Master of Science in Computer Science Engineering
May 2023

Prioritized Database Synchronization using Optimization Algorithms

Sai Sumeeth Alladi

Faculty of Faculty, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Faculty at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Computer Science Engineering. The thesis is equivalent to Weeks weeks of full time studies.

The authors declare that they are the sole authors of this thesis and that they have not used any sources other than those listed in the bibliography and identified as references. They further declare that they have not submitted this thesis at any other institution to obtain a degree.

Contact Information:
Author(s):
Sai Sumeeth Alladi
E-mail: saai21@student.bth.se

University advisor:
Hüseyin Kusetogullari
Department of Department of Computer Science

Faculty of Faculty
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Background. Database synchronization is an essential process due to the large amounts of data produced in today's industrial environment. Important data must be prioritized for synchronization in real time, and optimization algorithms can be used to produce a prioritized list of data based on an optimization objective. To prevent data loss and minimize data consumption for efficient synchronization, a synchronization framework with a prioritization policy is an effective solution.
Objectives. This research aims to develop an efficient synchronization framework with optimization algorithms that can obtain the optimal order of data for synchronization and identify the most suitable optimization algorithm for the objective. The proposed solution aims to reduce the burden on the On-Board (source) database and limit data consumption.
Methods. The research experiment was conducted on two optimization algorithms to identify the most suitable algorithm for the objective and determine the appropriate parameters. The algorithms were performed on a dataset provided by Volvo, and the synchronization time of rows (documents) across different networks was also measured.
Results. The results of the experiments indicate that the genetic algorithm and particle swarm optimization can generate a high-quality solution for the given objective of minimizing the data consumption during the synchronization in terms of fitness values and convergence speed. The genetic algorithm outperformed particle swarm optimization, and the synchronization framework can effectively synchronize data while handling network errors and concurrent updates.
Conclusions. Based on the experiments, it can be concluded that the genetic algorithm is the most effective optimization algorithm for this optimization objective, and the proposed synchronization framework works effectively. Further evaluation and testing are necessary to ensure industrial usage. The framework can be deployed to synchronize important data trucks produce on board.

Keywords: Database Synchronization, Optimization Algorithms, Prioritization, MongoDB Atlas

Acknowledgments

I want to thank everyone who supported me during my thesis. Firstly, I am grateful to my university supervisor, Hüseyin Kusetogullari, for his guidance, feedback, and support throughout the thesis. I would also like to thank my friends for their constant encouragement and support. I am grateful to Volvo and my company supervisor for allowing me to undertake this thesis. Moreover, I want to thank my family for their love and support throughout my academic journey. Lastly, I would like to thank God for guiding me, giving me strength and wisdom, and for the opportunity to undertake this thesis.

I am genuinely grateful for everything you have done for me. Your support and encouragement have meant the world to me.

Contents

List of Figures

List of Tables

Nomenclature

CDC Change Data Capture

CXPB Crossover probability

GA Genetic Algorithm

GNSS Global Navigation Satellite System

GUI Graphical User Interface

LAMBDA Offspring size

MUTB Mutation probability

MU Population size

NGEN Number of generations

OCC Optimistic Concurrency Control

Off-Board Database Destination Database

On-Board Database Source Database

PSO Particle Swarm Optimization

Chapter 1

Introduction

With the growing number of mobile devices, a lot of data is eventually produced. This data is stored in the device's local database. However, these devices have limited memory, less computational power, and less network bandwidth, making storing and processing this data more difficult [3]. This data can be valuable sometimes and needs to be stored in a database that stores, learns, or performs any actions. Database synchronization is maintaining data consistency between two or more databases while keeping the data clean and avoiding duplications or replications. Several commercial database synchronization frameworks were proposed for mobile databases and are already in use. The major problem with commercial database vendors providing the synchronization is that this forces the developers to use specific libraries and resources from the same vendor for any further development; this problem has been highlighted in many papers [3, 6, 8, 9].

This research focuses on synchronizing only prioritized data between two databases (On-Board and Off-Board) effectively. This thesis work was conducted in partnership with Volvo. The remote database or the on-board here is a truck equipped with a telemetry device, and the backend database is a database at the off-board. Data is eventually generated from the truck's telemetry devices; this data is stored in the local database and must be synchronized with the backend databases at Volvo. Also, there exists data that is more important and requires immediate synchronization than other generated data. So, it is necessary to prioritize what data to send first while synchronizing at the same time, minimizing the data consumption.

Wireless networking, as we speak, is evolving rapidly. Yet, a few problems are associated with it [21], like shaky connections, high-cost mobile data, etc. Since there are problems like this, we also need to set a prioritization policy for the synchronization depending on the network connection. A current study needs a synchronization framework that prioritizes the order of data to synchronize between the databases. We explore this gap and develop a synchronization framework that supports data prioritization while synchronizing the databases.

In this thesis, we implement optimization algorithms used for similar purposes in the industry to prioritize the data based on a defined prioritization policy and a framework for synchronizing the changed data between the databases. The synchronization framework must synchronize high-priority changed data rather than the random order of data or the first produced data because it will reduce the chances

of synchronizing essential data and consume more data. Keeping these in mind, the scope and objectives of the thesis are identified and described below.

1.1 Aim and objectives

This research aims to review the current literature on optimization algorithms used for similar problems and apply them to our research. Additionally, to create a framework for synchronization. The objectives of this study are,

1. Perform a literature review to understand the existing literature on optimization algorithms and select the algorithms that work to generate prioritized data according to the set synchronization policy.

2. To set up databases and create a synchronization framework to synchronize the changed data.

3. Integrating the optimization algorithm with the synchronization framework and evaluating it.

4. Create a simple GUI for the framework.

1.2 Research questions

1. **RQ1:** What are the existing optimistic algorithms and their applications for prioritization in various domains?
   Justification: Understanding the existing literature can potentially help the research, and the Literature Review enables us to get the knowledge and identify the methods for the problem and how to solve it.

2. **RQ2:** How do the different parameter settings affect the performance of the selected optimized algorithms in terms of finding the optimal solution?
   Justification: It is essential to evaluate the effectiveness of the devised framework. Comparing the selected algorithms with different parameter settings to further identify which algorithm produces better results is beneficial.

3. **RQ3** How well is the devised prioritized database synchronization framework working?
   Justification: Evaluating the synchronization framework performance is important as it identifies the framework's efficiency and the potential improvement areas.

1.3 Outline

To ensure a flow, this section outlines the main structure of this document as follows:

- **Chapter 1**: This chapter introduces the research topic. This section discusses the research's purpose, aim, and research questions.
- **Chapter 2**: This chapter provides the background and information required to understand the research from a reader's perspective.
- **Chapter 3**: This chapter provides the related work of the existing literature by other researchers on database synchronization and optimization algorithms.
- **Chapter 4**: This chapter provides the research methods used to answer the research questions and a detailed discussion on the approach to carry out the chosen research methods.
- **Chapter 5**: This chapter provides a detailed view of the synchronization framework devised, including the Graphical User Interface (GUI).
- **Chapter 6**: This chapter provides the findings from the results and analysis, issues faced when conducting the experiments, and validity threats.
- **Chapter 7**: This chapter provides a discussion of the results and the thesis in a more elaborate way
- **Chapter 8**: This chapter provides the conclusions on this research and potential future work opportunities for this research study.

Chapter 2

Background

In this section, we discuss the optimization algorithms and the synchronization framework used in the research,

2.1 Database Synchronization

Database synchronization is a critical and essential process in today's database environment. The increasing importance of data in various industries and organizations has made it essential to have backup systems in place and ensure speedy retrieval of data in case of database malfunctions. Real-time data synchronization is necessary for tasks such as data mining, data analysis, application support, and so on at the back end [16].

Database synchronization refers to the process of ensuring that two or more databases are always in sync with each other. In the modern-day environment, maintaining identical copies of data in multiple databases is often necessary, which may be located in different systems or locations. Synchronization ensures the data storage system's high availability, fault tolerance, and scalability.

2.1.1 Database Synchronization methods

There are several methods to achieve database synchronization, each with its advantages and limitations.

2.1.1.1 Master-slave

The most commonly used method is **master-slave**, where one database acts as the master or source database, and the other databases act as slaves or replicas [39]. The master database is responsible for updating the data, while the replicas receive updates from the master and apply them locally. This approach ensures that all databases have the same data, but there may be a delay in replicating updates to the replicas.

2.1.1.2 Master-Master

Another approach is **master-master** or **multi-master**, where each database acts as both a master and a replica [7]. This method provides better fault tolerance

and scalability, but it requires careful management of conflicts that may arise due to simultaneous updates on different databases. In this research, we follow master-master database synchronization. The process of database synchronization can be challenging due to the complexity of the data and the frequency of updates. It may also involve resolving conflicts between different updates or dealing with failures or disruptions in the network.

2.1.2 Synchronization using MongoDB Atlas

MongoDB Atlas is a cloud NoSQL database management system designed to support a wide range of use cases, integrating large amounts of data and Supporting hybrid and multi-cloud applications [17, 24]. One such use case is database synchronization, which involves maintaining consistent data across multiple databases. MongoDB offers various methods and techniques, including built-in features, to accomplish this goal.

2.1.2.1 Change Data Capture(CDC)

Change Data Capture is a real-time technique to capture modified data [22]. Change Streams is a feature in MongoDB that enables databases to receive real-time notifications for changes made to a collection. To create a Change Stream for a specific collection in MongoDB, the **watch()** method is used. This method returns a cursor that can iterate over the real-time changes as they occur. The changes in the source database collection (optimized) will be captured and streamed/sent as full documents to the destination database, as shown in Figure 2.1.

2.1.2.2 Connection Failures

In this synchronization framework, we use a method to handle network connectivity issues that may happen while synchronizing the source and destination databases. When there is a connection failure, the method waits for a set amount of time (starting at 1 second) and then tries to reconnect to the database. If the connection is not established, the method doubles the waiting time and tries to connect again, as shown in Figure 2.2. This continues until a connection is made or the maximum number of retries is reached, and once the connection is re-established, the synchronization starts.

It is important to have a connection retry mechanism during synchronization because network connectivity can be inconsistent. The connection may be lost or unstable, but the exponential time delay retry mechanism ensures that the synchronization process continues without interruption when the connection is re-established. Figure 2.2 depicts the whole process of how a connection failure will be handled.

2.1.2.3 Optimistic Concurrency Control (OCC)

Database management systems use *Optimistic Concurrency Control (OCC)* to manage concurrent access to data. In this research, OCC ensures that modifications made to documents in the source collection are properly synchronized to the destination

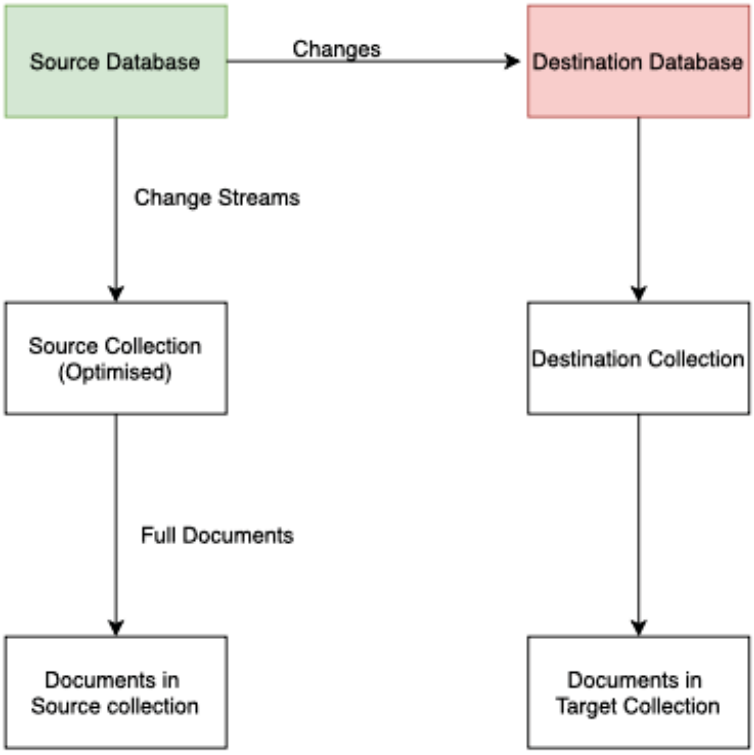

Figure 2.1: Change Data Capture

collection, even when concurrent modifications/updates occur.

When a document is updated in the source database, the framework adds a version number to the document and tries to update the corresponding document in the destination database. If the document's version number in the destination database is the same as that of the updated document in the source database, the update is successful. However, the document's version number in the destination database is different. In that case, another process has concurrently updated the document, and the framework must manage this situation accordingly.

Similarly, when a document is updated in the destination database, the updated document's version number is compared to the corresponding document in the source database. Figure 2.3 depicts this whole process of OCC; the arrows represent the acknowledgements between the databases.

2.2 Optimization Algorithms

Optimization algorithms are utilized in multiple fields, including engineering, finance, and computer science, to determine the most suitable solution to a problem. The primary objective of an optimization algorithm is to identify the ideal values of one or more variables that meet specific constraints and minimize or maximize an objective function.

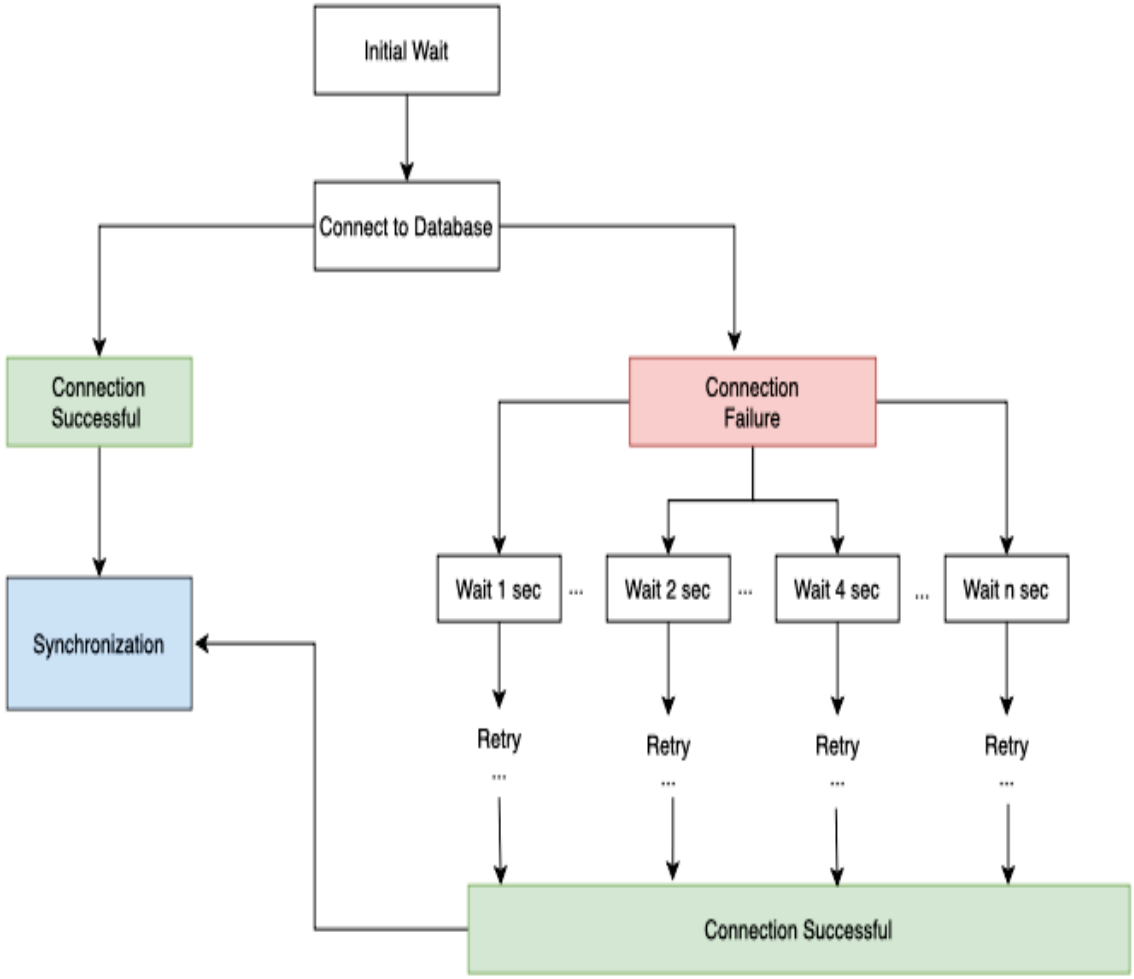

Figure 2.2: Connection Failure Handling

The Genetic Algorithm (GA) is a popular optimization method that draws on the principles of natural selection and genetics. With GA, a group of potential solutions evolves over generations through crossover, mutation, and selection operators, ultimately leading to improved solutions. GA has been utilized in several fields, such as engineering, finance, and biology, to tackle optimization problems [42].

Particle Swarm Optimization (PSO) is a widely used optimization algorithm inspired by the social behaviour of birds and animals. This method involves moving a population of particles around in the search space, guided by both their own best position and the best position found by the group, to find the optimal solution. PSO has effectively solved optimization problems across various fields, such as engineering, economics, and transportation.

There are several optimization algorithms available, such as Ant Colony Optimization (ACO), Simulated Annealing (SA), and Tabu Search (TS). ACO takes inspiration from the behaviour of ants and aims to find the shortest path between their nest and food source. This algorithm has been applied in different transportation and telecommunications sectors to solve optimization problems. On the other hand,

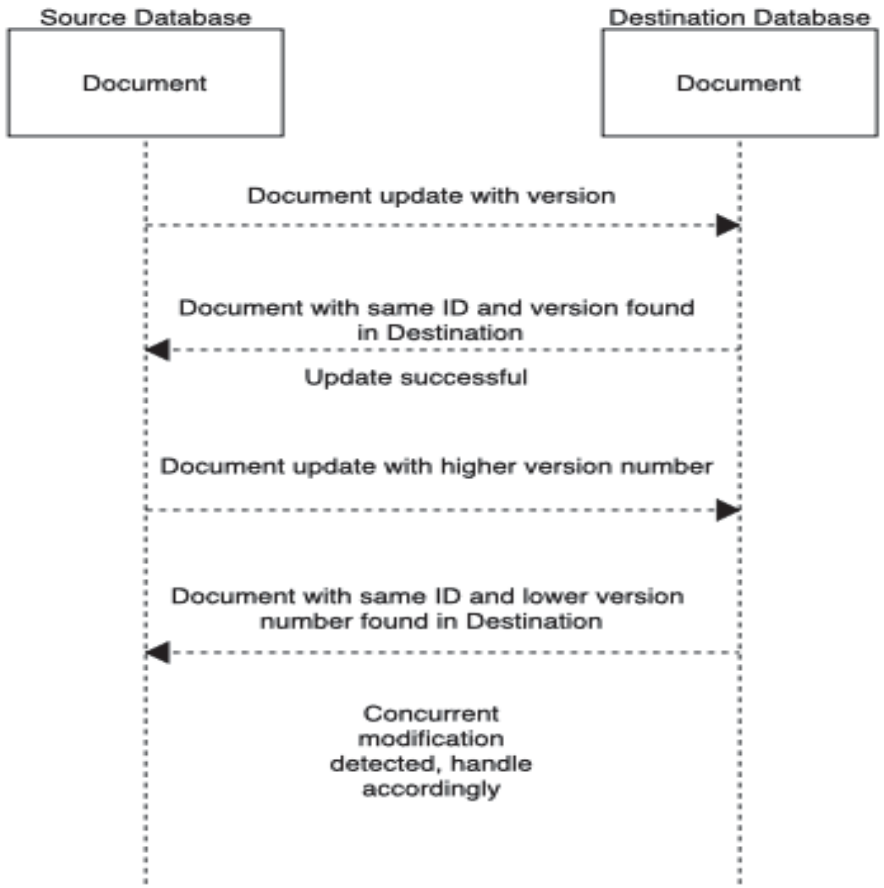

Figure 2.3: Optimistic Concurrency Control

Simulated Annealing is inspired by the process of annealing in metals and has been used in computer science and physics to solve optimization problems. Finally, TS is a local search algorithm that maintains a tabu list of previously visited solutions and has been used in manufacturing and logistics to solve optimization problems [10].

In conclusion, Optimization algorithms are extensively utilized in different industries to tackle intricate optimization issues. Selecting a suitable optimization algorithm depends on the problem and available resources. GA, PSO, ACO, SA, and TS have frequently employed optimization algorithms in diverse industries to address optimization problems [26].

Figure 2.4 represents some of the optimization algorithms divided and inspired by various factors.

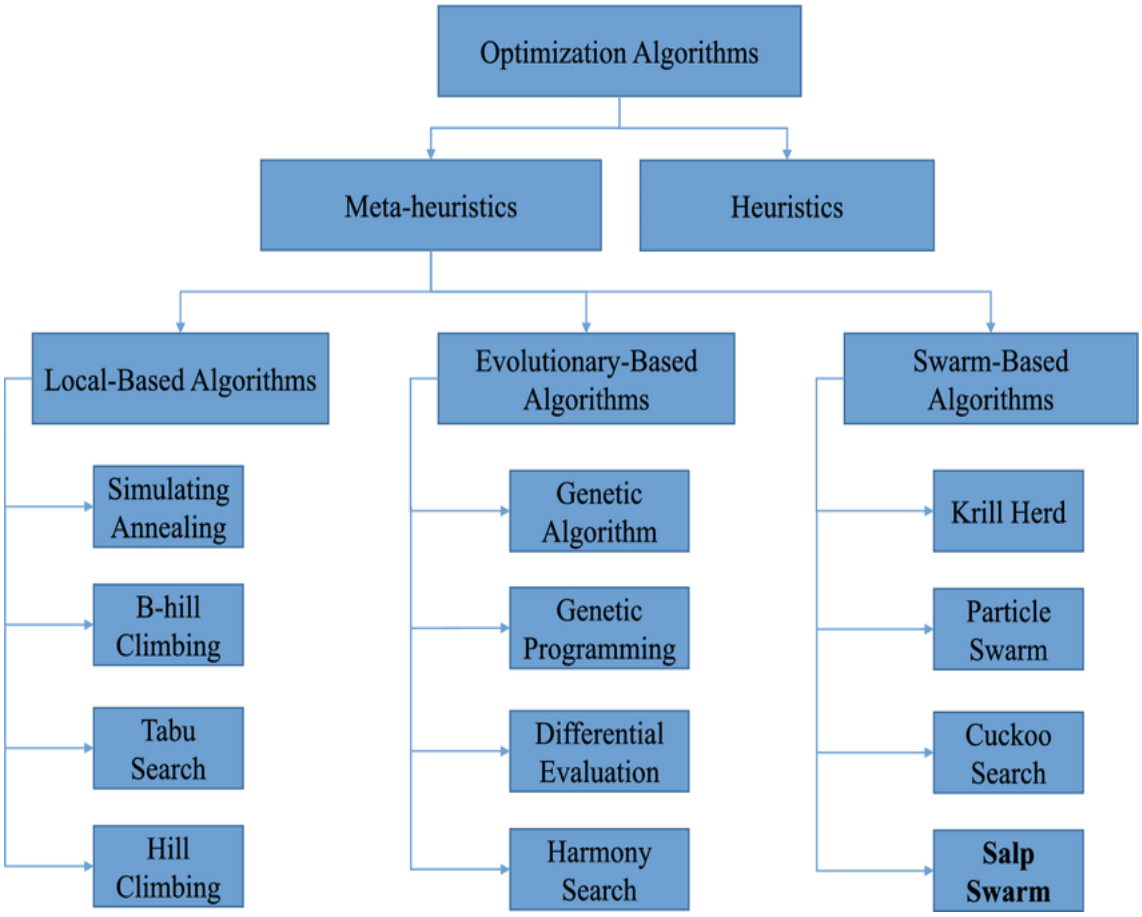

Figure 2.4: Optimization algorithms [1]

2.2.1 Genetic Algorithm (GA)

The Genetic Algorithm (GA) is an optimization algorithm that utilizes the principles of natural selection and genetics. It is a highly effective algorithm applied in various fields to resolve intricate optimization problems.

In the process of GA, a group of potential solutions go through evolution over many generations by employing selection, crossover, and mutation operators. Every solution within the population is a possible answer to the optimization problem..

In GA, a **solution** comprises a chromosome containing a string of genes. Each gene represents a specific variable or parameter related to the solution, and the value of each gene determines a different trait or feature of the solution.

The population in GA includes multiple chromosomes (or individuals), each representing a possible solution to the problem. The fitness function assesses the quality of each individual in the population by assigning a fitness value based on how well it solves the problem.
In GA, the **selection operator** picks the fittest individuals from the population to be the next generation's parents. These chosen individuals undergo crossover and mutation operations to create new offspring [37].

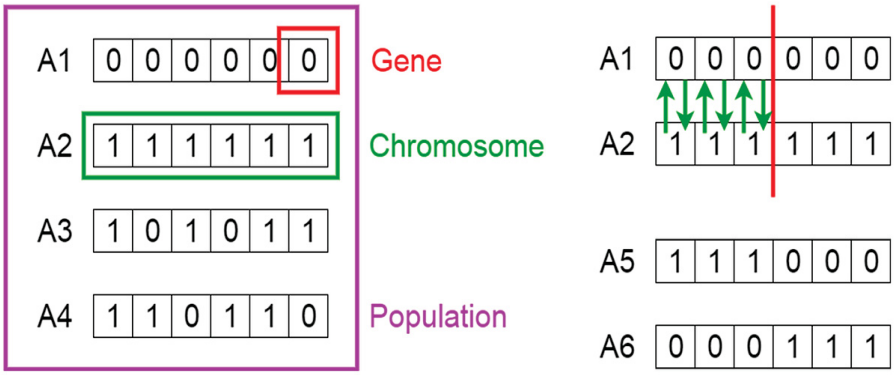

Figure 2.5: Gene, Chromosome and Population
[27]

Figure 2.5 represents the fundamental concepts of genetic algorithm such as gene, chromosome and population. In GA, the **crossover operator** combines two or more parents to produce new offspring. This method is applied to the selected parents to create new solutions. Some commonly used crossover methods in GA include one-point crossover, two-point crossover, and uniform crossover [34].

In GA, the **mutation operator** maintains genetic diversity by introducing random changes in the offspring. This mutation helps to explore new areas in the search space and avoid premature convergence. The mutation is only applied to the offspring with a low probability. Bit-flip, swap, and inversion mutation are commonly used methods for mutation in GA [42].

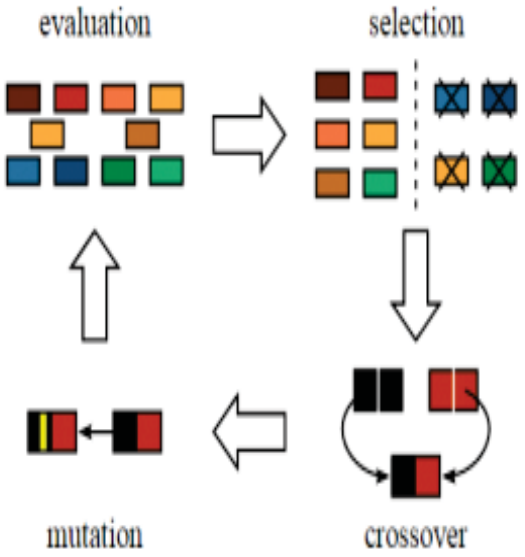

Figure 2.6: Genetic Operators
[31]

The GA algorithm repeats selection, crossover, and mutation operators to generate improved solutions over multiple generations. The algorithm stops running once a predetermined condition is met, such as finding an acceptable solution or reaching a specific number of generations. Figure 2.6 is a visual representation of the working of the different genetic operators.

In conclusion, GA has been successfully utilized in various fields, including engineering, finance, and biology, to address optimization problems. In summary, GA is a robust optimization algorithm that uses selection, crossover, and mutation operators to develop a population of possible solutions. Critical concepts in GA include genes, chromosomes, and populations. Commonly used methods include tournament selection, roulette wheel selection, one-point crossover, two-point crossover, uniform crossover, bit-flip mutation, swap mutation, and inversion mutation.

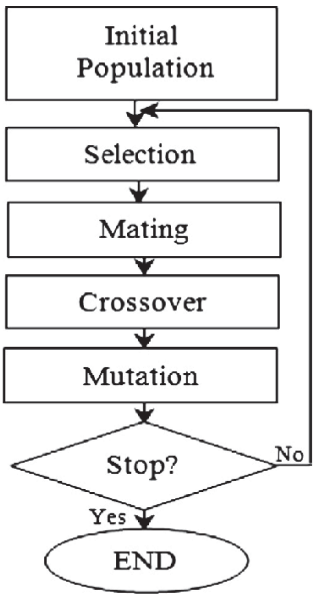

Figure 2.7: Implementation of genetic algorithm
[9]

2.2.2 Particle Swarm Optimization (PSO)

PSO is an optimization algorithm that imitates the social behaviour of birds and other animals. It works by moving a group of particles around in the search space, guided by their own best position and the best position found by the group. The algorithm has two main components: position update and velocity update.

In PSO, The process of **position update** updates the position of each particle based on its current position, velocity, and personal and global best positions. The update equation determines the new position of each particle by taking a weighted sum of the current position, personal best position, and global best position of the particle. In PSO, this equation is usually used to calculate the new position of each particle [41].

In PSO, **Velocity update** is a process that updates the velocity of each particle based on its current velocity and the difference between its personal and global best positions. The velocity update equation determines the new velocity of each particle based on the current velocity and the difference between its personal and global best positions. The velocity update equation is usually a weighted sum of the current velocity, the personal best position, and the global best position of the particle [29].

The PSO algorithm uses multiple iterations to update the position and velocity, ultimately finding the ideal solution. The algorithm stops when it meets a stopping criterion, such as reaching a set number of iterations or discovering a satisfactory solution.

PSO has been successful in optimizing engineering, economics, and transportation problems. In an Elsevier study, PSO was used to reduce costs in hybrid renewable energy systems, demonstrating promising outcomes.

In conclusion, PSO is a powerful optimization algorithm widely used in various fields to solve complex optimization problems. The algorithm moves a population of particles around in the search space, guided by their own best position and the best position found by the group, using position and velocity updates. Weighted sum, personal best position, and global best position are commonly used components in PSO. Figure 2.8 shows the complete implementation of the particle swarm optimization algorithm with the process as explained above.

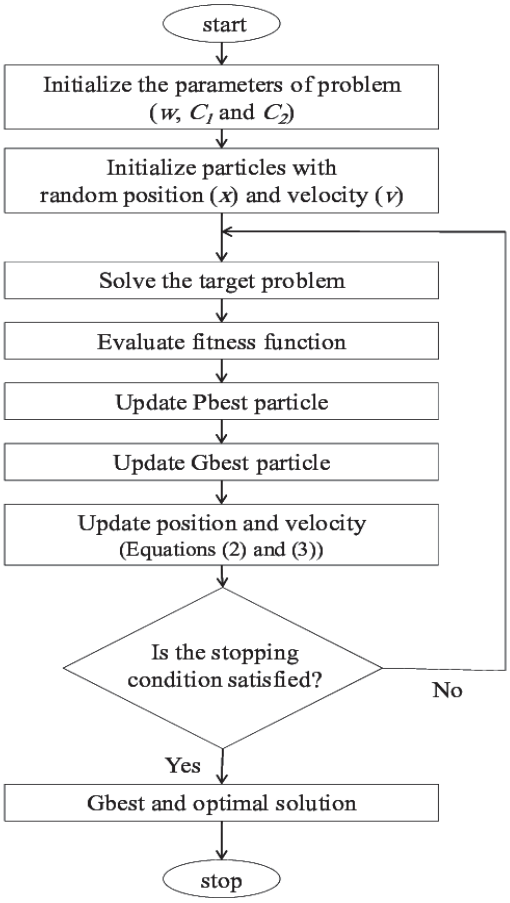

Figure 2.8: Implementation of Particle Swarm Optimization [43]

2.2.3 Fitness function

In optimization problems, the fitness function is a crucial component that evaluates the quality of a potential solution. The fitness function determines how well a solution meets the optimization/prioritization objective. Optimization algorithms aim to find the best solution(s), and the fitness function helps to find the best solution from the population for the given problem [23].

When trying to solve optimization problems, fitness functions can take various forms. They can be straightforward mathematical formulas, intricate simulations, or a combination of multiple criteria. Additionally, fitness functions can be static or dynamic, depending on whether the fitness value changes with time.

To ensure optimal results, it is crucial that the fitness function accurately reflects the optimization objective. A poorly designed fitness function can lead to suboptimal solutions or premature convergence. This is why it is essential to carefully design and validate the fitness function before using it in an optimization algorithm.

Chapter 3

Related Work

In the following, we emphasize the existing literature on database synchronization. These papers also describe the different database synchronization approaches.

We have reviewed several research papers to examine past efforts in database synchronization. It is important to note that while a considerable amount of research has been done on this topic, studies have yet to explore the prioritization of data synchronization between the source side and the destination side.

Mi-Young Choi et al. [8] emphasized the need for database synchronization. They proposed an algorithm called SAMD (Synchronization Algorithms based on Message Digest) for data synchronization between a server-side database and a mobile database. The SAMD algorithm uses message digest tables to compare two images and select the rows needed for synchronization. The proposed algorithm does not use any techniques that are dependent on specific database vendors, stored procedures, triggers or timestamps for picking up changed data; instead, it uses standard SQL functions for synchronization. However, in the paper, the algorithm uses a wired connection for data transmission. This paper has laid a foundation regarding the general framework of database synchronization.

Taqwa A et al. [3] analysed the literature on database synchronization. They discussed previously proposed solutions to the database synchronization problems, which included ISAMD [6], which uses standard SQL queries and does not require to compute message digest values of the databases. They propose a synchronization wireless algorithm based on message digest (SWAMD) to assist data synchronization between server-side and mobile device database. For picking up changed data, SWAMD also used message digest values to compare two images and spot the rows that need to be synchronized. They followed the same approach of [6] for pinpointing the rows for synchronization; however, their proposed algorithm is a wireless synchronization algorithm. They did a performance evaluation of SWAMD and concluded that it outperforms other synchronization algorithms in terms of network speed and execution time.

Madhu Ahluwalia et al. [2] discusses an algorithm for synchronizing source and target databases. Their algorithm arranges data into tuples and compares hashes of matching parts in the source and destination database. The authors have further compared their algorithm's performance against two variations of complete replica-

tion that employ different locking strategies. Based on the findings, the suggested approach proves more effective when the database undergoes minimal changes. Furthermore, the algorithm also outperforms row-level locking when fewer than 70% of the partitions have changes or modifications. The limitations of the algorithm are the hash function's capacity and the requirement to retrieve the number of data entries in a partition from the source-side database. Their approach, however, was different from [3, 6, 8] using message digest values to pinpoint the changed rows.

Kottilingam et al. [21] discuss about the challenge of synchronizing mobile data within high speed networks. They propose IDBSync (Improved database synchronization), designed to enhance the synchronization process between a server-side database and mobile database in the context of software defined networking (SDN). IDBSync uses a method called BaSyM (Batch Level Synchronization Methodology) that groups similar data cases, such as insertions, deletions, etc., for picking up changed data and synchronization. The synchronization server's tables are managed within the control plane. These tables comprise duplicates of the data tables found on both the server and client devices. These duplicates are obtained from the data planes and are used to implement the synchronization policy. This approach, according to them, minimized network usage and reduced energy consumption. The authors have compared the effectiveness of the proposed IDBSync mechanism against commercially available methods, and the results show that the proposed mechanism outperforms the commercial solutions in terms of network usage, energy consumption, and synchronization time. The authors furthermore concluded that the suggested approach can also be expanded to big data by NoSQL. This approach is completely different from the other papers [2, 3, 6, 8] where they have used message digest or hash values to compare two tables or images and capture changed data.

Gunasekaran Raja et al. [28] in another study introduces an innovative way to keep databases in sync, specifically between an SDN-assisted synchronization middlebox and mobile client databases. This method is all about dynamically managing caches to ensure data consistency. What sets this mechanism apart is its ability to handle different types of databases and adaptively synchronize data based on how often items are accessed. While this approach like [21] groups similar data cases, such as insertions, deletions, etc., for picking up changed data and synchronization, this approach also prioritizes data that is recent. The experiment conducted in this research show that average response time of this approach is faster than commercial solutions and faster than SAMD [8]. The authors conclude that the proposed mechanism not only matches but surpasses the performance of existing commercial solutions for database synchronization. as it's more energy-efficient, and it reduces the time it takes to synchronize databases.

3.1 Limitations and Research Gap

The previous research was mostly focused mainly on database synchronization basic architecture and using message digest, hash values or SDN-based approaches for picking up changed data and making the synchronization efficient in terms of speed and response time.

There has been notable lack of research focused on addressing the issue of synchronizing only certain data on the need and urgency. Except for [28] where the authors have introduced a prioritization policy that synchronizes the most recent data and frequently accessed data no other research have focused on which data to sync. However, this research [28] is limited as the prioritization focus is only on most recent and frequently accessed data, we intend to fill this lack of research on prioritization of data in database synchronization by introducing optimization algorithm to prioritize the data on the need, importance and urgency of the data which reduces the mobile data consumption as only important and prioritized data will be synced first rather than syncing all data. As the urgency, need and importance of data changes over time the database synchronization framework should be able to adapt meaning, there should be an option to change the priority levels which will be incorporated in this research. To fill the research gap the framework should be able to generate an optimal or prioritized order of data for synchronization while handling issues such network interruptions and concurrency updates.

Although, extensive research is made on different change data capture methods and are incorporated to the database synchronization, there is lack of research where they have set up change streams, meaning synchronizing the data as soon as a change is detected. We also intend to set up change streams in the prioritized database synchronization.

Chapter 4

Method

In this chapter, we will discuss the research methods used in this research. There are mainstream research methods for Computer Science, such as experiments, Surveys, Case studies, and Literature reviews, but the chosen research method is Experiments. The first step involves selecting optimization algorithms for prioritization. Then, we proceed to do experiments to evaluate the performance of the identified optimization algorithms for the prioritized database synchronization framework.

4.1 Literature Review

A literature review is performed to gain additional insights from current studies, which will help us answer RQ1. Keywords related to optimization algorithms, database synchronization, and other related terms are used to identify the relevant literature. A set of inclusion and exclusion criteria is used in the literature review to choose the most appropriate research papers and articles. Most of the articles were searched on notable and trusted knowledge bases such as Google Scholar, BTH Summon, IEEE Xplore, etc. The snowballing search mechanism is employed for this literature review.

The steps followed for this literature review:

1. Selecting only the above keywords for searching articles.
2. Selecting the articles according to the inclusion and exclusion criteria.
3. Further selecting the most relevant papers that are related to this research.
4. Study the selected papers.
5. Noting down a summary of all the found relevant research papers to further use in the research.

Below are the search strings or keywords used to identify the literature

---

("Database Synchronization" OR "Data Synchronization" AND "Optimization Algorithms" OR "Optimization Algorithms for Prioritization" OR "Data prioritization Algorithms")

The **inclusion criteria**:

1. Articles that are written in the English Language.
2. Articles that are available in full text.
3. Articles that have been published after 2003.
4. Articles related to Optimization algorithms and database synchronization.
5. Articles that are journals, books and conference papers.

The **exclusion criteria**:

1. Articles that are not written in English.
2. Articles that are not available in full text.
3. Articles that are not related to the research.

4.1.1 Search Mechanism

The search Mechanism refers to the method used to identify relevant papers for this thesis. The snowballing mechanism is employed for the effective selection of relevant papers.

4.1.2 Snowballing

Snowballing or snowball sampling is a research method that involves starting with the initial paper, which fits the inclusion criteria of the thesis and following the relevant papers from the references. The process repeats, leading to identifying the relevant literature effectively. This snowballing can be both forward and backward. **Backward snowballing** is selecting a paper and tracing backwards in time to find the studies that have been referenced in the paper, while **forward snowballing** is selecting a paper and finding the papers that have referenced the paper, which is opposite to backward snowballing approach. We used these mechanisms for the literature study in the thesis [40].

4.2 Experimentation

To answer RQ2 and RQ3, we conducted an experiment using the experimental methodology, a type of analytical approach. The experiment involved incorporating a hypothesis and variables that the researcher controls into the experiment design, monitoring and adjusting as needed.

The hypothesis for the RQ2 and RQ3 are as follows,

- **RQ2:** How do the different parameter settings affect the performance of the selected optimized algorithms in terms of finding the optimal solution?
  Null hypothesis (H0): Different parameter settings do not affect the performance of the selected optimized algorithms in terms of finding the optimal solution.
  Alternative hypothesis (H1): Different parameter settings affect the performance of the selected optimized algorithms in terms of finding the optimal solution.

- **RQ3** How well is the devised prioritized database synchronization framework working?
  Null hypothesis (H0): The devised prioritized database synchronization framework is not working effectively.
  Alternative hypothesis (H1): The devised prioritized database synchronization framework is working effectively.

To answer RQ 2, we evaluated different parameter settings on the selected optimization algorithms for prioritized database synchronization through experimental evaluation. We compared the performance of the algorithms and the synchronization framework on various metrics and selected the suitable algorithm for prioritized database synchronization.

Dependent variables: Convergence speed, elapsed time, synchronization time, fitness values

Independent variables: Optimization algorithms used for prioritizing the rows and algorithm's parameters.

4.2.1 Experimental Setup

Hardware Environment

CPU	Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHZ 2.80 GHz
GPU	NVIDIA GeForce GTX 1050 Ti
RAM	16 GB

Software tools, libraries, and Environment: The software applications and libraries used in this study are listed below:

1. ***Python*** programming language is used to implement optimization algorithms and synchronization scripts. It offers a vast collection of libraries and has a user-friendly syntax, making it an ideal choice for implementing and evaluating these algorithms and scripts.

2. ***NumPy*** is a Python library that supports sizeable, multi-dimensional arrays and matrices. It also provides various mathematical operations that can be performed on these arrays.

3. ***Pandas*** is a powerful Python library used for analyzing and manipulating data. It offers easy-to-use data structures, such as DataFrames, that simplify working with structured data.

4. The ***DEAP*** library is a Python tool that enables the creation and execution of evolutionary algorithms, including genetic algorithms. Its flexible framework allows the development of custom operators and evaluation methods.

5. ***Matplotlib*** is a library for Python that offers a range of visualization tools, such as line plots, box plots, bar plots, and scatter plots. It is used to visualize fitness values and synchronization time.

6. ***SciPy*** is a library in Python that builds upon NumPy and provides additional capabilities for scientific computing. This includes functions for statistical analysis, linear algebra, and optimization, which help conduct statistical tests.

7. ***Jupyter Notebook*** is a web-based computing environment that allows one to create and share documents with live code, equations, visualizations, and narrative text. It is used for implementing, evaluating, and recording research experiments.

8. ***PyMongo*** is a Python library that offers tools to connect with the MongoDB Atlas. One can perform numerous database operations, connect quickly to a MongoDB instance, and insert and retrieve data. MongoDB integration and synchronization are made possible using PyMongo in this research.

9. ***Tkinter*** is a standard Graphical User Interface (GUI) library for Python. It is a toolkit that provides various tools and widgets for creating graphical user interfaces (GUIs). It is based on the Tk GUI toolkit.

4.2.2 Dataset

Volvo Group has provided the data for this experiment in an Excel format. The dataset includes over 21 attributes, and the relevant ones are listed in Tables 4.1 and 4.2 below. There are no missing values in the dataset, but the timestamps have been converted to BSON format for more straightforward interpretation by MongoDB.

Column Name	Description
Timestamp	Timestamp of data recorded
cruisetime	time spent in cruise time
brakeCount	number of times brake is used
WithoutTorqueTime	time spent without torque
selectedDriveMode	Selected mode of driving
tachographSpeed Speed	according to tachograph
stopCount	number of stops made
serviceDistance	Distance travelled in service
trueIdleTime	Idle time
vehicleSpeed	speed of the vehicle
vehicleMovingTime	time the vehicle is in motion
vehicleDistance	Distance travelled by the vehicle
loadTruckFront	load on front of the truck
loadTruckRear	load on the rear of the truck
loadTruck	Total load on the truck
ext_gnss_altitude	Altitude of the vehicle (GNSS)
ext_gnss_latitude	Latitude of the vehicle (GNSS)
ext_gnss_longitude	Longitude of the vehicle (GNSS)
ext_gnss_timeStamp	Timestamp of the GNSS data
ext_gnss_heading	Heading of the vehicle (GNSS)
ext_gnss_horizontalSpeed	Horizontal speed of the vehicle (GNSS)

Table 4.1: Dataset description

4.2.3 Research Design

In this section, we provide an overview of how we carried out the experimentation in steps as shown in the below figure,

1. **Step 1:** Studying literature to understand which optimization algorithms are used in the prioritization of data.

2. **Step 2:** Selecting the optimization algorithms which will be used in this research study to find the prioritized order of data.

3. **Step 3:** Pre-processing the data so that it can be given to the optimization algorithm.

4. **Step 4:** Feeding the data to the selected optimization algorithms and comparing the results.

5. **Step 5:** Setting up databases and a synchronization script for the synchronization of prioritized data picked up by the optimization algorithm.

6. **Step 6:** Analyzing the devised synchronization framework and the optimization algorithm.

4.2.4 Approach

We used two algorithms, the genetic algorithm and the particle swarm algorithm Algorithm, to prioritize database synchronization. We iteratively tested with different parameter settings and random seed values to find the best solution. We continued the process until we found an optimal solution that satisfied our requirements. This study compared these two algorithms' performance to determine which was better suited for the optimization objective.

Genetic Algorithm Implementation
In this section, we explain our approach to the genetic algorithm to prioritize the data,

1. **Parameter settings:** The parameters a number of generations (NGEN), population size (MU), offspring size (LAMBDA), crossover probability (CXPB), and mutation probability (MUTB) are defined.

2. **Objective function:** To achieve efficient synchronization between source and remote databases, we have developed an objective function that minimizes data consumption while ensuring that essential prioritized data is synchronized. Additionally, we have created a fitness function that evaluates the fitness of each possible solution based on the provided data. The objective function considers synchronization parameters such as sync interval and data size.

   Sync interval is a synchronization parameter, which means the data synchronization starts for every sync interval.
   data_size is a parameter that defines the maximum amount of data in size that is to be transferred for each interval. This can change during the execution, depending on the bandwidth.

3. **Fitness function:** A fitness function is created and tailored according to the current optimization problem of prioritizing the order of rows for synchronization while limiting the data consumption. The fitness function calculates the score of each possible solution(individual) based on the data provided. In our case, our fitness function is tailored to consider multiple factors such as column priorities/weights and the conditions to invoke the priority/weight and load balance to evaluate each of the possible solutions, with the goal of selecting the best possible solution that satisfies the above-defined objective.
   Row Score: In order to calculate the row score, we assess each row in the solution by considering the values in the columns and taking into account the priority/weight of each column. For each row in the individual, calculate the row_score based on column weights and conditions:

$$\text{row\_score} = \sum_i (\text{column\_weight}[i] \cdot I(\text{column\_condition}[i](\text{row\_data}[i]))) \tag{4.1}$$

   where i is the column index, and $I()$ is the indicator function (1 if the condition is true, 0 otherwise).

Data Field	Weight Factor	Condition
selectedDriveMode	0.3	$x > 100$
tachographSpeed	0.2	$x > 30$
vehicleSpeed	0.4	$x > 30$
loadTruckFront	0.3	$x > 5000$
loadTruckRear	0.3	$x > 5000$
loadTruck	0.5	$x > 10000$
ext_gnss_altitude	0.4	$x > 50$
ext_gnss_heading	0.4	$x > 5$
ext_gnss_horizontalSpeed	0.2	$x > 10$

Table 4.2: Data fields, weight factors, and conditions used in the fitness function.

In Table 4.2, each row corresponds to a data field used in the fitness function. The "Weight Factor" column indicates the weight/priority assigned to each data field in the fitness function, which determines the relative importance of each field in determining the overall fitness score of the row. The "Condition" column specifies the condition that must be met for a given value of the data field to be considered significant in calculating the row's score. The Volvo supervisor chose these specific weight factors and conditions and was not chosen from extensive research. These conditions and weight factors are deemed to change as the requirement and priority of the data to synchronize change at the backend, so there could be no fixed weight factors and conditions. Different conditions and weight factors produce other results, which is a validity threat to this research.

For example, the "selectedDriveMode" field has a weight factor of 0.3 and a condition of x $>$ 100, which means that if the value of this field is greater than 100, it will be considered significant in the calculation of the row's score. It will be multiplied by a weight factor of 0.3. Similarly, the "loadTruck" field has a weight factor of 0.5 and a condition of x $>$ 10000, which means that if the value of this field is more excellent than 10000, it will be considered significant in the calculation of the row's score and will be multiplied by a weight factor of 0.5.

Load Balance: Load balance is calculated by comparing the front and rear weights of the truck. A separate function computes the absolute difference between the front and rear weights, as shown below,

$$\text{load\_balance\_score} = \sum_{i \in \{3,4\}} w_i \cdot |\text{load\_front} - \text{load\_rear}| \tag{4.2}$$

$$\text{load\_balance} = \frac{|\text{load\_front} - \text{load\_rear}|}{\min(\text{load\_front}, \text{load\_rear})} \tag{4.3}$$

Final Score:

$$\text{score} = \text{score} + \text{row\_score} - (\text{load\_balance\_score} \cdot \text{load\_balance}) \tag{4.4}$$

$$\text{normalized\_score} = \frac{\text{score} - \text{min\_score}}{\text{max\_score} - \text{min\_score}} \tag{4.5}$$

4. **Initialization:** We randomly generate an initial population of individuals (chromosomes). The individual (chromosome) represents an array of indices corresponding to rows in the provided data. Each individual or chromosome represents a potential prioritized order of rows.

5. **Fitness evaluation:** Using the fitness function devised, we calculate each individual's fitness in the population.

6. **selection:** In order to choose individuals for reproduction (mutation and-crossover), selection operators such as tournament selection, roulette wheel selection, and elitist selection are used. These operators prioritize individuals with better fitness values.

7. **Crossover:** We used the crossover on selected solutions(individuals) to create offspring. In this implementation, we employ the one-point crossover technique, in which the segments of the parent chromosomes are switched to create offspring at a random crossover point.

8. **Mutation:** We introduce diversity into the population by randomly modifying some of the offspring's genes (rows) using a mutation operation (MUTPB). This helps to delay premature convergence. We modify a few of the offspring's genes (rows in this case) with a certain probability by applying the mutation operation to them (MUTPB). The population is given diversity through this process, which delays premature convergence.

9. **Replacement:** Few of the least fit solutions are replaced by adding the offspring to the population.

10. **Termination:** Steps 4 to 8 are repeated until the given number of generations have passed (NGEN). The best individual is returned as the solution once a number of generations have passed.

Particle swarm optimization implementation:
In this section, we explain our approach to particle swarm optimization (PSO) to prioritize the rows in the given data,
Steps 1-3 are the same as the GA implementation, Particle Swarm Optimization (PSO) Implementation:

- **Initialization**: We create an initial swarm of particles where each particle represents a possible order of rows for data synchronization. The position and velocity of each particle are initialized randomly within the problem bounds.
- **Fitness evaluation**: We compute the fitness of each particle in the swarm using the pso_fitness function, which evaluates the particle based on the same objectives as the GA.
- **Personal and global best update**: We update the personal best positions for each particle and the global best position for the entire swarm based on their fitness values.
- **Velocity update**: We update the velocities of the particles using their current velocities, personal best positions, global best position, and some random factors. The velocity equation's inertia, cognitive, and social components determine how much the particle is influenced by its current velocity, personal best, and global best, respectively.
- **Position update**: We update the positions of the particles using their updated velocities. The new positions should be within the problem bounds.
- **Termination**: We repeat steps 2-5 for a predefined number of iterations (n_iterations). Once the termination criterion is met, we return the global best position found as the solution.

4.2.5 Metrics

1. **Convergence speed** describes how quickly an algorithm finds a solution. It can be determined by finding the slope of a line connecting the starting and ending fitness values and dividing that value by the number of generations for the two fitness values to converge. A lower convergence speed indicates that the algorithm found the solution more quickly [32].

$$c_s = \frac{y_n - y_0}{n} \tag{4.6}$$

Where:

- c_s is the convergence speed
- y_n is the fitness value of the last generation
- f_0 is the fitness value of the first generation
- n is the number of generations required for convergence

2. **Max fitness** is the highest fitness value attained by any individual in the population throughout all generations. It represents the best possible solution discovered by the algorithm [4].

$$\max = \max_{i=1}^{N} f(x_i) \tag{4.7}$$

3. **Avg fitness** refers to the average fitness value of all individuals in the population across all generations. It represents the overall quality of solutions discovered by the algorithm [19].

$$\text{avg} = \frac{1}{N} \sum_{i=1}^{N} f(x_i) \tag{4.8}$$

4. The **Standard deviation** of fitness values refers to the degree of diversity in fitness values across all individuals in the population throughout all generations. It indicates the diversity of solutions found by the algorithm, with a smaller standard deviation indicating a more homogeneous population and a more significant standard deviation indicating a more diverse population [4].

$$\text{std} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (f(x_i) - \text{avg})^2} \tag{4.9}$$

Where $f(x_i)$ is the fitness function value for the ith individual, and N is the population size.

As there is no baseline solution for this optimization problem, it is not possible to use metrics such as accuracy. Instead, we use fitness values and convergence speeds to evaluate the algorithms. Generally, in maximization optimization problems, if the fitness values are increasing over a generation, it means that the algorithm is reaching an optimal or sub-optimal solution, and it is vice versa for minimization problems. In research conducted by Nguyen et al. [13] for performance comparison of different algorithms in an experiment, their algorithm MIGA performed well, and the fitness function values over generations were used for the analysis. In another study conducted by Sourabh et al. [20], they reviewed applications and research of genetic algorithms. In their results, they often mentioned studies that used fitness function as a metric and concluded based on the fitness values. For instance, in a gaming application, gomkoku [38], the authors have concluded that GA based approach finds the solution with higher fitness than other methods implying higher fitness values for a maximization problem is a good sign. In another study by Khader et al. [12], the optimization is of minimization problem, and their fitness function was Mean Squared Error (MSE) was assigned as a fitness function and in the results, they used the fitness function as a metric and showed that the MSE is decreasing over the generations meaning the algorithm was able to converge to an optimal or

sub-optimal solution for the minimization problem. Hence, we determined to use fitness values and convergence speeds as evaluation metrics for the algorithms.

Chapter 5

Synchronization Framework

The synchronization framework is created by integrating the synchronization script and the optimization algorithm for generating prioritized documents/rows.

5.1 Database Setup

The database setup is discussed below,

1. The databases were set up on different MongoDB Atlas clusters.
2. The Source database has two collections, Primary data collection and Sync SRC data collection, while the Destination database has one collection, Sync DST data collection.
3. Truck/Telemetry Data is added to the Primary data collection of the Source database.

5.2 Synchronization Approach

In this section, we explain our approach to synchronizing the prioritized data between two databases, and Figure 5.1 depicts the overall framework.

1. At the source side, the primary data collection gets the telemetry data. The optimization algorithm fetches data using a query, and the optimal order of data for sync is sent to "Sync SRC data collection."
2. The Python script connects to the configured MongoDB databases and collections using the connection strings of the Source database and Destination database.
3. The source side, "Sync SRC data collection", and the destination side ", Sync DST data collection", were configured in the above step.
4. For Source collection, the script creates a '**change stream**' using the '**watch**' method and listens for all the changes, such as Insertions, Deletions, and Update operations.
5. Only the changes happening in "Sync SRC data collection" is in sync with the destination side database, which only has the prioritized documents/rows generated by the algorithm.

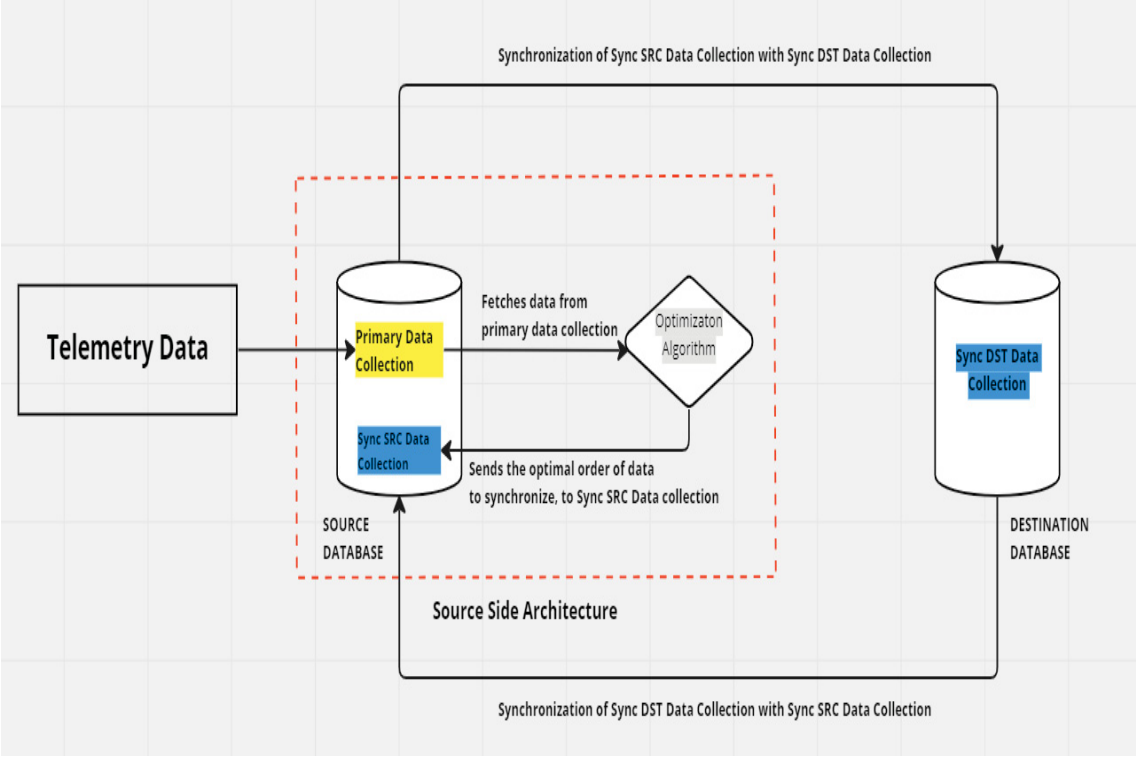

Figure 5.1: Prioritized Synchronization Framework

Figure 5.1 is the complete framework with the optimization algorithm integrated with synchronization script that works as above.

5.3 Graphical User Interface

The GUI for the framework was created using the Tkinter library in Python. The GUI has the following input fields,

1. **Source database connection string:** This string is used to connect to the source database.

2. **Destination database connection string:** This string is used to connect to the destination database.

3. **Source database name:** The name of the source side database we intend to interact with should be specified.

4. **Source collection name:** The name of the source side collection we indent to interact with should be specified

5. **Destination database name:** The name of the destination side database we intend to interact with should be specified.

6. **Destination collection name:** The name of the destination side collection we indent to interact with should be specified

7. **Timeout(seconds):** The amount of time the synchronization framework should run can be specified here

8. We have two buttons to start the synchronization script and run the genetic algorithm to generate

9. The output box prints the time taken to synchronize for every 100 documents/rows.

Figure 5.2 is the GUI after inputting the parameters needed to start synchronization. It is to be noted that the GUI shows synchronization time from the time we run the synchronization script and the actual time taken for synchronizing those 100 documents.

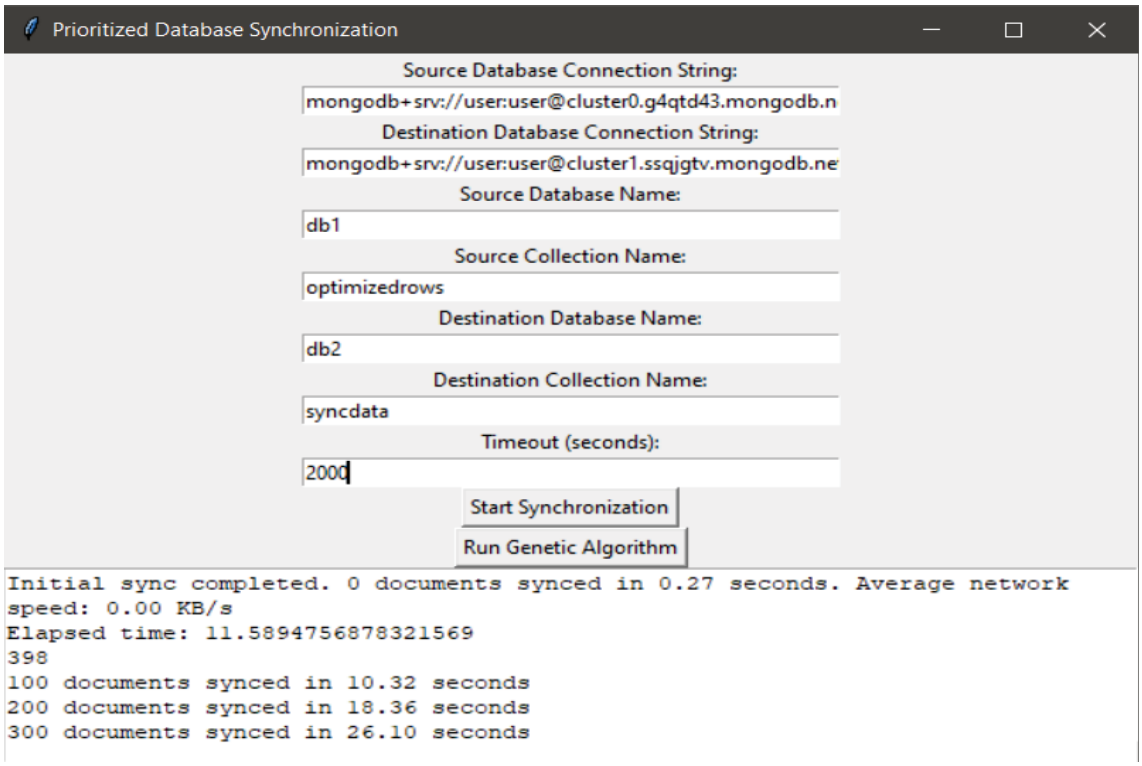

Figure 5.2: GUI with Input parameters

Chapter 6

Results and Analysis

6.1 Literature Review

In this chapter, we show the results of the literature review performed, followed by the analysis of the results found,

6.1.1 Literature Review Results

Article	Result
A systematic literature review of test case prioritization using genetic algorithms [5]	In this paper, the authors present a systematic literature review on the use of genetic algorithms for test case prioritization. The review looks at how genetic algorithms are currently used for test case prioritization, including the effects of various factors on solution quality and the authors finally conclude that genetic algorithms have the potential for solving test case prioritization problems.
Interactive requirements prioritization using a genetic algorithm [35]	In this paper, the authors present an Interactive Genetic Algorithm(IGA) for requirement prioritization. IGA combines incremental knowledge acquisition with constraints such as dependencies and priorities to produce accurate requirements ordering. This algorithm is compared in terms of effectiveness, efficiency, and robustness with the Incomplete Analytic Hierarchy Process (IAHP) method. The results show that this presented new IGA outperforms IAHP. This paper contributes to requirements prioritization by introducing an interactive genetic algorithm as a search-based technique.

Article	Result
A genetic algorithm-based approach for test Case prioritization [11]	In this paper, the authors present a genetic algorithm-based approach for test case prioritization in software maintenance and regression testing. The genetic algorithm is compared with a random technique using a benchmark program, and the experiment shows that the genetic algorithm approach outperforms the random technique in terms of the Average Percentage of Fault Detection (APFD). The research shows the effectiveness of a genetic algorithm-based approach for test case prioritization.
Multi-objective test prioritization via a genetic algorithm [30]	In this paper, the authors propose a multi-objective test prioritization approach using a genetic algorithm. The aim is to determine the criticality of software components at the design level and rank them in order of importance for testing. Test cases are then chosen from a large pool using a genetic algorithm-based technique. The research contributes to the understanding of genetic algorithm-based techniques for multi-objective test prioritization.
Optimal test suite selection in regression testing with test case prioritization using modified Ann and Whale optimization algorithm [14]	In this paper, the authors focus on the issue of selecting the best test suite for regression testing with test case prioritization. The aim is to increase fault detection during testing. The method in this paper involves creating test cases, clustering them using the kernel fuzzy c-means clustering technique, and then ranking the pertinent test cases in order of importance. Modified Artificial Neural Network (ANN) classification algorithms are used for test case prioritization, and the Whale Optimization Algorithm is used for weight optimization. The study seeks to increase the likelihood of spotting source code flaws early. The research contributes to regression testing and test case prioritization.
Regression test optimization and prioritization using Honey Bee optimization algorithm with fuzzy rule base [25]	In this paper, the authors discuss the problem of regression test optimization and prioritization. The aim is to increase the fault detection rate in regression testing by setting up the test case suite to catch errors earlier in the testing process. A Bee algorithm is modelled from the strategies of honey bee swarms. The algorithm is implemented and evaluated on two projects, and the prioritization results are quantified using the APFD metric. The proposed algorithm outperforms all other techniques in terms of fault detection rate. The research contributes to regression testing and test case prioritization.

Article	Result
Applying particle swarm algorithm to Prioritizing Test Cases for Embedded Real-Time Software Retesting [15]	In this paper, the authors propose a particle swarm algorithm (PSO) algorithm for prioritizing test cases in regression testing of embedded real-time software. The aim is to prioritize test cases in a new order and choose higher-priority test cases for the regression testing process. The PSO algorithm used in the paper demonstrates the effectiveness and efficiency of prioritizing test cases in test suites. The research contributes to applying the PSO algorithm to test case prioritization in embedded real-time software retesting.
Test case prioritization using multi-objective particle swarm optimizer [36]	In this paper, the authors present a 3-phase approach for test case prioritization in regression testing. The aim is to reduce redundant test cases and prioritize them based on fault coverage and execution time. Redundant test cases are removed in the first stage. Using a multi-objective particle swarm optimization algorithm, test cases are chosen in the second phase to have the shortest execution times while covering all faults. The test cases are given priority in the third phase based on the fault coverage to execution time ratio, with higher values indicating a higher priority.

Table 6.1: Results of the literature review

These research papers used in the literature review are retrieved from different databases such as IEEE, Science Direct, Springer, and ACM. Three papers from IEEE, three from Springer, one from Science Direct and one from ACM, making a total of 8 papers.

6.1.2 Literature Review Analysis

The papers reviewed for this literature review and their results are presented in the above table. To assess RQ1, the literature was helpful in determining the optimistic algorithms used for prioritization purposes. Although the application domain of these optimistic algorithms used by the authors is software engineering, software testing, regression testing and test case prioritization, the concept of using optimization algorithms for prioritization can be extended to other domains, such as in this thesis. From the literature review, it can be seen that various optimization algorithms have been used for prioritization purposes in the software engineering industry, such as the Genetic algorithm, particle swarm algorithm, Honey Bee optimization algorithm, Modified Ann and Whale optimization algorithm, and Ant colony optimization. Considering the time and scope of the research and also the popularity of the genetic and particle swarm optimization from the literature review, we have determined to use genetic and particle swarm optimization in this research.

6.2 Experimental research

Two optimization algorithms, genetic algorithm and particle swarm algorithm (PSO) determined from the literature review, were applied to similar problems and the same dataset for conducting the desired experiment. We iteratively conducted experiments with different parameter settings for each algorithm and selected three parameter settings across different ranges that are best in the tested parameter settings. We first present the comparison of 3 different selection operators for the Genetic algorithm and compare the results of different parameter settings for each algorithm.

Conducting Performance analysis on genetic and particle swarm optimization can help identify the best parameters. It, in turn, helps the generation of high-quality solutions [33].

6.2.1 Genetic Algorithm

In order to optimize the problem of obtaining the most prioritized rows, a genetic algorithm is used with parameter settings, as indicated in Table 6.1. The selection operators for each parameter setting are presented in Table 6.2. The resulting fitness values were plotted over 'n' generations and are depicted in Figures 6.1, 6.2, and 6.3. The Y-axis of these plots represents the fitness Values, while the X-axis denotes the generations.

Parameter settings	NGEN	MU	LAMBDA	CXPB	MUTPB
1	50	100	50	0.6	0.6
2	100	50	100	0.7	0.6
3	200	150	200	0.8	0.8

Table 6.2: Parameter settings for Genetic Algorithm

6.2.1.1 Parameter settings 1

The results in Figure 6.1 show that the genetic algorithm with parameter setting 1 optimized the objective function, as the fitness values of all three selection operators are increased over 50 generations. From Table 6.4 and Figure 6.1, it is also evident that each selection operator is performing differently, indicating that the algorithm with different parameter settings and operators can have an impact on the results.

Of the three selection operators (tournament, roulette wheel, and elitist), the roulette wheel operator performed better than the tournament operator and elitist operator regarding results, which is also clearly evident in Figure 6.1. The elitist selection operator has the lowest avg fitness value, std deviation and max fitness value, meaning that it may have reached a point in finding a near-optimal solution, but the other operators have found better near-optimal solutions. Regarding elapsed time, the roulette wheel operator is slightly faster than the others.

Selection Operators	**Function**	**Description**
selTournament	Tournament Selection	Selects individuals based on their fitness scores in a tournament. It randomly selects a population subset and chooses the individual with the highest fitness score.
selRoulette	Roulette Wheel Selection	Selects individuals with probabilities proportional to their fitness scores. The probability of selecting an individual is determined by dividing its fitness score by the sum of all the fitness scores in the population.
selBest	Elitist selection	Selects the best-performing individuals based on their fitness scores. It chooses the top individuals with the highest fitness scores in the population.

Table 6.3: Selection operators in Genetic Algortihm

Operator	Elapsed Time(s)	Max fitness	Avg	Std deviation
Tournament Selection	18.53	0.000097	0.000036	0.000036
Roulette Wheel Selection	16.9	0.00017	0.000053	0.000058
elitist selection	18.53	0.000048	0.000017	0.0000179

Table 6.4: GA Parameter 1 Results

Selection operator	Convergence speed
Tournament selection	1.02e-06
Roulette selection	2.49-06
elitist selection	3.83-07

Table 6.5: GA parameter settings 1 Convergence speeds

From Table 6.5, the elitist selection operator has the fastest convergence speed, followed by tournament and roulette wheel operators.

Overall, it can be concluded that the roulette wheel selection operator performed the best regarding fitness values, while the elitist Selection operator performed the best regarding convergence speed.

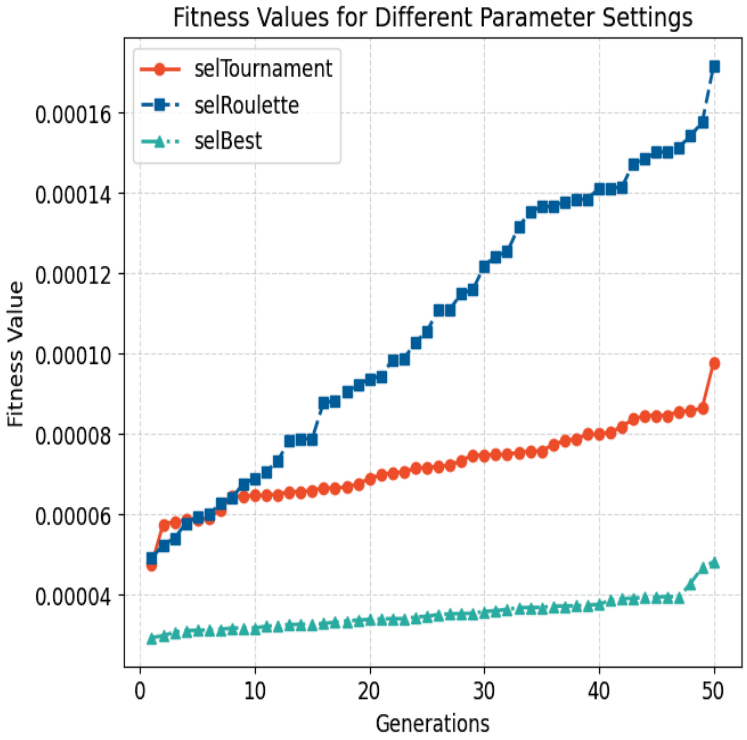

Figure 6.1: GA Parameter settings 1 plot

6.2.1.2 Parameter settings 2

With these parameter settings, GA successfully optimized the objective function, as the fitness values of all three selection operators increased over 100 generations. From Table 6.6 and Figure 6.2, it is also evident that the fitness value statistics also improved with these parameter settings, indicating that the algorithm with these parameter settings was able to find near-optimal solutions than parameter settings 1. When the parameter settings are changed, the algorithm behaviour is also slightly changing in these settings. Figure 6.2 shows fewer fluctuations than in parameter setting 1, and the algorithm reached a near-optimal solution.

Regarding convergence speed from Table 6.7, the tournament selection operator has the fastest speed, followed by the elitist and roulette Wheel operators.

Overall, the roulette Wheel Selection operator performed the best regarding fitness values. In contrast, the tournament Selection operator performed the best regarding convergence speed.

Operator	Elapsed Time(s)	Max fitness	Avg fitness	Std deviation
Tournament Selection	16.73	0.00011	0.00006	0.00001
Roulette Wheel Selection	18	0.00021	0.0.0001	0.00003
elitist selection	18.38	0.00005	0.00003	0.00005

Table 6.6: GA Parameter 2 Results

Selection operator	Convergence speed
Tournament selection	6.82e-07
Roulette selection	2.04e-06
elitist selection	2.87e-07

Table 6.7: GA parameter settings 2 Convergence speeds

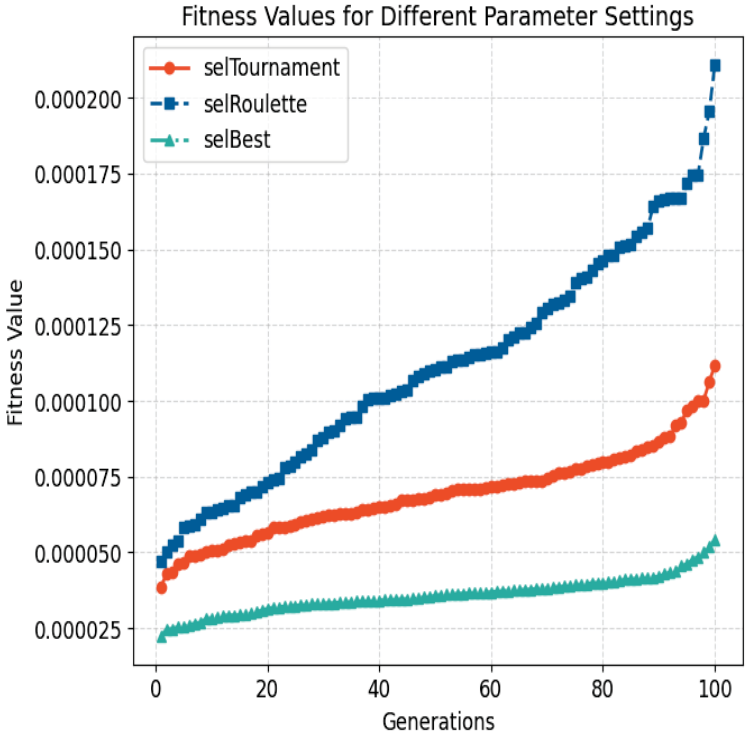

Figure 6.2: GA Parameter settings 2 plot

6.2.1.3 Parameter settings 3

With these parameter settings of adjusted parameters, the genetic algorithm successfully optimizes the objective function. As shown in Figure 6.3, the fitness values increase over time, giving an optimal order of data for synchronization while limiting

the data consumption. However, it is to be noted that the elapsed time has drastically increased when compared to other parameter settings, and the max fitness value 0.0001 of the roulette operator, according to Table 6.8, is still less than the parameter settings 2.

From Table 6.9, the tournament selection operator has the fastest convergence speed, followed by the elitist and roulette wheel operators.

Overall, it can be concluded that the roulette wheel Selection operator performed the best regarding fitness values. In contrast, the tournament Selection operator performed the best regarding convergence speed.

Operator	Elapsed Time(s)	Max fitness	Avg fitness	Std deviation
Tournament Selection	107.17	0.00006	0.00004	0.00045
Roulette Wheel Selection	114.27	0.0001	0.00006	0.00001
elitist selection	117.03	0.00004	0.00003	0.000003

Table 6.8: GA Parameter 3 Results

Selection operator	Convergence speed
Tournament selection	1.43e-07
Roulette Wheel Selection	4.48e-07
elitist selection	9.0395e-08

Table 6.9: GA parameter settings 3 convergence speeds

The results of parameter setting 3 for the genetic algorithm are not better than parameter setting 2, which has fewer generations than parameter setting 3. Though the fitness values progress over time, the maximum fitness value of parameter setting 3 is still less than the one with 100 generations (parameter setting2). This means the algorithm is getting stuck and is unable to find even better results. So, further increasing the number of generations would only produce less optimal solutions and take much time to find an optimal solution as the number of generations increases. So, there is no point in increasing the generations because it is evident that the results of parameter setting 3 are not better than parameter setting 2, which has even fewer generations, meaning the algorithm is getting stuck in the process of finding optimal solutions when the number of generations is increased.

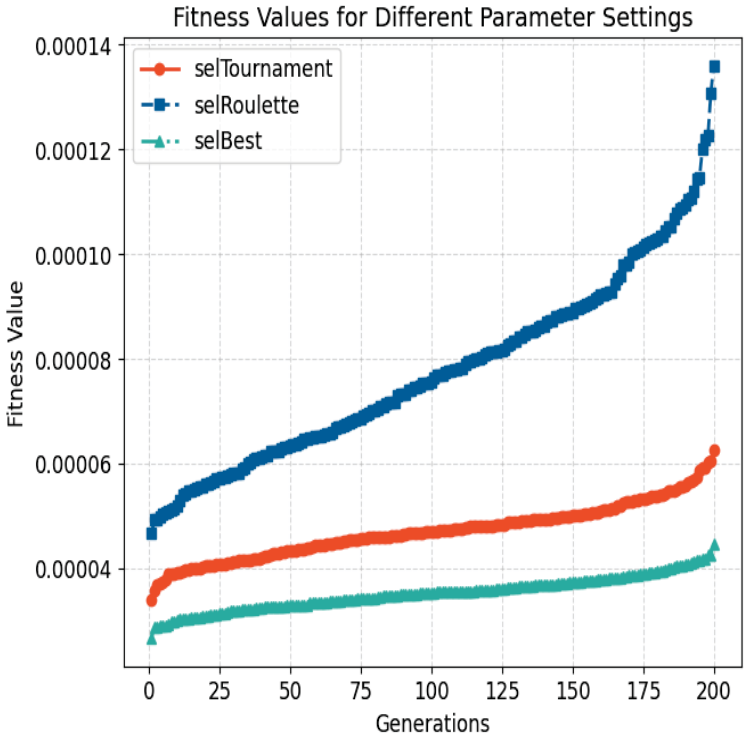

Figure 6.3: GA Parameter settings 3 Plot

6.2.2 Particle Swarm Optimization

Parameter settings	Num of Particles	Num of Iterations	Inertia Weight	Weights
1	50	100	0.9	1.5, 1.5
2	100	200	0.7	2.0, 2.0
3	200	300	0.6	2.5, 2.5

Table 6.10: Particle Swarm Optimization parameter settings

The results of parameter setting 3 for particle swarm optimization are not better than parameter setting 2, which has fewer generations than parameter setting 3. Though the fitness values progress over time, the maximum fitness value of parameter setting 3 is still less than the one with 100 generations (parameter setting2). This means the algorithm is getting stuck and is unable to find better solutions or results. So, further increasing the number of generations would only produce fewer results and take much time to find an optimal solution as the number of generations increases.

After analyzing the data provided in Table 6.11 and the insights from Figure 6.4, parameter setting 2 is the most effective for the particle swarm algorithm as it produced the highest fitness value and a balanced elapsed time comparably, indicating the ability to get better solutions than other parameter settings. The fitness values

of all parameter settings increased over iterations, suggesting that they are able to find near-optimal solutions. The Y-axis of the plots represents the fitness values, and the X-axis represents the number of iterations. It is evident that parameter setting 2 had the highest fitness value from Figure 6.4.

Regarding the convergence speeds from Table 6.12, the results show that parameter settings 3 has the fastest convergence speed and is more efficient in optimizing the objective, followed by settings 2 and 1.

Overall, parameter setting 2 performed the best regarding fitness values, while parameter setting 3 performed the best regarding convergence speed.

Parameter	Elapsed Time(s)	Fitness Value		
		Max	Avg	Std
P1	19.18	0.00018	4.30e-05	3.92e-05
P2	85.59	0.00027	2.63e-05	4.27e-05
P3	250.12	0.00019	2.60e-05	3.64e-05

Table 6.11: PSO algorithm results

Parameter setting	Convergence speed
1	3.82883903e-06
2	2.74865423e-06
3	1.00124971e-06

Table 6.12: PSO parameter settings Convergence speeds

To determine which algorithm performs better for this optimization objective, we can compare the best results of both Genetic and Particle Swarm optimization.

After examining the results of parameter settings 2 of both the algorithms, including elapsed time, fitness value plots, and convergence speeds, it is evident that the parameter settings of the genetic algorithm outperform those of the particle swarm algorithm. Though parameter setting 2 of particle swarm optimization has slightly better fitness values than parameter setting 2 of the genetic algorithm, they are almost the same, and the elapsed time of the genetic algorithm is much less than the particle swarm optimization. Therefore, we conclude that the genetic algorithm performs better than the particle swarm algorithm in this research.

For RQ2, The results of the experiments confirmed our hypothesis. Different parameter settings affect the performance of the selected optimized algorithms in terms of finding the optimal solution, as presented in the above results.

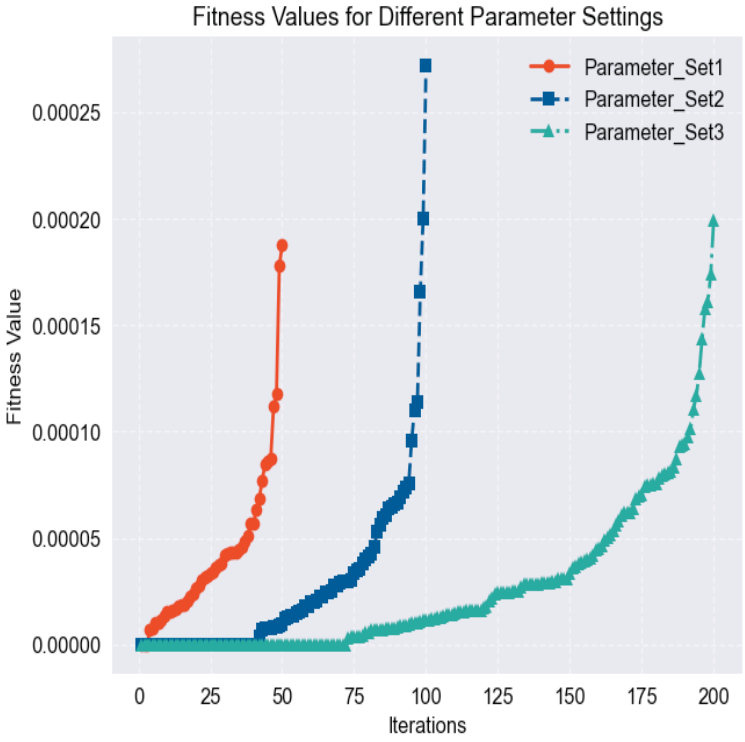

Figure 6.4: PSO parameter settings Plot

6.3 Synchronization results

To evaluate the synchronization framework, we measured the time it took to synchronize all the changes to the remote database from the source database over different networks. We inserted over 800 documents into the source side collection and then initiated the sync. Table 6.13 shows the measured speeds of different networks during synchronization, and Table 6.14 and Figure 6.5 shows the time taken for different numbers of documents over different network speeds over time. Since in the framework, we are using MongoDB Atlas, a cloud database, the network speeds between the framework server and the MongoDB servers can have a significant effect. It is expected that faster speeds can result in faster data access and retrieval times.

Network	Download Speed (MB/sec)	Upload Speed (MB/sec)
WiFi	106.4	51.07
4G	37.5	4.93
3G	14.49	4.72

Table 6.13: Network speeds

Figure 6.5 shows the time taken to synchronize different numbers of documents over different networks and speeds. As expected, the time taken for synchronization is strongly affected by the type of network, with WiFi being the fastest and 3G being the slowest across different numbers of documents. Table 6.13 shows the results; from that, we can conclude in exact numbers that WiFi took 31.42 seconds to complete the synchronization of over 800 documents and the same, 4G took 50.21 seconds,

Number of Documents	WiFi Time (s)	4G Time (s)	3G Time (s)
100	4.8	6.35	8.08
200	9.84	12.36	17.21
300	11.92	18	25.56
400	16.58	23.9	34.96
500	19.41	27.93	44.46
600	24.7	37.16	49.63
700	28.18	37.72	58.57
800	31.42	50.21	68.16

Table 6.14: time taken to synchronize documents over different networks

followed by 3G, which took 68.16 seconds. From this, we can conclude that network speeds can significantly affect the synchronization time.

We have also evaluated the features of this framework; the changed data capture is

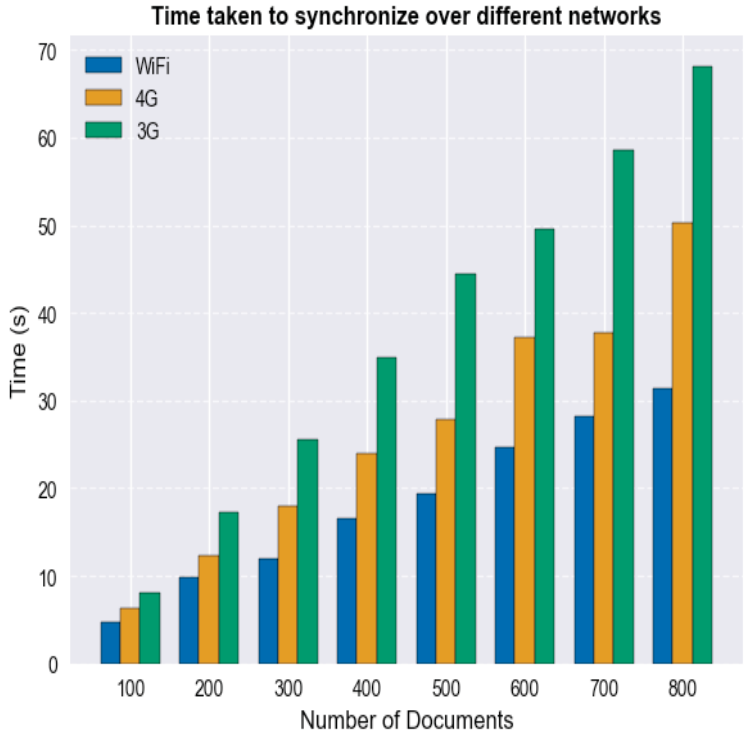

Figure 6.5: Time taken for synchronization

working well, as only the changed data was synchronized. We disrupted the network at the times of synchronization, and the synchronization framework, for the most time, did not raise any error, waited in exponential time delay, and later connected when the connection was back to normal. We updated the same document/row in both databases. The framework prioritized the most recently updated document/row over the other; thus, the concurrent changes are handled. Figure 6.6 shows the before and after synchronization of the destination side database. It is to be observed that the changes from the source side are successfully synchronized. Figure 6.6 and

6.7 shows the MongoDB Atlas interface before and after synchronization. All 398 documents on the source side were successfully synchronized, which we can see in the "After Sync" picture.

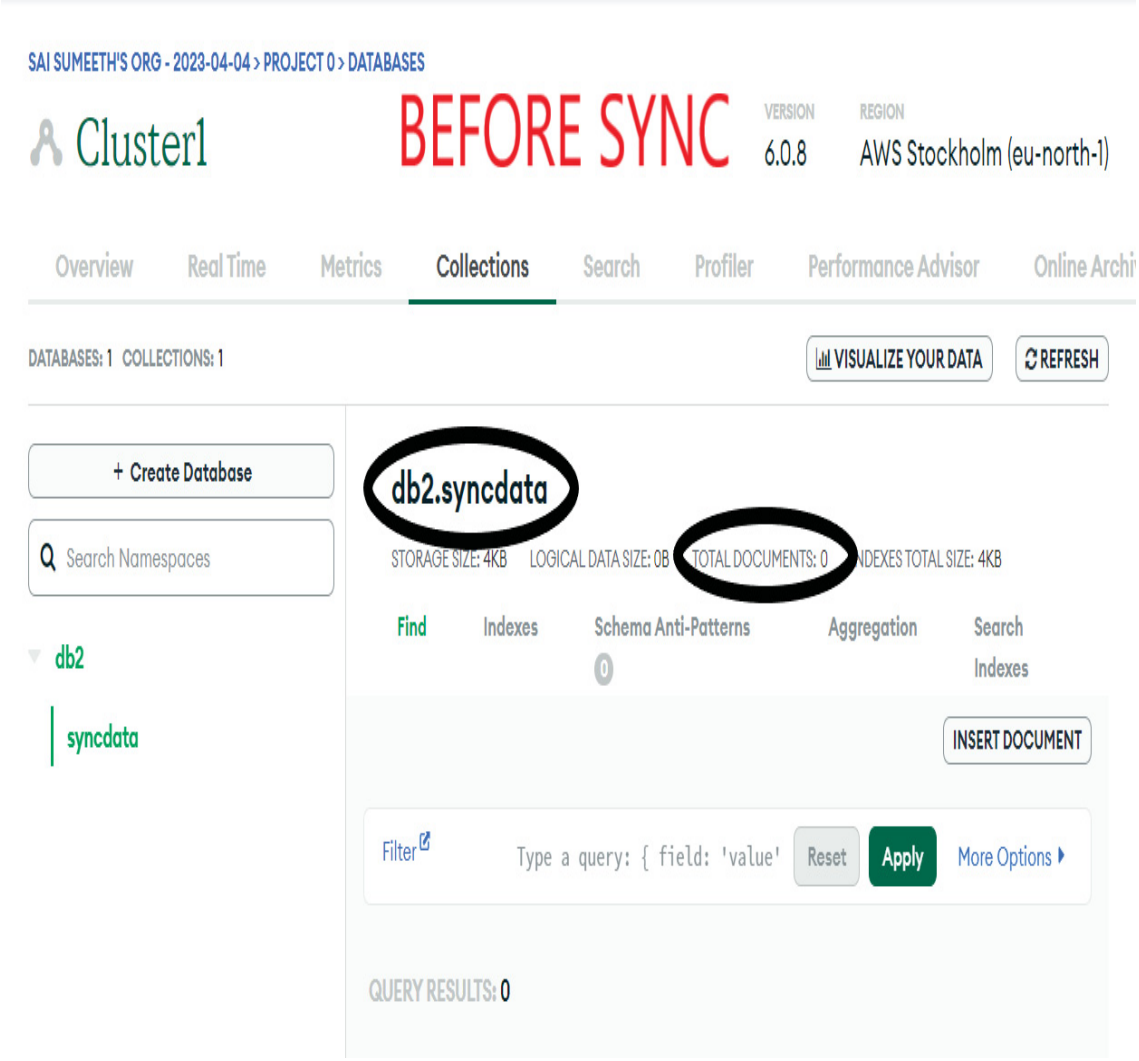

Figure 6.6: Before Synchronization (Destination Side)

For RQ3, The results of the experiments and synchronization times confirmed our hypothesis. The devised framework is working effectively, but as stated above, the synchronization times are high, so further research is needed, and this is the limitation of this framework.

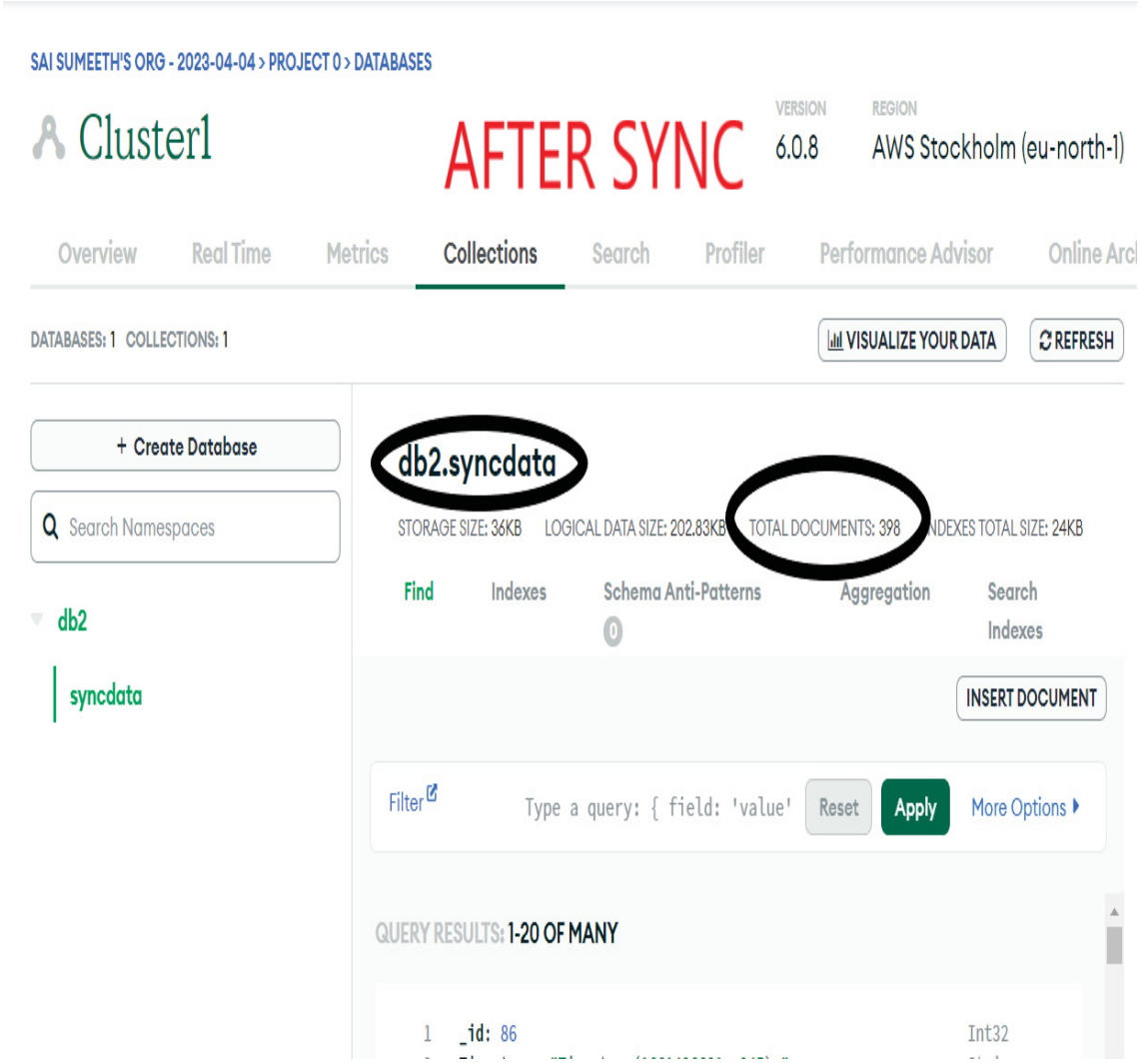

Figure 6.7: After Synchronization (Destination Side)

Chapter 7

Discussion

In this chapter, the discussion will explore the study results, consider their potential significance, and suggest directions for future research.

In the thesis, we have proposed a framework for synchronizing prioritized data between source and remote databases in the context of telemetry data of trucks at Volvo. From a literature review, we have selected two optimization algorithms, the genetic algorithm and the particle swarm algorithm, for this problem. We have also devised a synchronization framework that integrates with this algorithm.

For Answering RQ1, a literature review on existing optimistic algorithms used for prioritization purposes was conducted, and the literature review was helpful in answering this question. Optimization algorithms from the literature review were selected for the experimentation and implementation of the framework addressing the next research questions. This literature review has served as the basis for the further parts of the research.

For Answering RQ2 and RQ3, as discussed above, an experimental research method was conducted. For the algorithm, different parameter settings for GA and PSO and operators for GA were used to identify the best parameters and give quality solutions. These are evaluated using Maximum fitness, Avg fitness, Standard deviation, and convergence speeds. We aim to minimize data consumption while the necessary prioritized data is synchronized between the source and remote databases. By prioritizing rows based on their level of importance and specific conditions, the optimization algorithm can ensure that critical data is synchronized while minimizing unnecessary data consumption. This approach helps to improve the efficiency and effectiveness of the synchronization process, as only the most important data is being synchronized, reducing the load on the database and the network. It also helps reduce the risk of data loss or corruption, as the most critical data is prioritized and synchronized first.

For RQ2, the algorithms were run with different parameter settings. The experiments showed that changing and adjusting the parameters of the genetic algorithm and particle swarm algorithm significantly affected their performances. Parameter tuning for the genetic algorithm shows that it significantly influenced the quality of solutions obtained. Increasing the number of generations (NGEN) and the population size (MU and LAMBDA) while also increasing the crossover and mutation probabilities (CXPB and MUTPB) resulted in better solutions. Parameter setting 2 for GA with the roulette operator is the most effective for optimizing the objective

function because, in these settings, the algorithm achieved a higher maximum fitness value. For a maximization optimization problem, it indicates that the algorithm could find quality solutions. It also had a good balance of exploration and exploitation of the search space with increased CXPB and MUTPB (0.7 0.7). Additionally, the execution time compared is relatively less, and the standard deviation is also comparably low than parameter setting 3, which makes it better.

The experiments also showed that changing and adjusting the parameters of the PSO algorithm can significantly affect its performance. Increasing the number of particles and iterations allowed for more exploration and exploitation of the search space while adjusting the inertia weight and weights helped to balance these aspects of the algorithm. Parameter setting 2 was the best, with the highest, maximum, average fitness values and standard deviation indicating a good balance between exploration and exploitation. These findings suggest that careful parameter tuning can lead to improved optimization performance.

The synchronization, the Change Data Capture (CDC) technique, is utilized, which captures only modified data using the MongoDB method "watch()". Since intermittent, unstable network connections and failures are likely to occur, an exponential time delay method is employed to wait before attempting to reconnect. This approach ensures that a minimum load is placed on the source database. It is also possible for the same row or document to be changed in both databases. The synchronization framework employs the Optimistic Concurrency Control (OCC) technique to address this issue. This technique handles concurrent modifications and prioritizes the most recently modified document when handling these changes.

Based on the results, parameter setting 2 with the roulette operator was found to be most effective for optimizing the objective function for the genetic Algorithm. For the particle swarm optimization algorithm, its PSO parameter setting two was the most effective for optimizing the objective function. Overall, the framework with the genetic algorithm integrated into it is working well, but comparing it with other approaches of synchronization and other prioritization techniques would actually be a more effective way to evaluate and understand the results.

For RQ3, when integrated with the genetic algorithm, the framework was working fine. It can be observed in Figure 5.2 that when run first, the framework starts an initial sync and synchronizes any changes. Once we run the genetic algorithm, the algorithm takes some time to run and sends the data to another collection of the source database, which is in sync with the destination side database. The GUI prints the time taken for the genetic algorithm to run and also prints the time taken to synchronize for every 100 documents. Figure 6.5 shows the time taken to synchronize different numbers of documents. Although the speeds are relatively higher than normal, it was able to synchronize the changed documents, which were the optimized rows/documents generated by the genetic algorithm. We conclude that the overall framework is also working fine, accepting our initial hypothesis for this research question.

The other optimization algorithms might also optimize this problem and generate

reasonable solutions, but considering the time frame of this research, we did not consider the synchronization; it seems that the time taken to synchronize the documents is relatively high, and the synchronization framework needs more sophisticated features that would synchronize the documents faster, more about is discussed in the next chapter.

7.1 Validity Threats

In this section, we discuss the validity threats in this research,

7.1.1 Internal validity

Internal validity generally is about how well the experiment is designed and how accurately it measures what it intends to measure. The data for the experiment is provided by Volvo, and it is accurate. However, there is a selection bias; only the genetic algorithm and particle swarm algorithm to optimize the objective function of generating the optimal order of data for synchronization. Using other optimization algorithms might produce different results.

7.1.2 External validity

External validity is about how the research findings could be applied in other scenarios outside the research. The data used is from Volvo; however, with different data and the right data prepossessing, these algorithms could be applied to it. However, the other scenarios would not get the same results because the priority/weight and conditions to invoke are specific to the Volvo Supervisor and are not chosen from extensive research. The results are only for these settings, and the other settings might produce other results. It would take a considerable amount of time to find the appropriate parameter settings for the problem, so these results could not be applied to other problems. The synchronization framework, however, can be used without any modifications for the intended use or could be modified if more features are needed.

7.1.3 Construct validity

Conclusion Validity is about the conclusions drawn from accurate results and can be generalized. Since no baseline solution exists for this optimization objective, we compared the parameter settings and algorithm with statistic tests, fitness values, and convergence speeds. However, these metrics also efficiently compare the parameter settings and algorithms and draw conclusions from them.

7.1.4 Selective reporting bias

Selective Reporting Bias is about deliberately not fully including results. Although many parameter settings across different ranges were tested for both algorithms, only three parameter settings results were included as they were the best among the tested ones. The experiment was to see if the different parameter settings yield different results and select the best among them.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

The thesis aims to develop a prioritized database synchronization framework that mainly focuses on prioritizing the order of rows for synchronization with the destination database. The features of the framework include

1. Fetches data from the source database using PyMongo and performs the basic data transformation.

2. Prioritizing the rows based on their attribute values

3. Effective synchronization using MongoDB **'watch'** method to listen to all the changes made

4. Handles connection errors and retries for the synchronization operation after an exponential delay.

5. Handles concurrent changes using OCC.

We can conclude that optimization algorithms have also performed well regarding the fitness values over generations, convergence speeds and elapsed time. Still, the genetic algorithm outperformed the particle swarm algorithm and yielded better results. It is evident that different parameter settings have a significant effect on the algorithm's performance. Altering these can change the results of the algorithms. The overall framework, when implemented together, as seen in Figure 5.2, works well.

8.2 Future Work

Genetic algorithm and particle swarm optimization were used in this research, so considering other optimization algorithms or another method for prioritizing the documents/rows could be a potential future work.

The synchronization framework developed for the thesis is more generalized and would take considerable time to transform into an industry-ready framework and handle all the connections.

While the synchronization framework works effectively when there is a good connection, it might struggle and raise errors if there are continuous disruptions in the network. It is to be noted that the devised synchronization framework does not handle different network conditions. Making the framework to handle the intermittent connections is a possible future scope, given that the devised synchronization framework in this thesis is a simple generalized synchronization framework focused on prioritizing the order of data for sync.

In the devised framework, the algorithm fetches data from the source database and returns the optimal order of data ready for sync back to the source database, but the different collection; this puts additional load and occupies additional space on the source database. Minimizing the load on the source database would be a potential future work.

One potential solution could be using MQTT protocol [18], making the algorithm directly send the data to MQTT which not only puts less load on the source database but also effectively transmits and receives the data over a constrained network with only limited bandwidth. This thesis did not consider the approach, considering the scope and the constrained time.

References

[1] L. Abualigah, M. Shehab, M. Alshinwan, and H. Alabool, "Salp swarm algorithm: a comprehensive survey," *Neural Computing and Applications*, vol. 32, pp. 11 195–11 215, 2020.

[2] M. Ahluwalia, R. Gupta, A. Gangopadhyay, Y. Yesha, and M. McAllister, "Target-based database synchronization," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, 2010, pp. 1643–1647.

[3] T. A. Alhaj, M. M. Taha, and F. M. Alim, "Synchronization wireless algorithm based on message digest (swamd) for mobile device database," in *2013 INTERNATIONAL CONFERENCE ON COMPUTING, ELECTRICAL AND ELECTRONIC ENGINEERING (ICCEEE)*, 2013, pp. 259–262.

[4] M. Ali, M. Pant, and V. Singh, "Two modified differential evolution algorithms and their applications to engineering design problems," *World J Model Simul*, vol. 6, no. 1, pp. 72–80, 2010.

[5] A. Bajaj and O. P. Sangwan, "A systematic literature review of test case prioritization using genetic algorithms," *IEEE Access*, vol. 7, pp. 126 355–126 375, 2019.

[6] V. Balakumar. and I. Sakthidevi., "An efficient database synchronization algorithm for mobile devices based on secured message digest," in *2012 International Conference on Computing, Electronics and Electrical Technologies (ICCEET)*, 2012, pp. 937–942.

[7] E. Cecchet, *Middleware Support for Database Replication and Caching*. Boston, MA: Springer US, 2009, pp. 1738–1743. [Online]. Available: https://doi.org/10.1007/978-0-387-39940-9_1538

[8] M.-Y. Choi, E.-A. Cho, D.-H. Park, C.-J. Moon, and D.-K. Baik, "A database synchronization algorithm for mobile devices," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 2, pp. 392–398, 2010.

[9] J. Domingos, N. Simões, P. Pereira, C. Silva, and L. Marcelino, "Database synchronization model for mobile devices," in *2014 9th Iberian Conference on Information Systems and Technologies (CISTI)*, 2014, pp. 1–7.

[10] M. Dorigo and T. Stützle, *Ant colony optimization: overview and recent advances*. Springer, 2019.

[11] G. M. Habtemariam and S. K. Mohapatra, "A genetic algorithm-based approach for test case prioritization," in *Information and Communication Technology for*

Development for Africa, F. Mekuria, E. Nigussie, and T. Tegegne, Eds. Cham: Springer International Publishing, 2019, pp. 24–37.

[12] K. M. Hamdia, X. Zhuang, and T. Rabczuk, "An efficient optimization approach for designing machine learning models based on genetic algorithm," *Neural Computing and Applications*, vol. 33, pp. 1923–1933, 2021.

[13] N. T. Hanh, H. T. T. Binh, N. X. Hoai, and M. S. Palaniswami, "An efficient genetic algorithm for maximizing area coverage in wireless sensor networks," *Information Sciences*, vol. 488, pp. 58–75, 2019.

[14] S. Harikarthik, V. Palanisamy, and P. Ramanathan, "Optimal test suite selection in regression testing with testcase prioritization using modified ann and whale optimization algorithm," *Cluster Computing*, vol. 22, no. 5, pp. 11 425–11 434, 2019.

[15] K. H. S. Hla, Y. Choi, and J. S. Park, "Applying particle swarm optimization to prioritizing test cases for embedded real time software retesting," in *2008 IEEE 8th International Conference on Computer and Information Technology Workshops*. IEEE, 2008, pp. 527–532.

[16] M. I. Hossain and M. M. Ali, "Sql query based data synchronization in heterogeneous database environment," in *2012 International Conference on Computer Communication and Informatics*. IEEE, 2012, pp. 1–5.

[17] C. Huang, M. Cahill, A. Fekete, and U. Röhm, "Data consistency properties of document store as a service (dsaas): Using mongodb atlas as an example," in *Performance Evaluation and Benchmarking for the Era of Artificial Intelligence: 10th TPC Technology Conference, TPCTC 2018, Rio de Janeiro, Brazil, August 27–31, 2018, Revised Selected Papers 10*. Springer, 2019, pp. 126–139.

[18] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "Mqtt-s—a publish/subscribe protocol for wireless sensor networks," in *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08)*. IEEE, 2008, pp. 791–798.

[19] M. Isiet and M. Gadala, "Sensitivity analysis of control parameters in particle swarm optimization," *Journal of Computational Science*, vol. 41, p. 101086, 2020.

[20] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimedia Tools and Applications*, vol. 80, pp. 8091–8126, 2021.

[21] K. Kottursamy, G. Raja, J. Padmanabhan, and V. Srinivasan, "An improved database synchronization mechanism for mobile data using software-defined networking control," *Computers Electrical Engineering*, vol. 57, pp. 93–103, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790616300064

[22] K. Ma and B. Yang, "Log-based change data capture from schema-free document stores using mapreduce," in *2015 International conference on cloud technologies and applications (CloudTech)*. IEEE, 2015, pp. 1–6.

[23] S. Mahajan, S. D. Joshi, and V. Khanaa, "Component-based software system test case prioritization with genetic algorithm decoding technique using java platform," in *2015 International Conference on Computing Communication Control and Automation*. IEEE, 2015, pp. 847–851.

[24] MongoDB, "Why use mongodb?" https://www.mongodb.com/why-use-mongodb, accessed: May 4, 2023.

[25] S. Nayak, C. Kumar, S. Tripathi, N. Mohanty, and V. Baral, "Regression test optimization and prioritization using honey bee optimization algorithm with fuzzy rule base," *Soft Computing*, vol. 25, no. 15, pp. 9925–9942, 2021.

[26] A. G. Nikolaev and S. H. Jacobson, "Simulated annealing," *Handbook of metaheuristics*, pp. 1–39, 2010.

[27] K. Ogunyale, "Understanding the genetic algorithm," *https://medium.com/@kennyrich/understanding-the-genetic-algorithm-4eac04a07a59*, 2019.

[28] G. Raja, K. Kottursamy, S. H. Chaudhary, A. Hassan, and M. Alqarni, "Sdn assisted middlebox synchronization mechanism for next generation mobile data management system," in *2017 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computed, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, 2017, pp. 1–7.

[29] A. Ratnaweera, S. K. Halgamuge, and H. C. Watson, "Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients," *IEEE Transactions on evolutionary computation*, vol. 8, no. 3, pp. 240–255, 2004.

[30] M. Ray and D. P. Mohapatra, "Multi-objective test prioritization via a genetic algorithm," *Innovations in Systems and Software Engineering*, vol. 10, no. 4, pp. 261–270, 2014.

[31] J. Scholz, "Genetic algorithms and the traveling salesman problem a historical review," *arXiv preprint arXiv:1901.05737*, 2019.

[32] A. Shukla, H. M. Pandey, and D. Mehrotra, "Comparative review of selection techniques in genetic algorithm," in *2015 international conference on futuristic trends on computational analysis and knowledge management (ABLAZE)*. IEEE, 2015, pp. 515–519.

[33] C. Srinivas, B. R. Reddy, K. Ramji, and R. Naveen, "Sensitivity analysis to determine the parameters of genetic algorithm for machine layout," *Procedia materials science*, vol. 6, pp. 866–876, 2014.

[34] D. Srivatsa, T. K. Teja, I. Prathyusha, and G. Jeyakumar, "An empirical analysis of genetic algorithm with different mutation and crossover operators for solving sudoku," in *Pattern Recognition and Machine Intelligence: 8th International Conference, PReMI 2019, Tezpur, India, December 17-20, 2019, Proceedings, Part I*. Springer, 2019, pp. 356–364.

[35] P. Tonella, A. Susi, and F. Palma, "Interactive requirements prioritization using a genetic algorithm," *Information and software technology*, vol. 55, no. 1, pp. 173–187, 2013.

[36] M. Tyagi and S. Malhotra, "Test case prioritization using multi objective particle swarm optimizer," in *2014 International Conference on Signal Propagation and Computer Technology (ICSPCT 2014)*. IEEE, 2014, pp. 390–395.

[37] J. Vesterstrom and R. Thomsen, "A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems," in *Proceedings of the 2004 congress on evolutionary computation (IEEE Cat. No. 04TH8753)*, vol. 2. IEEE, 2004, pp. 1980–1987.

[38] J. Wang and L. Huang, "Evolving gomoku solver by genetic algorithm," in *2014 IEEE workshop on advanced research and technology in industry applications (WARTIA)*. IEEE, 2014, pp. 1064–1067.

[39] G. Wen, M. Z. Chen, and X. Yu, "Event-triggered master–slave synchronization with sampled-data communication," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 63, no. 3, pp. 304–308, 2015.

[40] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[41] Z. Xiao-Jun, Y. Chun-Hua, G. Wei-Hua, and D. Tian-Xue, "A particle swarm optimization algorithm with variable random functions and mutation," *Acta Automatica Sinica*, vol. 40, no. 7, pp. 1339–1347, 2014.

[42] X.-S. Yang and S. Deb, "Cuckoo search via lévy flights," in *2009 World congress on nature & biologically inspired computing (NaBIC)*. Ieee, 2009, pp. 210–214.

[43] M. Yousefi, M. Omid, S. Rafiee, and S. Ghaderi, "Strategic planning for minimizing co2 emissions using lp model based on forecasted energy demand by pso algorithm and ann," *International Journal of Energy and Environment (Print)*, vol. 4, 2013.

Faculty of Faculty, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden