EXAMINE AND COMPARE DATABASE TECHNOLOGIES TO TIME SERIES DATA

Yosafe Fesaha Oqbamecail

Master's Thesis in Applied Computer Science and Engineering

Engineering Data Science Specialisation

Faculty of Technology, Environmental and Social Sciences

Western Norway University of Applied Sciences

21st October 2025

ABSTRACT

This is where you write the abstract.

ACKNOWLEDGMENTS

I would like to thank . . .

Contents

CHAPTER 1

INTRODUCTION

The seas and oceans play a key role in the operation of global ecosystems, climate regulation, food security, and energy production[9]. Monitoring of such large and complex environments requires advanced technology. Among these, underwater sensor networks (UWSNs) have become valuable resources for continuous real time data collection in marine ecosystems, subsea structures, and oceanic processes[13].

The SFI Smart Ocean project is designed to create an autonomous and flexible wireless marine observation program. The system will allow for large scale, long term monitoring of underwater spaces as well as installations by combining UWSNs and cloud based big data solutions. The infrastructure is intended to deal with multi parameter observations, ensuring reliable data for both scientific research as well as industrial applications[36, 38].

Limitations of acoustic communication, long range UWSN deployments are impacted by environmental and technical challenges. These consist of power restrictions, minimal data rates and the inability to recalibrate once deployed at depth. Sensors are subjected to extreme conditions. Data quality can be affected by pressures, biofouling, corrosion, along with electronic drift with time. Moreover, environmental measurements are affected by parameters such as temperature and salinity can introduce additional uncertainty[42]. Once data reaches surface nodes, there is a need for efficient data handling. The storage infrastructure needs to be able to deal with large volumes of inbound data and maintain longterm availability.

The thesis investigates techniques for managing time series data produced by the Smart Ocean sensor network. The emphasis is on evaluating the development of database technologies that can deal with large scale storage and effective querying of time stamped observations. The work contributes to the wider picture of the Smart Ocean which aims to promote data driven decision making in marine operations. This research aligns with the ACM Computing Classification System (CCS) under Information systems → Data management systems → Database administration → Database performance evaluation[1]. The study focuses on

evaluating and comparing time series database technologies to improve the efficiency and reliability of underwater sensor data management.

1.1 Context and Motivation

1.1.1 The SFI Smart Ocean Platform

The Norwegian Research Council has provided funding for the SFI Smart Ocean project. Research institutes as well as industry partners work together on key issues in building smart ocean systems[39]. The initiative concentrates on 3 main areas:

1. **Underwater Sensor and Measurement Technology**: Focuses on developing autonomous sensors and methods for real time monitoring of underwater environments. These sensors include features like data collection, acoustic communication, and energy efficient operations.
2. **Underwater Wireless Sensor Networks Based on Acoustic Communication**: Addresses the establishment of reliable communication networks for data transmission from underwater sensors.
3. **Smart Ocean Platform for Cloud Based Data and Application Services**: Consists of developing the Smart Ocean platform to incorporate, process, and visualize ocean information. The platform uses Standard APIs & efficiently formats to deal with data from diverse underwater sensors.

1.1.2 Motivation

The deployment of Smart Ocean's sensor network will generate a large volume of time series data. The underwater sensors continuously measure parameters over extended periods, resulting in a continuous stream of timestamped readings. The data volume rapidly reaches big data scales with dozens or possibly hundreds of sensors reporting in real time (each measuring several variables). The system needs to store years of data reliably, deal with high frequency ingestion, and support queries for real time alerts and long term analysis a major challenge.

The selection of suitable data management technologies is crucial given these requirements. Standard relational databases, although robust, are not specifically designed to deal with high volume time series workloads where data arrives sequentially and is mainly queried by time. In recent years, specialized Time Series Management Systems (TSMSs) have emerged in response to the high volume, high velocity nature of data generated by IoT devices and sensors. Unlike general purpose databases, TSMSs are architected to efficiently store, query, and process time stamped data streams, making them uniquely suited for real time sensor workloads[26]. These systems are designed to efficiently ingest and index time stamped data and provide built in functions for time window queries, downsampling, and time centric analytics. Dedicated TSDBs are increasingly considered a natural fit for fast sensor data streams, yet with a wide variety of database options available, it is not obvious which technology is most suitable for the Smart Ocean platform.

1.2 Problem Description and Research Questions

the best way to efficiently store and manage the large amounts of time series data generated by the Smart Ocean platform in the above mentioned context is to determine what database technology (or combination of technologies) best meets the platform's demands for scalability, functionality as well as dependability in dealing with underwater sensor data. It involves comparing various approaches. For example traditional relational databases with time series extensions versus purpose built time series databases to figure out their relative advantages and disadvantages for Smart Ocean's use case.

The investigation is guided by the following research question and sub questions:

- **Main Research Question:** *Which database technology is most suitable for managing large scale time series data*
- **RQ1:** What are the key requirements and challenges in managing sensor data (e.g., data volume, frequency of data, query patterns, real time access needs, and deployment constraints)?
- **RQ2:** Which existing database systems (relational, NoSQL, and dedicated time series databases) are potential candidates for this task, and what are their expected advantages or limitations
- **RQ3:** How do selected candidate databases perform under ingestion rate, query performance, scalability, storage efficiency, and other relevant metrics?

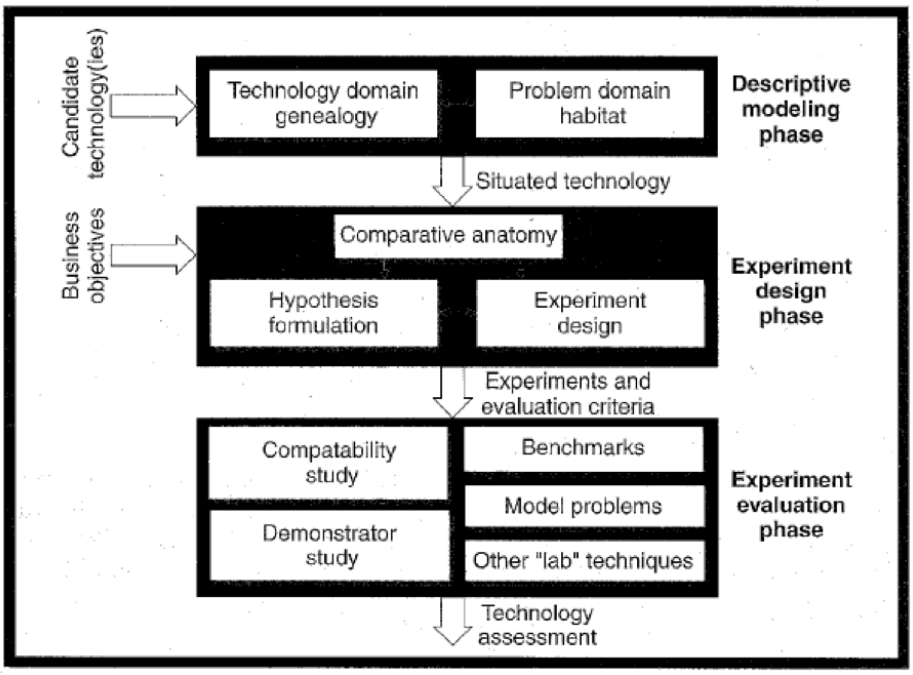

Fig. 1.1: Software technology evaluation framework.

1.3 Research Method

To answer the research questions, this thesis applies the evaluation methodology proposed by Brown and Wallnau[5]. Their framework provides a structured way to assess software technologies through the definition of measurable benchmarks or technology deltas, that highlight how one technology performs relative to another. The methodology is organized into three phases: descriptive modeling, experiment design, and experiment evaluation. Figure1.1 illustrates the overall structure of this framework.

The descriptive modeling phase consists of identifying the relevant features. This involves determining which requirements are most relevant for evaluation. These requirements then form the basis for defining performance benchmarks such as ingestion throughput, query latency, scalability, and storage efficiency.

The experiment design phase serves as the planning stage of the evaluation. In this phase, hypotheses are formulated about how different database technologies are expected to behave under representative workloads. Workload models and test harnesses are developed to simulate large scale time series data, ensuring that the experiments are both realistic and repeatable.

Finally, in the experiment evaluation phase, the selected databases are tested using the defined benchmarks. The resulting measurements are collected, analyzed, and compared to highlight the deltas between technologies. This process enables a systematic assessment of strengths, weaknesses, and trade offs, ultimately guiding conclusions about which database technologies are most suitable for handling time series data at scale.

1.4 Thesis Outline

The remainder of this thesis is organized as follows:

- **Chapter 2 Background and Related Work:** Provides the necessary background and context. Reviews time series data characteristics and challenges, and surveys existing database technologies for time series data management. Discusses related work and prior evaluations of database performance in IoT and sensor data scenarios.
- **Chapter 3 Design and Analysis:** Outlines the design of the experimental evaluation, including the selection of database technologies, the design of test workloads, and the metrics used for assessment.
- **Chapter 4 Implementation and Prototypes:** Describes the implementation of the experimental setup, including the deployment of database systems and the creation of test environments. Details the development of prototypes or models used for evaluation.
- **Chapter 5 Evaluation and Results:** Presents the empirical results of the comparative evaluation of database technologies. Includes ingestion throughput, query performance, storage efficiency, and other findings.
- **Chapter 6 Conclusion and Future Work:** Summarizes key findings and contributions, provides recommendations for the Smart Ocean platform, and suggests directions for future work.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 General Database Concepts

This particular section covers the basic ideas of databases, building on the relational model by Codd[8] and widely used database textbooks such as Silberschatz, Korth, and Sudarshan[41]. The understanding of relational as well as non relational database systems is based on these concepts.

2.1.1 Core Concepts

Database (DB): A database is an organized collection of data that is intended to be used. Instead of a randomly generated group of files, a database offers structure as well as relationships that allows for effective management and retrieval of information[41].

Database Management System (DBMS): The DBMS is the software layer that manages databases. It ensures efficient storage, safe access, and concurrent use, allowing multiple users or applications to interact with the data without conflict[35].

Schema / Data Model: The schema describes how the data is structured, such as tables in relational databases or collections in document oriented databases. It acts like a blueprint that defines the format and relationships of stored data.

Record / Row / Document / Node / Key-Value Pair: The record is the basic unit of data. Depending on the model, it may be a row in a table, a JSON document, a node in a graph, or a key-value entry.

Field / Attribute / Property: These are the smallest data elements, such as a column in a table, a field inside a document, or a property of a graph node.

Index: The DBMS maintains an index as an additional structure to facilitate more quickly searching. Much like a book index which helps locate pages quickly, a database index enables fast retrieval without requiring a full dataset scan. This increases performance but incurs additional storage and update costs[37].

Query: A query is an inquiry for information. For relational systems, Typically, this is expressed in SQL, but in non-relational systems it could be expressed using JSON syntax or API calls.

Transaction: A transaction is a collection of operations that are regarded as a single unit of work. It ensures that either all operations are successful or even none are applied, which is essential for preserving correctness of applications such as financial systems[22].

2.1.2 Relational Databases

Table (Relation): Data in relational databases is organized into tables (relations), which consist of rows and columns[8].

Row: Each row represents a single record or instance of data.

Column (Attribute): A named field that describes one aspect of the data in a table, such as "Name" or "Date of Birth."

Keys: Special fields that uniquely identify records or define relationships between them, such as primary and foreign keys.

Constraints: Rules enforced by the DBMS to maintain data integrity, such as `NOT NULL` if data must be present, `UNIQUE` if values must be distinct, or `CHECK` for custom conditions.

SQL (Structured Query Language): The standard language for defining and manipulating relational data.

Joins: Operations that combine data from multiple tables based on relationships, enabling richer queries across datasets.

Normalization / Denormalization: Techniques for organizing schema. Normalization reduces redundancy, while denormalization may intentionally duplicate data for performance gains.

2.1.3 NoSQL Databases

Key–Value Stores: Store data in keys and values in pairs. They are simple, fast, and scale very easily, usually used in real time systems and caching. Contemporary surveys describe how these systems are constructed to make use of higher throughput storage methods (such as flash media) and also deal with Big data volumes.[11].

Data: NoSQL databases often store data in semi-structured documents (e.g., JSON). Documents can have flexible fields, and indexes enable efficient search. Common use cases include web applications and mobile applications[18].

Column-Oriented Stores: Organize data by columns instead of rows, which allows efficient storage and retrieval for large datasets. Inspired by Google's Bigtable, they scale well for analytical workloads[6].

2.1.4 Querying and Processing

Querying and processing are central to the functionality of any database management system (DBMS). They determine how efficiently data can be retrieved, aggregated, and analyzed to generate meaningful insights. Modern databases employ declarative query languages, such as SQL, which allow users to specify how to obtain it. The DBMS then interprets, optimizes, and executes these queries using various internal mechanisms[41].

Aggregation: Aggregation operations such as `SUM`, `AVG`, and `COUNT` summarize groups of records into meaningful statistical values. These functions are fundamental to analytical workloads, providing insights into data distributions and trends. Aggregations are frequently combined with `GROUP BY` clauses or window functions for reporting, monitoring, and business intelligence purposes.

Query Optimizer: A DBMS component that decides how a query should be executed most efficiently, using cost based or rule based strategies[37].

2.1.5 Performance and Storage

Performance and storage mechanisms play a pivotal role in the design of modern database systems. Efficient query execution and data retrieval depend not only on algorithmic optimizations but also on how data is physically laid out, distributed, and accessed. Equally important is how storage is managed—techniques such as compression, partitioning, and caching directly impact throughput, latency, and resource usage[41].

Indexes: Structures maintained to make queries faster, such as B-trees or hash tables.

Sharding: Distributing data across servers to scale horizontally.

Replication: Storing copies of data across multiple machines for fault tolerance and availability.

Partitioning: Dividing data within a table by rows (horizontal) or columns (vertical) for scalability and manageability.

Compression: Techniques like dictionary or delta encoding to save space and speed up scans.

Caching: Temporarily storing results in memory to improve performance of frequently repeated queries.

2.2 Time Series Databases

Time series databases (TSDBs) are specialized systems designed to efficiently store, manage, and analyze data that is inherently ordered by time. Unlike general purpose databases, which treat time as just another attribute, TSDBs are built around the temporal dimension, making them well suited for workloads

such as sensor monitoring, financial transactions, or system metrics[4]. The motivation for their existence stems from the unique properties of time series data: it is often high volume, append only, and queried in large sequential ranges.

2.2.1 Core Properties of Time Series Databases

A defining feature of TSDBs is their ability to handle extremely high write throughput, as new measurements are continuously generated at fine time intervals. This creates append only workloads, where new entries are written sequentially rather than updating past records[2]. Efficient indexing strategies optimized for time enable queries that filter or aggregate over specific ranges, such as hourly averages or daily maxima.

Another core property is data lifecycle management. Since time series datasets grow rapidly, older measurements are often downsampled into coarser summaries (e.g., weekly or monthly averages) to save space while retaining long term trends[31]. Similarly, compression techniques take advantage of the temporal correlation between consecutive values, significantly reducing storage costs without sacrificing query performance.

2.3 Time Series Databases Advantages and Limitations

Time series databases (TSDBs) specialize the storage and processing stack around the temporal dimension in order to handle append heavy workloads, long retention, and range queries over contiguous time intervals efficiently[26]. In contrast to general purpose DBMSs, TSDB engines align file formats, compression, and indexing with time centric access patterns and provide built in operators for windowing, downsampling, and interpolation, which together yield substantial improvements in write throughput, storage footprint, and time range query latency in real world sensor/IoT and monitoring scenarios[25, 26].

This section synthesizes the main advantages of TSDBs—*compression*, *indexing*, and *scalability*—and then discusses *limitations* and a short *survey based synthesis*. The discussion draws on recent surveys and representative system/algorithm studies spanning compression techniques[7, 15, 24], similarity/indexing methods[16, 27, 28], and scalable TSDB architectures and deployments[14, 19, 46].

2.3.1 Compression

Compression is a first class concern in TSDBs because it reduces both storage costs and I/O, directly impacting scan and aggregation performance. A recent survey classifies time series compression into (a) dictionary based, (b) functional approximation (e.g., wavelets, Fourier, polynomials), (c) autoencoder-based neural methods, and (d) sequential/bit-level schemes (e.g., delta, delta-of-delta, XOR, RLE), highlighting the trade offs between compression ratio, reconstruction

error, and speed[7]. Lossy polynomial segmentation with explicit error bounds can yield extreme reductions while preserving analytics utility. Eichinger[15] reports piecewise regression with user chosen maximum deviation achieving up to several thousand fold reductions on smart grid data, without degrading forecasting beyond acceptable tolerances[15]. Conversely, lossless schemes such as Gorilla style timestamp/value encodings (delta-of-delta for timestamps; XOR for floating point values) preserve exactness and excel on smoothly evolving or repetitive signals; comparative benchmarks show that lossy Chebyshev polynomial compression can dominate on highly variable, weakly correlated series, while Gorilla tends to win when consecutive values are similar, underscoring that no single method is universally best[24]. Modern systems combine multiple codecs within a columnar/page format and choose adaptively per column/segment, integrating compression tightly with the storage engine and scan operators[25].

2.3.2 *Indexing*

Beyond time partitioning, TSDBs benefit from representations and indexes that accelerate similarity and subsequence search. Classic results established a transform then index paradigm: reduce dimensionality (e.g., DFT, DWT, PAA), then index coefficient vectors in a multidimensional structure with lower bounding distances (no false dismissals)[16]. Subsequence matching over arbitrary lengths can be supported by sliding windows that form *trails* in feature space; partitioning trails into minimum bounding rectangles enables efficient R*-tree search with large speedups over sequential scans[16]. For whole series search, locally adaptive piecewise constant approximation (APCA) permits variable length segments with tighter bounds than fixed size methods, and can itself be indexed; APCA delivered one to two orders of magnitude faster search than DFT/DWT/PAA on real datasets[27]. Shape oriented binning and pruning schemes further accelerate exact/approximate retrieval by grouping similar subsequences and exploiting best so far distances[28]. Collectively, these techniques exemplify a TSDB advantage: domain specific representations that are *both* compressible and indexable, yielding fast time series similarity/range queries at scale.

2.3.3 *Scalability*

TSDB scalability spans high velocity ingestion, long term retention, and low latency analytics over large temporal windows. Cloud native and cluster capable TSDBs partition series by device/time and replicate data for availability; ingestion paths favor append only LSM/columnar layouts with background compaction. Apache IoTDB demonstrates an integrated approach with a native time series file format, time aware encodings, and both edge and cloud deployments, supporting high throughput writes and sub second aggregations on billions of points while integrating with big data frameworks for offline analytics[46]. Empirical cloud benchmarks on industrial monitoring workloads show that

TSDBs backed by horizontally scalable stores (e.g., Cassandra) can achieve near linear scaling in write throughput and remain resilient under node failures, whereas alternatives may plateau or require heavy tuning[19]. At the application level, polyglot architectures that pair a TSDB for raw, compressed time series with an RDBMS for metadata/annotations improve end to end scalability and collaboration in large analysis platforms[14]. These results evidence another TSDB advantage: architectural fit for distributed, elastic environments with sustained ingestion and mixed historical/near real time analytics.

2.3.4 *Limitations*

Despite their strengths, TSDBs exhibit limitations that motivate ongoing research. Surveys note incomplete support for unified stream and batch processing and limited *native* real time analytics, often necessitating external stream processors[25, 26]. Heterogeneous, irregular, or multivariate series can complicate schema design and compression choices; cross series joins and ad-hoc relational predicates may be less efficient than in mature RDBMSs[26]. Compression introduces trade offs between ratio, CPU cost, and (for lossy methods) bounded error that must match application tolerance[7, 15]. Finally, many scalable TSDBs relax full ACID guarantees and strong consistency to prioritize availability and throughput, which can constrain transactional workloads[25, 26].

2.3.5 *Synthesis of Survey and Review Literature*

Comprehensive surveys consistently justify the specialization of TSDBs: general purpose DBMSs struggle with high rate ingestion and long range time scans at scale, whereas TSDBs align storage, indexing, and operators with temporal access patterns[25, 26]. They catalog architectural families (standalone engines, DBMS extensions, external store designs) and features (approximate queries, interpolation/forecasting operators, compression, retention), and identify open problems around elastic distributed designs, unified streaming/historical processing, and standardized interfaces[25]. Earlier reviews from a data mining perspective emphasize foundational issues of representation, indexing, similarity search, and visualization that continue to underpin modern TSDB capabilities and trade offs[17]. Overall, the literature positions TSDBs as the preferred substrate for time centric workloads, with clear advantages in compression, indexing, and scalability, tempered by limitations in heterogeneity handling, complex relational querying, and integrated real time analytics[7, 25, 26].

2.4 Database Technologies for Time Series Data

while working with this thesis, according to the DB Engines ranking[10], the most popular time series databases include InfluxDB, Kdb+, Prometheus, Graphite, TimescaleDB, Apache Druid, QuestDB, GridDB, DolphinDB, and TDengine. These systems differ in architecture, scalability, and cloud offerings, but all are

optimized for managing temporal workloads.

2.4.1 InfluxDB

InfluxDB is an open source TSDB optimized for high ingest and analytics. It supports retention policies, continuous queries, and both SQL like (InfluxQL) and functional (Flux) query languages. Cloud deployment is offered via InfluxDB Cloud[23].

2.4.2 Kdb+

Kdb+ is a columnar, in memory TSDB written in the q language. It is widely used in finance for ultra low latency analytics on tick data. It supports real time and historical tiers and scales through Kdb Insights on Kubernetes[30].

2.4.3 Prometheus

Prometheus is a Cloud Native Computing Foundation (CNCF) project focused on monitoring and alerting. It stores multi dimensional time series identified by labels, queried using PromQL. It is pull based (scraping) and integrates tightly with Kubernetes and Grafana[33].

2.4.4 Graphite

Graphite is an older TSDB that stores metrics in Whisper files. It provides simple retention based downsampling and integrates with StatsD and Grafana. It is suitable for smaller scale infrastructure monitoring[20].

2.4.5 TimescaleDB

TimescaleDB is a PostgreSQL extension with hypertables for automatic time partitioning. It supports continuous aggregates, native compression, and standard SQL. A fully managed cloud version is available[44].

2.4.6 Apache Druid

Apache Druid is a distributed columnar database optimized for real time OLAP queries on event streams. It partitions data by time and dimensions, enabling sub second analytics at scale[3].

2.4.7 QuestDB

QuestDB is a high performance open source TSDB with native SQL. It supports Influx line protocol ingestion and PostgreSQL wire compatibility, allowing millions of rows per second ingestion[34].

2.4.8 GridDB

GridDB, developed by Toshiba, is an open source TSDB optimized for IoT and big data. It uses a key container model with native time series containers and supports clustering for horizontal scalability and high availability. Both SQL like and NoSQL APIs are provided, making it adaptable to different application needs[45].

2.4.9 DolphinDB

DolphinDB is a distributed analytics platform with a built in TSDB engine. It combines stream and batch processing, supports hybrid row–column storage, and uses advanced compression (e.g., LZ4, Delta-of-delta). It is ACID-compliant, offers sub-millisecond queries, and integrates with tools like Python, R, and Grafana, making it suitable for finance and IoT[12].

2.4.10 TDengine

TDengine is an open source distributed TSDB optimized for IoT and the Industrial Internet of Things (IIoT). It provides super tables, SQL support, and cloud deployment with built in stream processing[43].

2.5 Benchmarking and Evaluation Approaches

Benchmarking and evaluating time series databases (TSDBs) is crucial for understanding their performance characteristics, scalability, and suitability for specific workloads. This section reviews common methodologies, metrics, and tools used in TSDB benchmarking, drawing on recent literature and industry practices.

2.5.1 General Database Benchmarking

2.6 Related Work

CHAPTER 3

DESIGN AND ANALYSIS

CHAPTER 4

IMPLEMENTATION AND PROTOTYPES

```
public static void main(String[] args) {

  int b, h, d;
  String btext, htext, dtext;

  [ ... ]

  int volum = b * h * d;

  String respons =
      "Volum [" + htext + "," + btext + "," + dtext + "] = " + volum;

}
```

CHAPTER 5

EVALUATION AND RESULTS

Table 5.1 gives an example of how to create a table.

Config	Property	States	Edges	Peak	E-Time	C-Time	T-Time
22-2	A	7,944	22,419	6.6 %	7 ms	42.9%	485.7%
22-2	A	7,944	22,419	6.6 %	7 ms	42.9%	471.4%
30-2	B	14,672	41,611	4.9 %	14 ms	42.9%	464.3%
30-2	C	14,672	41,611	4.9 %	15 ms	40.0%	420.0%
10-3	D	24,052	98,671	19.8 %	35 ms	31.4%	285.7%
10-3	E	24,052	98,671	19.8 %	35 ms	34.3%	308.6%

Table 5.1: Selected experimental results on the communication protocol example.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

BIBLIOGRAPHY

[1] ACM computing classification system. ACM Digital Library website, 2012. [Data set; ontology]. Retrieved from https://dl.acm.org/ccs, accessed 2025-08-28. 1

[2] M. P. Andersen and D. E. Culler. BTrDB: Optimizing storage system design for timeseries processing. In *FAST '16: 14th USENIX Conference on File and Storage Technologies*, pages 39–52. USENIX Association, 2016. Available at: https://www.usenix.org/conference/fast16/technical-sessions/presentation/andersen (accessed: 05.09.2025). 2.2.1

[3] Apache Software Foundation. Apache druid documentation, 2025. Available at: https://druid.apache.org/docs/ (accessed: 16.09.2025). 2.4.6

[4] A. Bader, O. Kopp, and M. Falkenthal. Survey and comparison of open source time series databases. In *BTW 2017 – Workshopband*, volume P-266 of *Lecture Notes in Informatics (LNI)*, pages 249–268. Gesellschaft für Informatik (GI), 2017. Available at: https://dl.gi.de/items/1d3c3186-8d50-4b51-b097-7695027076fd (accessed: 05.09.2025). 2.2

[5] A. W. Brown and K. C. Wallnau. A framework for evaluating software technology. *IEEE Software*, 13(5):39–49, 1996. Available at: https://doi.org/10.1109/52.536457 (accessed: 26.08.2025). 1.3

[6] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2006)*, pages 205–218. USENIX Association, 2006. Open access on USENIX: https://www.usenix.org/conference/osdi-06/bigtable-distributed-storage-system-structured-data (accessed: 04.09.2025). 2.1.3

[7] G. Chiarot and C. Silvestri. Time series compression survey. *ACM Comput. Surv.*, 55(10), Feb. 2023. 2.3, 2.3.1, 2.3.4, 2.3.5

[8] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970. Available at: https://dl.acm.org/doi/10.1145/362384.362685 (accessed: 30.08.2025). 2.1, 2.1.2

[9] Copernicus Marine Service. Why ocean health matters for food security and human health. Available at: https://marine.copernicus.

eu/explainers/why-ocean-important/food-security. (accessed: 27.08.2025). 1

[10] DB-Engines. Db-engines ranking of time-series dbms, 2025. Available at: https://db-engines.com/en/ranking/time+series+dbms (accessed: 16.09.2025). 2.4

[11] K. Doekemeijer and A. Trivedi. Key-value stores on flash storage devices: A survey. *ArXiv preprint arXiv:2205.07975*, 2022. Available at: https://arxiv.org/abs/2205.07975 (accessed: 04.09.2025). 2.1.3

[12] DolphinDB Inc. Dolphindb documentation, 2025. Available at: https://docs.dolphindb.com/en/about_dolphindb.html (accessed: 16.09.2025). 2.4.9

[13] F. P. F. Domingos et al. Underwater communication systems and their impact on aquatic ecosystems. *Electronics*, 14(1):7, 2024. Available at: https://www.mdpi.com/2079-9292/14/1/7 (accessed: 27.08.2025). 1

[14] E. Duarte, D. Gomes, D. Campos, and R. L. Aguiar. Distributed and scalable platform for collaborative analysis of massive time series data sets. In *Proceedings of the 8th International Conference on Data Science, Technology and Applications (DATA)*, pages 141–152, Praia de Rocha, Portugal, 2019. SCITEPRESS - Science and Technology Publications. 2.3, 2.3.3

[15] F. Eichinger, M. Karnstedt, A. Fuchs, S. Plantikow, and K. Böhm. A time-series compression technique and its application to the smart grid. *The VLDB Journal*, 24(2):193–218, 2015. 2.3, 2.3.1, 2.3.4

[16] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, SIGMOD '94, page 419–429, New York, NY, USA, 1994. Association for Computing Machinery. 2.3, 2.3.2

[17] Garima and S. Rani. Review on time series databases and recent research trends in time series mining. In *2014 5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence)*, pages 109–115, 2014. 2.3.5

[18] F. Gessert, W. Wingerath, S. Friedrich, and N. Ritter. Nosql database systems: A survey and decision guidance. *Computer Science – Research and Development*, 32(3–4):353–365, 2017. Available at: https://link.springer.com/article/10.1007/s00450-016-0334-3 (accessed: 04.09.2025). 2.1.3

[19] T. Goldschmidt, A. Jansen, H. Koziolek, J. Doppelhamer, and H. P. Breivold. Scalability and robustness of time-series databases for cloud-native monitoring of industrial processes. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 602–609, 2014. 2.3, 2.3.3

[20] Graphite Project. Graphite documentation, 2025. Available at: https://graphite.readthedocs.io/ (accessed: 16.09.2025). 2.4.4

[21] P. Grzesik and D. Mrozek. Comparative analysis of time series databases in the context of edge computing for low power sensor networks. In *Computational Science - ICCS 2020: 20th International Conference, Proceedings, Part V*, volume 12141 of *Lecture Notes in Computer Science*, pages 371–383. Springer, 2020. Available at: https://link.springer.com/chapter/10.1007/978-3-030-50426-7_28 (accessed: 26.08.2025).

[22] T. Härder and A. Reuter. Principles of transactionoriented database recovery. *ACM Computing Surveys*, 15(4):287–317, 1983. Available at: https://dl.acm.org/doi/10.1145/289.291 (accessed: 04.09.2025). 2.1.1

[23] InfluxData. Influxdb documentation, 2025. Available at: https://docs.influxdata.com/ (accessed: 16.09.2025). 2.4.1

[24] O. Iqbal and R. B. Keskar. Techniques to compress time-series data. In *2021 10th International Conference on Power Science and Engineering (ICPSE)*, pages 56–60, 2021. 2.3, 2.3.1

[25] S. Jensen, T. Pedersen, and C. Thomsen. *Time Series Management Systems: A 2022 Survey*. Association for Computing Machinery (ACM), United States, Dec. 2022. 2.3, 2.3.1, 2.3.4, 2.3.5

[26] S. K. Jensen, T. B. Pedersen, and C. Thomsen. Time series management systems: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 29(11):2581–2600, 2017. 1.1.2, 2.3, 2.3.4, 2.3.5

[27] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, SIGMOD '01, page 151–162, New York, NY, USA, 2001. Association for Computing Machinery. 2.3, 2.3.2

[28] E. Keogh and M. Pazzani. An indexing scheme for fast similarity search in large time series databases. In *Proceedings. Eleventh International Conference on Scientific and Statistical Database Management*, pages 56–67, 1999. 2.3, 2.3.2

[29] M. Kleppmann. A critique of the cap theorem. *ArXiv preprint arXiv:1509.05393*, 2015. Available at: https://arxiv.org/abs/1509.05393 (accessed: 04.09.2025).

[30] Kx Systems. Kdb+ documentation, 2025. Available at: https://code.kx.com/q/ (accessed: 16.09.2025). 2.4.2

[31] J. Mostafa, S. Wehbi, S. Chilingaryan, and A. Kopmann. SciTS: A benchmark for time-series databases in scientific experiments and industrial internet of things. *arXiv*, 2022. Available at: https://arxiv.org/abs/2204.09795 (accessed: 05.09.2025). 2.2.1

[32] D. Pritchett. Base: An acid alternative. *ACM Queue*, 6(3):48–55, 2008. Available at: https://dl.acm.org/doi/10.1145/1394127.1394128 (accessed: 04.09.2025).

[33] Prometheus Authors. Prometheus documentation, 2025. Available at: https://prometheus.io/docs/introduction/overview/ (accessed: 16.09.2025). 2.4.3

[34] QuestDB Team. Questdb documentation, 2025. Available at: https://questdb.io/docs/ (accessed: 16.09.2025). 2.4.7

[35] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, 3rd edition, 2003. Available at: https://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/ (accessed: 04.09.2025). 2.1.1

[36] Research Council of Norway. SFI Smart Ocean - prosjektbanken. https://prosjektbanken.forskningsradet.no/project/FORISS/309612, 2020. (accessed: 26.08.2025). 1

[37] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, pages 23–34. ACM, 1979. Available at: https://dl.acm.org/doi/10.1145/582095.582099 (accessed: 04.09.2025). 2.1.1, 2.1.4

[38] SFI Smart Ocean. About sfi smart ocean. https://sfismartocean.no/about/. (accessed: 26.08.2025). 1

[39] SFI Smart Ocean. Sfi smart ocean hub. https://smartoceanplatform.github.io/. (accessed: 26.08.2025). 1.1.1

[40] B. Shah, P. M. Jat, and K. Sashidhar. Performance study of time series databases. *arXiv*, 2022. Available at: https://arxiv.org/abs/2208.13982 (accessed: 05.09.2025).

[41] A. Silberschatz, H. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill, 7th edition, 2020. Available at: https://www.db-book.com/ (accessed: 04.09.2025). 2.1, 2.1.1, 2.1.4, 2.1.5

[42] A. M. Skålvik, C. Sætre, K.-E. Frøysa, R. N. Bjørk, and A. Tengberg. Challenges, limitations, and measurement strategies to ensure data quality in deep-sea sensors. *Frontiers in Marine Science*, 10, 2023. Available at: https://www.frontiersin.org/articles/10.3389/fmars.2023.1152236/full (accessed: 27.08.2025). 1

[43] TDengine Authors. Tdengine documentation, 2025. Available at: https://docs.tdengine.com/ (accessed: 16.09.2025). 2.4.10

[44] Timescale. Timescaledb documentation, 2025. Available at: https://docs.timescale.com/ (accessed: 16.09.2025). 2.4.5

[45] Toshiba. Griddb documentation, 2025. Available at: https://docs.griddb.net/ (accessed: 16.09.2025). 2.4.8

[46] C. Wang, X. Huang, J. Qiao, T. Jiang, L. Rui, J. Zhang, R. Kang, J. Feinauer, K. A. McGrail, P. Wang, D. Luo, J. Yuan, J. Wang, and J. Sun. Apache iotdb: time-series database for internet of things. *Proc. VLDB Endow.*, 13(12):2901–2904, Aug. 2020. 2.3, 2.3.3

APPENDIX A

SOURCE CODE

Where to find the source code . . . (if applicable)

APPENDIX **B**

RESEARCH DATA

Where to find the research data . . . (if applicable)

List of Figures

List of Tables