 Concurrency and Parallelism

This section provides a brief overview of a number of challenges arising in parallel and concurrent programming, including concurrent data structures and correctness notions. Parallelism is typically studied in two communication models [10]: *message passing*, common in distributed systems, or *shared-memory*, as in a multiprocessor system with several processor cores or threads with simultaneous access to the memory.

Distributed Parallelism A distributed system consists of a number of separate nodes. Nodes can perform processing using private, local resources, and communicate by exchanging messages over a network in order to coordinate or share information. For data pipelines and distributed processing, data, tasks, and results must be sent over the network, and communication cost becomes an important constraint for the performance of the system [27].

Shared-Memory Parallelism In shared-memory systems, multiple processors or threads have efficient access to a shared global memory space, and execute asynchronously [31]. When two or more such processes or threads operate on the shared memory, without any coordination, their respective sequences of operations can effectively *interleave* in arbitrary ways. It is up to the programmer to introduce the necessary *synchronisation* between threads in order to prevent the undesirable incorrect interleavings, and for the result of the program to be *correct* (e.g. with respect to a given specification of the program in a sequential setting) [54].

Atomic operations are fundamental primitives for implementing synchronisation, which ensure that a sequence of operations is executed as an uninterruptible, indivisible unit. A common technique to guarantee atomic execution of a sequence of operations is to require *mutual exclusion*, where only a single process at a time is permitted to execute the operations in this *critical section*. Locks are common abstractions for implementing mutual exclusion in concurrent programs, doing so by effectively preventing concurrency for parts of the program.

When reasoning about the correctness of a realistic-size concurrent program, it becomes infeasible to consider all possible interleavings of operations to verify that only correct ones are possible. Instead, a theoretical model is needed to provide guarantees regarding the *safety* and *liveness* properties of the execution. Safety properties, informally, assert that "bad things never happen", for example, that at most one thread can be executing inside a critical section at any point, or that all dequeue operations on a concurrent queue must return items that have been previously enqueued —, i.e., that the program adheres to the correct sequential order of operations and does not violate invariants (*sequential correctness*). Liveness properties additionally guarantee that the system makes *progress* — informally: "good things eventually happen"; e.g., one thread will eventually (within bounded time) enter the critical section if it is currently free and one or more threads are waiting to enter —, and that operations which have started will eventually complete.

2.2.1 Concurrent Data Structures

In order to achieve high performance, concurrent data structures need to maximise the amount of work that can be performed concurrently, by keeping critical sections small, and using efficient synchronisation primitives [31]. For example, fine-grained locking around smaller parts of a data structure can permit non-conflicting operations to proceed concurrently, e.g. a concurrent FIFO queue may permit an enqueue and a dequeue operation to happen concurrently at opposites sides of the data structure, while two dequeue operations would need to synchronise and serialise their work.

Beyond fine-grained synchronisation, another approach is to *relax* correctness requirements in order to permit more concurrency and higher performance. Of course, the concurrent execution must not deviate from a correct one further than a predetermined tolerable extent, i.e., the relaxation must be *bounded* [30].

2.2.2 Intermediate Value Linearisability (IVL)

Linearisability [32] is a common model for operation ordering requirements in concurrent data structures. Considering an execution consisting of many operations, each having a start (call) and end (return) time, performed across several threads, linearisability requires that each operation appears to take effect instantaneously (at its *linearisation point*), maintaining their real-time ordering. For example, consider a shared counter being concurrently read and updated by two threads; linearisability guarantees that the read will observe either the counter value before or after the update, determined by the relative ordering of the linearisation points of the two operations.

However, implementing linearisable concurrent data structures can be expensive, and the guarantees provided stronger than necessary for approximate processing applications [51], where relaxation is a familiar and necessary approach to achieve high performance. To this end, Intermediate Value Linearisability (IVL) relaxes the guarantees of linearisability in a controlled fashion, trading (stricter-than-necessary) correctness guarantees for increased concurrency and

performance. Considering again the shared counter example from above, but instead of requiring the read to return either exactly the value before or after the update, IVL permits the read to also return any *intermediate* value between these – an intuitive and often acceptable result.

2.3 Data Sketches

Data sketches describe a family of data summarisation algorithms, using a small (relative to the input data size) amount of memory to provide approximate answers to a query about the input data [20]. Sketch summaries can be built and maintained incrementally, making them suitable as components in data processing pipelines for various types of queries that are otherwise too expensive to compute accurately, such as item frequencies, top-k elements, heavy hitters, frequency moments, or range queries.

The accuracy of a sketch is typically tied to its size; a larger summary will be able to retain more detailed information in its state, permitting more accurate approximation of the query result. Approximation accuracy is commonly described in terms of the absolute or relative error ε of the estimated query result with respect to the accurate result [21]. For summaries building on randomisation, such as sketches, there is a small failure probability δ of violating this ε error bound. The sketch literature provides theoretical analyses of the trade-off between sketch memory requirements and the (ε, δ) approximation guarantees that can be provided.

2.3.1 Estimating Item Frequencies

Given a stream of arriving items sampled from a domain $\mathcal{U}$ (IP-address pairs, for example), the aim is to provide approximations for the number of occurrences of any item in the domain seen so far, also called *point queries*. Accurate counting would need memory on the order of the size of the domain $|\mathcal{U}|$, which is infeasible for many real-world applications. For example, a network monitoring system tracking the frequency of all 2^{64} IP address source-destination pairs, using a dedicated counter for each to guarantee accuracy, would require thousands of petabytes of memory. Instead, summarising the frequencies using sketching can achieve high accuracy while using orders of magnitude fewer counters, hence less memory, by effectively sharing counters for multiple elements.

Count-Min Sketch

The Count-Min Sketch (CMS) [18] is a widely adopted algorithm (e.g., found in Redis [50] or Apache Spark [7]) for estimating the frequencies of elements. It builds on the above-mentioned idea of sharing counters for tracking multiple items. It consists of a 2D array of counters, or *buckets*, with H rows (height) and K buckets per row (width), where $K \ll |\mathcal{U}|$. The (ε, δ) approximation guarantees are achieved by setting $H = \lceil e/\varepsilon \rceil$ and $K = \lceil \ln(1/\delta) \rceil$. Associated with each row i is a hash function h_i, which uniformly maps items from the input domain to bucket indices, i.e., the range $[1, 2, \ldots, K]$.

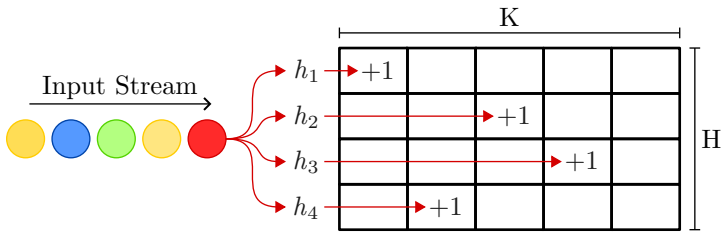


Figure 2.3: Count-Min Sketch update operation.

Update As illustrated in Figure 2.3, for each incoming data item, the hash function at every row determines which counter in the row to update; that counter is then incremented. As such, the CMS update operation is very light-weight, supporting high-rate updates of the sketch.

Query To perform a point query, the hash functions of all rows are evaluated to find the buckets at the indices which the queried item is mapped to. As each of the H selected counters contains the exact frequency of the queried item as well as frequencies of various other items mapped to the same bucket, the query returns the min of these counters as the approximation, thus minimising the overestimation introduced by other items being counted using the same counters.

As multiple items are assigned to the same counter, each counter stores the sum of their frequencies. Hence, the value in the counter will always be greater or equal to the true frequency of an item mapped to that counter, causing the query result to be an overestimation. The magnitude of this overestimation depends on how how many occurrence of other items are counted in the same counter. For skewed input streams, particularly frequent (heavy) elements will cause their assigned counters to take on large values, causing decreased approximation accuracy (increased error) for other items stored there. This is why the query operation returns the minimum count over multiple rows; the hash functions at each row are randomised and will group items differently. The probability that a given (light) item collides with heavy items *in every row* decreases exponentially with the number of rows (this bring the relationship between δ and number of rows H).

Merge Merging two sketches yields a new sketch, which is a summary of the union of the input streams summarised by the sketches. For the CMS, merging requires both sketches to use the same H, K, and per-row hash functions, and is then performed by summing counters in corresponding positions in both arrays.

Augmented Sketch

The Augmented Sketch (ASketch) [53] is an extension of the CMS to improve operation throughput and accuracy by reducing the impact of collisions with

heavy items. To this end, some memory of the CMS is reallocated (usually by reducing K) to a new *filter* data structure that works as a simple key-value lookup table. The filter contains dedicated counters for accurately counting the top-k heaviest elements (k being the number of counter slots in the filter; typically $k = 16$), instead of tracking them in the CMS. Less frequent items are stored in the CMS as before.

The query begins by checking the filter to determine whether the queried item is present there. If so, the query is answered directly from the filter, with improved accuracy due to exact counting, while light items stored in the CMS, and queried as before, also see improved accuracy due to reduced collisions with the heavy items. Care needs to be taken during updates in order to maintain the invariant of the filter always tracking the current top-k elements; if an item no longer belongs to the top-k, it is evicted from the filter, its frequency count moved to the CMS, and the filter slot reassigned to the new member of the top-k heavy items.

2.3.2 Estimating Frequency Moments

Frequency moment sketches estimate functions of the distribution of item frequencies in the input [39]. The frequency of item $a \in \mathcal{U}$ is denoted $f(a)$. Then, the n-th frequency moment is given by:

$$F_n = \sum_{a \in \mathcal{U}} f(a)^n$$

Their uses are many: The 0-th frequency moment, F_0, is the number of unique items in the stream. The first frequency moment, F_1, is the total number of items in the stream (i.e., the sum of all of their frequencies). The second frequency moment, F_2, is the sum of squared item frequencies, and is of particular interest for several applications including self-join size estimation in relational databases, or identifying anomalies that manifest as changes in input skew. It is also referred to as the *surprise number*, as it indicates 'how surprising' it might be to see a new item in the stream; F_2 encodes information about the skewness of the frequency distribution of the stream. Consider two streams of the same length, but with high and low skew in their frequency distribution. For the stream with high skew, some item frequencies will be very large (for the heaviest items), and the calculated F_2 will be large in comparison to the other stream with low skew, where frequencies are relatively similar for all items, and the sum of squares will be lower. Sketches for estimating frequency moments, particularly F_2 and self-join sizes, have been studied since early on [2, 3, 16].

2.3.3 Estimating Ranks and Quantiles

While frequency moment sketches summarise information about the frequency distribution into a single number, other sketches and summaries are needed to query more detailed information about the distribution. To this end, a *rank query* for a given item a returns its rank, given by $\sum_{b<a} f(b)$, that is

the sum of occurrences of all items b that are smaller than a (this requires the domain $\mathcal{U}$ to be ordered). A *quantile query* returns the item which has a given rank [20]. The Karnin-Lang-Liberty (KLL) Sketch [36] is one of several summaries for approximating rank and quantile queries, featuring state-of-the-art approximation error guarantees. It uses multiple levels arrays called *compactors*, which store up to W sampled items from the input stream. When a compactor becomes full, it compacts its contents into $W/2$ items, and doubles the 'weight' associated with the surviving items by promoting them to the next level compactor. Compactors at higher levels thus represent exponentially more input data items. Rank queries are answered by traversing contents of the compactors and summing the weights of all stored items smaller than the queried item.

2.3.4 Parallel Sketches

A key aspect of data sketches are the efficient and simple procedures for building and querying them. However, as data rates and volumes continue to grow, parallelising sketch operations becomes a relevant area of research in order to support the necessary throughput. This section provides an overview of some recent works that aim to enable parallelism for sketch updates, queries, or both.

Delegation Sketch

The Delegation Sketch [55] is one of the first frameworks for distributing sketch update and query operations over multiple hardware threads. A key idea is to split the domain of input items into partitions, where each thread is responsible for one partition. Each thread has a local CMS for tracking frequencies of the items belonging to its partition, and the owning thread has exclusive access to it. Input data is arbitrarily distributed to threads, for example in a round-robin fashion, and any thread can process the next item; if the item belongs to a the threads own partition, it updates its local CMS. Other items are placed into pre-allocated *delegation buffers*, one for each other partition in the system. Once such a buffer is full, it is handed over to the thread owning the contained data to be inserted into its CMS in bulk. Buffering updates reduces communication between threads compared to handing over each individual item.

When querying for the frequency of a specific item, the first step is to determine which partition the item belongs to. Then, the query procedure knows which few components of the whole data structure can contain items belonging to that partition: the CMS of the owning thread, and the corresponding delegation buffers of all other threads. Delegation Sketch supports queries concurrently with updates, so care must be taken regarding concurrent access to these structures. By performing the query on the thread responsible for the partition owning the queried item, a correct and safe query result can be obtained in an efficient manner, without requiring heavy-handed mutual exclusion primitives such as locks. The original paper [55] discusses some claims

regarding the consistency of the concurrent operations in Delegation Sketch, associating with consistency models for bulk operations [47], such as queries which may or may not consider overlapping concurrent updates in their results. A more precise study of the consistency properties of the delegation framework is part of the contributions of this thesis in Chapter A.

Other Related Work

Around the same time as Delegation Sketch, Rinberg et al. introduced "Fast Concurrent Data Sketches" [52], a general model for parallelising any sketch, requiring mergeability. Threads build local summaries, which are periodically merged into a global sketch. Queries are performed on snapshots of this global sketch.

IVL (Section 2.2.2) is highly suited for concurrent data summaries, as its goals regarding relaxation to improve efficiency and performance align directly with the light-weight, efficient operations needed for high-rate data summarisation. Further, Rinberg and Keidar extend the study of IVL by applying it to (ε, δ) data summarisation structures, including CMS, and show that IVL implementations of concurrent sketches maintain the (ε, δ) guarantees of the original sequential sketch.

Other works investigate parallel sketching on architectures other than multiprocessors: SKT [12] targets FPGA-based acceleration of sketching for 100 Gbit/s network packet processing by building multiple sketch instances in parallel, requiring merging of the partial sketches before querying can be done, hence queries cannot be concurrent with updates. Sketching on GPUs has also been studied in works such as [58].

2.3.5 Multi-Query Sketches

Sketches are typically designed to provide an approximate answer to a single type of query. However, it is common for data analytics processes to require various types of statistics about the input data, which would require constructing separate sketches for each such statistic. Despite the efficient algorithms and light-weight space cost of sketches, this may become prohibitively expensive. To this end, sketches which only need to be built once, but are capable of answering multiple types of queries about the data, are a powerful building block for more efficient data analytics pipelines.

In the original Count-Min Sketch paper [18], the authors mention the possibility to estimate additional queries about the data, such as top-k or heavy hitters, although this requires auxiliary data structures such as a heap, in order to store the identities of these heaviest elements (as those are otherwise not recorded, as part of the information discarded during summarisation by the CMS). Later, Cormode and Muthukrishnan in [19] introduced methods to estimate the second frequency moment from an existing Count-Min Sketch, requiring no additional data structures.

SKT [12] also answers multiple types of queries, by constructing instances of three existing sketches concurrently. As the design is FPGA-based, the pre-

viously mentioned time cost for constructing multiple sketches is not applicable, but is instead replaced by a space cost to be paid in hardware.

Estimating range queries, that is, counting the frequency of items in a given range, for high-dimensional data is another area of active research. Here, the challenge lies in being able to accept arbitrary predicates defining the range, i.e., any combination of dimensions to filter for; a naïve solution to this problem would require an number of sketches exponential in the number of attributes of the input. Recent works have advanced the state of the art significantly. Hydra [41] avoids the problem of tracking each subpopulation separately by using a "sketch of sketches" design to maintain memory efficiency. OmniSketch [49] followed the next year, providing a new algorithm with significantly better time complexity (scaling linearly in the number of dimensions of the input data, rather than exponentially as in Hydra).

2.3.6 Resizeable Sketches

Sketches described so far all have one thing in common: their size is static after initialisation, and cannot be adjusted. This prevents several useful functionalities, such as increasing the size of the sketch to maintain desired (ε, δ) guarantees if the input data grows larger than the original sketch was budgeted for, elastically resizing the sketch at runtime depending on the amount of available memory, or shrinking sketches before transmitting them over the network. Several recent works have begun to address this gap, by allowing dynamic resizing of a data sketch.

Elastic Sketch [59] targets high-rate network traffic monitoring, and composes a limited array of counters (for counting frequent, heavy flows) with a CMS (for light flows) to balance high accuracy with memory usage. The data structure can be compressed to enable sending it over low-bandwidth links, while the heavy part can be expanded during execution to permit tracking a larger number of heavy flows, thus supporting varying traffic characteristics.

Dynamic Count-Min Sketch (DCMS) [61] also identifies the challenge of preselecting static sketch parameters before execution, usually without knowledge of the future data stream length. DCMS uses a linked list of Count-Min Sketches, appending a new CMS instance when the current head of the list has reached a certain fill level; queries traverse the list to aggregate occurrences tracked across the instances. Hence, the expansion capability is coarse-grained, and DCMS lacks the ability to shrink to free memory if needed, additionally requiring several auxiliary data structures for determining fill-levels which need to be maintained during updates.

Geometric Sketch (GS) [11], like Elastic Sketch, also targets flow frequency estimation, aiming to be a flexible approach with fine-grained resizability in both directions, enabling adaptability to stream length during runtime and dynamically balancing the memory-accuracy trade-off as necessary to maintain acceptable estimation error.

Chapter 3

Challenges and Research Questions

This chapter presents the challenges relative to the state of the art and the three central research questions which are addressed by the works in this thesis.

3.1 Parallelising Efficient Data Volume Reduction

The primary approach to improving performance of data pipelines investigated in this work centres around reducing data volume by summarisation or compression, often to address a transfer bandwidth bottleneck in the pipeline (Section 2.1.5). Both approaches introduce some additional processing work to the pipeline, often at the edge, introducing additional contention for limited resources and implying added latency. However, as parallel architectures featuring multiprocessors and simultaneous multithreading are widely available, sensibly exploiting them leads to more efficient resource utilisation and the possibility for *latency hiding* — while one thread or process is stalled waiting for data access or transfer, others can continue to make progress.

Lossless data compression of large data can sometimes be multi-threaded [1, 26]. For large quantities of files, it may make sense to instead process several files concurrently. However, parallel compression of files introduces additional reading and writing of multiple files in parallel, placing more demand on bandwidth of memory and storage subsystems, contending with, and possibly starving, other pipeline tasks (Section 2.1.5). Additionally, disk accesses are dispersed over various non-contiguous locations, which can severely impede the performance of hard disk-based bulk storage. In turn, pipeline performance (both in latency and throughput) may decrease as the additional compression work introduces new bottlenecks.

Parallel Data Summarisation and Sketching has been studied in a number of recent works (described in Section 2.3.4), each studying a sub-problem within

this area. To serve as a powerful building block in a long-lived continuous data pipeline, a parallel data sketch could be used to summarise incoming data items, while simultaneously supporting querying by 'downstream' tasks in the pipeline. To this end, efficient designs for synchronisation amongst threads are necessary, such as [55] (Section 2.3.4). However, a data pipeline may require tracking more than one statistic about the contents of the input data stream, e.g. also monitoring skewness and stream length alongside the frequency of individual elements (Section 2.3.5). To track each of these, multiple separate sketch instances could be constructed in parallel. As identified in [12], however, sketch algorithms often perform very similar operations. Hence, eliminating redundant work by constructing, in a parallel fashion, a single sketch with the capability of answering multiple relevant queries from the same summary seems a promising direction to improve efficiency in data pipelines.

These discussions lead to the first research question:

Research Question 1: Parallelism

How can available parallelism (shared-memory or distributed) be exploited for significantly and efficiently reducing data volume in big data pipelines?

3.2 Managing the Balancing Act Around Performance Metrics

When introducing data reduction into a data pipeline, additional work is introduced. As hinted to in Section 3.1, this shifts resource demands, latency, and throughput performance by way of various intricate and complex interdependencies between components of the system. For example, the aforementioned introduction of lossless data compression to improve throughput also introduces additional processing and data movement steps, which could starve other pipeline tasks of those resources and, in the worst case, instead lead to an overall reduction in performance. Hence, care needs to be taken to avoid introducing new system bottlenecks when adding data compression to pipelines.

Even when taking care to perform data volume reduction work *efficiently*, as asked in Research Question 1, several design choices can impact the balance between key metrics in the pipeline. Compression and summarisation work introduce additional pipeline tasks, thus in turn adding latency — though parallelisation can allow latency hiding to alleviate the impact — while improving throughput due to parallelised compression, in turn reducing data volume for subsequent tasks (e.g., network transfers).

In the case of data summarisation, the built-in parameters for balancing query estimation accuracy against summary size allow direct control over this trade-off (Section 2.3). Lower-accuracy approximations can be achieved at lower space and time cost, hence finding a sweet-spot between the accuracy requirements of the application and processing capabilities of the target system is necessary. This requires a deep understanding of the available parameters

and their intricate interplay. For example, improving accuracy by increasing the size of the summary may lead to worse throughput if the data structure now spills out of CPU cache, hence introducing additional latency to update and query operations, in turn aggravating staleness of query results, where updates becomes visible later, and ultimately worsening estimation accuracy relative to the real value at the time of the query.

This is to say that there exist many complex interdependencies between various components in parallel processing systems, and no parameter can be tuned in isolation. Understanding, studying, describing, and managing the available trade-offs becomes a critical step for introducing parallelism-enabled data compression or summarisation into pipelines, and motivates the second research question:

Research Question 2: Balancing

How to balance accuracy and latency (freshness) with efficiency and throughput performance of data reduction algorithms for high-rate, parallel big data pipelines?

3.3 Concurrency Semantics of Parallel Data Sketches

Further challenges arise when implementing concurrent data structures into data pipelines. Data summarisation as a building block in continuous pipelines implies the need for both update and query operations to happen concurrently. Ideally, queries see a fresh summary, one that is an up-to-date view of the complete input data stream, containing all of the data that has been produced so far. This is complicated by the fact that updates may happen concurrently with the query (unless requiring mutual exclusion, though this would slash throughput). Hence, the concurrency semantics and synchronisation designs of parallel data sketch algorithms need to be described, along with their impact on accuracy and freshness:

Research Question 3: Semantics

What can be said regarding the concurrency semantics and estimation accuracy of parallelism-enabled data sketches for high-rate, continuous data pipelines?

Chapter 4

Thesis Contributions

This chapter introduces the research contributions which constitute this thesis, with a particular focus on the answers they provide to the research questions. For each of the three chapters, a brief overview of the work is presented, including the problem, challenges, and approach, followed by a discussion of the links to the research questions. An overview of the links between contributions and research questions is also provided in Table 4.1.

Table 4.1: Overview of chapters and research questions.

Chapter	**RQ1: Parallelism**	**RQ2: Balancing**	**RQ3: Semantics**
LMQ-Sketch [Chapter A]	Concurrent operations (queries and updates), just-enough synchronisation.	Memory-accuracy-concurrency trade-off, partitioning for accuracy and efficiency.	Describing semantics using IVL, Monotonicity of Scans [24], and operation latency.
ReSketch [Chapter B]	Parallel processing with dissimilar configurations permitted by enhanced mergeability.	Novel feature-set requires some additional components and overhead; profit from it to achieve much-improved approximation accuracy.	Instance provenance DAG describes impact of structure-altering operations on estimation error.
FORTE [Chapter C]	Batch-parallel pipelining of work tasks, with latency hiding.	In-memory processing and task scheduling lead to improved latency and throughput despite doing more work.	

4.1 LMQ-Sketch: Lagom Multi-Query Sketch for High-Rate Online Analytics

Lagom[1] *Multi-Query Sketch* (*LMQ-Sketch*) aims to combine two important points of interest in the world of data sketches: (1) answering multiple types of queries from a data summary built in one pass over the input data, in an effort to be more efficient than building multiple separate sketches, and (2) supporting concurrent queries and updates using multiple threads to parallelise operations in order to achieve higher throughput, while clearly describing behaviour and semantics of the resulting concurrent data structure.

4.1.1 Problem & Challenges

In solving this problem, several fundamental challenges arise:

(a) How can multiple queries be supported by a single data sketch? Some answers are presented in the works described in Section 2.3.5, though several of these do not support queries overlapping other queries, or updates, as they target applications constructing sketches first, and querying them later. For the targetted continuous processing applications of LMQ-Sketch, questions of concurrent query result consistency need to be answered.

(b) How can operations on the data structure be distributed across multiple threads, in a safe, correct, yet efficient manner? Works discussed in Section 2.3.4 such as the Delegation Sketch [55] feature proven, efficient multi-threading designs for constructing sketches, using a partitioning and work delegation scheme, though only supporting queries targeting information contained in a single partition. Bringing support for additional query types, with the ability to gather data from multiple partitions, thus becomes a rewarding leap forward, though requiring solutions to a multitude of consistency, synchronisation, and efficiency challenges.

(c) What impact does synchronisation have on freshness, latency, and accuracy of queries? The aforementioned synchronisation may introduce a certain level of overhead for operations, impacting latency and throughput performance. Relaxed consistency models, such as IVL [51], and synchronisation designs can allow significant performance benefits compared to strict lock-based approaches (Section 2.2.2), but what is the impact on accuracy?

(d) How can the behaviour of the operations be described using established theoretical models for concurrency semantics? Downstream consumers of the data structure must be aware of its behaviour under concurrent operations, such as possible (in)consistency between query results, arising from the use of relaxed consistency models, such as time-lapse effects [24].

[1] Lagom (`Wikipedia`) is Swedish and describes 'just the right amount', indicating appropriateness rather than suggesting lack.

What guarantees can be made regarding consistency, accuracy, or freshness of these query results?

4.1.2 Approach

LMQ-Sketch addresses these challenges by presenting a framework for a concurrency-enabled data sketch with support for multiple types of queries: item frequencies (point queries, Section 2.3.1), and the first and second frequency moments, F_1 and F_2 (Section 2.3.2). The domain of input tuples is partitioned, and each partition is summarised in a separate data sketch, which is maintained by a dedicated thread. This domain-splitting approach significantly improves item frequency estimation accuracy compared to using a single large sketch at identical memory budgets, because collisions are restricted to occur only within each partition, similar to [55]. To lessen communication overhead, threads buffer a (bounded) number of updates belonging to another partition before handing them over to the responsible thread which inserts them into the partition sketch in bulk.

Building on the Augmented Sketch (Section 2.3.1, [53]), itself an extension of the Count-Min Sketch (Section 2.3.1, [18]), LMQ-Sketch inherits support for point queries, which can be answered entirely from the *single* partition owning the queried data item. Additional components are introduced to the data structure to support estimating F_1 and F_2 — so-called *global queries*, as they require information from *all* partitions — in a lightweight fashion, utilising per-partition partial results maintained by updater threads to help the query operation progress faster. A lightweight synchronisation design ensures that queries efficiently coordinate their work with concurrent data structure updates, and the resulting concurrency semantics are described in the publication.

4.1.3 Results & Answers to Research Questions

LMQ-Sketch approaches the research questions from a perspective of parallelising data summarisation operations, in order to provide a powerful component for constructing larger streaming data pipelines.

Research Question 1: Parallelism The resulting design of LMQ-Sketch supports concurrent sketch update and query operations, utilising multiple available processor threads in parallel to achieve high throughput. Further, multiple query types are supported from a single data structure, bringing efficiency and consistency benefits compared to building multiple separate sketches, which may not be in sync at all times. The domain-splitting and partitioning approach permits highly scalable and performant operation for updater threads by reducing inter-thread communications, while light-weight synchronisation between updates and queries permits a high degree of concurrency. Performed benchmarks provide quantitative insight to the impact of design choices and parameters on the efficiency of the shared-memory-parallel design of LMQ-Sketch.

Research Question 2: Balancing Several key aspects with intricate interactions need to be balanced by LMQ-Sketch. To navigate this, LMQ-Sketch contributes a balanced synchronisation design, named Lagom, which enables low-latency, concurrent querying, even for global queries, while balancing freshness, timeliness, accuracy, and high throughput. A detailed empirical evaluation compares the design to a lock-based approach, strictly correct but orders of magnitude slower, and one lacking any synchronisation, permitting maximal concurrency but showing severe losses in the resulting consistency. The middle-ground of the balanced approach — by slightly relaxing consistency requirements in a controlled fashion in order to achieve high performance — enables handling the complexity of concurrently supporting three types queries on the same data structure.

Research Question 3: Semantics The consistency properties of the Lagom synchronisation design used in LMQ-Sketch are described in terms of *Intermediate Value Linearisability* (IVL) [51] and *monotonicity of scans* [24] (Section 2.2), two established concepts from the literature. The theoretical analysis describes the achieved semantics of each query type, including bounding the number of concurrently-performed updates that a query may miss, i.e., the worst-case staleness of its result. For the most complex F_2 queries, a step-by-step analysis highlights how each constituent component — partitioning, delegation buffers, and partial results; each introduced to improve latency, throughput, and memory efficiency — affects the achieved semantics, while also revealing that IVL-ness itself is not enough to characterise estimation accuracy: long query latency can lead to large, yet IVL-permitted, uncertainty. This provides further insight to *operation latency* as another dimension of the consistency-concurrency trade-off navigated by LMQ-Sketch.

4.2 ReSketch: A Mergeable, Partitionable, and Resizable Sketch

Data sketches are a practical solution for monitoring approximate queries over high-velocity data streams. Further, the ability to merge data sketches enables constructing them in parallel and distributed systems, for higher throughput performance. However, merging sketches traditionally requires them to be configured with identical parameters, set statically at sketch initialisation time. Chapter B proposes a novel methodology, ReSketch, to address these limitations.

4.2.1 Problem & Challenges

Static parameter selection for sketches can lead to restricted usefulness in applications experiencing dynamic memory budgets or workloads, or heterogeneity in capabilities across nodes. As an illustrative example, Chapter B considers a distributed network monitoring system. Nodes with varying processing capabilities observe network traffic flows, and summaries frequency information in

local sketches. A central coordinator receives sketches from the devices, and would merge them to produce a single, representative sketch which can be queried. Additionally, monitoring nodes may join and leave the system due to network topology reconfiguration or elastic scaling, e.g. in cloud systems, to manage dynamic load.

Traditional sketches lack the necessary flexibility to support three necessary key properties: (1) Resizability (Section 2.3.6): Only few sketches support expanding [61] or shrinking [59] (or both [11]) their size in order to align with dynamic resource availability; expanding to improve accuracy, shrinking to free memory or compress the summary size. (2) Enhanced Mergeability: Traditionally, merging two sketches requires them to have the same size. Chapter B identifies and defines *enhanced mergeability*, the ability to merge two sketches of different sizes, and selecting the size, hence estimation accuracy, of the resulting sketch. (3) Partitioning: partitioning the input to different sketches can improve throughput and accuracy, and has been shown in prior works such as [29, 33, 34, 46, 55], but is usually static. Employing partitioning to perform load balancing, e.g. in distributed processing, however requires adjusting partitions dynamically in order to adapt to changes in the input. Further, as nodes may join and leave the distributed processing system, load balancing partitions also must change to reflect such changes in the topology.

4.2.2 Approach

A key insight to address the described challenges is the need for more flexible hash functions. Traditional hash functions used in data sketches map input items to buckets using a static mapping, commonly involving a modulo operation using the sketch width as the modulus. However, in order to efficiently resize sketches, without moving contents of all buckets to new locations, Chapter B employs *consistent hashing* [35] instead. In consistent hashing, the hash space is modelled as a logical ring, and buckets as contiguous, non-overlapping segments. While the function that maps an input item to a point on the ring remains static, resizing the sketch, i.e., adding or removing buckets, is performed by adjusting the bucket boundaries on the ring, and only items that find themselves in a new bucket need to be moved.

A "sketch of sketches" design is employed, replacing the usual counters inside the sketch with quantile sketches (Section 2.3.3), to enable splitting or merging of buckets during resize or partition-adjustment operations. Although the nested sketches use additional memory, they also provide extra information for frequency query estimation with improved accuracy, hence becoming part of the memory-accuracy trade-off.

Finally, ReSketch introduces *partition-aware hashing*, to enable efficient and dynamic (re-)partitioning of a sketch. Essentially, a new, superordinate hash ring is used to assign input data items to a partition, described as a segment on this global partition ring. Each partition is summarised by a sketch instance, which internally uses the above-mentioned consistent hashing rings to determine bucket placements for incoming items. This additional level of hashing permits merging and splitting partitions, e.g. for supporting processing

nodes that join or leave during continuous monitoring scenarios.

4.2.3 Results & Answers to Research Questions

ReSketch is the first sketch design to permit arbitrary resizing, merging of sketch instances with different dimensions, and dynamic re-partitioning (merging and splitting) while maintaining high estimation accuracy and operation throughput throughout the operations. It enhances sketches with a new set of efficient structural operations to enable their use in highly dynamic data pipeline applications.

Research Question 1: Parallelism The enhanced mergeability of ReSketch enables parallelism without enforcing homogeneity for participating threads, processes, or nodes, no longer requiring the use of identical sketch configuration parameters in order for their produced sketches to be mergeable. Hence, powerful nodes can produce larger summaries with higher approximation accuracy, while less powerful nodes, or nodes with less available memory, may produce smaller sketch instances, yet their contributions to the monitoring system are mergeable.

Research Question 2: Balancing The new estimator used in ReSketch utilises available memory budget for tracking quantiles and permits orders of magnitude better accuracy (measured in terms of average absolute and relative error) compared to Count-Min Sketch and other baselines using the same memory budget. Thus, it provides a new balance between overhead needed for novel functionality and approximation accuracy.

Furthermore, ReSketch permits elasticity, for example to adapt to available memory by resizing the sketch as needed, or dynamically selecting a suitable compression level for the sketch when it needs to be sent over a limited bandwidth connection. The ability to resize the sketch allows altering the traditionally static memory-accuracy balance of the sketch dynamically, in a fine-grained manner that was previously not possible.

Research Question 3: Semantics A theoretical analysis for the estimation accuracy of the ReSketch estimator is provided, offering insight to the worst-case performance of the algorithm in terms of the usual (ε, δ) bounds for sketches. Further, the impact of the sketch structure-altering operations, needed for dynamic and distributed sketching, is also described, e.g., the approximation accuracy bounds achieved when merging two sketch instances. To this end, the work introduces an *instance provenance DAG* to model the execution history of each sketch instance, consisting of its lineage of ancestor sketches and the structure operations that have occurred, to derive their combined impact on estimation error of the current sketch.

4.3 FORTE: An Extensible Framework for Robustness and Efficiency in Data Transfer Pipelines

As Big Data scales up across many types of domains, building pipelines to automate handling, managing, transferring, and structuring the data become essential. Chapter C presents FORTE, a framework for implementing real-world, large-scale, Big Data transfer pipelines for industrial applications, with a particular focus on bulk transfers of batch data.

4.3.1 Problem & Challenges

Data often originates from a multitude of devices at the edge of networks, and must then be transferred and consolidated in a central location for processing and storage. Automating such transfers is an essential part of Big Data operations, but must also satisfy additional requirements in an industrial setting, including ensuring security (confidentiality and integrity) of the transferred data, governance (monitoring, cataloguing, and traceability), as well as scalability, robustness, and efficiency of the pipeline itself.

4.3.2 Approach

FORTE is an extensible framework for robust, efficient big data transfer pipelines. The data pipeline is modelled as a DAG of tasks, enabling granular reasoning about performance, bottlenecks, and exploration of the design space to tackle associated challenges.

Transferring big data in an environment with limited bandwidth demands a way of reducing the data transfer volume. Depending on application demands, the loss of accuracy from data summarisation may not be acceptable, and lossless compression is required instead. A particular focus is placed on performance engineering of the compression stage, rather than decompression, as data originating and compressed at edge devices must contend for the limited resources available there, while the receiving data centre performing the decompression typically has comparatively abundant processing capacity. To this end, FORTE offsets the additional latency and resource demand due to data compression work by merging adjacent pipeline tasks, allowing them to execute together and share data to each other directly via in-memory buffers, thus avoiding the need for slow reads and writes of bulk data to and from storage disks. This technique is conceptually similar to operator fusion found in stream processing engines such as Apache Flink [4], where it optimises operator code generated from pipeline DAGs for better efficiency by improved code and data locality [40, 45]. These tasks are merged into a large task which is then scheduled as a single unit.

Beyond this, scheduling of pipeline tasks is also studied as a key technique for managing pipeline performance. Scheduling task execution in a pipelined fashion allows latency hiding of, e.g., slow transfers, by batching data and

pipelining their processing. Different scheduling policies, such as FIFO and shortest-job-first (SJF), also affect the behaviour of the system, for example in average throughput or starvation-freedom guarantees; Chapter C compares the various trade-offs of these policies and their impact on pipeline performance.

4.3.3 Results & Answers to Research Questions

FORTE is evaluated in a real-world industrial application for automating the transfer of high-volume data. The available trade-offs in the design space are studied and evaluated, achieving improvements in transfer latency and sustained transfer rates while also bringing support for additional required features including monitoring, tracing, and data security.

Research Question 1: Parallelism FORTE operates on batches of data, which pass through a sequence of steps that constitute the data pipeline. As such, processing of batches can occur in a pipeline-parallel fashion, effectively harnessing compute resources available at distributed locations (e.g., on both ends of a long-distance transfer). Lossless compression provides the necessary reduction in volume in order for the transfer throughput to be sufficient for the expected daily volume of input data, despite adding additional demand on limited CPU and storage bandwidth.

Research Question 2: Balancing The cost of data compression must be carefully considered and balanced in order to yield a net benefit to pipeline performance (throughput and reduced latency). FORTE manages the impact of integrating data compression into transfer pipelines by fusing pipeline tasks, enabling in-memory processing of data for efficient communication, to offset the additional performance cost (in terms of resource demand, latency, and throughput) of performing data compression tasks.

FORTE also studies the effect of scheduling strategies on balancing latency and throughput in the pipeline; while FIFO scheduling ensures liveness and starvation freedom, heterogeneous batch sizes induce pipeline bubbles and convoy effects, causing slowdowns for subsequent batches. Utilising shortest-job-first scheduling shows benefits to average per-batch latency and throughput, but may cause large batches to queue for potentially unbounded amounts of time. Hence, the task scheduling strategy in such a distributed data pipeline has significant impact on the balance of latency and throughput performance.

Chapter 5

Conclusion and Future Work

This thesis studies various aspects of managing ever-growing Big Data in data pipelines. Using efficient techniques for summarising the data promises great benefits in resource requirements at edge devices and data centres, and significant reductions in data transfer costs. Research on data sketches has spanned multiple decades, and as they are now finding adoption in real-world applications, aspects of their integration into data pipelines and systems need to be studied.

To this end, three proposed approaches have been fully implemented and evaluated as part of this thesis. LMQ-Sketch studies challenges arising in multi-threaded data sketching, where operations can overlap arbitrarily, and where the resulting performance and semantics aspects need to be carefully balanced. ReSketch proposes a new set of techniques for enhancing sketches with additional operations to overcome limitations following from their traditionally static configuration. Example applications include distributed data processing systems, where heterogeneity and dynamism in resources and topology are common. The work contributes the capabilities needed for deploying sketches in modern elastic platforms and applications, by allowing dynamic adjustment of sketch parameters for adapting to current workload and available resources. Zooming out, FORTE considers end-to-end systems aspects of entire Big Data transfer pipelines, and how to make them reliable, robust, and efficient while supporting critical features for industrial settings including data security and governance, as well as also timely delivery in resource- and bandwidth-constrained systems.

Several interesting follow-up questions for future research directions arise along the trajectory of these works. LMQ-Sketch lays a foundation which can be extended with support for additional query types, for example top-k queries, building on gained insights regarding semantics and consistency, to further enhance its applicability as a component for memory-efficient, concurrent streaming data processing pipelines. Further, as both ReSketch and LMQ-Sketch each offer a novel set of capabilities, exploring the trade-offs and

possibilities arising when combining their respective features to enable dynamic resizing, merging, and (re-)partitioning — e.g., adjusting the number of partitions and threads — of concurrency-enabled multi-query sketches, would enable an interesting direction to explore on the way towards bringing data summaries into demanding dynamic pipeline applications. Building on the benchmarks in Chapter B, an important follow-up would investigate the performance of ReSketch in further detail, shedding light to the effect of its tuning knobs in conjunction with data- and execution-dependent performance behaviours, for example in distributed edge-fog data analytics applications.

Beyond the immediate horizon of these works, several other aspects and applications of data summarisation appear relevant for enabling their future use in Big Data analytics systems and pipelines. An interesting topic for further study of data sketches is the hash function computations at the core of their otherwise simple algorithms, and how they could more efficiently harness features of modern computing architectures such as vectorised processing to improve performance for high-throughput applications. A relevant example of such applications is network monitoring, e.g. for security purposes such as sketch-based rate limiting [38, 48], where line-rate processing using high-performance packet processing technologies allows offloading the processing directly to network interface hardware [13, 25], enabling the necessary high throughput for packet processing pipelines. Earlier works have studied hierarchical data processing architectures, where distributed nodes propagate local summaries towards a central site for tracking statistics [16, 27]. Processing-in-network technologies using programmable switches can be used to offload processing and data aggregation along the route of the data towards such central sites, permitting high-throughput processing and pipelining as well as latency hiding for data analytics in transit [42]. Furthermore, the intersection between data summarisation and data privacy, e.g. in differential privacy [23, 60], appears as a timely and relevant area. Moreover, there are clear links to current topics of large-scale data mining and machine learning, where massive data structures and vectors are common and techniques for dimensionality reduction are highly sought after to improve efficiency and accelerate data processes [14].

References

[1] Mark Adler, *Pigz - Parallel Gzip* Aug. 20, 2023. URL: https://github.com/madler/pigz.

[2] Noga Alon, Phillip B. Gibbons, Yossi Matias, and Mario Szegedy. "Tracking Join and Self-Join Sizes in Limited Storage". In: *Proceedings of the Eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. PODS '99. New York, NY, USA: Association for Computing Machinery, May 1, 1999, pp. 10–20. DOI: 10.1145/303976.303978.

[3] Noga Alon, Yossi Matias, and Mario Szegedy. "The Space Complexity of Approximating the Frequency Moments". In: *Journal of Computer and System Sciences* 58.1 (Feb. 1, 1999), pp. 137–147. ISSN: 0022-0000. DOI: 10.1006/jcss.1997.1545.

[4] *Apache Flink® — Stateful Computations over Data Streams*. URL: https://flink.apache.org/ (visited on Apr. 7, 2026).

[5] *Apache Hadoop*. URL: https://hadoop.apache.org/ (visited on Apr. 1, 2026).

[6] Apache Software Foundation. *Apache Airflow*. July 2021.

[7] Apache Software Foundation. *CountMinSketch (Spark 4.1.1 JavaDoc)*. Jan. 9, 2026. URL: https://spark.apache.org/docs/4.1.1/api/java/org/apache/spark/util/sketch/CountMinSketch.html (visited on Mar. 24, 2026).

[8] *Apache Spark™ - Unified Engine for Large-Scale Data Analytics*. URL: https://spark.apache.org/ (visited on Sept. 18, 2022).

[9] *Argo Workflows*. URL: https://argoproj.github.io/workflows/ (visited on Apr. 12, 2026).

[10] M. Ben-Ari. *Principles of Concurrent and Distributed Programming*. 2nd ed. Harlow, England; New York: Addison-Wesley, 2006. 361 pp. ISBN: 978-0-321-31283-9.

[11] Dvir Biton, Roy Friedman, and Rana Shahout. "Geometric Sketch: The Inflatable-Shrinkable Sketch". In: *Advanced Information Networking and Applications*. Cham: Springer Nature Switzerland, 2025, pp. 270–281. DOI: 10.1007/978-3-031-87766-7_24.

[12] Monica Chiosa, Thomas B. Preußer, and Gustavo Alonso. "SKT: A One-Pass Multi-Sketch Data Analytics Accelerator". In: *Proceedings of the VLDB Endowment* 14.11 (July 2021), pp. 2369–2382. ISSN: 2150-8097. DOI: 10.14778/3476249.3476287.

[13] Cilium Authors. *XDP Program Type — Cilium 1.20.0-Dev Documentation*. Apr. 12, 2026. URL: https://docs.cilium.io/en/latest/reference-guides/bpf/progtypes/#xdp (visited on Apr. 12, 2026).

[14] Graham Cormode. "Current Trends in Data Summaries". In: *ACM SIGMOD Record* 50.4 (Jan. 31, 2022), pp. 6–15. ISSN: 0163-5808. DOI: 10.1145/3516431.3516433.

[15] Graham Cormode. "Sketch Techniques for Approximate Query Processing". In: (2010). URL: https://dimacs.rutgers.edu/~graham/pubs/papers/sk.pdf.

[16] Graham Cormode and Minos Garofalakis. "Sketching Streams through the Net: Distributed Approximate Query Tracking". In: *Proceedings of the 31st International Conference on Very Large Data Bases*. VLDB '05. Trondheim, Norway: VLDB Endowment, Aug. 30, 2005, pp. 13–24. ISBN: 978-1-59593-154-2.

[17] Graham Cormode, Minos Garofalakis, Peter J. Haas, and Chris Jermaine. "Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches". In: *Foundations and Trends in Databases* 4.1–3 (2011), pp. 1–294. ISSN: 1931-7883, 1931-7891. DOI: 10.1561/1900000004.

[18] Graham Cormode and S. Muthukrishnan. "An Improved Data Stream Summary: The Count-Min Sketch and Its Applications". In: *Journal of Algorithms* 55.1 (Apr. 1, 2005), pp. 58–75. ISSN: 0196-6774. DOI: 10.1016/j.jalgor.2003.12.001.

[19] Graham Cormode and S. Muthukrishnan. "Summarizing and Mining Skewed Data Streams". In: *Proceedings of the 2005 SIAM International Conference on Data Mining*. Proceedings of the 2005 SIAM International Conference on Data Mining. Society for Industrial and Applied Mathematics, Apr. 21, 2005, pp. 44–55. DOI: 10.1137/1.9781611972757.5.

[20] Graham Cormode and Ke Yi. *Small Summaries for Big Data*. Cambridge: Cambridge University Press, 2020. DOI: 10.1017/9781108769938.

[21] *Data Stream Management: Processing High-Speed Data Streams*. Data-Centric Systems and Applications. Berlin, Heidelberg: Springer, 2016. DOI: 10.1007/978-3-540-28608-0.

[22] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: *6th Symposium on Operating Systems Design & Implementation (OSDI 04)*. San Francisco, CA: USENIX Association, Dec. 2004. URL: https://www.usenix.org/conference/osdi-04/mapreduce-simplified-data-processing-large-clusters.

[23] Differential Privacy Team, Apple. *Learning with Privacy at Scale*. Apple, 2017. URL: https://docs-assets.developer.apple.com/ml-research/papers/learning-with-privacy-at-scale.pdf (visited on Apr. 12, 2026).

[24] Cynthia Dwork, Maurice Herlihy, Serge Plotkin, and Orli Waarts. "Time-Lapse Snapshots". In: *SIAM Journal on Computing* 28.5 (Jan. 1999), pp. 1848–1874. ISSN: 0097-5397. DOI: 10.1137/S0097539793243685.

[25] *eBPF - Introduction, Tutorials & Community Resources*. URL: https://ebpf.io (visited on Apr. 12, 2026).

[26] *Facebook/Zstd*. URL: https://github.com/facebook/zstd (visited on May 12, 2022).

[27] Minos Garofalakis. "Distributed Data Streams". In: *Encyclopedia of Database Systems*. Boston, MA: Springer US, 2009, pp. 883–890. DOI: 10.1007/978-0-387-39940-9_137.

[28] Phillip B. Gibbons. "Big Data: Scale Down, Scale Up, Scale Out". Keynote. 29th IEEE International Parallel & Distributed Processing Symposium (IPDPS) (Hyderabad, India). May 28, 2015. DOI: 10.1109/IPDPS.2015.123.

[29] Liyuan Gu, Ye Tian, Wei Chen, Zhongxiang Wei, Cenman Wang, and Xinming Zhang. "Per-Flow Network Measurement With Distributed Sketch". In: *IEEE/ACM Transactions on Networking* 32.1 (Feb. 2024), pp. 411–426. ISSN: 1558-2566. DOI: 10.1109/TNET.2023.3286879.

[30] Thomas A. Henzinger, Christoph M. Kirsch, Hannes Payer, Ali Sezgin, and Ana Sokolova. "Quantitative Relaxation of Concurrent Data Structures". In: *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '13. New York, NY, USA: Association for Computing Machinery, Jan. 23, 2013, pp. 317–328. DOI: 10.1145/2429069.2429109.

[31] Maurice Herlihy, Nir Shavit, Victor Luchangco, and Michael Spear. *The Art of Multiprocessor Programming*. Second. Cambridge, MA, United States: Elsevier, Morgan Kaufmann Publishers, 2021. 553 pp. ISBN: 978-0-12-415950-1.

[32] Maurice P. Herlihy and Jeannette M. Wing. "Linearizability: A Correctness Condition for Concurrent Objects". In: *ACM Trans. Program. Lang. Syst.* 12.3 (July 1, 1990), pp. 463–492. ISSN: 0164-0925. DOI: 10.1145/78969.78972.

[33] Martin Hilgendorf and Marina Papatriantafilou. "LMQ-Sketch: Lagom Multi-Query Sketch for High-Rate Online Analytics". In: *39th International Symposium on Distributed Computing (DISC 2025)*. Vol. 356. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025, 36:1–36:24. DOI: 10.4230/LIPIcs.DISC.2025.36.

[34] Victor Jarlow, Charalampos Stylianopoulos, and Marina Papatriantafilou. "QPOPSS: Query and Parallelism Optimized Space-Saving for Finding Frequent Stream Elements". In: *Journal of Parallel and Distributed Computing* 204 (Oct. 1, 2025), p. 105134. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2025.105134.

[35] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web". In: *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing - STOC '97*. The Twenty-Ninth Annual ACM Symposium. El Paso, Texas, United States: ACM Press, 1997, pp. 654–663. DOI: 10.1145/258533.258660.

[36] Zohar Karnin, Kevin Lang, and Edo Liberty. "Optimal Quantile Approximation in Streams". In: *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. 2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS). New Brunswick, NJ, USA: IEEE, Oct. 2016, pp. 71–78. DOI: 10.1109/FOCS.2016.17.

[37] Martin Kleppmann. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. First edition. Sebastopol, CA: O'Reilley, 2017. 1 p. ISBN: 978-1-4493-7332-0.

[38] Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, and Yan Chen. "Sketch-Based Change Detection: Methods, Evaluation, and Applications". In: *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*. IMC '03. New York, NY, USA: Association for Computing Machinery, Oct. 27, 2003, pp. 234–247. DOI: 10.1145/948205.948236.

[39] Jurij Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets*. Third edition. New York, NY: Cambridge University Press, 2020. ISBN: 978-1-108-47634-8.

[40] Ron Liu. *FLIP-315 Support Operator Fusion Codegen for Flink SQL - Apache Flink - Apache Software Foundation*. Apache Flink. Jan. 3, 2024. URL: https://cwiki.apache.org/confluence/display/FLINK/FLIP-315+Support+Operator+Fusion+Codegen+for+Flink+SQL (visited on Apr. 7, 2026).

[41] Antonis Manousis, Zhuo Cheng, Ran Ben Basat, Zaoxing Liu, and Vyas Sekar. "Enabling Efficient and General Subpopulation Analytics in Multidimensional Data Streams". In: *Proceedings of the VLDB Endowment* 15.11 (July 2022), pp. 3249–3262. ISSN: 2150-8097. DOI: 10.14778/3551793.3551867.

[42] Gonçalo Matos, Salvatore Signorello, and Fernando M. V. Ramos. "Generic Change Detection (Almost Entirely) in the Dataplane". In: *Proceedings of the Symposium on Architectures for Networking and Communications Systems*. ANCS '21. New York, NY, USA: Association for Computing Machinery, Jan. 18, 2022, pp. 113–120. DOI: 10.1145/3493425.3502767.

[43] *Modern Data Orchestrator Platform | Dagster*. Dagster. URL: https://dagster.io/ (visited on Apr. 12, 2026).

[44] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. "A Modern Primer on Processing in Memory". In: *Emerging Computing: From Devices to Systems: Looking Beyond Moore and Von Neumann*. Singapore: Springer Nature, 2023, pp. 171–243. DOI: 10.1007/978-981-16-7487-7_7.

[45] Thomas Neumann. "Efficiently Compiling Efficient Query Plans for Modern Hardware". In: *Proc. VLDB Endow.* 4.9 (June 1, 2011), pp. 539–550. ISSN: 2150-8097. DOI: 10.14778/2002938.2002940.

[46] Vinh Quang Ngo and Marina Papatriantafilou. "Cuckoo Heavy Keeper and the Balancing Act of Maintaining Heavy Hitters in Stream Processing". In: *Proc. VLDB Endow.* 18.9 (May 1, 2025), pp. 3149–3161. ISSN: 2150-8097. DOI: 10.14778/3746405.3746434.

[47] Yiannis Nikolakopoulos, Anders Gidenstam, Marina Papatriantafilou, and Philippas Tsigas. "A Consistency Framework for Iteration Operations in Concurrent Data Structures". In: *2015 IEEE International Parallel and Distributed Processing Symposium*. 2015 IEEE International Parallel and Distributed Processing Symposium. May 2015, pp. 239–248. DOI: 10.1109/IPDPS.2015.84.

[48] Jonas Otten. *Raking the Floods: My Intern Project Using eBPF*. The Cloudflare Blog. Sept. 18, 2020. URL: https://blog.cloudflare.com/building-rakelimit/ (visited on Apr. 12, 2026).

[49] Wieger R. Punter, Odysseas Papapetrou, and Minos Garofalakis. "OmniSketch: Efficient Multi-Dimensional High-Velocity Stream Analytics with Arbitrary Predicates". In: *Proc. VLDB Endow.* 17.3 (Nov. 1, 2023), pp. 319–331. ISSN: 2150-8097. DOI: 10.14778/3632093.3632098.

[50] Redis. *Count-Min Sketch | Docs*. Mar. 12, 2026. URL: https://redis.io/docs/latest/develop/data-types/probabilistic/count-min-sketch/ (visited on Mar. 24, 2026).

[51] Arik Rinberg and Idit Keidar. "Intermediate Value Linearizability: A Quantitative Correctness Criterion". In: *Journal of the ACM* 70.2 (Apr. 18, 2023), 17:1–17:21. ISSN: 0004-5411. DOI: 10.1145/3584699.

[52] Arik Rinberg, Alexander Spiegelman, Edward Bortnikov, Eshcar Hillel, Idit Keidar, Lee Rhodes, and Hadar Serviansky. "Fast Concurrent Data Sketches". In: *ACM Transactions on Parallel Computing* 9.2 (Apr. 11, 2022), 6:1–6:35. ISSN: 2329-4949. DOI: 10.1145/3512758.

[53] Pratanu Roy, Arijit Khan, and Gustavo Alonso. "Augmented Sketch: Faster and More Accurate Stream Processing". In: *Proceedings of the 2016 International Conference on Management of Data*. SIGMOD '16. New York, NY, USA: Association for Computing Machinery, June 14, 2016, pp. 1449–1463. DOI: 10.1145/2882903.2882948.

[54] Michael L. Scott and Trevor Brown. *Shared-Memory Synchronization*. Synthesis Lectures on Computer Architecture. Cham: Springer International Publishing, 2024. DOI: 10.1007/978-3-031-38684-8.

[55] Charalampos Stylianopoulos, Ivan Walulya, Magnus Almgren, Olaf Landsiedel, and Marina Papatriantafilou. "Delegation Sketch: A Parallel Design with Support for Fast and Accurate Concurrent Operations". In: *Proceedings of the Fifteenth European Conference on Computer Systems*. EuroSys '20. New York, NY, USA: Association for Computing Machinery, Apr. 17, 2020, pp. 1–16. DOI: 10.1145/3342195.3387542.

[56] Volvo Autonomous Solutions. *Autonomous Hauling without a Safety Driver: Brønnøy Kalk Limestone Mine*. Aug. 31, 2023. URL: https://www.youtube.com/watch?v=VMhhkdbz9Lw (visited on Mar. 24, 2026).

[57] Volvo Autonomous Solutions. *Volvo Autonomous Solutions Removes Safety Driver at Brønnøy Kalk*. Aug. 30, 2023. URL: https://www.volvoautonomoussolutions.com/en-en/news-and-insights/press-releases/2023/aug/volvo-autonomous-solutions-removes-safety-driver-at-bronnoy-kalk.html (visited on Mar. 24, 2026).

[58] Theophilus Wellem and Yu-Kuen Lai. "An OpenCL Implementation of Sketch-Based Network Traffic Change Detection on GPU". In: *2012 Fifth International Symposium on Parallel Architectures, Algorithms and Programming*. 2012 Fifth International Symposium on Parallel Architectures, Algorithms and Programming. Dec. 2012, pp. 279–286. DOI: 10.1109/PAAP.2012.46.

[59] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. "Elastic Sketch: Adaptive and Fast Network-wide Measurements". In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM '18. New York, NY, USA: Association for Computing Machinery, Aug. 7, 2018, pp. 561–575. DOI: 10.1145/3230543.3230544.

[60] Fuheng Zhao, Dan Qiao, Rachel Redberg, Divyakant Agrawal, Amr El Abbadi, and Yu-Xiang Wang. "Differentially Private Linear Sketches: Efficient Implementations and Applications". In: *Proceedings of the 36th International Conference on Neural Information Processing Systems*. NIPS '22. Red Hook, NY, USA: Curran Associates Inc., Nov. 28, 2022, pp. 12691–12704. ISBN: 978-1-7138-7108-8.

[61] Xiaobo Zhu, Guangjun Wu, Hong Zhang, Shupeng Wang, and Bingnan Ma. "Dynamic Count-Min Sketch for Analytical Queries Over Continuous Data Streams". In: *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*. 2018 IEEE 25th International Conference on High Performance Computing (HiPC). Bengaluru, India: IEEE Computer Society, Dec. 2018, pp. 225–234. DOI: 10.1109/HiPC.2018.00033.