UPPSALA
UNIVERSITET

Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 2095

Scalable Data Management for Internet of Things

KHALID MAHMOOD

ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2021

ISSN 1651-6214
ISBN 978-91-513-1346-7
URN urn:nbn:se:uu:diva-458420

Dissertation presented at Uppsala University to be publicly examined in Room 2446, Polacksbacken, Lägerhyddsvägen 2, Uppsala, Friday, 14 January 2022 at 13:15 for the degree of Doctor of Philosophy. The examination will be conducted in English. Faculty examiner: Professor Vera Goebel (Department of Informatics, University of Oslo).

Abstract

Mahmood, K. 2021. Scalable Data Management for Internet of Things. *Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology* 2095. 44 pp. Uppsala: Acta Universitatis Upsaliensis. ISBN 978-91-513-1346-7.

Internet of Things (IoT) often involve considerable numbers of sensors that produce large volumes of data. In this context, efficient management of data could potentially enable automatic decision making based on analytics of sensors on equipment. However, these sensors are often geographically distributed and generate diverse formats of data in form of sensor streams at a high rate. The combination of these properties of IoT pose significant challenges for the existing database management systems (DBMSs) to provide scalable data storage and analytics.

The problem of providing efficient data management of distributed IoT applications using DBMS technologies is addressed in this thesis. Initially, we developed a prototype system, Fused LOg database Query Processor (FLOQ), which enables general query processingover collections of relational databases that are deployed locally on distributed sites to store sensor measurement logs. Although FLOQ provides efficient query execution when scaling the number of distributed databases, it exhibits complexity and scalability issues for large IoT applications having heterogeneous data. The limitations of FLOQ are primarily inherent to its use of relational database backends for storage of sensor logs.

When a relational database is used to store large-scale IoT data, it exhibits several challenges. The loading of massive logs produced at high rates is not fast enough due to its strong consistency mechanisms. Furthermore, it could demonstrate a single point of failure that limits the availability, and the inflexible schemas make it difficult to manage heterogeneity. In contrast to relational databases, distributed NoSQL data stores could provide scalable storage of heterogeneous data through data partitioning, replication, and high availability by sacrificing strong consistency. To understand the suitability of NoSQL databases, this thesis also investigates to what degree NoSQL DBMSs provide scalable storage and analytics of IoT applications by comparing a variety of state-of-the-art relational and NoSQL databases for real-world industrial IoT data.

The experimental evaluations reveal that the scalability can be provided by the distributed NoSQL data stores; however, the support of advanced data analytics is difficult due to their limited query processing capabilities. Furthermore, data management of distributed IoT applications often requires seamless integration between a real-time edge analytics platform, a distributed storage manager, effective data integration, and query processing techniques for handling heterogeneity. Therefore, in order to provide a holistic data management solution, this thesis developed the Extended Query Processing (EQP) system, which enables advanced analytics for supporting both edge and offline analytics for large-scale IoT applications.

These contributions enable efficient data management of large-scale heterogeneous IoT applications and supports advanced analytics.

Keywords: NoSQL, IoT, Smart Computing, MongoDB, IIoT, Data Streams, Edge Computing

Khalid Mahmood, Department of Information Technology, Division of Computing Science, Box 337, Uppsala University, SE-75105 Uppsala, Sweden.

ISSN 1651-6214
ISBN 978-91-513-1346-7
URN urn:nbn:se:uu:diva-458420 (http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-458420)

To my parents,

পারভীন জাহান, আজাদ সোবহান

List of papers

This thesis is based on the following papers, which are referred to in the text by their Roman numerals.

I Minpeng Zhu, Khalid Mahmood, Tore Risch, Scalable Queries Over Log Database Collections, 30th British International Conference on Databases (BICOD 15), Edinburgh, UK, July 6-8, 2015, *In Proc. BICOD 2015, pp. 173-185.*

II Khalid Mahmood, Thanh Truong, Tore Risch, NoSQL Approach to Large Scale Analysis of Persisted Streams, 30th British International Conference on Databases (BICOD 15), Edinburgh, UK, July 6-8, 2015, *In Proc. BICOD 2015, pp. 152-156.*

III Khalid Mahmood, Tore Risch, Minpeng Zhu, Utilizing a NoSQL Data Store for Scalable Log Analysis, 19th International Database Engineering & Applications Symposium (IDEAS 15), Yokohama, Japan, July 13 - 15, 2015, *In Proc. IDEAS 2015, pp. 49-55.*

IV Khalid Mahmood, Kjell Orsborn, Tore Risch, Comparison of NoSQL Datastores for Large Scale Data Stream Log Analytics, 2019 IEEE International Conference on Smart Computing (SMARTCOMP 19), Washington, DC, USA, June 12 - 15, 2019, *In Proc. SMARTCOMP 2019, pp. 478-480.*

V Khalid Mahmood, Kjell Orsborn, Tore Risch, Wrapping a NoSQL Datastore for Stream Analytics, 2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI), Las Vegas, NV, USA, August 11-13, 2020, *In Proc. IRI 2020, pp. 301-305.*

VI Khalid Mahmood, Kjell Orsborn, Tore Risch, Analytics of IIoT Data Using a NoSQL Datastore, 2021 IEEE International Conference on Smart Computing (SMARTCOMP 21), Irvine, California, USA, August 23-27, 2021, *In Proc. SMARTCOMP 2021, pp. 97-104.*

VII Khalid Mahmood, Tore Risch, Scalable Real-Time Analytics for IoT Applications, 2021 IEEE International Conference on Smart Computing (SMARTCOMP 21), Irvine, California, USA, August 23-27, 2021, *In Proc. SMARTCOMP 2021, pp. 404-406.*

Reprints were made with permission from the publishers.

Contents

1. Introduction

The fields of Industrial Internet [14], Cyber Physical Systems [27], and Internet of Things (IoT) [19], [50] have been proliferated with the growing number of devices distributed over the internet. To enable automatic decision-making from industrial processes, it is often necessary to manage and analyze data generated at high rates from these devices. In this context, providing efficient processing and analysis of massive volumes of data over numerous distributed data sources become a major challenge.

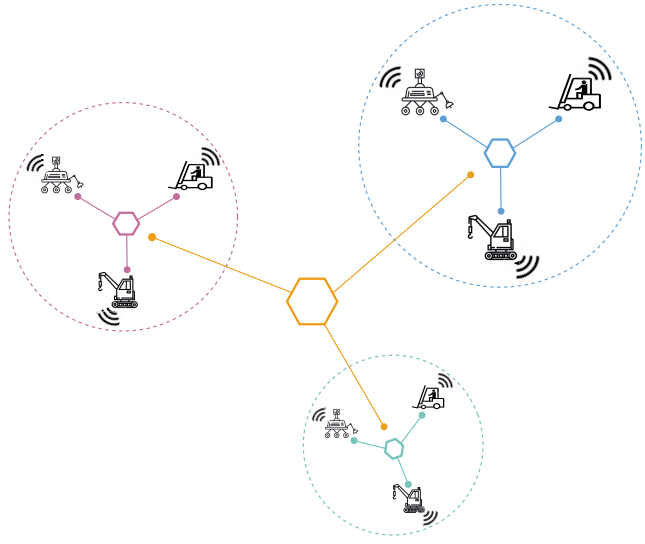

Figure 1.1. Distributed IoT application.

An example of large-scale data analysis can be drawn from the field of Internet of Things (IoT) [31], [33], [52]. IoT applications as depicted in Fig. 1.1 often consist of distributed *sites*, where clusters of machines equipped with sensors are deployed. Data is continuously received from these sensors, thus forming streams of values. The streams are generated at high rates often producing large volumes of data. Furthermore, the data streams from different types of sensors are heterogeneous by nature, i.e., represented using different data formats. Another feature of these types of applications is that the sensors can be frequently added or removed from the sites. The combination of these properties of a distributed IoT application poses significant challenges for the existing data management and analysis techniques.

Data analysis typically involves running programs written by developers in e.g., Python, R, or Java to perform computational tasks to analyze data. For scalable analyses, a distributed programming paradigm such as map-reduce [10] is often used. In contrast, *data analytics* provides high-level tools to enable analysts to perform interactive data analysis. For example, Database Management Systems (DBMSs) provide tools for scalable storage and analyses of data, with several advantages over conventional data analysis [39]. It allows structuring data through schemas, analytics through ad hoc queries, and scalable search by utilizing indexes. However, enabling data analytics over distributed IoT applications requires several DBMS techniques to be integrated seamlessly.

When performing analytics locally at each site, the stream generation rate from the sensors can be very high; therefore, adopting disk-based DBMS techniques is not feasible due to latency caused by slow write operations from high incoming data rates. An in-memory Data Stream Management System (DSMS) [17] is therefore more feasible in this context for supporting real-time data analytics [32]. Unlike DBMSs, which execute queries over data stored in tables, a DSMS provides online analytics over continuously produced streams of elements. Typically, a DSMS is implemented with in-memory techniques, which eliminates the latency of disk-based storage. However, summarized and/or filtered data streams still need to be stored in a persistent DBMS for further historical analytics. Two approaches could potentially be adopted in this scenario: storing data into a local DBMS deployed at each site or uploading it into a distributed DBMS from the local devices. Both of these approaches exhibit several trade-offs:

- When a relational-DBMS (RDBMS) is deployed locally at each site, data-integration techniques, e.g., *view-based data integration* [21], [26], can be applied to facilitate global analytics over unified views of local RDBMS tables. However, such an approach often leads to various problems:
  - Large IoT applications typically consist of diverse sensors. These sensors often require frequent addition and removal from the application, which can lead to significant modification of local relational schemas. As a result, the global schema also requires adaptation; therefore, static DBMS data-integration techniques caused problems with complexity and scalability for storing heterogenous sensor data.
  - RDBMS having strong consistency model slow down the injection of a large volume of streams.
  - A single point of failure may also occur as the data is stored locally at each site and access through a central database.

  However, utilizing the RDBMS exhibits several advantages as well:

  - An RDBMS has an advanced query language that allows complex tasks to be expressed easily, making it suitable to perform a variety of IoT data analytics even by novice programmers.
  - RDBMSs are equipped with various indexing techniques that can be leveraged by sophisticated query optimizers, which enable efficient and scalable query execution.
- Rather than having separate RDBMSs deployed at each site, a distributed DBMS can be used to achieve fault tolerance using data replication. However, several issues need to be considered for utilizing distributed DBMSs:
  - A distributed database that favors a strong consistency model, often disrupts the availability and performance of the entire IoT application. To provide availability and scalability, a type of distributed database called a *NoSQL* data store [4], that sacrifices some database-wide strong consistency, is more suitable. A particular category of NoSQL data stores, called *document stores*, are capable of persisting objects having dynamic key-value associations (e.g., JSON objects [40]). Such a distributed database system is well suited for injecting heterogeneous sensor data where different kinds of data having various formats need to be added or removed.
  - NoSQL data stores typically have simple query functionality, usually restricted to put, get, and update, which limits advanced analytics. Therefore, among numerous NoSQL data stores [4], the choice of an appropriate kind is critical for providing scalable data management for IoT applications.

The problems of providing scalable data analytics for large-scale sensor logs from distributed IoT applications using persistent DBMSs is addressed in this thesis. The work is comprised of seven papers that utilize sensor logs from real-world IoT applications [13], [31], [33], [52] to investigate how different systems perform under realistic IoT workloads. The following research questions are investigated in the thesis:

1. The overall research question is how to provide scalable storage and analytics of sensor logs from distributed heterogeneous IoT applications.
2. How can relational DBMSs be utilized to provide IoT data analytics?
3. Compared to relational DBMSs, to what degree are distributed NoSQL data stores able to provide high-performance data persistence and advanced query processing by leveraging indexing techniques?
4. Among different kinds of distributed NoSQL data stores [4], what are the trade-offs in choosing different data partitioning and indexing techniques for scalable query execution of IoT data streams?
5. How would data-management solutions utilizing relational DBMSs or NoSQL data stores be designed to combine high volume heterogeneous data injection and advanced query processing for distributed IoT applications?

Table 1.1. *Relationship between the research questions (1-5) and the papers (I-VII).*

	1	2	3	4	5
I	✓	✓			✓
II	✓		✓	✓	✓
III	✓		✓		
IV	✓			✓	
V	✓				✓
VI	✓	✓	✓		✓
VII	✓				✓

The above research questions are addressed in *Paper I-VII*. Table 1.1 shows the relationships between each research question and the corresponding papers. The contributions of the papers are briefly summarized below, whereas the detailed technical contributions are provided in Chapter 3.

In *Paper I*, we developed our first prototype system, *Fused LOg database Query processor (FLOQ)*, to facilitate data analytics for distributed IoT applications. FLOQ utilizes relational *log databases* at each site to store locally generated sensor measurements. It also maintains a global *meta-database*, describing the properties of all the sensors operating in the entire federation of sites. FLOQ resembles the classic *view-based data integration* approach [21], where a global relational view is maintained over all the local log databases. Utilizing this view, FLOQ enables global queries to analyze the measurement data from the sensors stored in the local log databases. A particular challenge in this scenario is to perform scalable joins between the meta-database and the log databases. We proposed two new distributed joining strategies, *Parallel Bind Join (PBJ)* and *Parallel Bulk-Load Join (PBLJ)* to perform parallel query processing over autonomous log databases by utilizing standard DBMS APIs. FLOQ was evaluated with real-world IoT data from Bosch Rexroth - Hägglund [13]. Paper I provides answers to research questions **one**, **two**, and **five**.

Future IoT applications tend to grow rapidly with the introduction of many heterogeneous sensors that are dynamically added or removed from the federation. A data-management approach using a rigid relational schema in FLOQ will lead to scalability issues as it requires frequent schema updates. Furthermore, the insertion of heterogeneous sensor measurements into a relational table with predefined columns is not flexible (Section 2.1.1). By contrast, distributed NoSQL data stores that support inserting objects having dynamic associations of key-value pair objects (e.g., JSON) are more suitable for handling the heterogeneity. However, it was unknown how this approach provides scalable storage and query processing for large-scale data analytics. To under-

stand the suitability of NoSQL databases for large IoT applications, *Paper II* provides the scope to formulate research questions **one**, **three**, **four**, and **five**.

The proposals from Paper II serve as a basis for *Paper III* for investigating to what degree the NoSQL data store MongoDB [36] can achieve scalable storage and query processing for large-scale sensor logs. In the paper, MongoDB is compared with a relational DBMS from a major commercial vendor and with a popular open-source relational DBMS. MongoDB allows sacrificing transactional consistency in order to achieve high-performance inserts and updates. Furthermore, it provides both primary and secondary indexing, which is required for scalable query execution and analytics. Our study shows that MongoDB is a viable alternative compared to relational databases for large-scale IoT data analytics. The results of this paper provide an answer to questions **one** and **three**.

The performance of MongoDB motivated us to work on *Paper IV* to explore the suitability of other kinds of distributed NoSQL data stores for large IoT applications. In the paper, we compare three principal categories of state-of-the-art distributed NoSQL data stores [4]. We choose Cassandra [1] from the column store family, MongoDB [36] as a document store, and Redis [41] as a distributed main memory key-value store. This work enabled us to study the trade-offs between different types of storage and data-partitioning mechanisms used in these new kinds of distributed databases. Our results show that the in-memory data store Redis is a good choice over other NoSQL databases when there is sufficient main-memory. This is expected as in-memory techniques perform better compared to their disk-based counterparts. However, most IoT applications produce large volumes of data that do not fit into the main-memory; therefore, disk-based NoSQL data stores are often required. Furthermore, when the persistent NoSQL data stores Cassandra and MongoDB are used, we observe that if the analytics uses queries involving inequalities, the range-based data partitioning of MongoDB facilitated by B-tree indexing scales better than Cassandra's distributed hashing techniques. Paper IV provides the answers to research questions **one**, and **four** and motivated us to develop a data management solution for IoT by utilizing MongoDB as back-end storage and analytics platform.

A conclusion from Paper III and Paper IV is that, while scalability can be provided by distributed NoSQL data stores such as MongoDB, advanced data analytics is restricted in these types of systems due to its limited query processing capabilities. In *Paper V*, we discussed how a prototype system combining distributed scalability of MongoDB with a relationally complete [7] query processor can provide advanced analytics. We adopted the wrapper-mediator approach [44] by integrating MongoDB with the in-memory mediator database Amos II [43], [42] that offers advanced query processing compared to the more limited query processing in MongoDB. The preliminary testing by injecting large volume sensor streams shows that our prototype system provides improved performance compared to state-of-the-art RDBMSs, even though it

has some overhead caused by local query processing performed in the wrapper outside of DBMS. Paper V provides answers to research questions **one** and **five**. The wrapper developed in Paper V is enhanced in *Paper VI*, including further implementations and empirical evaluations.

Data-management solutions for large distributed IoT applications require seamless integration between a real-time data analytics platform, a distributed data manager, and effective data integration techniques for handling heterogeneity. *Paper VI* proposes a system architecture by comparing two different data management solutions for IoT applications using either an RDBMS or MongoDB as backend. In this work, we compare both approaches and discuss how heterogeneity can be handled better using the schema-less data model provided by MongoDB. By adopting de-normalization [45], MongoDB enables combining both meta-data and sensor data having diverse measurements into a single JSON object. Rather than applying static view-based data integration techniques over local relational log databases as in Paper I, this approach eliminates expensive join operations between a global meta-database and local log databases. The wrapper presented in Paper V is completed in Paper VI including the *Extended Query Processing (EQP)* system, which provides relationally complete query processing and numerical operators over MongoDB. Hence, the prototype system leverages the distributed scalability of MongoDB for backend storage of heterogonous sensor data, while providing advanced analytics through the complex query processing capabilities of EQP. Paper VI provides the answers to research questions **one**, **two**, **three**, and **five**.

Finally, *Paper VII* is a demonstration paper that utilizes the NoSQL approach for IoT data-management presented in Paper VI. In the paper, we integrate the data stream management system, *sa.engine* [47] with EQP to perform real-time audio anomaly detection on edge devices for IoT applications. Paper VII provides answers to research questions **one** and **five**.

The next chapter provides an overview of the technologies used in this thesis. Chapter 3 provides elaborated discussions and technical contributions together with stating my own contributions in each of Papers I-VII. Finally, Chapter 4 provides directions for future work.

2. Background

This chapter describes the general technologies and background related to this thesis. At first, the relational database concept is discussed, with a focus on how it can be utilized to store heterogeneous data and provide large-scale data analytics. Then follows an overview and comparison of different categories of distributed NoSQL data stores. Finally, the main-memory mediator database system, Amos II is discussed by highlighting its query processing strategies, which are utilized in this thesis for system development.

2.1 Relational Databases

In a relational database [6], data is typically categorized and stored in a collection of *tables*. These tables are logically related to each other; hence, the name *relational database*. Each table consists of a set of attributes called *columns*. The instances of the data in a table are stored as *rows* and the *values* are organized within the columns for each row. Typically, a user of relational databases accesses and modifies the data without altering the table structure (i.e., the database schema) as table modification is usually an expensive operation.

2.1.1 Relational Databases for Storing Heterogeneous Data

Relational databases work best with structured data having predefined attributes of fixed lengths, represented as columns of a table. For example, customer data having similar attributes such as names, phone, email, etc., fit well with the relational database model, e.g., having a customer table with a fixed number of columns. However, a relational database has complexity and performance issues when storing heterogeneous data. Consider the example in Fig. 2.1, for storing heterogeneous sensor data into the `measurements` table in a relational database.

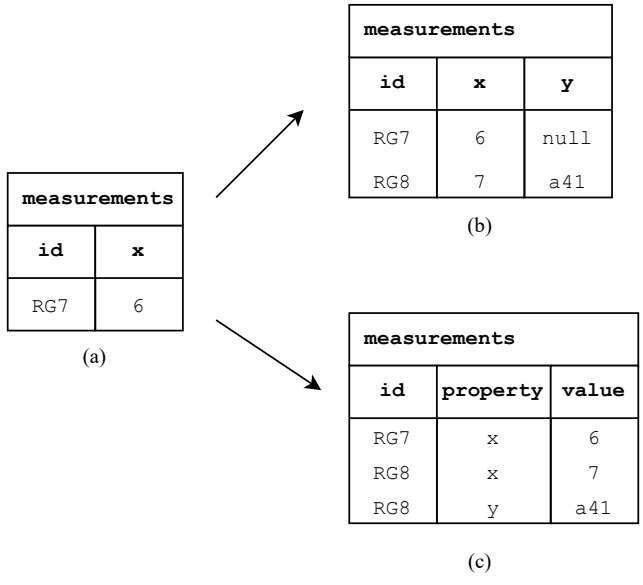

Figure 2.1. Heterogeneous data insertion in a relational table.

Initially, we start inserting the data for a particular sensor with id `RG7`, which has a measured value `x` as `6`. Therefore, we can create a `measurements` table as depicted in Fig. 2.1a having two columns representing the two attributes, `id`, and `x`. However, in typical IoT applications, new types of sensors often have to be added to the systems quite rapidly. In this context, a particular problem arises if we try to dynamically add new kinds of sensors consisting of different properties. For example, instead of having the single property `x` of sensor `RG7`, if a new sensor `RG8` has another property `y`, we are forced to modify the schema for the `measurements` table. To resolve this issue, one option, as depicted in Fig. 2.1b, is to append one more column for attribute `y` in the `measurements` table. However, schema modifications by adding a new column for each new type of sensor with more attributes than the previous ones, are expensive, hence, not scalable. Furthermore, in case of different sensors having different attributes, this solution is not space-efficient, as null values must be be assigned for missing attributes (e.g., in Fig. 2.1b, for sensor `RG7`, a `null` value for property `y` is assigned). Furthermore, querying over tables with many attributes and many `null` values becomes more complicated.

An alternative approach shown in Fig. 2.1c is to use a modified `measurements` table having three fixed columns, `id`, `property`, and `value`. Here, for each measured value in the sensor data, we assign `property` and `value` columns. Therefore, for sensor `RG7`, we need to insert one row: `property x` and its `value 6`, while sensor `RG8` need to insert two rows: `property x` having `value 7` and `property y` with `value a41` as a *character string*.

Although this approach offers tentative storage for heterogeneous data, query execution is highly inefficient. For example, in the column `value`, we have mixed data types of integers and character strings, hence, we needed to define the data type of the column as a character string. Therefore, any range query will be slow as it must perform a full-table scan because it is not possible to utilize an index over mixed data type of integer and character strings. A database with native support for heterogeneous data is a simpler and more scalable solution to this problem.

2.1.2 Relational Database for Large-Scale Data Analytics

Traditionally, relational databases are designed to provide analytics and transaction processing for business applications. Analytics is usually performed by dynamic query execution, leveraging the indexing to speed up query processing. In addition, transaction processing ensures the correctness of updates and queries; however, it is a resource and computation-intensive task. A substantial time for transaction processing is spent in logging, latching, locking, B-tree, and buffer management operations, which may cause 20 times performance overhead in a relational DBMS [22]. In recent years, new applications have proliferated that do not require full transaction support. Hence, a variety of high-performance databases have been designed to drive the analytics for large-scale data; for example, in-memory, transaction-less, NoSQL, and single-threaded databases have been developed by eliminating some of the overheads of the transaction processing systems.

To support large-scale data analytics, a horizontal scaling by distributing the data over many commodity servers may be preferable. However, performing transactions in a distributed environment is difficult; hence, relational databases often prefer a single computing server. When the data and transaction volume increases, a relational DBMS prefers vertical scaling-up by adding more computing resources, such as disk space, memory, and CPUs to the server. A high-performance single server is usually significantly more expensive than commodity servers, while prone to single-point-of-failure. Therefore, if full transactional consistency is not required, a distributed database with horizontal scaling of data to support large-scale analytics is preferable.

To provide data analytics for IoT applications, this thesis utilizes relational databases in *Paper I,II,III,V*, and *VI*.

2.2 NoSQL Data Stores

While a relational database provides a structured data model, dynamic query execution of SQL queries, and strong transactional consistency; a NoSQL data store sacrifices some of these attributes. Hence, the definition of NoSQL refers

to “Not Only SQL” or “Not Relational” [4]. Rather than the complex transactional processing capabilities of relational databases, NoSQL databases typically provide only simple online transaction processing (OLTP), such as insert, update, and look-up for a single record, in order to achieve higher availability and scalability. Therefore, performing advanced analytics is usually limited in NoSQL data stores due to the lack of a complete query language. The design principle of these systems often adheres to the Eric Brewer's CAP theorem [16] [3], which states that a distributed system can only have two out of three properties: consistency, availability, and partition-tolerance. Therefore, a NoSQL database is often designed to choose either availability or consistency, since network partitions in a distributed environment are unavoidable. A detail overview of NoSQL data stores can be found in Rick Cattell's paper [4].

Implementations of contemporary NoSQL data stores are inspired by the three early systems: Memcached [9], Google's BigTable [5], and Amazon's DynamoDB [11]. Memcached utilized distributed hashing to scale data inside the main memory of multiple nodes, while BigTable demonstrated that a persistent sparse table can be scaled up to thousands of commodity servers. DynamoDB pioneered the idea of *eventual consistency*, where the updates are not atomic compared to relational databases, but propagate to the distributed nodes eventually. Hence, DynamoDB achieves higher availability and scalability by sacrificing strong consistency.

2.2.1 Categories of NoSQL Data Stores

A consistent and systematic way to categorize NoSQL data stores is based on their data models [4]. The three main categories of NoSQL data stores [4] based on data models are key-value, document, and column stores, which are utilized in this thesis:

- **Key-value Stores**: These systems model data based on key-value pairs and utilize distributed hashing over the keys for finding the appropriate node to store its values. Pioneered by Memcached, these data stores can be an in-memory system like Redis [41] or use persistent disk storage similar to Project Voldemort [28]. Distributed hashing techniques, such as *consistent hashing* [25] are often adopted in these types of systems. In a distributed computing environment having m servers, consistent hashing partitions n keys and maps each of them to m servers based on a hash function. When a single server shutdown occurs, consistent hashing offers on average a minimal number of n/m keys remapping; while in most of the *modulo-based hashing* techniques, nearly all keys needed to be remapped. Hence, consistent hashing is well suited for fault tolerance and load-balancing of large-scale data in distributed commodity servers.

- **Document Stores**: A document store allows values to be vectors, lists, and even nested documents (e.g., JSON), where the attribute names are dynamically defined for each document at runtime. While relational databases use fixed-length tuples defined in a global schema, the attributes of a document are not pre-defined, and varying data types of values are permitted. Hence, a document store is well suited for heterogeneous data storage. A document store can utilize consistent hashing techniques or *range-based partitioning* [38] for data distribution. If consistent hashing techniques are used by the system, it provides efficient load-balancing of data. In contrast, range-based partitioning techniques cluster data based on keys with adjacent values, which speeds up range search. Unlike key-value stores, some document stores, such as MongoDB [36] offer B-tree indexing and dynamic query execution, which is used in this thesis to provide scalable analytics over heterogeneous data.
- **Column Stores**: These data stores are often referred to as *wide-column stores* or *extensible record stores*, and offer flexibility in between *tuples* and *documents*. These data stores can inject data having a very large number of dynamic columns; however, unlike relational databases, new columns with various formats can be added without schema modification. In contrast to document stores, a column store cannot support nested records. It is also expensive to assemble objects from many separately stored columns; hence, it is less flexible. A column store can be seen as a two-dimensional key-value store, which performs data-partition based on both row keys and columns. Cassandra [1] is an example of a column store, which was initially open-sourced by Facebook [4].

Utilizing NoSQL data stores for IoT data management is the primary focus of this thesis, which is discussed in the *Paper II-VII*.

2.3 Overview of Amos II

Amos II [43] [42] is an extensible mediator database system, which allows different kinds of data sources to be integrated and queried. It provides a functional and object-oriented query language, *AmosQL*. In Amos II, objects are classified by *types* and stored inside in its main-memory object-store. *Functions* are used to define the properties of objects and the relationships between different objects. Typically, relational databases use a tuple-calculus based query language, such as SQL, where variables are bounds to tuples (i.e., rows of tables). In contrast to tuple-calculus, AmosQL is a domain calculus query language, where queries are defined over instances of typed *objects* that are often used as function arguments. Amos II also offers an extensible query processor, which supports implementing interfaces as *foreign functions* written in regular programming languages (e.g., C, Lisp, Java, or Python). These for-

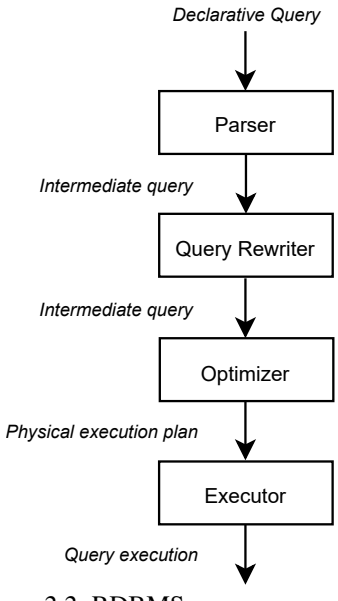

Figure 2.2. RDBMS query processing

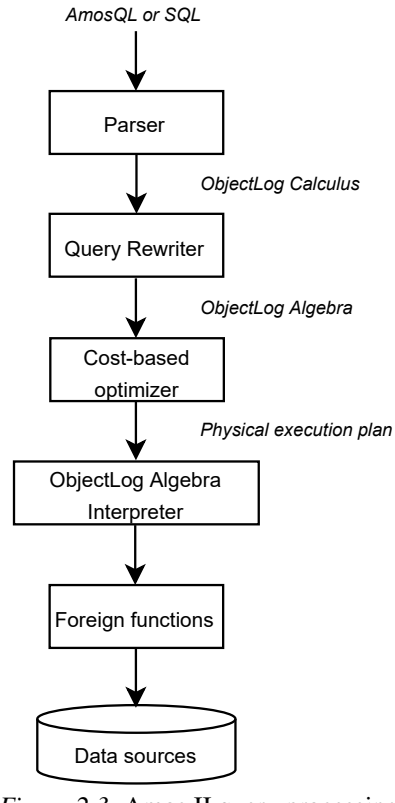

Figure 2.3. Amos II query processing

eign functions can also access different kinds of external data sources, which facilitates developing various *wrappers* to enable query processing over external data processing systems (e.g., RDBMSs, NoSQL data stores, or cloud infrastructures). These extensibility feature of Amos II are utilized in this thesis to facilitate domain-specific query processing over data sources in the prototype systems *Fused LOg database Query processor (FLOQ)* for accessing relational databases and in *Extended Query Processing (EQP)* system for accessing MongoDB.

An overview of typical query processing strategies in an RDBMS can be found in the paper by Hellerstein et al. [23]. Such query processing systems mainly consist of four modules: *parser*, *query rewriter*, *optimizer*, and *executor*, as depicted in Fig. 2.2. Query processing in Amos II [29] [15], as shown in Fig. 2.3, adheres to query processing steps similar to RDBMSs, with some nuances. Both query processing strategies are compared below to highlight how the prototype systems FLOQ and EQP leverage query processing techniques of Amos II:

- **Parser**: A relational database *parser*, depicted in Fig. 2.2, takes a *declarative query*, such as an SQL query; parses and validates it to

convert it into an intermediate query representation, often equivalent to *tuple relational calculus* [12, pp. 268]. In contrast, Amos II provides parsers for both *SQL* and *AmosQL* and converts the queries to a *domain relational calculus* language called *ObjectLog* [29], which is an object-oriented dialect of Datalog. The queries to both FLOQ and EQP are expressed as AmosQL.

- **Query rewriter**: The function of the query *rewriter* of Amos II is to transform the *ObjectLog* query representation for simplification and optimization without modifying its semantics nor accessing the actual storage. In Amos II, transformations are first made on the domain calculus before translating to an algebraic representation [15]. One important task of both relational and Amos II rewriters, is view expansion, which is usually performed by replacing the view with actual tables and predicates that reference the view. The rewriters often perform some query simplifications by arithmetic/logical expression evaluations, redundant join evaluation, and flattening of nested queries to improve the performance of the query *optimizer*. In our prototype system FLOQ, the query rewrite strategies are applied to decompose the query into sub-queries and bind to a particular log-database for facilitating the parallel join (i.e., PBJ or PBLJ [52]). In EQP, query rewrite rules are applied for accessing our external data source, MongoDB.
- **Optimizer**: Given the internal representation of a query, a query *optimizer* typically produces an efficient plan for executing the query. The approach is first described by Selinger, et al. on the development of the System R optimizer [46], which inspired the development of most query optimizers found in contemporary databases. Finding an optimal plan often involves exploring all possible join orders, which is computationally expensive (equivalent to the order of Catalan number [49] and its lower bound complexity is $\Omega(2^n)$ [8, pp. 327]). Instead, the System R relational optimizer employs a faster optimization strategy by exploring only *left-deep* query plans [23], where the right-hand side of each join input must be a single table. Furthermore, the optimizer prunes the search space based on estimated costs that are typically dominated by disk accesses. Hence, the trade-off is between choosing a near-optimal query plan and slower optimization time. In contrast to System R, Amos II is a main-memory database; therefore, the exponential query complexity cost of the System R optimizer is not justified as the number of disk access is not a dominating factor. Hence, the query optimizer in our prototype system EQP explores the search space in quadratic time, using greedy heuristics to find the cheapest reordering based on [29]. After query optimization is performed within EQP, further optimization is performed by the MongoDB database system. However, unlike state-of-the-art commercial databases, MongoDB (up until version 4.0), does not model the cost based on estimates of table statistics [37], which can

potentially lead to inefficient query plans. As shown in Fig. 2.2 and Fig. 2.3, the output of the query optimizers is a *physical execution plan*.

- **Executor**: A query *executor* interprets the physical execution plan and invokes procedures to access physical tables. In the case of our prototype systems, FLOQ and EQP, the query *optimizer* of Amos II generates a physical execution plan as an *ObjectLog algebra* [15], which is interpreted by the query *executor* to invoke *foreign functions* for accessing the external *data sources*, RDBMS and MongoDB. For developing EQP, a substantial amount of the implementation of the query processing system consisted of developing foreign functions in C for the MongoDB external data source in order to provide data manipulation, query execution, and other database control operations (e.g., index creations, exploring query execution plan and statistics).

The next chapter provides discussions and technical contributions of the Papers I-VII.

3. Contributions and Discussions

The discussion and technical contributions of this Thesis are summarized below for each paper to guide the reader on how papers relate to the research questions provided in Chapter 1.

3.1 Paper I

To provide data analytics for IoT applications, we first developed a prototype system, *Fused LOg database Query processor (FLOQ)*, which is presented in *Paper I*. The architecture of FLOQ is illustrated in Fig. 3.1.

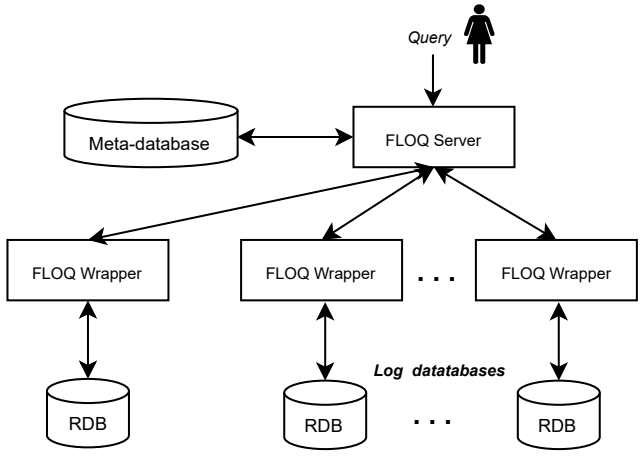

Figure 3.1. FLOQ system architecture.

In this paper, we consider a real-world IoT application scenario, where machines such as trucks, pumps, kilns, etc. are widely distributed at different geographic locations (aka sites). The sensors on the machines produce large volumes of log streams stored locally in autonomous relational *log databases* (depicted as *RDB* in Fig. 3.1). A global *meta-database* is utilized to describe the properties of machines, sensors, measurements, etc. FLOQ adopts the *local-as-a-view* approach [21] [26], which provides a global view by maintaining logical *union-all* of the local measurement tables stored in each RDB. FLOQ provides general query processing on the global view over all the autonomous RDBs integrated through the meta-database.

A user of FLOQ typically issues a *query* to the *FLOQ server*, which processes the query by first looking up the *meta-database* to find the identifiers required for querying RDBs containing the desired data. The FLOQ server then in parallel sends distributed queries to the RDBs, encapsulated by *FLOQ wrappers*. Finally, the FLOQ server collects and merges the distributed query results to obtain the final result.

A particular challenge in this scenario is a scalable way to process queries that join meta-data with data selected from the collection of autonomous RDBs using standard DBMS APIs. To speed up queries combining meta-data with distributed logged sensor readings, sub-queries to the log databases need to be run in parallel. This paper proposes two strategies to perform such joins, namely *Parallel Bind-Join (PBJ)* and *Parallel Bulk-Load Join (PBLJ)*, which are implemented in FLOQ. With both PBJ and PBLJ, streams of selected meta-data variable bindings are distributed to the wrapped log databases and processed there in parallel. After the parallel processing, the result streams are merged asynchronously by FLOQ. PBJ generalizes the *bind-join (BJ)* [20] operator, which is a state-of-the-art algorithm for joining data from an autonomous external data source with a central database. With PBJ the streams of bindings selected from the meta-database are bind-joined in the distributed wrappers with their encapsulated log databases. One problem with bind-join in our scenario is that large numbers of sub-queries will be sent to the log databases for execution; one for each parameter combination selected from the meta-database, which is particularly slow. To overcome this problem, PBLJ was developed, which first bulk-loads the selected bindings in parallel into a binding table in each log database, then performs a regular join between the bulk-loaded bindings and the local measurements. This approach eliminates executing a large number of sub-queries on log-databases and performs substantially better compared to bind-join and PBJ for scaling a large number of bindings.

To investigate the performance of PBJ and PBLJ, we developed a cost model to evaluate the efficiency of each strategy. The cost model provides the hypothesis that the PBLJ performs better compared to PBJ when scaling the number of (i) log databases (ii) return tuples, and (iii) parameter bindings from the meta-database. The conclusions from the cost model serve as a basis to develop experimental evaluations to prove the hypothesis empirically.

The *local-as-a-view* data integration approach adopted by FLOQ has a particular advantage over other view-based data integration techniques (e.g., global-as-a-view [26]), as it allows the addition of new distributed sites having autonomous databases. Since the global view is derived from the local measurement tables stored in each local database, any addition of new sites scales well in the local-as-a-view approach. We confirm this by providing experimental results showing that FLOQ scales well when new sites are added to the system. However, a major challenge arises when an IoT application requires frequent addition of heterogeneous sensors that generate data streams of diverse for-

mats. Each addition of a different type of sensor needs altering the schema of the RDB table or denormalizing the sensor data for inserting into a single RDB table, which is less intuitive and inefficient (Section 2.1.1). Furthermore, each alteration of the RDB schema requires modification of the meta-database schema. Hence, adopting any view-based data integration approach using relational databases results in complexity for IoT applications having diverse sensors. Therefore, to provide heterogeneous data integration for IoT applications, our future research was directed towards adopting scalable NoSQL data stores that allow storing dynamic key-value association objects such as JSON.

This paper provides answers to research questions **one**, **two**, and **five**.

I am a co-author of this paper. I developed the cost model that guides the experimental evaluation. I participated in the experimental evaluations and the implementation discussions. I also contributed in writing a significant part of the manuscript.

3.2 Paper II

Providing scalable persistence and analytics of large-scale streaming logs are central for data management of numerous emerging fields, such as IoT, Cyber Physical Systems, and Industrial Internet. The primary problem for persisting large volumes of streaming logs with conventional relational databases is that loading a large volume of data logs produced at high rates is not fast enough due to the strong consistency model and high cost of indexing. As a possible alternative, state-of-the-art NoSQL data stores that sacrifice transactional consistency to achieve higher performance and scalability can be utilized. This paper highlights six major data management challenges for providing scalable analytics for large-scale streaming logs with RDBMS and NoSQL databases:

1. *Archiving large volume of streaming logs*: In relational DBMSs, the high cost of maintaining the indexes and full transactional consistency can degrade the bulk-loading of a large volume of data logs. Is it feasible to leverage the weak consistency level of a NoSQL or relational database to boost the performance?
2. *Indexing strategies*: Unlike relational databases, most NoSQL data stores do not provide both primary and secondary indexing, which is required for scalable processing of analytics queries over data logs.
3. *Index utilization*: Providing high-performance and scalable analytics for a large volume of stream logs often requires a substantial amount of storage for indexing. Therefore, providing memory and disk efficient indexing strategies are needed for both RDBMS and NoSQL databases.
4. *Query processing*: Unlike relational databases, most NoSQL data stores do not provide a query optimizer. Although NoSQL data stores, e.g., MongoDB, provide a query language; the sophistication of the query optimizer needs to be investigated for scalable analytics.

5. *Advanced analytics*: Relational DBMS features for advanced analytics, such as joins or numerical expressions are limited in NoSQL data stores. Therefore, it requires to be investigated how advanced numerical analytics over large-scale data logs could be provided by NoSQL data stores.
6. *Parallelization of data*: NoSQL data stores have the ability to distribute data over many machines, which can provide parallel query execution. Therefore, the performance of data partitioning of distributed NoSQL data stores needs to be investigated for query execution over distributed data logs.

The above challenges from *Paper II*, provide the scope to formulate research questions **one**, **three**, **four**, and **five**. Based on these challenges, two research proposals were suggested, namely (i) developing *stream log analytics benchmarks* to understand the suitability of RDBMs and NoSQL data stores, and (ii) enabling *advanced analytics over NoSQL* data stores.

We developed a stream analytics benchmark in *Paper III* and *Paper IV*. *Paper III* compares the performance of two state-of-the-art RDBMSs with a NoSQL data store MongoDB, while *Paper IV* compares the performance of three different NoSQL data stores.

From the benchmarks, the performance and suitability of MongoDB for IoT workloads motivated us to develop a query processing system over MongoDB to support advanced IoT data analytics, which is presented in *Paper V-VII*.

I am the primary author of this paper. Tore Risch contributed to the discussions and the revisions of the manuscript. Thanh Truong participated in the revisions of the manuscript.

3.3 Paper III

A potential problem for persisting large volumes of data logs with a conventional relational database is that the loading of massive logs produced at high rates is not fast enough due to the strong consistency model and high cost of indexing. Unlike relational DBMSs, some NoSQL databases can sacrifice strong consistency by providing so-called eventual consistency compared with the ACID transactions of regular DBMSs for providing high-performance updates. Typically, NoSQL databases are designed to perform simple tasks with high scalability. Therefore, distributed NoSQL databases can be utilized for large-scale historical analysis of log data or numerical log analytics where full transactional consistency conforming ACID compliance is not required.

In this paper, we investigate to what degree a NoSQL database can achieve high-performance persisting and fundamental analyses of large-scale data logs from a real-world application [13]. For the evaluation, a state-of-the-art NoSQL database, MongoDB, is compared with a relational DBMS from a major commercial vendor and with a popular open-source relational DBMS. The main contribution of the paper is a performance evaluation that compares the suit-

ability of the two kinds of database systems for large-scale log analytics with various indexing strategies and reveals the trade-offs between data loading under various consistency configurations.

In the performance evaluation, for persisting large-scale data logs, our results revealed that relaxing the consistency did not provide substantial performance enhancement for neither relational nor NoSQL databases. We discovered that both commercial and open-source relational databases provide less than 25% performance improvement for bulk-loading with relaxed transaction consistency. For MongoDB, weak consistency configuration of bulk loading provides around 26% improvement. However, this performance is expected to be much higher on a highly distributed system as it requires updates in multiple replica nodes where relaxing consistency has a substantial impact. Overall, MongoDB performs significantly better compared to the open-source database and has comparable performance with the commercial database.

To understand the scalability of analytics workloads for the databases, we defined three fundamental queries for investigating the performance impact of query processing and index utilization. The properties of the chosen queries are key selection, range search, and aggregation. The queries are fundamental to analytics of IoT workloads and provide basic building blocks of the queries presented in *Paper I* and [48]. We made the experimental queries simplistic in nature, which is desirable for domain-specific benchmarks [18]. Compared to the complex queries presented in *Paper I* and [48], these queries allow us to investigate the internals of the DBMSs for a better understanding of the performance trade-offs. Our subsequent papers also used these queries.

In the performance evaluation for query execution, we found that the key look-up query that utilizes the primary key index scales well in all the DBMSs. Furthermore, we observe that range queries in commercial databases scale better for non-selective queries, while MongoDB is faster for selective ones. The reason is that unlike MongoDB, the commercial database switches from a non-clustered index scan to a full table scan when the selectivity is sufficiently low, while MongoDB continues to use a non-clustered index scan. Finally, an aggregation query scales well for all systems by utilizing the secondary index when computing an aggregated value. Here, distributed MongoDB scales best compared to single-instance MongoDB, commercial, and open-source DBMS; as parallel scans without sending lots of results among distributed nodes speed up query execution.

From the overall performance evaluation involving data archival and query analytics, we found that both MongoDB and the commercial relational DBMS perform significantly better compared to the open-source relational DBMS. Furthermore, the commercial relational DBMS sometimes has performance advantages in query execution compared to MongoDB by having a sophisticated query optimizer. By contrast, distributed MongoDB is an alternative to vertical scaling of a relational DBMS for inherently parallel IoT workloads.

We concluded that for high-performance loading and analytics of data logs, MongoDB is a viable alternative compared to relational databases for queries where the choice of an optimal execution plan is not critical.

This paper provides answers to research questions **one** and **three**.

I am the primary author of this paper. Tore Risch contributed to the discussions and the revisions of the manuscript. Minpeng Zhu participated in the revisions of the manuscript.

3.4 Paper IV

The performance of MongoDB from *Paper III* motivated us to explore the suitability of other kinds of distributed NoSQL data stores for large IoT applications.

Large IoT applications typically produce a massive volume of data, which makes single server storage incapable of scaling analytics workloads. An additional challenge arises when a DBMS is deployed centrally, which might prone to a single point of failure that could be fatal for mission-critical applications. In order to alleviate these issues, a modern distributed NoSQL data store can be utilized to achieve high-performance, scalability, and availability. These data stores can be deployed over many machines, which can eliminate single point of failure. In this paper, we provided a performance comparison of three principal categories of distributed state-of-the-art NoSQL data stores [4] to evaluate their applicability and efficiency for large-scale analytics of sensor logs from real-world IoT applications. For the performance evaluation, one state-of-the-art data store is selected from each of the three main categories of NoSQL data stores: MongoDB as a *document store*, Cassandra as a *column store* and Redis as a *distributed main memory key-value store*. These data stores use different kinds of data partitioning, indexing, and storage mechanisms, which are also used in a variety of other contemporary NoSQL systems and major cloud service providers [24]. For example, the proof-of-concept implementation of a key-value store in Amazon Web Services is DynamoDB [11], while Google Cloud Platform uses Bigtable [5]. Therefore, understanding the architectural differences of these systems provides crucial insights for optimizing analytics performance for a variety of large-scale IoT applications.

In the performance evaluation, we investigated two tasks that represent typical IoT data analytics. These tasks are similar to the *key-lookup* and *range* queries presented in *Paper III*. We executed these tasks over 100 million sensor data logs, distributed across 10 nodes for each of the three distributed NoSQL data stores.

The first task of finding a particular log record that resembles a key look-up query, scales well in all three systems as it runs under 10 milliseconds. This task is leveraged by utilizing a primary key index supplied by all systems. In

the case of Cassandra and Redis, a primary index was supported by the use of a consistent hashing technique, while for MongoDB a range-based hashing technique was used. This essentially demonstrates that both hashing techniques are equally capable of finding a specific record from a large data set. Internally, a hash value based on the key attribute determines which node is responsible for retrieving the specific record. Then the request is dispatched to the specific server node for further processing of the task and retrieving the result for the client. These steps are included in all three systems and, therefore, exhibit equivalent performance for the key lookup task.

The second type of task, involving a range query, reveals more interesting performance trade-offs for the three types of systems. To speed up this task, we used a secondary ordered index which in the case of MongoDB was a B-tree index, while for Cassandra an inverted index was used. Since Redis is an in-memory data store, we used its in-memory sorted set. As expected, the in-memory data store Redis outperforms both the disk-based NoSQL data stores Cassandra and MongoDB. On the other hand, MongoDB performs better compared to Cassandra as it's B-tree index often resides in memory causing fewer disk accesses than Cassandra's inverted index. We later on investigated the performance of an aggregated task, which is not presented in this paper. This task is essentially an aggregation over the range query that uses the same secondary ordered index for calculating the aggregation result. Therefore, it demonstrated a similar performance of range query, making in-memory Redis the fastest, followed by MongoDB.

From the performance evaluation, it is clearly noticeable that for MongoDB, when we utilize the range-based hashing technique for data partition together with the in-memory B-tree indexing, the performance of a range query is significantly better compared to Cassandra's disk-based inverted index. However, it is noteworthy that the range-based partitioning in MongoDB is also more prone to skewed load balancing when the distribution of the partition key is biased. As a result, the B-tree index over skewed data residing in a particular node could potentially be a bottleneck, which leads to overall performance degradation of a range query. Apart from data partitioning, in-memory data stores, Redis, demonstrated significant performance advantages over disk-based counterparts such as MongoDB and Cassandra. Though Redis uses a consistent hashing technique compare to the range-based hashing of MongoDB, its in-memory data structures plays a significant role in enabling such a performance boost. However, this performance advantage comes at a cost of sacrificing the persistence guarantees provided by disk-based data stores, which could be prone to data loss that is critical in many IoT applications. Furthermore, most IoT applications require large volumes of data to be analyzed, so providing enough main-memory is not practical; therefore, disk-based NoSQL data stores are required.

While MongoDB offers better performance compared to Cassandra and provides disk-based implementation compare to Redis, it also supports dis-

tributed storage of JSON objects. These features of MongoDB makes it suitable for large-scale heterogeneous storage for IoT applications compared to other NoSQL data stores. Therefore, in future research, we planned to develop an advanced query processing system to overcome the limited query capability of MongoDB for enhancing IoT data analytics. This work is presented in *Paper V-VII*.

This paper provides answers to research questions **one** and **four**.

I am the primary author of this paper. My co-authors have contributed to the discussions and the revisions of the manuscript.

3.5 Paper V

A conclusion from *Paper III* and *Paper IV* is that, while scalability can be provided by the distributed NoSQL data store MongoDB, advanced data analytics is difficult to provide due to its limited query processing capabilities. In this paper, we discussed how a prototype system that combines the distributed scalability of MongoDB with a relationally complete [7] query processor can provide advanced analytics. We adopted the wrapper-mediator approach [44] by integrating MongoDB with the in-memory mediator database Amos II [43] [42]. Amos II provides an object-oriented data model, a relationally complete query language AmosQL, and numerical operators. Therefore, utilizing Amos II as a mediator system will provide advanced query processing compared to the more limited query processing of MongoDB. Furthermore, the Amos II kernel has been extended to provide streaming query processing in the systems SVALI [51] and sa.engine [47] . Therefore, our implemented system enables query-based online data stream analytics in front of the MongoDB distributed NoSQL data store to support large-scale data analytics over distributed IoT applications.

The main-memory database Amos II provides a *foreign function* interface that supports several programming languages (C, Java, Python, or Lisp) for accessing external data stores, systems, and DBMSs. To provide high-performance access to MongoDB, we implemented a set of foreign functions in C. Internally MongoDB represents objects and expresses its queries as JSON. Since JSON is equivalent to the *Record* data type in Amos II, we provided an efficient implementation of type conversion between JSON and type *Record* to interchange data between the two systems. The implemented foreign functions utilized type *Record* to express queries over MongoDB, while the AmosQL is operated over these foreign functions to enable advanced query processing. The preliminary testing by injecting large sensor streams shows that our prototype system provides improved performance compared to state-of-the-art relational databases, even though it has some overhead caused by local query processing performed in the Amos kernel. This work was a part of our on-

going implementation of a fully-functional wrapper, which was enhanced in *Paper VI* with further implementation and empirical evaluations.

Paper V provides answers to research questions **one** and **five**.

I am the primary author of this paper. My co-authors have contributed to the discussions and the revisions of the manuscript.

3.6 Paper VI

Contemporary distributed IoT applications consist of large numbers of sensors, producing massive volumes of heterogeneous sensor streams at high rates. The combination of these features of IoT applications poses substantial challenges for existing DBMSs in providing scalable data analytics. For example, data management of distributed IoT applications often requires seamless integration between a real-time data analytics platform, distributed data storage, and effective data integration techniques for handling heterogeneity. In this paper, we discussed a holistic data management approach for supporting edge and offline analytics for distributed IoT applications. This paper primarily has three contributions. Firstly, we demonstrate how to utilize relational DBMSs effectively to support scalable analytics for IoT. Then, we discuss the limitations of the relational DBMS approach, especially for supporting heterogeneity, which led us to adopt the alternative approach of utilizing a NoSQL DBMS to mitigate the challenges. Finally, we discuss the limitations of the query execution of the NoSQL database MongoDB and developed an *Extended Query Processing (EQP)* system to support advanced analytics.

3.6.1 Relational DBMS Approach

Large IoT applications often consist of geographically distributed sites. In such a scenario, locally at each site, real-time edge analytics is often preferred in order to deliver immediate responses. Since providing real-time analytics with disk-based technology is slow, an in-memory *edge data stream management system (EDSMS)* can be deployed close to the sensors for supporting scalable online analytics data streams. However, the data streams at each site often need to be stored for further offline analytics, which can be offered by deploying local relational log databases. This IoT application context becomes equivalent to the application scenario described in *Paper I*, where general query processing is performed by FLOQ over the global view derived from local log databases by integrating their meta-data. As discussed in Section 3.1, the local-as-view approach adopted by FLOQ scales well for large distributed IoT applications. However, this view-based data integration approach using relational databases leads to complexity for heterogeneous sensors data. Therefore, in this paper, we developed a data management system by adopting

the document store NoSQL database MongoDB that provides heterogeneous data storage through JSON objects.

3.6.2 NoSQL Data Stores for IoT Data Management

The approach of integrating a NoSQL data store in an IoT application is utilized by deploying the distributed MongoDB to eliminate the local relational databases of the RDBMS approach. Instead of archiving stream logs at each local log database, we inject streams from the EDSMSs directly into the distributed MongoDB. Since distributed MongoDB can easily shard and replicate data over many computing nodes, this approach provided scalability and avoids a single point of failure.

When injecting large volumes of sensor logs from EDSMSs to DBMSs, any use of disk-access techniques should possibly be avoided. NoSQL data stores like MongoDB allow multiple data insertions without using any intermediate disk-based storage. In this paper, we implement the *online bulk-loading (OBL)* strategy that first accumulates a large volume of stream elements in-memory and then converts them into MongoDB's native data format, *binary-JSON (BSON)* [35]. After that, a single bulk insertion command to MongoDB is issued, which injects all staged in-memory data stream elements into MongoDB without using any disk-based techniques.

Typically, relational databases use normalization [6] to eliminate data duplication to reduce update anomalies. When we use relational DBMSs in our previous approaches, both sensor data and meta-data tables are normalized. Hence, the distributed join strategies (i.e., *PBJ* or *PBLJ*) of FLOQ is required to perform the analytics. This approach exhibits difficulty in managing heterogeneity due to the rigid relational schemas; furthermore, distributed joining of multiple tables proposed in FLOQ, is inherently expensive. By contrast to normalization, if we adopt a denormalization approach [45] in distributed MongoDB by combining sensor properties within a single JSON object, the distributed joins can be replaced with a single MongoDB collection lookup. Furthermore, the JSON storage capability of MongoDB also enables the injection of data streams of various formats into a single MongoDB collection, which overcomes the limitations of earlier approaches of using a relational database for handling heterogeneity. Hence, by using MongoDB, we can adopt the denormalization approach combined with a dynamic schema-free paradigm to support heterogeneity better.

3.6.3 Advanced Query Processing

Although integrating distributed MongoDB provides storage for heterogeneous sensor logs, it lacks the advanced query processing of regular RDBMSs (e.g., relationally complete queries [7], joins, and numerical operations). There-

fore, we implemented the *Extended Query Processing System (EQP)* on top of MongoDB to enable these advanced analytics. Similar to FLOQ, EQP also enables general query processing over the integrated log databases. From our ongoing work on a MongoDB wrapper presented in *Paper V*, in this paper, we implemented the query processing of both EQP and OBL as a unified system in the in-memory mediator database Amos II. The fully functional query processing system is developed inside the Amos II kernel that utilized the foreign functions implemented in C to access the external data source MongoDB.

Since AmosQL is relationally complete, it enables EQP to perform complex queries involving joins, aggregations, and numerical operations (e.g., sum, stdev). Therefore, the query execution of EQP overcomes the limitations of MongoDB for performing advanced analytics.

Paper VI provides answers to research questions **one**, **two**, **three**, and **five**.

I am the primary author of this paper. My co-authors have contributed to the discussions and the revisions of the manuscript. Tore Risch also contributed to the implementations.

3.7 Paper VII

In this paper, we demonstrate a data management solution by enabling both edge and offline analytics based on an IoT use case involving sound anomaly detection on distributed equipment. In this work, we adopted the NoSQL approach presented in *Paper VI* for providing a data management solution for our application scenario. We utilized the EDSMS sa.engine [47] to support edge analytics for real-time sound anomaly detection, while we used EQP to provide further offline analytics over the MongoDB backend. In our previous experimental evaluations for the NoSQL approach provided in *Paper VI*, we used Amos II to simulate the DSMS by emitting CSV streams to MongoDB. In contrast, this paper used sa.engine to perform the edge analytics and the output streams were injected through EQP. As a result, this paper demonstrated how EQP provides a holistic data management solution for a realistic IoT application scenario.

Paper VII provides answers to research questions **one** and **five**.

I am the primary author of this paper. Tore Risch contributed to the discussions and the revisions of the manuscript.

4. Future Work

Indexing plays a significant role for improving query performance. We observed the impact of indexing in our work for scalable query execution over large-scale sensor logs stored in both relational and NoSQL databases [34] [30]. However, we noticed in *Paper II* that the size of the indexing can be a major overhead, sometimes equivalent to the size of the stored data. For large-scale IoT applications that are often distributed across many computing nodes, we typically need to store gigabytes of heterogeneous data in various format. Therefore, future research could focus on developing memory-resident distributed indexing strategies. The designing criteria of such domain specific indexing would be to provide space-efficient data structures (e.g., using compression and probabilistic data structures), which could offer scalable query execution over heterogeneous data format, such as JSON.

For the experimental evaluations of this thesis presented in *Paper III-IV*, and *VI*, we utilized fundamental queries involving key-lookup, aggregation and range search. The primary principle for designing queries that are simplistic in nature was to observe the internal architecture of the DBMSs, such as their storage mechanisms, query processing approaches, and index utilization for our domain-specific application scenario. Furthermore, adopting consistent benchmark settings comprising of the same queries across different data processing platforms allowed us to compare and contrast the architectural difference of various relational and NoSQL databases. The outcome of the benchmarks motivated us to choose MongoDB for developing prototype system Extended Query Processing (EQP) to support complex query execution. In the future, more complex queries (e.g., generalizing the numeral queries as in *Paper I*) are required for evaluating the performance of EQP. Future research would focus on emploing domain-specific IoT benchmarks, such as IoTAbench [2] for performance evaluation of FLOQ and EQP to understand how relational and NoSQL databases perform under generic IoT application scenarios.

In this thesis, we have discovered various trade-offs in choosing relational and NoSQL databases for different application contexts. For example, a relational DBMS can be a good choice for storing structured data where its sophisticated query optimizer facilitates efficient analytics. In contrast, NoSQL databases provide fault-tolerance and load-balancing of large distributed applications, but with limited query capabilities. Among a variety of NoSQL data stores, the document stores support heterogeneous data formats, the column stores enable efficient online analytics (OLAP) by clustering columns for

efficient filtering of projection operators, and the in-memory key-value stores provide distributed caching of frequently accessed data. For optimizing the performance of distributed applications, combining the features of both relational and NoSQL databases can bring the benefit of both worlds. Therefore, future research would focus on developing wrapper-mediator systems combining relational databases and a variety of NoSQL data stores. This will enable global query optimization over a variety of data sources; which will leverage the strength of each system and overcome limitations (e.g., supporting query processing over contemporary NoSQL databases).

NoSQL databases, such as document stores, typically store data as dynamic key-value objects (e.g., JSON). While there are several advantages of native JSON data stores, the absence of schema information makes query optimization difficult. Relational query optimizers rely on meta-data such as the schemas and statistics of data in columns within a table. Sophisticated query optimizers include cost models based on these statistics to find efficient plans for query execution. In absence of schemas and statistics in a JSON storage manager, building cost models to facilitate query optimization is challenging. Therefore, interesting future work would be to overcome the limitation of a schema-less dynamic key-value storage by developing efficient dynamic query optimization strategies.

5. Summary in Swedish

Alltmer företag använder utrustning som är uppkopplad mot Internet, ofta kallat *Sakernas Internet* (eng. *Internet of Things, IoT*) Ofta görs verksamhetskritiska beslut baserat på information från sådan uppkopplad utrustning. Uppkopplad IoT-utrustning kan innehålla många sensorer som producerar mycket stora mängder data, ofta i mycket hög takt. Allteftersom mängden uppkopplad utrustning blir större ökar därför behovet av effektiv hantering av data i form av kontinuerliga flöden, s.k. *dataströmmar*, av mätvärden från sensorer på geografiskt distribuerad uppkopplad utrustning. Kombinationen av stor mängd utrustning med många sensorer som producerar stora datavolymer ger stora utmaningar för existerande databashanteringssystem (eng. *Data Base Management System, DBMS*) att tillhandahålla verktyg för att utföra skalbar lagring, sökning och analys över stora datamängder. S.k. *frågespråk* i DBMS gör det möjligt för användare att utföra avancerad analys av databasens innehåll på en hög nivå utan detaljerad programmeringskunskap. Dagens DBMS-marknad domineras av tabellorienterade databashanteringssystem, s.k. *relationsdatabaser*, med frågespråket *SQL*.

Denna avhandling analyserar problem med att tillhandahålla effektiv hantering av data från distribuerade IoT-applikationer med hjälp av DBMS-teknologier. Inledningsvis utvecklade vi ett prototypsystem, *Fused LOg database Query processor (FLOQ)*, som möjliggör sökning m.h.a. ett frågespråk i samlingar av geografiskt distribuerade relationsdatabaser där sensorloggar från IoT-utrustning kontinuerligt lagras. På så vis tillhandahåller FLOQ effektiv och kraftfull sökning över många databaser innehållande sensorloggar. Emellertid uppvisar FLOQ problem med begränsad flexibilitet och skalbarhet för IoT-applikationer där lagrade data är heterogena med olika format och där utrustningen inte är ständigt uppkopplad. Begränsningarna för FLOQ beror främst på dess krav på centralt uppkopplade relationsdatabastabeller med tabellorienterad representation av sensorloggar.

Sammanfattningsvis, när en relationsdatabas används för att lagra IoT-data i stor skala, uppvisar den flera begränsningar:

- En relationsdatabas är inte alltid tillräckligt snabb för att ladda datarika loggar som produceras med höga hastigheter, beroende på dess starka konsistensmekanismer.
- Om data från uppkopplad utrustning integreras m.h.a. en central relationsdatabas som i FLOQ kan databasen utgöra en systemkritisk felpunkt, eftersom delar av databasen tidvis blir otillgänglig eller inaktuell när utrustningen tidvis är bortkopplad.

- Oflexibel tabellorienterad representation av data i relationsdatabaser gör det komplicerat att dynamiskt lägga till eller ta bort sensorer med olika dataformat.
- Tabellorienterade SQL-frågor gör det besvärligt och ineffektivt att söka i heterogena data från sensorer med olika dataformat.

Medan en relationsdatabas tillhandahåller en strukturerad datamodell, dynamiska SQL-frågor och stark transaktionskonsistens, offrar s.k. *NoSQL-databaser* några av dessa attribut. Termen *NoSQL* hänvisar till "Inte bara SQL" eller "Inte relationell". För att uppnå högre tillgänglighet och skalbarhet än relationsdatabaser, tillhandahåller NoSQL-databaser vanligtvis enbart enkla operationer som att infoga, uppdatera eller slå upp en enda post, snarare än de komplexa transaktions- och frågemöjligheterna hos relationsdatabaserna. NoSQL-databaser har dock vanligtvis begränsningar i att utföra avancerad analys med på grund av deras bristfälliga frågespråk.

En hypotes som undersöks i denna avhandling är i vilken grad distribuerade NoSQL-databaser kan tillhandahålla skalbar lagring och analys av heterogena data för IoT-tillämpningar bättre än relationsdatabaser. Distribuerade NoSQL-databaser kan lätt skalas upp genom utspridning (partitionering) och replikering av data. Vidare kan hög tillgänglighet åstadkommas genom att offra den starka konsistens m.h.a. transaktioner som tillhandahålles av relationsdatabassystem. För att förstå lämpligheten hos NoSQL-databaser undersöker avhandlingen i vilken grad olika sorters NoSQL-databaser är tillämpliga för IoT-applikationer genom att jämföra prestanda hos ett antal väl kända relations- och NoSQL-databaser för industriella IoT-tillämpningar.

De experimentella utvärderingarna visar att utmärkt skalbarhet kan tillhandahållas av distribuerade NoSQL-databaser. Emellertid har de begränsat stöd för avancerad dataanalys på grund av avsaknad av eller svagt frågespråk. Dessutom kräver datahantering för distribuerade IoT-applikationer ofta att man kombinerar lokal analys i realtid direkt på utrustningen med distribuerad central datalagring utan kontinuerlig uppkoppling. Eftersom sensorer och andra datakällor på utrusningen producerar data i olika format krävs vidare effektiv integration av data i olika format kombinerat med frågebehandlingstekniker för att hantera heterogena data. Dessutom måste arkitekturen tillåta att man enkelt och effektivt kan lägga till och ta bort datakällor allteftersom uppkopplad utrustning ändras.

För att tillhandahålla en holistisk datahanteringsarkitektur som undviker ovanstående begränsningar hos konventionella DBMS utvecklades ett prototypesystem benämnt *Extended Query Processing (EQP)* som kombinerar avancerad analys av strömmande och lagrade data på distribuerad utrustning med central analys i en skalbar NoSQL-databas.

6. Acknowledgements

First of all, I would like to thank *Professor Tore Risch*. Being my primary supervisor, you have helped me to grow as an independent researcher with great patience and care. You shared your knowledge, which helped me to learn the internals of database systems that I valued the most. Thank you for your help in the technical implementations and scientific writings. I have also greatly enjoyed the time passing with you, especially when traveling to Edinburgh.

I am grateful to my supervisor *Kjell Orsborn* for helping me in the toughest time of my Ph.D. study. A path towards the Ph.D. is not always a straight line and without your help, I could not be able to reach at this point of my thesis.

I am indebted to *Matteo Magnani*. You, not only helped me in my research on numerous occasions but also assisted me personally with your valuable suggestions. I am lucky to meet you during my Ph.D. career.

Davide Vega D'Aurelio, thank you for showing me the dark side of the moon :-). I learned from you that there are many important stuff in life apart from coding and research. Yes, I am talking about the board games, cooking, and many fun activities that you have introduced to me. Thank you very much for remaining beside me in all these years.

I would like to thank *Georgios Fakas* and *Erik Zeitler* for assisting me during my Ph.D study. *Georgios*, I really enjoyed the time on teaching database courses with you. *Erik*, your kind words were very supportive during the time when it was needed the most.

Thank you, *Professor Pierre Flener*, for being so supportive during the final year of my study.

Thank you, *Justin Pearson* for your great help for reviewing my doctoral dissertation.

To my dear friend *Amin Kaveh* and *Georgios Kalamatianos*, thank you for being close to me during my happy and gloomy moments. Both of you have pacified my soul during the turmoil of all these years.

I am grateful to my UDBL fellow members, *Silvia Stefanova*, *Lars Melander*, *Minpeng Zhu*, *Andrej Andrejev*, *Thanh Truong*, *Sobhan Badiozamany*, and *Miran Nadir*. *Minpeng*, you helped me in many ways not only in research but also in different aspects of my personal life. I learned a lot being close to you. Thank you *Sobhan* for the help that you have provided me in various aspects of my life. *Miran*, I truly enjoyed passing time with you.

Thank you, *Ulrika Andersson*. You made my life in Sweden easy by providing me numerous administrative supports, such as visa applications, housing, and many more.

I am really grateful to my family here in Sweden for the enormous support that you have provided me. Thank you, *Valo Khalamoni* and *Valo Khalu - Kamrul Islam* and *Jasmin Jahan Islam*. The continuous support that you have provided me during the last nine years during my stay in Sweden is incredible. Without you, I could never be able to come up to this point. You have loved and cared me with your heart, for which I am eternally grateful. And, to my little brother *Tausif Islam*, I am thankful for your help. For the last nine years, I have found you always beside me, whenever needed. Your suggestions during some of my passing hardships were extremely helpful. I have learned a great deal from you.

To you *Mehbuba Tabassum*, during my life in Sweden, you were always beside me like a protecting angel. You were the part of my crazy ideas, insane dreams, ridiculous visions. You always supported my foolishness. Thank you for believing me. Your calm and serene ways of taking care of my myriads of problems were truly a haven. Thank you for your support during the hardest time of my life. You are awesome!

To my dear parents, thank you *Abbu* and *Ammu*, I am eternally grateful to you. The amount of sacrifice that you have made in your life for my career and education is unimaginable. Even until today, you are always beside me and tried to support me to the fullest, against all the odds, while never thinking of yourself. No son can be as fortunate as me to find you in my life. Because of you, I am able to come this far. And thank you, my dear little sister, *Nusrat Sharmeen* for being so kind to me.

Finally, I would like to thank my grandfather *Abul Hossain*. Thirty years ago, when I was a little kid, you sow the dream of studying Computer Science in my heart. You are my superhero, my North Star. When I am in doubt and facing a tough time, I always looked into your life to get inspiration. Thank you for your unconditional love. I wish of being a great human being like you.

This project is supported by the Swedish research program eSSENCE and European Union FP7 project Smart Vortex.

References

[1] Apache Software Foundation. Cassandra. `http://cassandra.apache.org`. [accessed 12 November 2021].

[2] Martin Arlitt, Manish Marwah, Gowtham Bellala, Amip Shah, Jeff Healey, and Ben Vandiver. IoTabench: an internet of things analytics benchmark. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, pages 133–144, 2015.

[3] Eric A Brewer. Towards robust distributed systems. In *PODC*, volume 7, pages 343477–343502. Portland, OR, 2000.

[4] Rick Cattell. Scalable SQL and NoSQL data stores. *Acm Sigmod Record*, 39(4):12–27, 2011.

[5] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):1–26, 2008.

[6] Edgar F Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, June 1970.

[7] Edgar F Codd. *Relational completeness of data base sublanguages*. Citeseer, 1972.

[8] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, third edition, 2009.

[9] Danga Interactive. Memcached. `https://memcached.org`. [accessed 12 November 2021].

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

[11] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. *ACM SIGOPS operating systems review*, 41(6):205–220, 2007.

[12] Ramez Elmasri and Sham Navathe. *Fundamentals of database systems*. Pearson, seventh edition, 2017.

[13] European Union. Smart Vortex Project. `http://smartvortex.eu`. [accessed 12 November 2021].

[14] Peter C Evans and Marco Annunziata. Industrial internet: Pushing the boundaries. *General Electric Reports*, pages 488–508, 2012.

[15] Gustav Fahl and Tore Risch. Query processing over object views of relational data. *The VLDB Journal*, 6(4):261–281, 1997.

[16] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *Acm Sigact News*, 33(2):51–59, 2002.

[17] Lukasz Golab and M Tamer Özsu. Data stream management. *Synthesis Lectures on Data Management*, 2(1):1–73, 2010.

[18] Jim Gray. *Benchmark handbook: for database and transaction processing systems*. Morgan Kaufmann Publishers Inc., 1992.

[19] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[20] Laura M. Haas, Donald Kossmann, Edward L. Wimmers, and Jun Yang. Optimizing queries across diverse data sources. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, pages 276–285, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[21] Alon Halevy, Anand Rajaraman, and Joann Ordille. Data integration: The teenage years. In *Proceedings of the 32nd international conference on Very large data bases*, pages 9–16, 2006.

[22] Stavros Harizopoulos, Daniel J. Abadi, Samuel Madden, and Michael Stonebraker. *OLTP through the Looking Glass, and What We Found There*, pages 409–439. Association for Computing Machinery and Morgan & Claypool, 2018.

[23] Joseph M Hellerstein and Michael Stonebraker. Anatomy of a database system. *Readings in Database Systems,*, 2005.

[24] Google Inc. Map AWS services to Google Cloud Platform products. `http://cloud.google.com/free/docs/map-aws-google-cloud-platform/`. [accessed 12 November 2021].

[25] 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, pages 654–663, New York, NY, USA, 1997. Association for Computing Machinery.

[26] Yannis Katsis and Yannis Papakonstantinou. View-based data integration. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 4452–4461, New York, NY, 2018. Springer New York.

[27] Jay Lee, Behrad Bagheri, and Hung-An Kao. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing Letters*, 3:18 – 23, 2015.

[28] LinkedIn. Project Voldemort. `http://www.project-voldemort.com/`. [accessed 12 November 2021].

[29] Witold Litwin and Tore Risch. Main memory orientated optimization of oo queries using typed datalog with foreign predicates. *IEEE Transactions on Knowledge and Data engineering*, 4(6):517–528, 1992.

[30] Khalid Mahmood, Kjell Orsborn, and Tore Risch. Comparison of NoSQL datastores for large scale data stream log analytics. In *2019 IEEE International Conference on Smart Computing*, pages 478–480, Washington D.C., USA, 2019.

[31] Khalid Mahmood, Kjell Orsborn, and Tore Risch. Wrapping a NoSQL datastore for stream analytics. In *2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI)*, 2020.

[32] Khalid Mahmood and Tore Risch. Scalable real-time analytics of IoT

applications. In *2021 IEEE International Conference on Smart Computing*, pages 478–480, California, USA, 2021.

[33] Khalid Mahmood, Tore Risch, and Kjell Orsborn. Analytics of IIoT data using a NoSQL datastore. In *2021 IEEE International Conference on Smart Computing*, pages 478–480, California, USA, 2021.

[34] Khalid Mahmood, Tore Risch, and Minpeng Zhu. Utilizing a NoSQL data store for scalable log analysis. In *Proceedings of the 19th International Database Engineering & Applications Symposium*, pages 49–55. ACM, 2015.

[35] MongoDB Inc. BSON Types. `https://docs.mongodb.com/manual/reference/bson-types/`. [accessed 12 November 2021].

[36] MongoDB Inc. MongoDB. `http://www.mongodb.com`. [accessed 12 November 2021].

[37] MongoDB Inc. Query Plan. `https://docs.mongodb.com/manual/core/query-plans/`. [accessed 12 November 2021].

[38] MongoDB Inc. Ranged Sharding. `https://docs.mongodb.com/manual/core/ranged-sharding/`. [accessed 12 November 2021].

[39] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J Abadi, David J DeWitt, Samuel Madden, and Michael Stonebraker. A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 165–178, 2009.

[40] Felipe Pezoa, Juan L Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. Foundations of json schema. In *Proceedings of the 25th International Conference on World Wide Web*, pages 263–273, 2016.

[41] Redis Labs. Redis. `http://redis.io`. [accessed 12 November 2021].

[42] Tore Risch and Vanja Josifovski. Distributed data integration by object-oriented mediator servers. *Concurrency and computation: Practice and experience*, 13(11):933–953, 2001.

[43] Tore Risch, Vanja Josifovski, and Timour Katchaounov. Functional data integration in a distributed mediator system. In Peter M. D. Gray, Larry Kerschberg, Peter J. H. King, and Alexandra Poulovassilis, editors, *The Functional Approach to Data Management: Modeling, Analyzing and Integrating Heterogeneous Data*, pages 211–238. Springer, 2004.

[44] Mary Tork Roth and Peter M. Schwarz. Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In *Proceedings of the VLDB*, pages 266–275, 1997.

[45] G. Lawrence Sanders and Seungkyoon Shin. Denormalization effects on performance of rdbms. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)*, USA, 2001.

[46] P Griffiths Selinger, Morton M Astrahan, Donald D Chamberlin, Raymond A Lorie, and Thomas G Price. Access path selection in a relational database management system. In *Readings in Artificial Intelligence and Databases*, pages 511–522. Elsevier, 1989.

[47] Stream Analyze. sa.engine. `http://streamanalyze.com/under-the-hood/`. [accessed 12

November 2021].

[48] Thanh Truong and Tore Risch. Scalable numerical queries by algebraic inequality transformations. In *International Conference on Database Systems for Advanced Applications*, pages 95–109. Springer, 2014.

[49] Wikipedia. Catalan number. `http://en.wikipedia.org/wiki/Catalan_number`. [accessed 12 November 2021].

[50] Felix Wortmann and Kristina Flüchter. Internet of Things. *Business & Information Systems Engineering*, 57(3):221–224, 2015.

[51] Cheng Xu, Elisabeth Källström, Tore Risch, John Lindström, Lars Håkansson, and Jonas Larsson. Scalable validation of industrial equipment using a functional dsms. *Journal of Intelligent Information Systems*, 48(3):553–577, 2017.

[52] Minpeng Zhu, Khalid Mahmood, and Tore Risch. Scalable queries over log database collections. In *British International Conference on Databases*, pages 173–185. Springer, 2015.

Acta Universitatis Upsaliensis

Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 2095

Editor: The Dean of the Faculty of Science and Technology

A doctoral dissertation from the Faculty of Science and Technology, Uppsala University, is usually a summary of a number of papers. A few copies of the complete dissertation are kept at major Swedish research libraries, while the summary alone is distributed internationally through the series Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology. (Prior to January, 2005, the series was published under the title "Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology".)

Distribution: publications.uu.se
urn:nbn:se:uu:diva-458420

ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2021