DMBench: Load Testing and Benchmarking Tool for Data Migration

Fares Hamouda

A Thesis Submitted to the Faculty of Graduate Studies
In Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science

Graduate Program in Electrical Engineering and Computer Science

York University
Toronto, Ontario

August 19, 2024
©[Fares Hamouda], [2024]

Abstract

Data migration refers to the set of tasks around transferring data over a network between two systems, either homogeneous or heterogeneous, and the potential reformatting of this data. Combined with large volumes of data, resource constraints and variety in data models and formats, data migration can be critical for enterprises, as it can consume a significant amount of time, incur high costs, and pose a significant risk if not executed correctly. The ability to accurately and effectively predict these challenges and plan for proper resource, time and budget allocation is vital for the proper execution of data migration. In this work, we introduce the concept of load testing and benchmarking for data migration to allow decision-makers for higher efficiency and effectiveness when planning for such tasks. Our framework aims for extensibility and customizability to enable the execution of a greater variety of tests. Here, we present a prototype architecture, a roadmap of how the development of such a platform should proceed and a simple case study of how it can be used in practice.

Acknowledgment

To my Parents, I am grateful for your constant support. Your prayers and kind messages have been my daily source of strength and motivation.

To Feriel, thank you for always being by my side. Your endless support especially during challenging times, has been precious.

To Professor Marios, working with you has been an absolute pleasure. Advancing in a healthy environment, exchanging ideas for the only purpose of reaching the best solution is what has been motivating me to work on this project.

I would also like to thank Professor Shahin Kamali , Professor Jarek Szlichta and Professor Arash Habibi Lashkari for accepting to be part of my examining committee. Your willingness to evaluate my work is greatly appreciated.

With all my heart, I would like to thank all my friends from all over the world who wanted to be part of my defense and share in my happiness with me. Your support means the world to me.

Finally, I would like to thank Mitacs and York University for the exceptional opportunity and financial support to pursue this prestigious degree.

Preface

This research project has been conducted in collaboration with our industrial partner IBM, specifically working with Dariusz Jania. The project, titled "1149 - An Intelligent and Dynamic Planner for Adaptive Migration of Big Data Systems," has been carried out under the expert guidance and financial support of Professor Marios Fokaefs.

The work presented in this thesis has also been generously funded by several esteemed organizations. I gratefully acknowledge the financial support provided by Mitacs through the Globalink Graduate Fellowship, the NSERC CREATE Program in Dependable Internet of Things Applications (DITA), and York University.

Furthermore, part of this research has been accepted and presented at the prestigious ICPE 2024 conference in London. The paper, titled "DMBench: Load Testing and Benchmarking Tool for Data Migration" , was co-authored by Fares Hamouda, Marios Fokaefs, and Dariusz Jania, and published by the Association for Computing Machinery.[1]

Contents

List of Tables

List of Figures

1 Introduction

Technological advancements and paradigm shifts have long motivated large-scale migrations of software and data systems to different and new platforms [3]. One such advancement was cloud computing, which offered significantly larger capacity both in computational power and storage, revolutionizing how enterprises manage and process data [4]. More recently, the rise of distributed and large-scale data models, such as data lakes, has spurred another wave of data migration [5]. These migrations are driven by the need to leverage advanced capabilities, enhance efficiency, scalability, and cost-effectiveness in managing large volumes of data, and respond to evolving business requirements.

Data migration, defined as the process of transferring data between storage types, formats, or computer systems, is essential in this context. It ensures that data remains accessible and usable across different platforms and technologies. However, this task is fraught with challenges. The sheer volume of enterprise data puts immense strain on networks during the transfer process. This large data volume increases the probability of errors and failures, making reliability a critical concern [6]. Furthermore, the migration process itself can be complex, requiring meticulous planning and execution to avoid data loss, corruption, or prolonged downtime.[7]

Another layer of complexity is introduced by the diversity of legacy and new target systems. Migrating data across different systems often requires significant modifications to existing migration strategies and tools. This diversity necessitates comprehensive planning and robust strategies to ensure data integrity, security, and compliance with various regulations throughout the migration process[8, 9]. For instance, migrating from an on-premises data warehouse to a cloud-based data lake involves not only technical adjustments but also considerations for data governance and access control policies [10].

When considering data migration within a business context, it is crucial to account for how the task is planned and the impact it can have on the enterprise's operations. Migration is typically viewed as a maintenance task that does not generate direct value. However, it demands substantial resources, including infrastructure, time, and personnel, which might otherwise be allocated to value-generating activities [8]. Effective planning must address potential downtime, ensure minimal disruption to business activities, and incorporate comprehensive risk management strategies [11]. Additionally, the financial implications of migration projects cannot be overlooked, as they can significantly affect the company's budget and resource allocation.

The stakes are even higher for businesses operating in highly regulated industries, such as healthcare or finance, where compliance with legal and regulatory standards is mandatory. In such cases, ensuring that data migration processes adhere to relevant regulations (e.g., GDPR, HIPAA)[12] becomes paramount . This includes maintaining accurate audit trails, ensuring data encryption during

transit, and implementing robust access controls .

To address these challenges, we propose the development of a performance testing platform that allows for easy, fast, and flexible experimentation with different migration scenarios and configurations. This platform aims to simulate various conditions and constraints, providing data for optimal decision-making. By enabling the testing of multiple migration strategies in quick succession, the platform can identify the most efficient and reliable approaches tailored to specific organizational needs. This approach not only mitigates risks but also provides actionable insights that can inform future migration projects.

The contributions include:

- A versatile tool for data migration performance testing that can be easily adapted to diverse scenarios and requirements, available publicly as an open-source project on GitHub [1] , with comprehensive documentation to help configure and run experiments.
- Support for various migration engines, offering users a wide range of options to suit their specific needs. This includes engines optimized for different types of data, such as structured, semi-structured, and unstructured data.
- A case study on the performance of a large-scale real-world system in different scenarios, providing practical insights of the performance testing platform. This study will highlight how different configurations impact performance, reliability, and cost-efficiency.

Potential Impact In practice, the tool provides an automated process that helps users determine which migration engine and configuration best suit their needs, ensuring efficient and informed decision-making. For research, the tool enables extensive data collection, which can be used to create models predicting the most suitable migration engines. These models can be further refined through machine learning algorithms, allowing for continuous improvement and adaptation to emerging technologies and methodologies.

Additionally, the tool can serve as the basis for dynamically adaptive migration engines, allowing for real-time adjustments and optimizations based on ongoing performance metrics and environmental changes. This dynamic adaptability is crucial in today's fast-paced digital landscape, where businesses must be able to respond quickly to changes in technology and market demands. Moreover, the tool's open-source nature fosters a collaborative environment where researchers and practitioners can contribute enhancements, share insights, and develop new features that address evolving challenges in data migration.

Ultimately, the proposed performance testing platform aims to bridge the gap between theoretical research and practical application, providing a valuable resource for both academia and industry.

[1] https://github.com/yorku-ease/DataMigrationBenchmarkingTool/tree/main/deployment

By facilitating better planning, execution, and optimization of data migration projects, these contributions can help organizations navigate the complexities of digital transformation with greater confidence and success.

2 Related Work

This section reviews the existing literature on data migration, focusing on migration tools, benchmarking and performance testing tools, and the performance of migrations.

2.1 Data Migration Techniques and Tools

Data migration is a critical process in moving data from one system to another, often involving complex procedures to ensure data integrity and security. Hussien [7] explores various strategies, methodologies, categories, and risks associated with data migration for cloud environments. He introduces the DMig 1 model, which enhances security during migration through encryption techniques. This research highlights the necessity of secure migration practices, which our platform can evaluate by benchmarking DMig 1 against other secure migration methods.

Shakya [13] presents a solution for secure data transfer in cloud computing environments, focusing on performance and data transfer security during migrations. This work is particularly relevant to understanding secure data transfer techniques that can inform future enhancements of our framework, even though our current focus is on performance.

Iqbal et al. [14] conducted a multivocal literature review on data migration in cloud environments, identifying key opportunities and challenges. They emphasize that there is no universally best migration technique; instead, the choice of technique depends on the specific requirements of the project. This insight aligns with the objective of our platform, which aims to help users compare and select the optimal migration approach based on their unique needs.

Mohammad [15] highlights the importance of evaluating migration tools based on data integrity and cost optimization criteria. This is directly relevant to our work, as our performance testing platform takes into consideration the overall cost-effectiveness of various migration strategies, helping users make informed decisions.

Dewi et al. [16] present an architecture for scalable data migration using data synchronization and transactional replication techniques. Their approach to maintaining data consistency and scalability during migration guides our platform's objective to provide robust support for various data types and migration scenarios, ensuring that data remains accurate and synchronized throughout the process.

Astrand [17] explores the development of data migration tools for software re-engineering, which is relevant to our platform's goal of supporting the development and testing of new migration tools and strategies.

2.2 Benchmarking and Performance Testing Tools

Benchmarking and performance testing are essential for evaluating the efficiency and effectiveness of data migration tools. The need for comprehensive performance comparisons is underscored by cloud migration benchmarking reports, such as those from DBTA [18]. These reports suggest expanding performance comparisons to include a broader range of migration scenarios, beyond just cloud-specific tools, to create a more versatile performance testing platform.

Research by EuroSTAR Huddle [19] delves into performance testing techniques specifically for database migrations. They stress the importance of capturing performance metrics like response times and throughput before and after migration. This aligns with our platform's focus on performance testing, as pre-migration performance evaluations can inform scenario selection and identify potential bottlenecks.

Understanding the factors influencing migration performance is crucial for optimizing data migrations. The 2021 Hadoop-to-Cloud Migration Benchmark Report [20] provides insights into the challenges faced during Hadoop environment migrations to the cloud, such as network bandwidth and data volume. While our platform offers a broader scope, encompassing various data sources and migration scenarios, this report provides valuable background on specific challenges encountered during data migrations.

Puliafito et al.[21] present MobFogSim, a simulation toolkit for modeling mobility and migration in fog computing environments, discussing the impact of dynamic migration on performance. This toolkit's insights are relevant to our platform's ability to test different migration strategies and their performance impacts.

2.3 Migration Performance Analysis

In addition to focusing on database migration, we also explored other works related to migration performance in various contexts. These studies provide valuable insights that can enhance our understanding and approach to optimizing data migration performance.

Shirvani et al. [22] explore various VM migration and server consolidation techniques in DVFS-enabled cloud data centers. Their findings on optimizing resource utilization and performance during VM migrations are particularly relevant to our platform's ability to test different migration strategies and their performance impacts, ensuring efficient resource use during migrations.

Shafiq et al. [23] review load balancing techniques in cloud computing environments, which are crucial for improving the performance of cloud services during data migration. Considering these load balancing techniques can help ensure minimal disruption and maintain high performance during the migration process, aligning with our objective of efficient migrations.

Aslanpour et al. [24] provide a comprehensive review of performance evaluation metrics for cloud,

fog, and edge computing, highlighting benchmarks and standards for future research. This review supports our platform's development by identifying key performance metrics necessary for comprehensive testing, enabling us to establish robust benchmarks for migration performance.
Karthikeyan et al. [25] analyze energy consumption in virtual machine migration using hybrid swarm optimization techniques. Understanding the techniques for measuring energy consumption impacts is useful for our performance testing platform, helping us evaluate and optimize energy efficiency during migrations.

2.4 Conclusion

In summary, existing research highlights the need for a comprehensive data migration performance testing platform. Current studies underscore the importance of secure and efficient data migration techniques, the necessity of robust benchmarking and performance testing tools, and the critical factors influencing migration performance. Our platform aims to address these needs by offering a generalized framework for experimenting with diverse migration approaches across various data sources and cloud environments. By providing comparative analysis, it guides users towards the optimal migration techniques tailored to their specific requirements.

3 Background

This section combines the provided information on data migration, containerization, and related research to establish the context for the proposed framework.

3.1 Data Migration: Source, Target, and Engines

Data migration is the process of transferring data between storage systems, databases, applications, or even business processes . This typically involves a **source** where the data resides initially, and a **target** where it will be transferred to[26, 27, 28, 29]. A critical component in this process is the **migration engine**, responsible for orchestrating the data movement. These engines can be deployed on-premises or in the cloud.

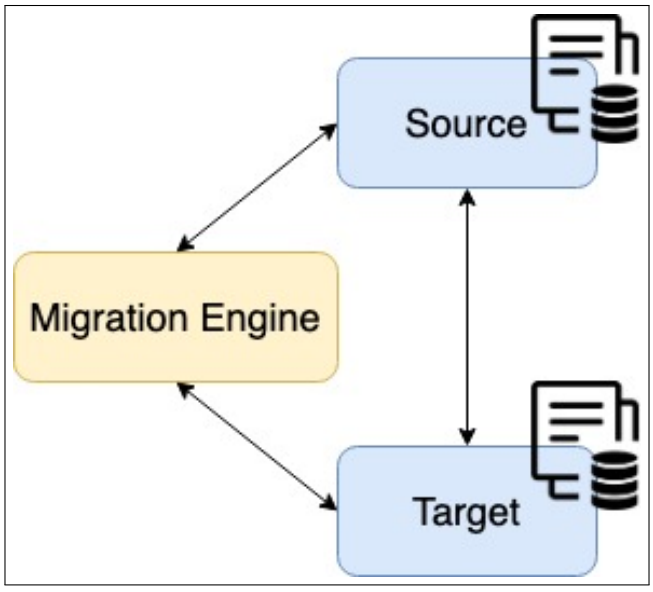

Figure 1: Data Migration Architecture Diagram.

Examples of migration engines include:

- **Db2 Migration Service** : This is a dedicated service from IBM for migrating data between Db2 databases [30]. Db2 itself is a relational database management system offered by IBM [31].
- **Custom Python Scripts** : Python [32] scripts leverage libraries like Paramiko[33] to facilitate data migration. Paramiko[33] specializes in Secure Shell (SSH[34]) connections, enabling secure file transfer between the source and target machines. These scripts can be used to dump a source database (e.g., MySQL[35]) into a file on the target machine and then

import it into the target database system. Encapsulating this entire process within a container ensures portability and simplifies deployment across various environments.

3.2 Containerization

Containerization[36] plays a vital role in the framework. By bundling migration engines, their dependencies, and configuration files into containers, we achieve several benefits:

- **Isolation**: Containers ensure each engine operates independently, preventing conflicts with other applications.
- **Resource Efficiency**: Containers are lightweight and use resources efficiently.
- **Deployment Flexibility**: Containerization allows for flexible deployment of migration engines. They can run on the same machines as source/target data machines, or on separate machines depending on user infrastructure and security constraints.

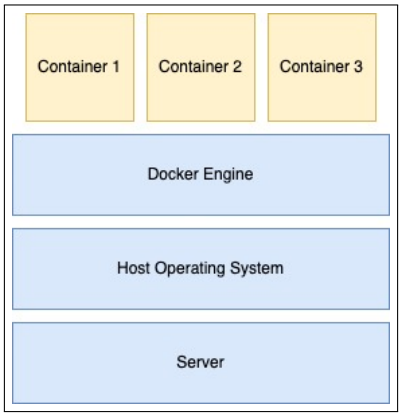

Figure 2: Docker Container Architecture Diagram

3.3 Message Broker : Kafka

In the context of data migration, a message broker like Kafka can be a crucial component for handling large data transfers. Apache Kafka is a distributed event streaming platform capable of handling high-throughput, fault-tolerant, and scalable messaging. It centralizes communication, decouples producers and consumers, and ensures reliable data transfer, thereby preventing data loss during migration processes.[37]
Kafka operates based on the concepts of producers, consumers, topics, and partitions:

- **Producers**: These are the applications or systems that send data to Kafka. In a data migration scenario, the producer could be the system exporting data from the source database.

- **Consumers**: These are the applications or systems that receive data from Kafka. In this case, the consumer could be the system importing data into the target database.
- **Topics**: Topics in Kafka are logical channels to which data is written by producers and from which data is read by consumers. Each topic can be divided into multiple partitions.
- **Partitions**: Partitions are a way to parallelize data processing in Kafka. Each partition can be hosted on a different server, allowing for high throughput and fault tolerance.

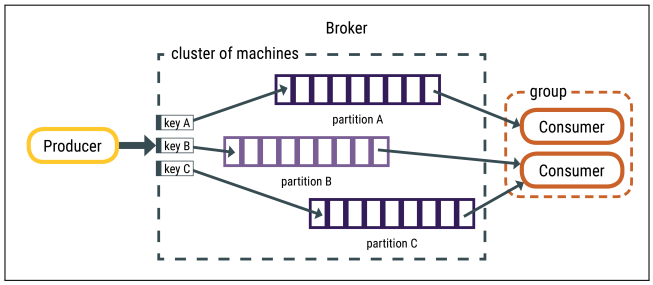

Figure 3: Apache Kafka Architecture Diagram, from [2]

3.4 Resource Monitoring

Monitoring resource consumption (disk, CPU, memory) of the migration engine is important, especially when dealing with infrastructure constraints. Effective resource monitoring helps in identifying bottlenecks, optimizing resource usage, and ensuring that the migration process does not negatively impact other applications running on the same infrastructure.

3.4.1 Tools for Resource Monitoring

cAdvisor [38]: cAdvisor (Container Advisor) is a tool developed by Google that provides insights into resource usage and performance characteristics of running containers. It collects, aggregates, processes, and exports information about running containers. cAdvisor offers detailed information about resource utilization within the container itself, including CPU, memory, disk, and network statistics. This granularity allows for precise monitoring and troubleshooting of individual containers, making it an essential tool for containerized environments.

node-exporter [39]: Node Exporter is a Prometheus exporter for hardware and OS metrics exposed by *nix kernels, written in Go with pluggable metric collectors. While node-exporter focuses on host-level metrics, it can still be valuable for monitoring the overall system health during the migration process. It provides metrics on CPU usage, memory usage, disk I/O, network I/O, and

more, which are crucial for understanding the performance and health of the physical or virtual machines running the containers.

Prometheus [40]: Prometheus is an open-source systems monitoring and alerting toolkit originally built at SoundCloud. Since its inception, it has been adopted by many organizations and has a vibrant ecosystem. Prometheus acts as the central repository for storing metrics collected from various exporters like cAdvisor and node-exporter. It supports a powerful query language (PromQL) to aggregate and query the collected metrics, making it possible to create complex queries to analyze resource consumption patterns and identify potential issues.

Grafana [41]: Grafana is an open-source platform for monitoring and observability. It allows you to query, visualize, alert on, and understand your metrics no matter where they are stored. Grafana visualizes data collected by Prometheus for monitoring resource consumption and setting up alerts. It provides a highly customizable dashboard interface where users can create and share dashboards to monitor key metrics. Grafana supports alerting rules that can notify users via various channels (email, Slack, etc.) when certain thresholds are exceeded, helping to proactively manage resource utilization and maintain system health.

4 DMBench: A Performance Testing Tool For Data Migration

4.1 Motivation

The complexity of data migration processes arises from the intricate interplay of various factors, including resource allocation, middleware functionality, and infrastructure heterogeneity. Optimal resource allocation is essential to minimize costs, migration time, and downtime, highlighting the critical role of the middleware in streamlining the migration process. To enhance efficiency, middleware solutions may employ parallelization techniques and employ architectures similar to MapReduce for data unloading, processing, and reloading. Furthermore, the distributed nature of modern infrastructures necessitates an adaptive middleware that is capable of dynamically adjusting to changing conditions during migration. Therefore, we seek to develop a decision-support system to optimize migration parameters and infrastructure configurations, ensuring efficient and reliable data migration across diverse environments.

4.2 Framework Overview

DMBench is designed to be primarily functional, usable and portable, but also flexible and extensible. To this end, it consists of multiple semi-independent modules with distinct roles that can be easily configured individually or as a whole, and can be replaced by alternative implementations with ease. The use of Docker containers makes the platform easy to deploy and redeploy at will and on demand. Next, we provide more details about the properties and relations between these modules.

The primary objective of DMBench is to allow testers and decision-makers to design and execute migration experiments as efficiently and as effectively as possible. Such a platform should allow for easy configuration of different experiments, fast deployment and execution of the experiment and fully automated and comprehensive presentation of the results. In principle, the platform should enable testers to execute as many "what-if" scenarios as possible to compare many alternatives or confirm many hypotheses. In practice, the platform guides the tester to prepare an environment to send data from one machine to another (potentially remote) with different types of configurations, and then monitors each part of the process.

DMBench has been designed according to a set of principles on the functional and non-functional requirements of a data migration testing tool, as described below.

4.2.1 Functional Requirements:

- **Easy setup:** The setup for a migration task refers to all steps relevant to preparing the source and target machines, the migration engine, any monitoring tools (e.g., logging or monitoring)

along with respective databases to save results, and the controller of the migration test or experiment. Setup can be a time-consuming step, especially when it is not automated. The ability to quickly complete the setup and make it easy to share between experiments is vital to the ability of testing multiple scenarios and configurations.

- **Easy configuration:** Numerous parts of the platform and of an experiment need to be configurable to allow for extensive control to the tester. If the configuration is complex, inconsistent or generally lacking, it may render experimentation results unusable.

- **Convenient and complete access to results:** For any benchmark to be useful, it needs to provide the results in a convenient way to enable further analysis and interpretation. While this can include reports with visualizations and descriptive statistics, it is important to also return all raw measurements and results in a format that can be easily digested by an analysis software.

4.2.2 Non-functional Requirements

- **Usability:** The user of the tool is expected to have some basic understanding of data migration and potentially of the source and target systems, and the data to be transferred. Besides that, the tools should hide as many details about the experimentation infrastructure as possible and expose any configuration or input points through a clear and easy-to-use system. The proposed framework follows a simplified approach, allowing users to initiate experiments with a few straightforward commands after the environment is set up and configured. The objective is to streamline the process, ensuring ease of use while maintaining simplicity.

- **Extensibility:** Through the configuration files, the user can provide outside input to the framework and provide any migration engine he wants to test by dockerizing it following the process proposed. In addition, all other modules, including logging and monitoring, are designed to be easily changeable in the future without requiring changes to the entire framework.

- **Accuracy and Reliability:** When configuring an experiment, the user has the possibility to request for each experiment to be executed a given number of times. By repeating the exact same experiment multiple times and averaging over the results of the iterations, we can ensure that any source of variability is excluded, and the results are accurate and reliable. In many analyses, it is required to perform statistical testing to confirm or reject the initial hypothesis. By repeating the experiments multiple times, we make it possible to run such tests with high confidence. No matter the number of repetitions, experiments in DMBench are executed deterministically and any variations originates from the environment or the use

case and not from the tool. Through complete access to the results, this is verifiable by the tester.

4.2.3 Architecture

Figure 4 shows the main components of DMBench, which are described in detail next. The framework is composed of the yellow components, while the blue components are exterior to the framework. The primary role of the framework is to test the Migration Engine within the blue components and assess its performance.

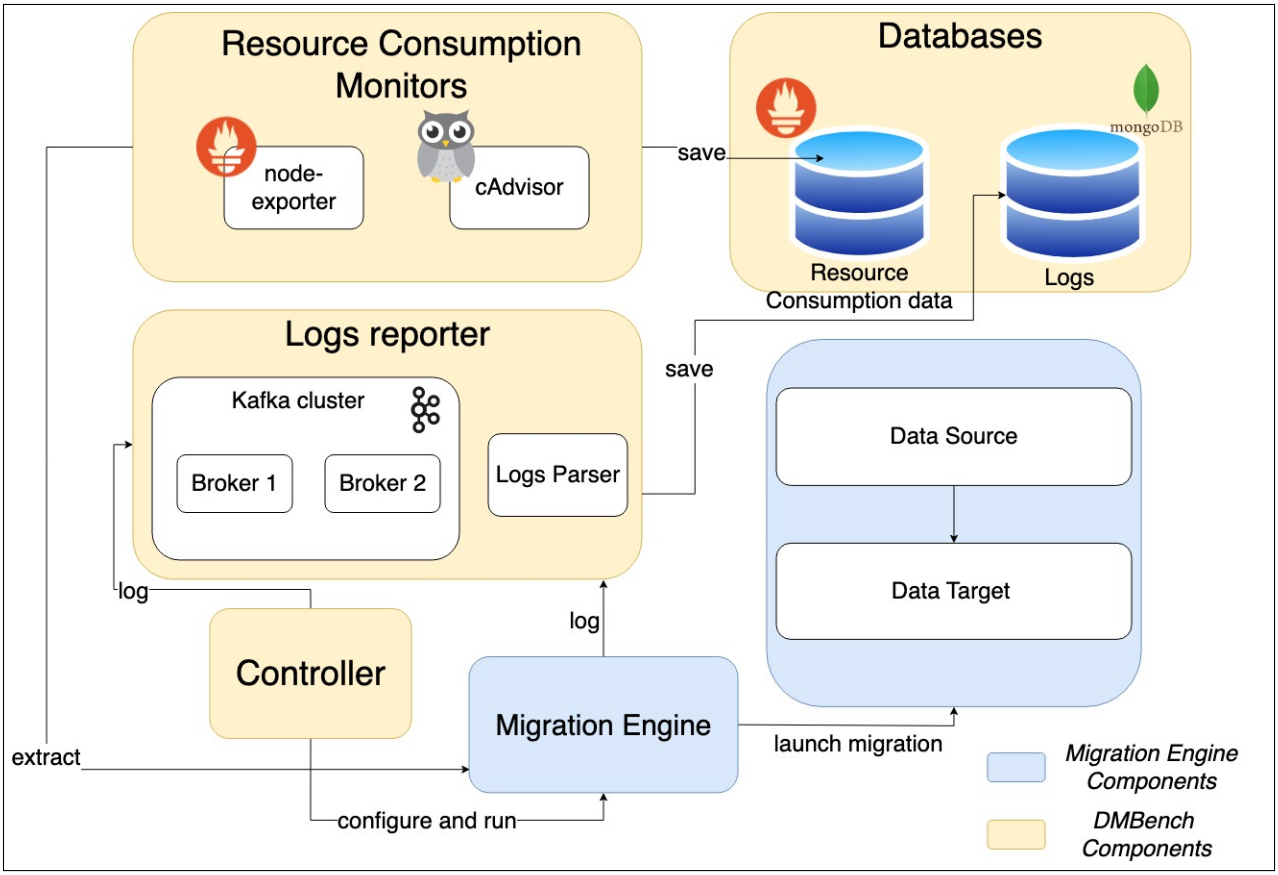

Figure 4: The general architecture of DMBench.

- **Migration Engine:** Central to the framework, the Migration Engine serves as the focal point for all benchmarking activities, but is primarily the module responsible for transferring data from the source to the target. All other components within the framework are designed to closely monitor and assess the performance of the engine. Users are afforded the flexibility to select any migration engine of their preference. Whether the chosen engine operates within a singular service architecture or spans multiple services, the only prerequisite is the

dockerization of the selected engine. The framework seamlessly manages the intricacies of the benchmarking process, offering a streamlined and user-friendly experience.

- **Data Source:** This component serves as the starting point for the Migration Engine to transfer data from here to a designated target machine. DMBench requires only an IP address and appropriate credentials to connect to the Data Source machine, but the specific deployment method (virtual machine, physical server, container) is irrelevant to the platform. The Data Source is usually a database system

- **Data Target:** This component is the destination for data transferred by the Migration Engine from the source machine. Similar to the sources, only an IP address and connection credentials are necessary. The Data target is usually a database system

- **Controller:** Responsible for orchestrating all experiments under consistent conditions, the Controller configures the migration engine for each experiment with varying parameters. It initiates and oversees the execution of the migration process, monitoring the engine's performance through recording and analysis of migration logs. Simultaneously, the Controller tracks resource consumption by deploying `cAdvisor` [38]and `node-exporter` [39] on the migration infrastructure.

- **Metrics & Logs Databases:** DMBench uses two databases for the monitored data. The first, a time series database based on `Prometheus` [40], aggregates resource consumption data collected from `cAdvisor` [38]. The second database, a `MongoDB` [42] instance, serves as the repository for all log data generated during the experiments. These databases work in tandem, with `Prometheus` [40] focusing on resource metrics and the second database storing all logs from both the framework and the migration engine, it ensures comprehensive and organized storage of experiment results.

- **Logs reporter:** Comprising two integral components, the Logs Reporter ensures a robust and organized handling of experiment logs. The first component involves a `Kafka` cluster, serving as the repository for all logs. Both the Controller and the Migration Engine publish their logs to `Kafka`, with a dedicated consumer responsible for retrieving and temporarily storing these logs in local files.

  The second component is the parser, which not only extracts data from the logs but also transforms it into a human-readable format. The parsed information is then exported into CSV files before being permanently stored in a `NoSQL` database, as mentioned above. This dual-component approach ensures a seamless and efficient process for managing, interpreting, and extracting insights from the experiment logs.

4.3 Framework Components Deep Dive

In this section, we will take a closer look at some of the key components of the framework, examining their roles, configurations, and interactions in detail. This detailed analysis will provide a deeper understanding of how each component contributes to the overall functionality and performance of the DMBench framework.

4.3.1 Databases

For the machine designated to host the databases, the necessary configuration involves modifying the ‘prometheus.yml‘ file. Specifically, **Prometheus**[40] needs to be given the IP address of the machine where the controller is deployed, as the migration engine will be deployed on that same machine. These modifications ensure that **Prometheus** can correctly scrape data from the specified endpoints, enabling it to monitor the necessary components effectively.
Both **Prometheus** and **MongoDB**[42] have Docker Compose files, which make their deployment straightforward once they are configured. After adjusting the configuration files to specify the correct endpoints and settings, the databases can be deployed with ease using Docker Compose.
Additionally, we have a **Grafana**[41] dashboard that is deployed alongside **Prometheus** and works closely with it. The **Grafana** dashboard is pre-configured to display real-time graphs and historical data of past experiments, providing comprehensive visualization and monitoring capabilities.

4.3.2 Logs Reporter

We have chosen **Kafka**[37] for the logs reporter due to its scalability and fault tolerance. **Kafka** can be deployed with multiple brokers, ensuring high availability and reliability. Docker Compose files have been set up to facilitate easy deployment. **Kafka** uses the credentials of the **MongoDB** database to know where to send all logs.
A consumer is set up to subscribe to multiple topics: ‘performancebenchmark‘, ‘migrationengine‘, and ‘framework‘. Each of these topics corresponds to a different type of log. Logs for the framework and migration engine are categorized under various types, such as info, debug, and errors. This categorization allows for easy troubleshooting if issues arise during the experiments.
For each topic, we have developed a parser to process the logs and store them in the database. Additionally, there is a need to have a parser for each migration engine to make better use of these logs. The deployment of the **Kafka** cluster is straightforward, as Docker Compose files are ready, and scripts for consumers and parsers are prepared to be run with a simple Python command.

4.3.3 Controller

The Controller utilizes a pivotal configuration file named ‘config.ini‘, crucial for providing essential settings to the Migration Engine. This configuration file is of paramount importance, guiding users in the dockerization of their migration engine.
The ‘config.ini‘ file consists of two integral parts:
First Part: This section is transmitted unaltered to the Migration Engine. Users can add as many key-value pairs as needed in this section to clearly identify the requirements for the migration engine. This part may contain information about the target server, source server, or any other necessary details to run the migration engine. It also includes information about the logs reporter so that the migration engine knows where to publish the logs. Additionally, it contains essential settings for the controller, such as the Docker image of the migration engine and the number of experiments, which denotes how many times an experiment should be repeated to obtain more accurate results.
Second Part: This section includes all the parameters for the migration scenarios that users want to test. The Controller picks each combination of these parameters and sends them to the Migration Engine one at a time, ensuring a thorough and systematic testing process.

- `[[experiment]]`
  - `file`: `file1.csv, file2.txt, file3.java`
  - `limit`: `1048576, 1048576`
  - `compressiontype`: `None, gzip, lz4`
  - `stream`: `3, 2, 1`
  - ...

The Controller checks all possible combinations when creating the configuration file for the Migration Engine. Here is an example of what the second part of the configuration file might look like:

- `[[experiment]]`
  - `file`: `file1.csv`
  - `limit`: `1048576`
  - `compressiontype`: `None`
  - `stream`: `3`

4.4 Setup The Framework

4.4.1 Physical requirements

Aside from the essential components required for the migration to be successful, such as the source database and the target database, the deployment of DMBench involves three additional components:

- **Controller:** Oversees all experiments and sets up the Migration Engine.
- **Databases:** Includes Prometheus for resource data and a secondary database for performance data.
- **Logs Reporter:** Organizes experiment logs using a Kafka cluster and a parser for human-readable logs.

To ensure the smooth operation of DMBench, the following prerequisites must be met:

- Docker must be installed on all machines involved in the deployment.
- Python must be installed on the machine that will be deploying the Logs Reporter.

Depending on the user's infrastructure, these three components can be deployed on more than one machine or all on the same machine. DMBench is designed to be flexible, allowing easy configuration for communication between components, even if they are deployed on separate machines. This flexibility ensures that the system can adapt to various deployment scenarios, facilitating seamless operation regardless of the physical setup.

4.4.2 Operational Steps

To run DMBench, follow the steps outlined in the flowchart (Figure 5). We'll explain each step briefly here, and more detailed information can be found in the Git repository[2].

[2] `https://github.com/yorku-ease/DataMigrationBenchmarkingTool/tree/main/deployment`

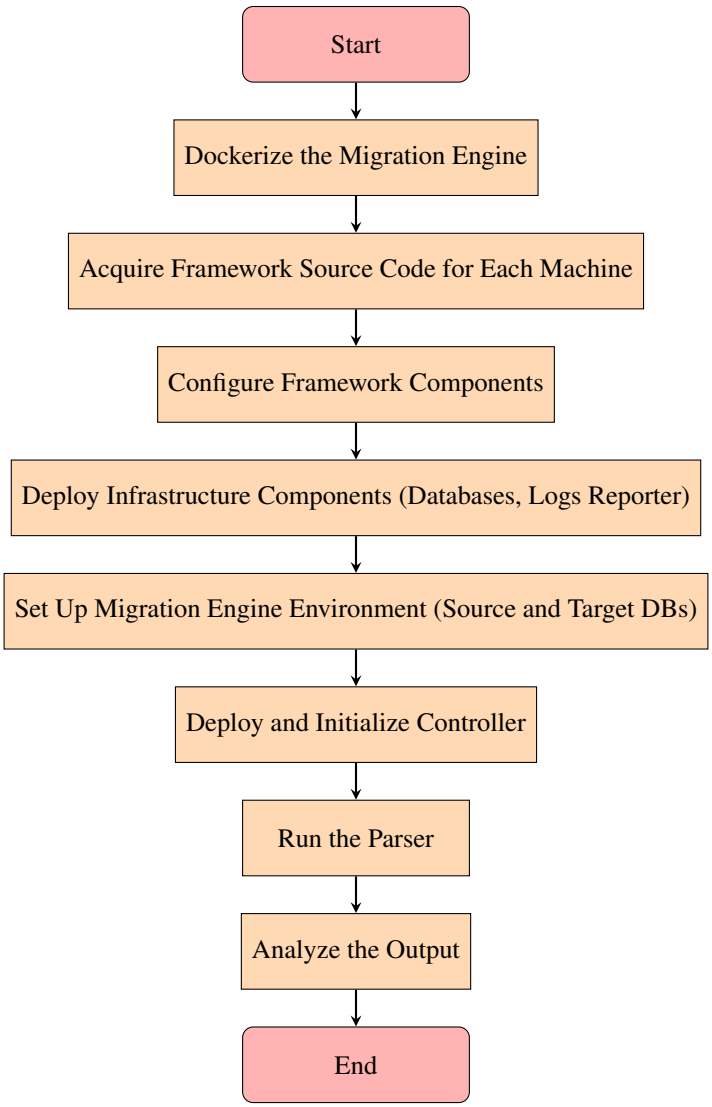

Figure 5: Flowchart showing the steps of running a performance test with DMBench.

1. Dockerize the Migration Engine To effectively test a migration engine, adherence to the framework's standards is essential. This implies dockerizing the migration engine in accordance with the specified guidelines.

For seamless execution, the user assumes that the source and target of the data are already operational and prepared for migration when running the container. Additionally, the Docker container should be configured to anticipate the configuration file generated by the Controller. Upon initiating the container, the migration process will commence smoothly.

By ensuring your Docker container is correctly set up, you can facilitate an efficient and reliable migration process, meeting all specified requirements and standards.

2. Acquire Framework Source Code for Each Machine In the Git repository[3], the user will find a dedicated deployment folder. This folder contains distinct sub-folders for each component of the framework:

- **Databases**
- **Controller**
- **Logs Reporter**

Each subfolder contains configuration files and setup scripts for deploying its designated component. These should be downloaded and deployed onto their respective machines. Ensure that the correct files are downloaded to the appropriate machine to facilitate seamless setup and operation of DMBench.

3. Configure Framework Components Each component of the framework requires specific configurations to ensure seamless operation. Detailed instructions on how to configure each component are provided in the Git repository. Configuration tasks may include specifying IP addresses, ports, and other essential settings tailored to each component's requirements. Proper configuration is crucial for the framework to function correctly and for the components to communicate effectively with each other.

4. Deploy Infrastructure Components (Databases, Logs Reporter) All components of the framework are encapsulated in Docker containers and have associated Docker Compose files. To deploy these components, the user needs to simply run the 'docker compose up' command for each component in the correct order. This process will set up all the necessary backend systems required for the data migration. Detailed instructions on the deployment order and any additional setup requirements can be found in the Git repository.

5. Set Up Migration Engine Environment (Source and Target DBs) Ensure that the source and target databases for the migration engine are ready. This involves preparing the databases that will be involved in the data migration process.

6. Deploy and Initialize Controller Deploy the controller, which will manage and execute the migration experiments. The controller is responsible for coordinating the various components and ensuring the experiments run smoothly.

[3] https://github.com/yorku-ease/DataMigrationBenchmarkingTool/tree/main/deployment

7. Run the Parser Upon completion of the experiment, indicated by the termination of the Controller container, it is essential to navigate to the Kafka cluster machine. Subsequently, the parser needs to be executed. Throughout the experiment, resource consumption logs were promptly stored in Prometheus. However, the logs pertaining to performance benchmarks remain localized on the Kafka machine. Running the parser becomes imperative at this juncture. Its role is twofold: to render the performance benchmark logs human-readable and to facilitate their exportation into JSON and CSV files. Furthermore, the parser ensures the archival of these logs in the MongoDB database for comprehensive analysis and reference. On the Kafka cluster machine, change the current working directory to the reporter/logsParser folder. Run the following command

8. Analyze the Output After following all the steps outlined earlier, the user will have collected a significant amount of data from the experiments. Every performance benchmark, including transfer time, is now neatly stored in the MongoDB database.
Simultaneously, PrometheusDB has captured all the detailed metrics about resource consumption, which can be easily visualized using the Grafana Dashboard. This combination ensures that the user has all the crucial information at your fingertips, allowing for a comprehensive analysis of your experiments.

4.5 Meeting Functional and Non-Functional Requirements with DMBench

In developing DMBench, specific functional and non-functional requirements were identified to ensure that the tool provides a comprehensive, efficient, and user-friendly environment for data migration benchmarking. Below is a summary of how DMBench addresses these requirements:

1. Easy Setup and Configuration: DMBench simplifies the setup and configuration process through the use of Docker containers and Docker Compose files, which encapsulate all necessary components, allowing for easy deployment with minimal manual intervention. The microservices architecture adopted by DMBench further enhances its usability by isolating different components (e.g., migration engine, logging, and monitoring) into manageable, self-contained services. This approach ensures that each component can be deployed and updated independently, reducing the complexity of the overall system. The config.ini file centralizes the configuration process, enabling users to define key parameters such as SSH credentials, data paths, and migration engine settings in one location. This makes the tool flexible and adaptable to various scenarios, while also ensuring a consistent setup across different environments.

2. Convenient and Complete Access to Results: DMBench provides robust logging and monitoring capabilities through a combination of Prometheus, MongoDB, and Kafka. Prometheus is

used to store time-series data for resource monitoring, while MongoDB holds all logs from both the framework and migration engine. The Grafana dashboard enhances the accessibility of results, enabling real-time visualization of data and easy review of past experiments.
Moreover, Kafka serves as a fault-tolerant and scalable logging system within DMBench, ensuring that logs are reliably captured and processed even in the event of failures. The use of Kafka, coupled with the microservices architecture, allows for efficient log management and scalability, making DMBench capable of handling high volumes of data and complex logging requirements. This architecture ensures that raw data is preserved for in-depth analysis, while also supporting the reliable and scalable operation of the logging system.

3. Extensibility: The modular, microservices-based architecture of DMBench supports extensibility, allowing users to integrate additional migration engines or replace existing components with minimal effort. Each component, including the migration engine, logging, and monitoring tools, can be customized or extended according to user needs. If users want to integrate their own engine, they simply need to provide the framework with the appropriate Docker image and specify any variables required to run the engine. Additionally, they must include the logger's configuration so the engine knows where to log data. The separation of concerns facilitated by microservices ensures that new features or engines can be added without disrupting the existing system. This flexibility ensures that DMBench can evolve alongside the changing requirements of data migration projects.

4. Accuracy and Reliability: DMBench ensures accuracy and reliability by allowing users to repeat experiments multiple times and average the results to minimize variability. The framework's deterministic execution of experiments guarantees that any observed variations are due to environmental factors or the specific use case, rather than the tool itself. The inclusion of Kafka as part of the logging system further enhances reliability by ensuring that logs are captured and processed consistently, even in the presence of failures. This rigorous approach to testing and logging enhances confidence in the results and supports robust decision-making.

4.6 Default Migration Engine

In this section, we present the first data migration engine that the framework supports. This engine is specifically designed to transfer files and has been dockerized in accordance with the framework standards.
The default migration engine serves as a reference implementation, showcasing the capabilities and flexibility of the DMBench framework. It is engineered to handle file transfers efficiently and reliably, making it an ideal starting point for users to understand the framework's functionalities and potential.

4.6.1 Architecture

The architecture of the default migration engine within the Docker container is depicted in Figure 6. This architecture includes several key components:

- **fileMigrator:** The principal class that orchestrates the entire migration process. It calls all other components to perform their respective tasks, ensuring the migration is executed smoothly.
- **Files Manager:** This module handles all tasks related to files, such as dividing files into streams, executing the actual transfer of files, and other file management tasks. All files to be transferred must be placed in the `data` folder within the source machine, ensuring the engine has direct access to them.
- **Connection Manager:** Responsible for establishing connections between the source and target machines. It provides tools for sending data, such as SSH connections for simple file transfers. The Connection Manager can be customized to connect to any type of target system by overriding specific methods.
- **Kafka Logger:** This component logs all experiment details to a Kafka cluster. It ensures that comprehensive logs are maintained for analysis and debugging.
- **Compressor:** The component responsible for compressing data. It has two subclasses:
  - **GzipCompressor:** Handles compression using the Gzip algorithm.
  - **Lz4Compressor:** Handles compression using the LZ4 algorithm.

  The design allows for easy integration of additional compression algorithms as needed.
- **Configuration File:** The `config.ini` file discussed in Section 4.4.2 is central to the operation of the migration engine. This file provides essential settings and parameters required for the migration process, which are passed from the Controller to the Migration Engine.

The migration engine must be deployed in the source machine, ensuring direct access to the files to be transferred. Proper placement and configuration are crucial for the smooth operation of the migration process.

4.6.2 Prerequisites

For the migration process to be successful, the following prerequisites must be met:

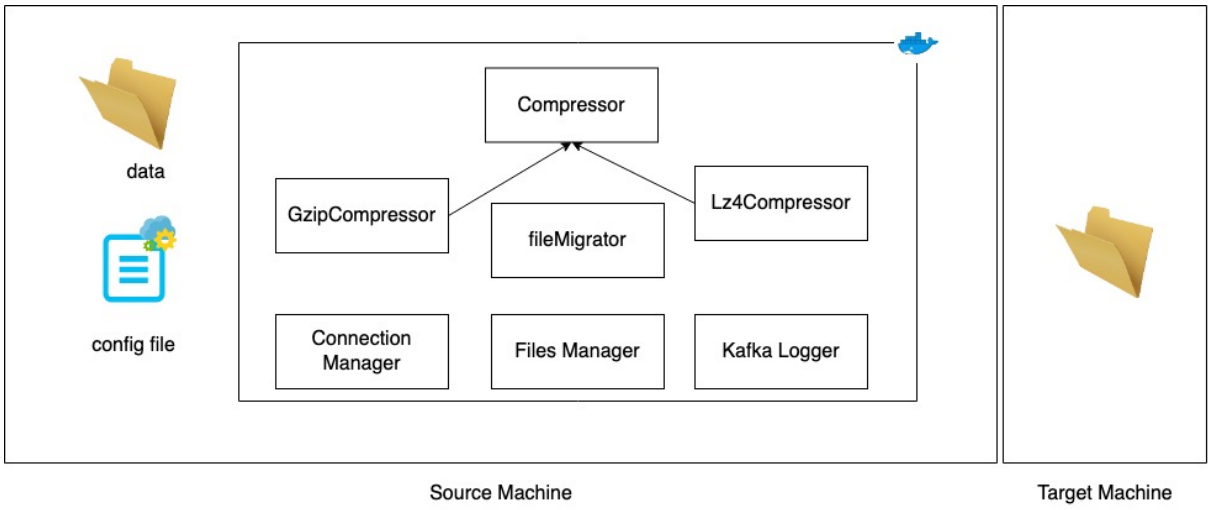

Figure 6: Architecture of the Default Migration Engine inside a Docker container.

- **Docker Installation:** Docker must be installed on the source server where the default engine will be running.
- **SSH Connection:** SSH connection must be enabled on the remote server where the files will be migrated to. Make sure that SSH is properly configured and that the necessary credentials (e.g., username and password) are set up for authentication.
- **Network Accessibility:** Both the source and target servers must be accessible over the network. Ensure that there are no firewall rules or network configurations that could block the communication between the source and target machines.
- **Configuration Setup:** Ensure that the `config.ini` file is correctly configured with all necessary parameters, including the IP addresses, usernames, passwords, and other settings required for the migration process. This configuration file is critical for the migration engine to function correctly and must be set up as mentioned in Section 4.4.2.
- **File Placement:** Ensure that all files to be transferred are saved in the `data` folder within the source machine. This ensures that the migration engine has direct access to the files and can manage the transfer process effectively.

By ensuring these prerequisites are met, the migration engine can effectively manage the transfer of data between the source and target machines, leveraging the robust architecture and components outlined above. Proper preparation and configuration will help to avoid potential issues and ensure a smooth and efficient migration process.

4.6.3 Configuration and Deployment

Starting the default migration engine is straightforward. A docker run command with the config.ini file mounted as a volume is sufficient to initiate the migration.
The configuration file should include:

- SSH credentials for both the source and target machines.
- Information about the exact location of files to be transferred and their destination.
- Specifics on how to transfer these files.
- Information about the logger, including where to send the logs.

4.7 IBM Db2 Migration Service

In this section, we present the IBM Db2 Migration Service [30], a sophisticated migration engine developed by IBM engineers, which we have integrated into the DMBench framework. This service enhances the efficiency of migrating data to and from IBM Db2 databases[31].

4.7.1 Integration and Setup

The integration of the IBM Db2 Migration Service into the framework involves creating a Docker image that encapsulates all necessary components and dependencies. We have downloaded the necessary binaries and software, and followed the steps mentioned in the Db2 Early Access program[43]. Refer to Section 4.4.2 for guidelines on creating Docker images within the framework.

Creating the Docker Image The Docker image for the IBM Db2 Migration Service includes the installation of required packages, configuration of environment variables, and setup of the migration platform. Below are the steps performed to create the Docker image:

1. Use a CentOS base image and perform system cleanup.
2. Copy necessary RPM files (e.g., `gskcrypt64`, `gskssl64`, `ibm_migration_platform`) into the container.
3. Install the RPM files using `yum localinstall`.
4. Update the system packages and install required Python packages from `requirements.txt`.
5. Copy the setup and logging scripts (`installConfluent.sh`, `myscript.sh`, `kafkaLogger.py`) into the container.
6. Run the scripts for initiating the migration and logging.

4.7.2 Configuration and Deployment

Starting the Db2 migration engine is straightforward. Use a docker run command with the config.ini file mounted as a volume to initiate the migration.
The configuration file must contain:

- Authentication details for the source and target databases.
- Logger settings, including the destination for the logs.
- Detailed instructions for executing the migration as per the documentation, covering parameters such as maximum streams, compression types, tables, and other necessary configurations.

4.8 Fivetran

In this section, we present the Fivetran data migration service, an advanced cloud-based data integration platform that has been integrated into the DMBench framework. This service enhances the efficiency of migrating data between various databases and cloud storage solutions.

4.8.1 Integration and Setup

The integration of the Fivetran service into the framework involves creating a Docker image that encapsulates all necessary components and dependencies. Refer to Section 4.4.2 for detailed guidelines on creating Docker images within the framework.

About Fivetran Fivetran [44] is a fully managed data pipeline service that automates the process of loading data from various sources into a data warehouse. It supports a wide range of data sources, including databases, cloud storage, and SaaS applications. Fivetran continuously syncs data from the source to the destination with minimal configuration and handles schema changes automatically, ensuring the destination is always in sync with the source.

Supported Data Sources and Destinations Fivetran supports numerous data sources, including:

- Databases: PostgreSQL, MySQL, SQL Server, etc.
- Cloud Storage: AWS S3, Google Cloud Storage, etc.
- SaaS Applications: Salesforce, Google Analytics, etc.

It can load data into various data warehouses, such as Snowflake, BigQuery, Redshift, and others.

Creating the Docker Image Unlike the Db2 Migration Service, the logic for Fivetran's data migration does not run within the Docker container but operates in the cloud as a black box. The Docker container communicates with the Fivetran API to manage the migration process. Fivetran acts as a replication engine, ensuring that the source and target data are synchronized. It uses connectors to connect to the source and target of data, as illustrated in Figure 7.

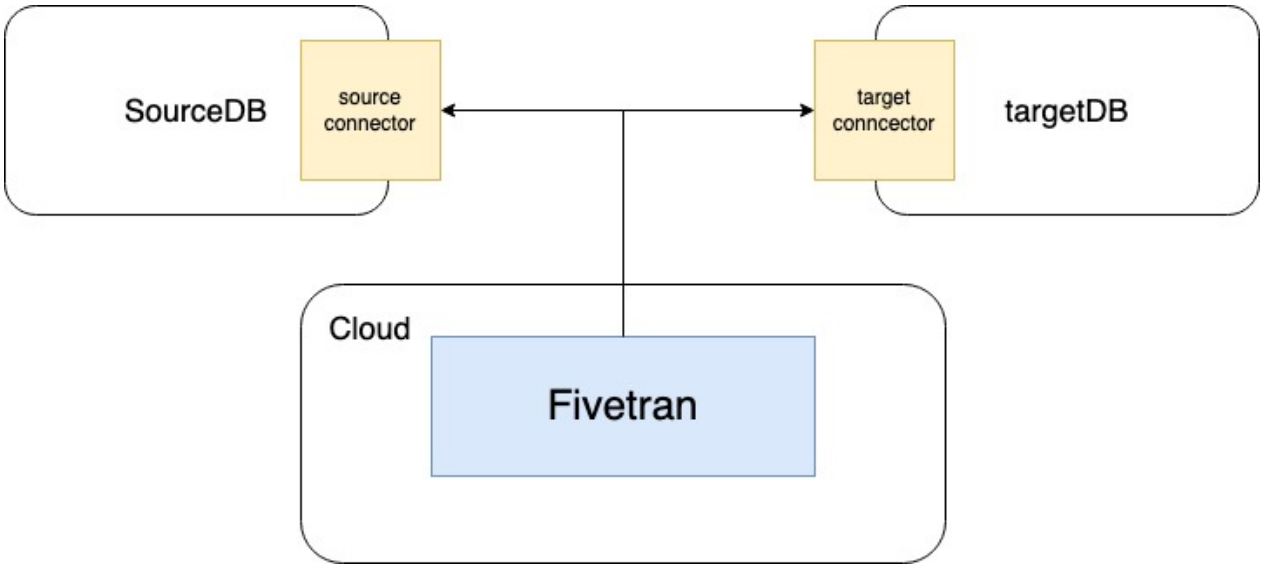

Figure 7: Fivetran Data Integration Architecture Diagram.

Before running the experiment, the user must create these connectors on the Fivetran dashboard and then provide the connector ID and group ID in the `config.ini` file. The container will force a resync of data from source to target, triggering the migration each time.

Main Script This Python script is the main script running in the container. To test the image, we created a MySQL connector, which can easily be extended to different other connectors. Below are some of main the steps performed by the script:

1. Drop the specified tables in the target database to ensure a clean migration.
2. Set the Fivetran connector's schedule type to "manual" to control sync operations programmatically.
3. Initiate a resync operation using the Fivetran API and wait for it to complete.
4. Wait for the connector to reach specific states (e.g., "syncing" or "scheduled") during the resync process.
5. Output the results of the migration process and log relevant information for further analysis.

4.8.2 Configuration and Deployment

Starting the Fivetran migration engine is simple. A docker run command with the config.ini file mounted as a volume is sufficient to initiate the migration.
The configuration file should include:

- Authentication details such as connector ID, API key, and API secret.
- Logger settings, specifying the destination for the logs.
- Details of the schema and tables to be migrated.

5 Db2 Migration Service Case Study

5.1 Motivation

Migration is a complex process that involves numerous configuration parameters, presenting challenges in terms of resources and time. Errors during migration can be costly, making it crucial for engineers to determine the optimal settings before executing the actual migration in a production environment. This preemptive approach allows for the identification and rectification of errors, unexpected delays, and potential costs, ensuring that the actual migration runs under optimal conditions with minimum cost and time.

One critical aspect of this preemptive strategy is the use of compression techniques to reduce data size during transfer. Compression is frequently employed to potentially accelerate the process and minimize resource consumption. However, different compression algorithms impact the migration process in diverse ways. Understanding these effects is essential for selecting the most suitable compression technique to optimize data migration. Evaluating the efficiency of binary compression, which can significantly reduce data size, provides insights into its practicality and benefits during Db2 data migrations. By addressing these aspects, the study aims to identify the most efficient and effective strategies for data migration using the Db2 Migration Service.

DMBench demonstrates its capability to provide this testing environment through a case study on IBM Db2.

5.2 Introduction

The IBM Db2 database [31] is a highly advanced and versatile relational database management system (RDBMS) designed to handle large-scale data environments. IBM Db2 supports a wide range of data types and structures and is widely used in enterprise settings due to its robust performance, scalability, and comprehensive feature set [45].

One of the key complexities of IBM Db2 lies in its ability to efficiently manage and query massive datasets. It includes sophisticated features such as advanced indexing, complex query optimization, in-memory processing, and support for both structured and unstructured data. These capabilities make IBM Db2 suitable for critical applications requiring high availability, security, and reliability. However, these same features also contribute to its complexity, necessitating careful planning and management to ensure optimal performance [46].

Given the intricate nature of IBM Db2 , performing data migrations can be a challenging task. Migration processes must account for the vast amount of data, the need to minimize downtime, and the requirement to maintain data integrity and security. An efficient migration engine is essential to handle these demands effectively. Such an engine should support various migration strategies, in-

cluding full and incremental migrations, and provide comprehensive mechanisms for error handling and performance monitoring [47].
Moreover, selecting the best configuration for migrating IBM Db2 databases is crucial to maximize its performance. Factors such as hardware resources, database settings, compression techniques, and parallel processing capabilities can significantly influence the efficiency of the migration process and the overall database performance. Compression, in particular, can reduce data transfer times and storage requirements, but the choice of compression algorithm and settings (e.g., enabling binary mode) must be tailored to the specific workload and system characteristics to achieve optimal results [48].
In summary, the complexity of the IBM Db2 database demands an efficient and well-configured migration engine to ensure smooth and effective data transfers. By carefully choosing the appropriate compression techniques and configurations, organizations can enhance the performance and reliability of their IBM Db2 environments, thereby supporting their critical business operations [49].
In this section, we present two comprehensive case studies conducted using the DMBench framework to evaluate the performance and efficiency of data migration processes with the IBM Db2 Migration Service [30]. The primary aim is to understand the impact of different compression techniques on data transfer and to assess the effectiveness of binary compression. By conducting these studies, we seek to answer the following research questions:

1. How do different compression types (None, Gzip, LZ4) impact the performance and efficiency of data migration using the Db2 Migration Service?

2. How efficient can compression be when using binary during data migration with the Db2 Migration Service?

These case studies provide valuable insights into optimizing data migration strategies, ensuring efficient and reliable data transfers, and informing best practices for future migration projects.

5.3 Experimental Setup

5.3.1 Preparing the Machines

As outlined in 4.4.1, the experimental setup involved creating dedicated machines for the databases and for the framework components (controller and logs reporter).

- **Framework Machines:**
  - **Machine F1 (Framework Database):** Set up to handle the databases needed for the framework with sufficient resources.

- **Machine F2/D1 (Controller & Logs Reporter & Source Database):** Configured to host the controller, the logs reporter.

- **Database Machines:**

  - **Machine F2/D1 (Source Database & Framework Components):** IBM Db2 deployed on a Docker container. This machine is the same Machine hosting the controller, and the logs reporter.
  - **Machine D2 (Target Database):** IBM Db2 deployed on a Docker container. Using Docker containers ensures consistency in the environment and simplifies the deployment process. Both source and target containers were configured with identical settings to maintain consistency in performance measurement.

The specifications of all machines according to their IDs are as follows:

Machine ID	vCPU	RAM (GB)	Disk (GB)	Image
Machine D2	16	120	576	Ubuntu-20.04.6-Focal-x64-2023-11
Machine F2/D1	16	120	576	Ubuntu-20.04.6-Focal-x64-2023-11
Machine F1	16	60	576	Ubuntu-20.04.6-Focal-x64-2023-11

Table 1: Machine Specifications Used to Run the Experiments for the Case Study

5.3.2 Data Preparation

The TPC-H benchmark was used to generate synthetic data. TPC-H is a well-known benchmark for evaluating the performance of database systems. It provides a set of standard queries and a data schema designed to emulate real-world business scenarios. Using TPC-H allows us to generate a diverse and complex dataset, making it ideal for testing data migration processes under realistic conditions. It is widely recognized and used in database performance benchmarking, ensuring that the results are comparable and credible. The dataset includes various types of data, such as customer orders and line items, which are representative of real-world business data.[50][51] The generated data was duplicated into multiple tables to allow for running more than one scenario. This duplication ensures that we can test different aspects of the migration process without altering the original dataset. The synthetic data was then imported into the source database.
By following these steps, we ensured that the databases and framework components were properly set up and ready for the migration experiments. This setup provides a controlled environment to accurately measure the performance and efficiency of the Db2 Migration Service under different compression scenarios.

5.3.3 Scenarios

The following scenarios are built based on real-world cases gathered from actual clients. While these scenarios are representative of the challenges faced in real data migration processes, the data sets used in this study have been deliberately down-scaled to smaller volumes. This down-scaling is necessary for feasibility, as working with full-scale data would require extensive computational resources, including high-performance servers, significant storage capacity, and prolonged processing times.

The data for these scenarios was generated using the TPC-H benchmark, as described in Section 5.3.2. The experiments we will run are based on these scenarios, with different configurations of the Db2 Migration Service applied each time. Moreover, the insights gained from these smaller data sets are scalable and applicable to larger data volumes. The principles and trends observed in these experiments can be extrapolated to understand how different approaches would perform at scale.

Scenario Number	**Scenario Name**	**Table Name**	**Number of Rows**	**Number of Columns**	**Data Size (MB)**
1	Few mixed tables	LINEITEM1	3,894,493	16	674
		ORDERS2	15,000,000	9	1,825
		ORDERS10	1,993,416	9	275
				Total Size	2,774
2	One gigantic table	GIGANTIC ORDERS	45,000,000	9	5,477
				Total Size	5,477
3	Multiple large tables	ORDERS2	15,000,000	9	1,825
		ORDERS3	15,000,000	9	1,825
		ORDERS4	15,000,000	9	1,825
		ORDERS5	15,000,000	9	1,825
		ORDERS6	15,000,000	9	1,825
		ORDERS7	15,000,000	9	1,825
		ORDERS8	15,000,000	9	1,825
				Total Size	12,775
4	One wide table with multiple columns	ORDERS WIDE2	15,000,000	27	5,337
				Total Size	5,337

Table 2: Tables Details for All Scenarios in Case Study.

5.4 Case Study 1: Impact of Different Compression Types on Migration Performance

In large-scale data migration projects, particularly those involving massive datasets that can take days to migrate, it is crucial for engineers to select the appropriate compression type to optimize transfer times and resource utilization. The decision on which compression algorithm to use can significantly impact the overall efficiency and performance of the migration process. This consideration is especially important when working with limited computational resources. LZ4 and Gzip are both popular compression algorithms, but they serve different needs based on their performance characteristics. LZ4 is designed for speed, offering extremely fast compression and decompression at the expense of a lower compression ratio, making it ideal for scenarios where quick data processing is crucial, such as in real-time systems or in-memory databases. On the other hand, Gzip, though slower in both compression and decompression, provides a significantly higher compression ratio, resulting in smaller file sizes. This makes Gzip more suitable for applications where storage efficiency and bandwidth reduction are prioritized, such as web content delivery or data archiving.[52] Therefore, understanding how different compression techniques affect migration performance is vital for making informed decisions.
This subsection addresses the first research question: *"How do different compression types (None, Gzip, LZ4) impact the performance and efficiency of data migration using the Db2 Migration Service?"*

5.4.1 Configuration

Here, we will specifically focus on the configuration of the Controller for the Db2 Migration Service, as detailed in Section 4.7.1.
The Controller configuration for the Db2 Migration Service is detailed in Appendix A
The ‘tables‘ variable in the ‘experiment‘ section specifies the scenarios we aim to run, which include scenarios 1, 2, 3, and 4. We will conduct experiments for three compression types: LZ4, GZIP, and NO compression. The ‘numberofexperiments‘ parameter in the ‘migrationEnvironment‘ section indicates that each experiment will be repeated six times to ensure the accuracy and reliability of the results.

5.4.2 Results

To ensure a thorough analysis, we performed several statistical tests to validate the assumptions of our evaluation metrics. The outcomes of these tests are documented for the performance metric (total transfer time) in Table 3 and for the resource consumption metrics in Tables 6, 7, 8, 9, 10 and 11.

- **Shapiro-Wilk Test**: This test examines the normality of the data. A p-value below 0.05 signifies a non-normal distribution.
- **Levene's Test**: This test assesses the homogeneity of variances. A p-value above 0.05 indicates equal variances across groups.
- **Kruskal-Wallis Test**: Due to the Shapiro-Wilk test indicating non-normal data ($p < 0.001$), we employed the Kruskal-Wallis test, a non-parametric alternative to ANOVA, to determine if there are statistically significant differences between groups. The Kruskal-Wallis test is robust against the assumption of homogeneity of variances, ensuring the reliability of the significant differences identified, regardless of the outcome of Levene's test.

Analysis of Total Transfer Time

To understand the impact of different compression techniques on the total transfer time, we analyzed the mean and standard deviation of transfer times across various scenarios, as well as the values of the statistical tests, as shown in Table 3. Pairwise comparisons performed using the Mann-Whitney U test to compare differences between groups of compression techniques and groups of scenarios are detailed in Tables 5 and 4. Additionally, we can compare the means in the boxplots that illustrate the total transfer time for different compression techniques and scenarios (Figures 8, 9, 10, 11).

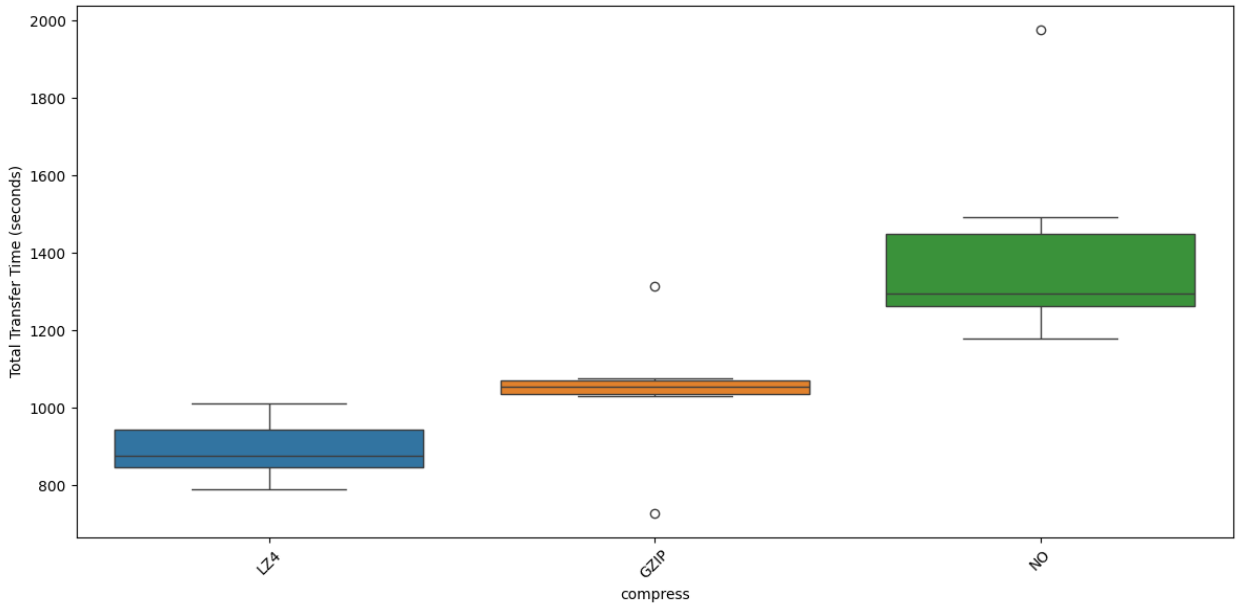

Figure 8: Box Plot of Total Transfer Time for Different Compression Methods (Scenario 1,Case Study 1)

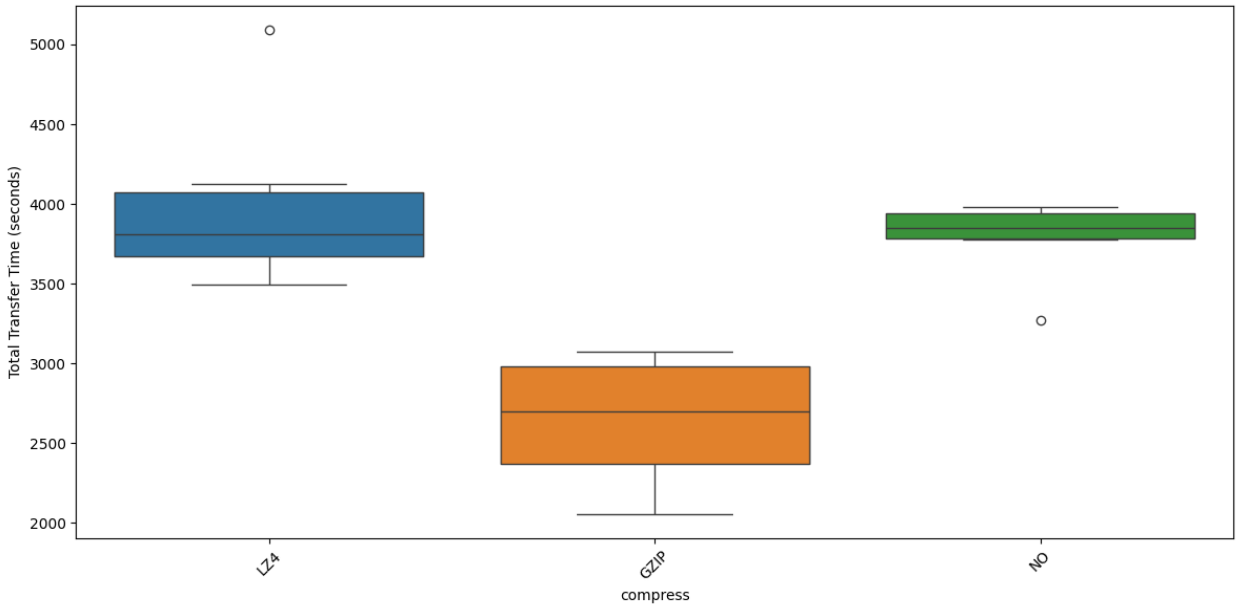

Figure 9: Box Plot of Total Transfer Time for Different Compression Methods (Scenario 2,Case Study 1)

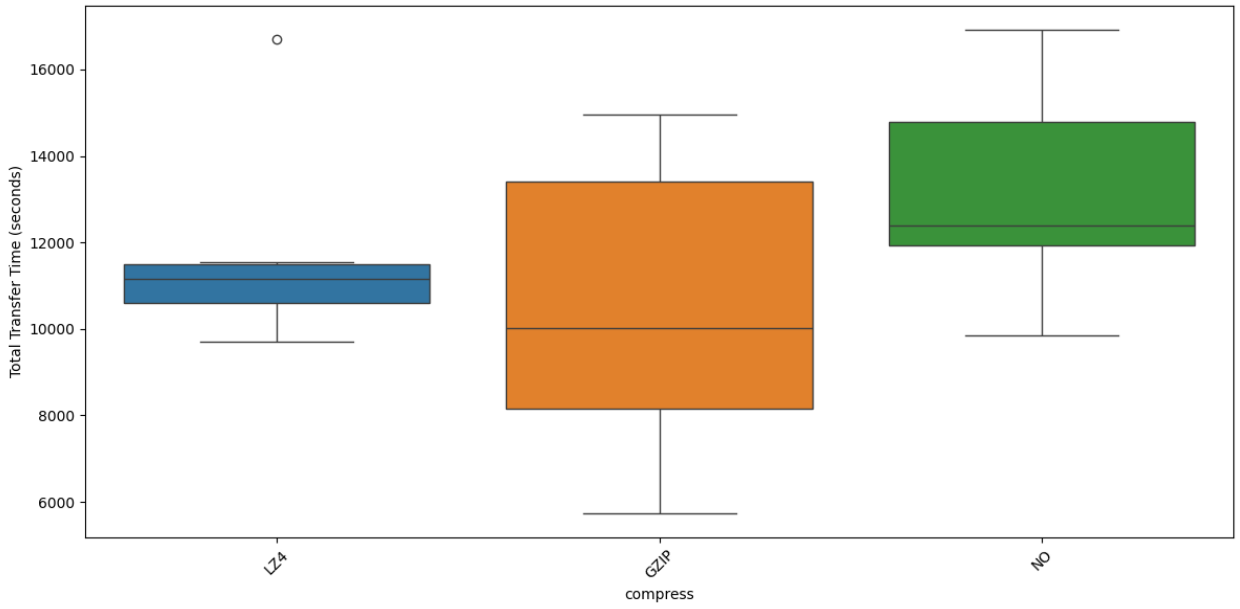

Figure 10: Box Plot of Total Transfer Time for Different Compression Methods (Scenario 3,Case Study 1)

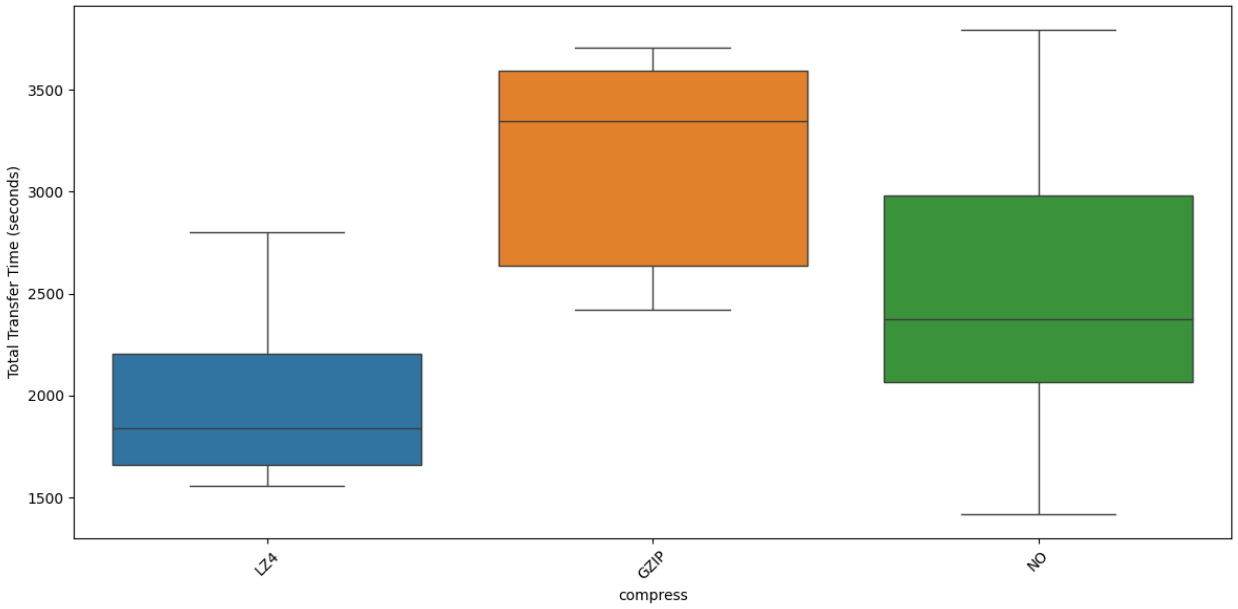

Figure 11: Box Plot of Total Transfer Time for Different Compression Methods (Scenario 4,Case Study 1)

Table 3: Total Transfer Time for Different Compression Types and Scenarios (Case Study 1)

Scenario	**Compress**	**Mean (s)**	**STD (s)**
Scenario 1	GZIP	1043.48	186.40
Scenario 1	LZ4	893.50	82.19
Scenario 1	NO	1416.92	293.63
Scenario 2	GZIP	2645.54	415.37
Scenario 2	LZ4	3998.07	578.48
Scenario 2	NO	3780.85	261.79
Scenario 3	GZIP	10470.75	3656.88
Scenario 3	LZ4	11790.44	2497.35
Scenario 3	NO	13158.02	2583.07
Scenario 4	GZIP	3149.73	578.36
Scenario 4	LZ4	1993.99	474.37
Scenario 4	NO	2517.58	842.78
Test		**P-value**	**Meaning**
Shapiro-Wilk		< 0.001	Not normal
Levene's Test		< 0.001	Not equal variances
Kruskal-Wallis		< 0.001	Significant differences

The results indicate several key patterns in the total transfer time across different scenarios and compression techniques:

In **Scenario 1 (Few mixed tables)** LZ4 compression achieves the lowest mean transfer time. This result aligns with LZ4's strength in speed. LZ4's faster compression and decompression make it more efficient for handling smaller, mixed datasets where quick processing is crucial. Gzip, while more effective in terms of compression ratio, takes longer to compress and decompress, resulting in higher transfer times compared to LZ4. The no-compression scenario shows the highest transfer times due to the lack of any size reduction, which requires more time for data transmission.

In **Scenario 2 (One gigantic table)**, Gzip significantly reduces transfer time compared to LZ4 and no compression. Gzip's higher compression ratio is beneficial for large datasets, as it results in smaller file sizes, which reduces transfer time despite the slower compression speed. LZ4, although faster, does not compress as well, leading to larger file sizes and longer transfer times compared to Gzip.

In **Scenario 3 (Multiple large tables)**, In this case, Gzip again outperforms LZ4 and no compres-

sion. The effectiveness of Gzip in reducing file size outweighs its slower speed in scenarios with multiple large tables. LZ4's faster processing is less beneficial here due to its lower compression ratio, resulting in larger file sizes and longer transfer times compared to Gzip. No compression results in the longest transfer times as there is no reduction in data size.

In **Scenario 4 (One wide table)**, LZ4 compression results in the lowest transfer time, followed by GZIP, with no compression leading to the highest transfer times.

The statistical tests (Table 5) confirm these observations: **Kruskal-Wallis Test**: With a p-value < 0.001, the test indicates significant differences between the groups, validating the observed differences in total transfer times.

Table 4: Pairwise Comparison of Scenarios in Case Study 1 Using Mann-Whitney U Test

Compression		**Scenario 1**	**Scenario 2**	**Scenario 3**	**Scenario 4**
GZIP	**Scenario 1**	-	< 0.01	< 0.01	< 0.01
GZIP	**Scenario 2**		-	< 0.01	0.13
GZIP	**Scenario 3**			-	< 0.01
GZIP	**Scenario 4**				-
LZ4	**Scenario 1**	-	< 0.01	< 0.01	< 0.01
LZ4	**Scenario 2**		-	< 0.01	< 0.01
LZ4	**Scenario 3**			-	< 0.01
LZ4	**Scenario 4**				-
NO	**Scenario 1**	-	< 0.01	< 0.01	< 0.01
NO	**Scenario 2**		-	< 0.01	< 0.01
NO	**Scenario 3**			-	< 0.01
NO	**Scenario 4**				-

The pairwise comparison between scenarios also confirms that there is a significant difference between scenarios, as shown in Table 4.

Table 5: Pairwise Comparison of Compression Methods in Case Study 1 Using Mann-Whitney U Test

Scenario 1				**Scenario 2**			
	LZ4	**NO**	**GZIP**		**LZ4**	**NO**	**GZIP**
LZ4	-	< 0.01	0.06	**LZ4**	-	0.94	< 0.01
NO		-	< 0.05	**NO**		-	< 0.01
GZIP			-	**GZIP**			-
Continued on next page							

Table 5 continued from previous page

Scenario 1				Scenario 2			
	LZ4	**NO**	**GZIP**		**LZ4**	**NO**	**GZIP**
Scenario 3				**Scenario 4**			
	LZ4	**NO**	**GZIP**		**LZ4**	**NO**	**GZIP**
LZ4	-	0.18	0.70	**LZ4**	-	0.31	< 0.01
NO		-	0.18	**NO**		-	0.24
GZIP			-	**GZIP**			-

However, the pairwise comparison between compression techniques in Table 5 shows that in the scenarios we tested, there isn't always a significant difference between the compression techniques. After thorough investigation, we concluded that this was related to a limitation of resources. An important observation from the experiments is that although the configuration specified a maximum of 10 streams, the logs indicated that only one stream was used for migration. This limitation particularly impacts both LZ4 and GZIP performance; however, it impacts LZ4 more as LZ4 is more CPU-intensive compared to GZIP. Utilizing only one stream limits the available CPU resources, which could have contributed to the slightly higher transfer times observed for LZ4.
Maximizing the number of active streams could potentially improve LZ4 performance by better leveraging CPU resources. This has limited the performance of LZ4 and GZIP and made them give similar results.

Conclusion Based on these results, there is no single best compression technique as we had initially expected; it depends on the scenario and the data we want to transfer. The choice of compression technique significantly impacts the total transfer time. LZ4 is generally the most efficient for scenarios with fewer tables or wide tables, while GZIP excels in handling very large datasets. However, due to limited resources, such as the use of only one stream for migration, there were instances where no compression performed better than the compression techniques. Therefore, it is crucial to choose the right compression technique based on the specific scenario and resource availability before starting the experiment.

Analysis of resources consumption metrics

To further understand the impact of different compression techniques, we analyzed the resource consumption metrics, including CPU usage, disk I/O service bytes, memory usage, memory cache, and network RX and TX bytes. The results are detailed in the following tables:

Table 6: CPU Usage for Different Compression Types and Scenarios (Case Study 1)

Scenario	**Compress**	**Mean**	**STD**
Scenario 1	GZIP	2.22%	0.41%
Scenario 1	LZ4	2.37%	0.59%
Scenario 1	NO	2.21%	0.32%
Scenario 2	GZIP	1.64%	0.33%
Scenario 2	LZ4	2.43%	1.25%
Scenario 2	NO	1.83%	0.28%
Scenario 3	GZIP	3.79%	0.44%
Scenario 3	LZ4	3.82%	0.29%
Scenario 3	NO	3.28%	0.76%
Scenario 4	GZIP	1.73%	0.37%
Scenario 4	LZ4	1.33%	0.27%
Scenario 4	NO	2.00%	1.02%
Test		**P-value**	**Meaning**
Shapiro-Wilk		< 0.001	Not normal
Levene's Test		< 0.05	Not equal variances
Kruskal-Wallis		< 0.001	Significant differences

- Across most scenarios, LZ4 shows the highest CPU usage, as seen in Table 6. This trend is consistent due to the computationally intensive nature of LZ4 compression, which prioritizes speed over compression ratio.
- Scenario 3, involving multiple large tables, demonstrates the highest overall CPU usage for all compression types, reflecting the substantial computational effort required for processing large volumes of data.
- It is important to confirm the limitation mentioned when analyzing the total transfer time: even though we launched the experiment with a maximum of 10 streams, it turns out that only one stream has been running. The machine used for these experiments has 16 CPUs. Since only one stream is running, only one CPU is being utilized, which constitutes approximately 6

Table 7: Memory Usage for Different Compression Types and Scenarios (Case Study 1)

Scenario	**Compress**	**Mean (MB)**	**STD (MB)**
Scenario 1	GZIP	178.95	64.77
Scenario 1	LZ4	201.08	7.89
Scenario 1	NO	183.12	62.96
Scenario 2	GZIP	261.47	9.16
Scenario 2	LZ4	156.74	92.46
Scenario 2	NO	183.91	92.14
Scenario 3	GZIP	235.31	96.82
Scenario 3	LZ4	273.33	81.30
Scenario 3	NO	269.51	75.96
Scenario 4	GZIP	205.49	84.40
Scenario 4	LZ4	217.83	59.83
Scenario 4	NO	174.92	88.00
Test		**P-value**	**Meaning**
Shapiro-Wilk		< 0.001	Not normal
Levene's Test		0.38	Equal variances
Kruskal-Wallis		< 0.01	Significant differences

Table 8: Memory Cache for Different Compression Types and Scenarios (Case Study 1)

Scenario	**Compress**	**Mean (MB)**	**STD (MB)**
Scenario 1	GZIP	8.06	6.43
Scenario 1	LZ4	8.49	1.39
Scenario 1	NO	9.10	6.64
Scenario 2	GZIP	24.81	2.47
Scenario 2	LZ4	8.29	11.84
Scenario 2	NO	12.11	12.60
Scenario 3	GZIP	18.62	14.05
Scenario 3	LZ4	22.81	11.04
Scenario 3	NO	24.06	11.41
Scenario 4	GZIP	15.69	11.69
Scenario 4	LZ4	17.38	8.44

Continued on next page

Scenario 4	NO	11.65	12.36
Test		**P-value**	**Meaning**
Shapiro-Wilk		< 0.001	Not normal
Levene's Test		0.42	Equal variances
Kruskal-Wallis		< 0.01	Significant differences

Table 9: Network RX Bytes for Different Compression Types and Scenarios (Case Study 1)

Scenario	**Compress**	**Mean (MB)**	**STD (MB)**
Scenario 1	GZIP	138.21	39.47
Scenario 1	LZ4	271.51	24.20
Scenario 1	NO	405.93	84.66
Scenario 2	GZIP	118.90	25.83
Scenario 2	LZ4	140.34	17.08
Scenario 2	NO	332.73	35.21
Scenario 3	GZIP	87.39	35.92
Scenario 3	LZ4	123.53	23.13
Scenario 3	NO	245.95	40.70
Scenario 4	GZIP	32.85	7.83
Scenario 4	LZ4	66.81	12.92
Scenario 4	NO	545.45	131.82
Test		**P-value**	**Meaning**
Shapiro-Wilk		< 0.001	Not normal
Levene's Test		< 0.01	Not equal variances
Kruskal-Wallis		< 0.001	Significant differences

Table 10: Network TX Bytes for Different Compression Types and Scenarios (Case Study 1)

Scenario	**Compress**	**Mean (MB)**	**STD (MB)**
Scenario 1	GZIP	137.15	39.13
Scenario 1	LZ4	270.03	23.77
Scenario 1	NO	405.35	85.50
Scenario 2	GZIP	118.31	25.68
Scenario 2	LZ4	139.90	17.02
Scenario 2	NO	331.96	35.11
Scenario 3	GZIP	87.60	36.07
Scenario 3	LZ4	123.70	23.10
Scenario 3	NO	245.97	40.96
Scenario 4	GZIP	32.46	7.61

Continued on next page

Scenario 4	LZ4	66.42	12.99
Scenario 4	NO	544.95	130.83
Test		**P-value**	**Meaning**
Shapiro-Wilk		< 0.001	Not normal
Levene's Test		< 0.01	Not equal variances
Kruskal-Wallis		< 0.001	Significant differences

- **Memory Usage (Table 7)**:
  - LZ4 consistently shows higher memory usage compared to GZIP and no compression, which is not what I expected. I expected no compression to use the most resources, but my assumption is that since no compression requires less computation, it simply transfers raw data without modifications, unlike GZIP and LZ4.
  - The highest memory usage is observed in Scenario 3 for all compression types, aligning with the scenario's complexity involving multiple large tables.

- **Network RX and TX Bytes (Tables 9 and 10)**:
  - No compression results in the highest network RX and TX bytes across all scenarios, which is expected as it involves transferring the most data.
  - LZ4 shows intermediate network RX and TX bytes, while GZIP results in the lowest network RX and TX bytes, reflecting its higher compression ratio.
  - The differences in network RX and TX bytes across scenarios and compression techniques highlight the significant role of data size reduction in optimizing network resource usage.

Table 11: Disk IO Service Bytes for Different Compression Types and Scenarios (Case Study 1)

Scenario	**Compress**	**Mean (MB)**	**STD (MB)**
Scenario 1	GZIP	4.07	3.04
Scenario 1	LZ4	6.53	0.47
Scenario 1	NO	3.99	3.01
Scenario 2	GZIP	6.19	0.28
Scenario 2	LZ4	2.15	3.21
Scenario 2	NO	3.12	3.34

Continued on next page

Scenario 3	GZIP	4.11	3.16
Scenario 3	LZ4	5.10	2.50
Scenario 3	NO	4.93	2.42
Scenario 4	GZIP	4.12	3.14
Scenario 4	LZ4	4.93	2.41
Scenario 4	NO	3.28	3.47
Test		**P-value**	**Meaning**
Shapiro-Wilk		< 0.001	Not normal
Levene's Test		0.34	Equal variances
Kruskal-Wallis		0.28	No significant differences

The Disk I/O service bytes do not show significant differences across different compression types and scenarios, as indicated by the Kruskal-Wallis test (p-value = 0.28), as shown in Table 11. One possible explanation for this lack of significant difference is that we are monitoring the engine, not the databases themselves. The disk I/O interactions would be significantly different on the databases, which interact more with the disk compared to the engine.
The correlation matrix heatmap (Figure 12) illustrates the relationships between various metrics, including total transfer time, CPU usage, disk I/O, memory usage, and network traffic. Key observations include:

- Moderate positive correlation (0.66) between `TotalTransferTime` and `cpu.usage.total_avg`, suggesting that higher CPU usage is associated with longer transfer times which suggests that optimizing CPU utilization could further reduce transfer times.
- Negative correlation between `TotalTransferTime` and network metrics (`network.rx_bytes_avg` and `network.tx_bytes_avg`), indicating that higher network throughput corresponds to shorter transfer times.

Conclusion Overall, the findings suggest that selecting an appropriate compression algorithm is crucial for optimizing data migration performance. Both GZIP and LZ4 offer substantial benefits, and the choice between them should be based on the specific requirements of the migration task, such as the importance of speed and resource utilization. Ensuring that the system is properly configured to utilize the maximum number of streams can also play a significant role in enhancing performance, especially for CPU-intensive compression algorithms like LZ4.
Further research could explore the impact of other compression algorithms and investigate additional factors, such as data encryption, to provide a more comprehensive understanding of data

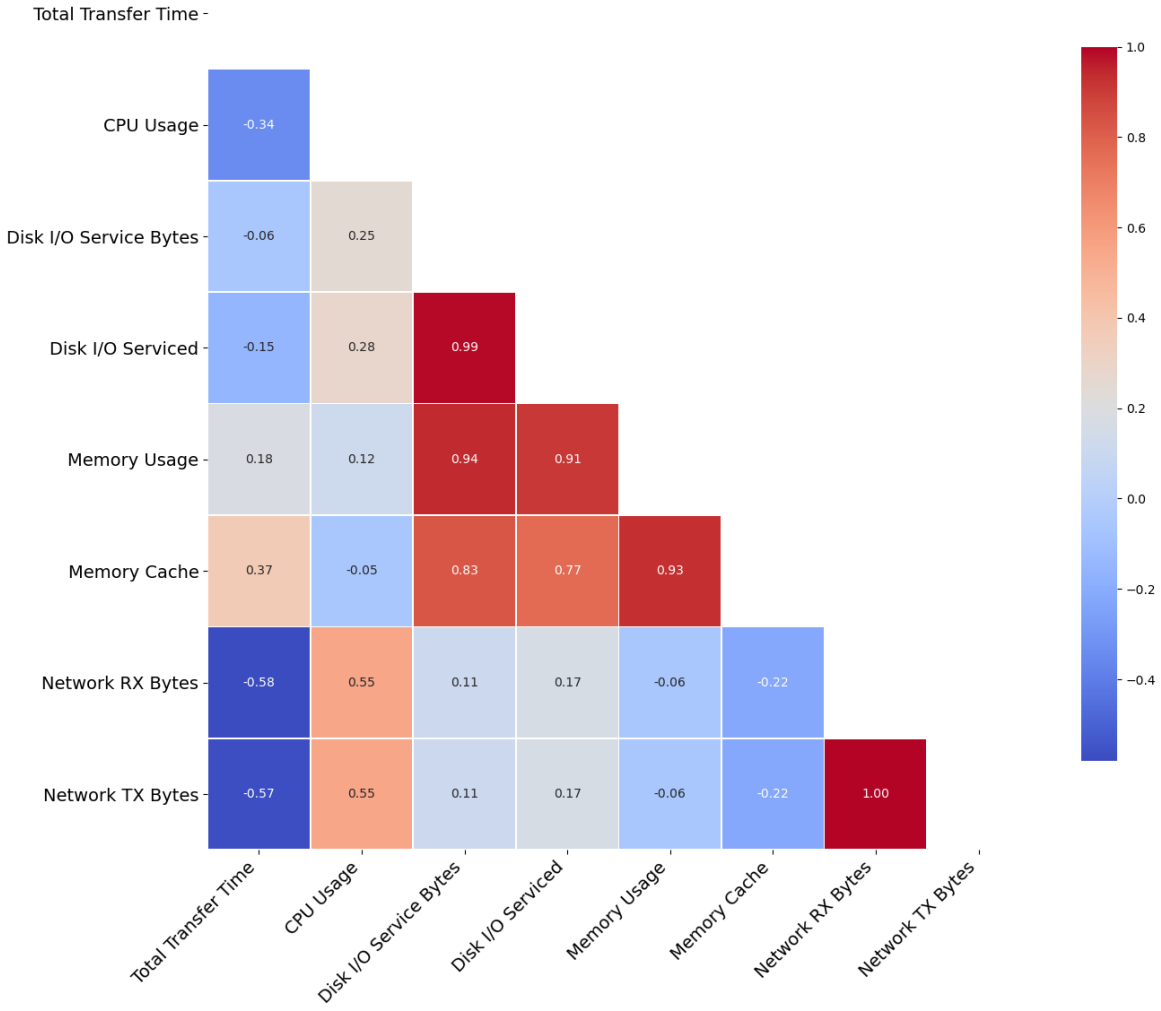

Figure 12: Correlation Matrix Heatmap for Performance Metrics (Case Study 1)

migration optimization strategies.

5.5 Case Study 2: Efficiency of Binary Compression During Data Migration

In binary compression, data is first transformed from its raw format to binary code, which is then more easily and more efficiently compressed. Data is reduced in size by identifying and encoding patterns within the data into shorter binary codes [53]. This potentially results into faster data processing and transfer subject to the data original format. Given the potential impact on migration efficiency, it is important to verify it through empirical testing. Understanding the efficiency of binary compression can help in making informed decisions to enhance the performance of data migration tasks.

This subsection addresses the second research question: *"How efficient can compression be when*

using binary during data migration with the Db2 Migration Service?"

5.5.1 Configuration

Rather than reviewing the full framework configuration here, we direct you to Section 4.4.2 for detailed steps. This section will focus specifically on configuring the Controller for the Db2 Migration Service, as outlined in Section 4.7.1.
The Controller configuration for the Db2 Migration Service is in Appendix B
The ‘tables‘ variable in the ‘experiment‘ section outlines the scenarios to be tested, covering scenarios 1, 2, and 4. We will run experiments for two compression types: LZ4 and GZIP, as well as evaluate two settings for the binary flag: true and false, to determine the efficiency of compression with the binary flag set to true. The ‘numberofexperiments‘ parameter in the ‘migrationEnvironment‘ section specifies that each experiment will be repeated six times to ensure result accuracy and reliability. The Docker image ‘fareshamouda/Db2migrationservice:test‘ is designated for the Db2 migration service , which the framework will utilize to perform the migration tasks.

5.5.2 Results

To ensure the robustness of our analysis, we conducted several statistical tests to verify the assumptions of the statistical tests. The results in Table 12 show that the Shapiro-Wilk test indicates non-normal distribution. In that case, we need to perform the Kruskal-Wallis test, which confirms significant differences between the groups. The Kruskal-Wallis test is robust against the homogeneity of variance assumption, meaning that the results are valid regardless of the outcome of Levene's test. This ensures that the significant differences identified are reliable.

Table 12: Summary of Total Transfer Time for Different Compression Types and Binary Settings (Case Study 2)

Scenario	**Compression**	**Binary**	**Mean Transfer Time (s)**	**STD (s)**
Scenario 1	GZIP	FALSE	1315.68	56.74
Scenario 1	GZIP	TRUE	973.51	110.51
Scenario 1	LZ4	FALSE	1285.83	52.48
Scenario 1	LZ4	TRUE	1014.28	186.65
Scenario 2	GZIP	FALSE	3647.76	263.52
Scenario 2	GZIP	TRUE	2335.50	296.23
Scenario 2	LZ4	FALSE	3952.50	548.67
Scenario 2	LZ4	TRUE	1561.66	197.91
Scenario 4	GZIP	FALSE	3564.72	294.24
Scenario 4	GZIP	TRUE	2333.82	216.08
Scenario 4	LZ4	FALSE	3950.74	480.85
Scenario 4	LZ4	TRUE	2255.93	249.19
Test			**P-value**	**Meaning**
Shapiro-Wilk			< 0.001	Not normal
Levene's Test			0.12	Equal variances
Kruskal-Wallis			< 0.001	Significant differences

To go more in-depth, we performed pairwise comparisons between independent groups using the Mann-Whitney test, which further validates these findings in Table 13. Significant differences were observed between binary true and false settings for both GZIP and LZ4 in most scenarios.

Table 13: Pairwise Comparison for GZIP and LZ4 Compression

Scenario	**GZIP Comparison (True vs False)**	**LZ4 Comparison (True vs False)**
Scenario 1	< 0.01	0.0556
Scenario 2	< 0.01	< 0.01
Scenario 4	< 0.01	< 0.01

When comparing the mean values in Table 12 and analyzing the boxplots that compare all scenarios in Figures 13, 14 and 15 we can conclude that enabling the binary flag resulted in a substantial reduction in transfer times for both GZIP and LZ4. This is evident from the lower mean transfer times.

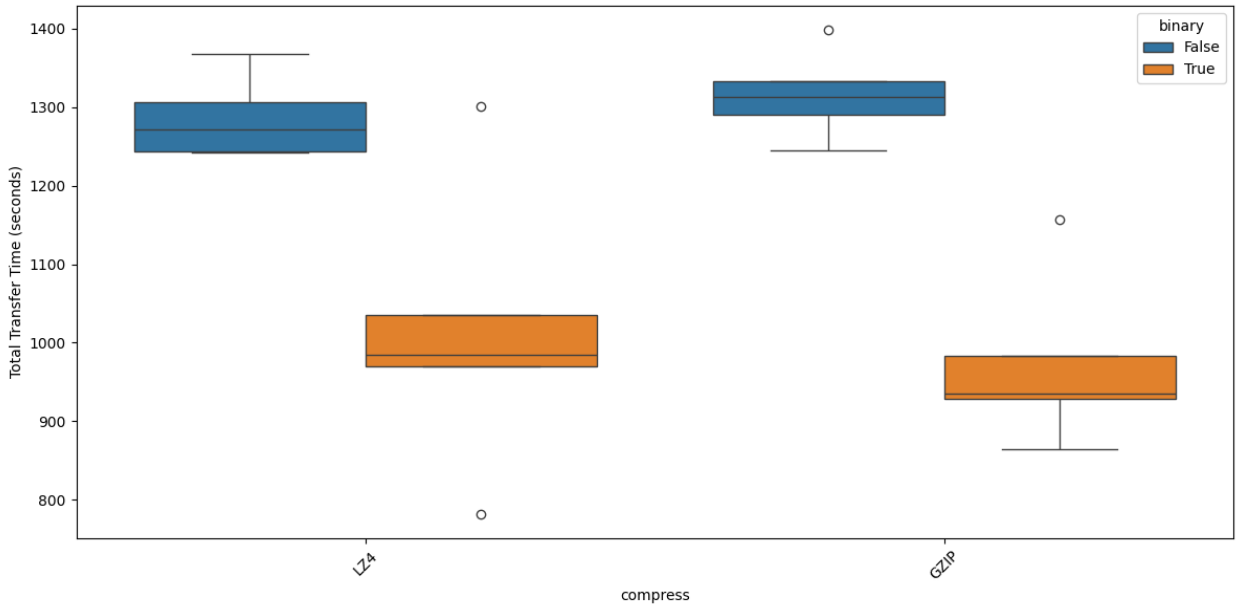

Figure 13: Box Plot of Total Transfer Time for Different Compression Methods and Binary Flags (Scenario 1,Case Study 2)

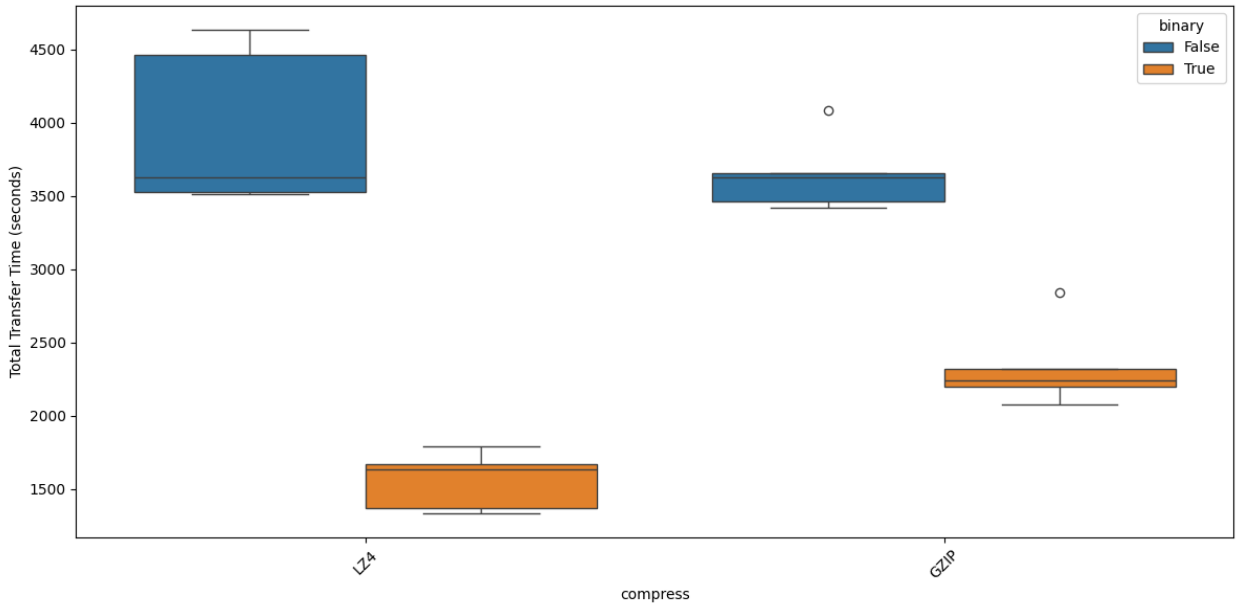

Figure 14: Box Plot of Total Transfer Time for Different Compression Methods and Binary Flags (Scenario 2,Case Study 2)

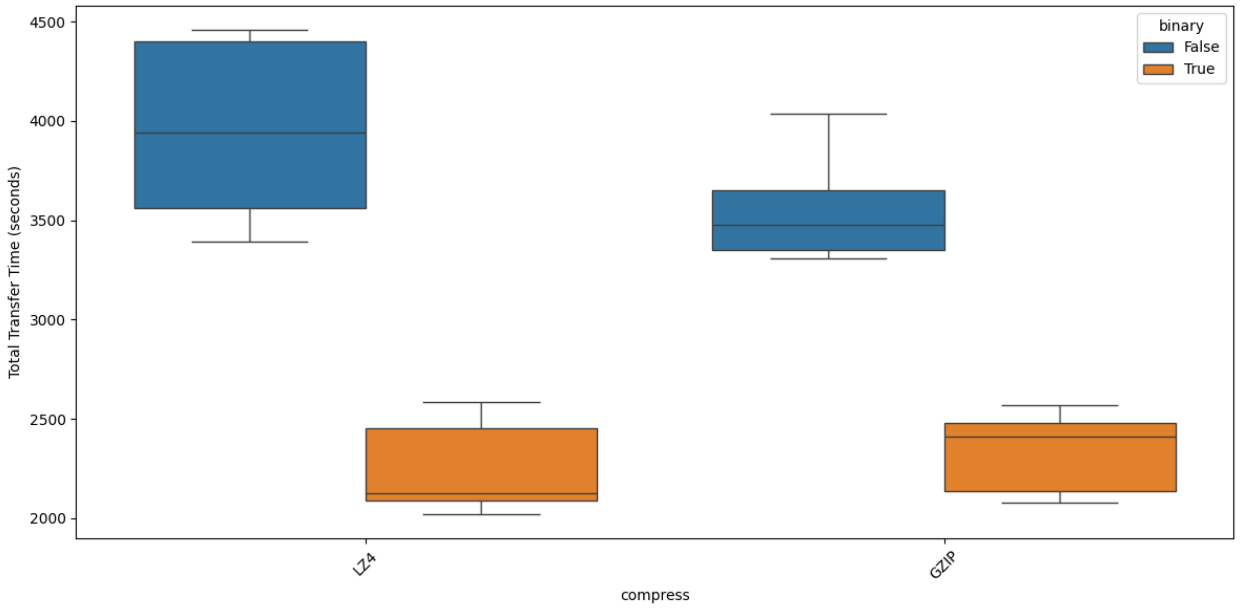

Figure 15: Box Plot of Total Transfer Time for Different Compression Methods and Binary Flags (Scenario 4,Case Study 2)

5.5.3 Conclusion

The empirical evidence from this study supports the engineers' claim that enabling binary mode during compression can significantly improve data migration efficiency. The findings highlight the importance of configuring the binary flag to true to achieve optimal performance in data migration tasks using the Db2 Migration Service.

5.6 Error Detection and Reporting

An important aspect of our framework is its comprehensive error detection and reporting mechanism. The framework is designed to log both informational messages and errors, allowing users to understand the problems that occurred during experiments. This capability ensures that any issues can be promptly identified and addressed, enhancing the reliability and usability of the framework. In this section, we present two examples of errors that the framework reported during the experiments:

- **Error 1: Query Error Detected**

  This error in table 14 is detected when the log "Query error: True" shows up. Following the trace of the error in the logs reveals an extraction process exiting with a non-zero return code, indicating an unexpected error. Further investigation shows a disk full error on the target system, which caused the SQL execution to fail.

Table 14: Log Details for Disk Full Error Detected

Log ID	Log Type	Time	Description
5,NO,testdb_testdb,Scenario4, 10, False	ERROR	16:35:54.085	<1003:MainThread> base_migration.py:267, Extraction process exited with non-zero return code! encountered an unexpected error
5,NO,testdb_testdb,Scenario4, 10, False	INFO	16:34:34.834	<1006:Thread-3>monitors.py:41, "***.***.***.***:50000 (TARGET):SQL3706N A disk full error was encountered on ""T17.49c20862d2eb4b2 b860bfce7 b49438aemigr.pipe.001.000 14557.0000002.log"". (-3706) (SQLExecDirectW)"

- **Error 2: Network Timeout and Connectivity Issues**

Table 15: Log Details for Network Timeout and Connectivity Issues

Log ID	Log Type	Time	Description
1-NO-testdb_testdb-scenario1-1-False-1717521515.2164514	ERROR	17:21:43.084	<1052:MainThread> query_executor.py:125, Timeout during the "***.***.***.***" connectivity check.
1-NO-testdb_testdb-scenario1-1-False-1717521515.2164514	ERROR	17:21:43.085	<1052:MainThread> query_executor.py:151, Db2 databases on hosts: [***.***.***.***] are not reachable.

This error in table 15 indicates that a network timeout occurred during a connectivity check. The log entries show that the Db2 databases on the specified hosts were not reachable, leading to a failure in connectivity verification.

These examples demonstrate the framework's ability to detect and report errors effectively, providing users with the information needed to troubleshoot and resolve issues. Each log entry includes a

timestamp of when the error occurred and the ID of the experiment, allowing users to identify the specific experiment and iteration in which the error happened.

6 Conclusion

The complexities inherent in data migration underscore the necessity for meticulous planning and robust tools. This thesis has explored the challenges and strategies associated with data migration, highlighting the importance of selecting optimal settings and configurations to ensure efficient and secure data transfers.

In this thesis, we have developed a comprehensive framework for monitoring data migration engines, suitable for both on-premise and cloud deployments. Chapter 4 presented the architecture of this framework and outlined the steps required for its use. We explored various migration engines supported by the framework, showcasing its versatility. In Chapter 5, we demonstrated the framework's efficiency through detailed case studies. Implementing this framework sets a solid foundation for further research, aimed at enhancing the data migration process and ensuring improved performance and reliability.

The case studies presented have provided valuable insights into the impacts of different compression techniques on the performance and efficiency of data migration. The findings underscore the importance of choosing appropriate compression algorithms and configurations tailored to specific workloads and system characteristics. These studies contribute to a deeper understanding of how to optimize data migration strategies, ensuring reliable and efficient data transfers.

In summary, the tools and methodologies developed and tested in this thesis offer significant advancements in the field of data migration. They provide a framework for future research and practical applications, enabling organizations to enhance the performance and reliability of their data environments. By addressing the challenges associated with data migration and offering practical solutions, this work supports the ongoing evolution of data management practices, ensuring that critical business operations can be maintained with minimal disruption and maximum efficiency.

6.1 Limitations

While this thesis provides comprehensive insights into the performance of different compression types and scenarios, it has certain limitations that must be acknowledged:

- **Size of the Data:** The datasets used for this analysis were relatively small. Testing the scalability of the framework and the engine with much larger datasets would provide more comprehensive insights.
- **Resource Consumption Monitoring:** Currently, the resource consumption data is only gathered for the migration engine. Including resource consumption data for the source and target databases would add significant value. This can be achieved by simply adding the containers of the databases to be monitored by cAdvisor.

- **Cloud-Based Engine Benchmarking:** Benchmarking the resource consumption of cloud-based engines presents challenges, as it is not always possible to gather detailed resource consumption metrics. However, using cloud-based software to gather logs about the engine can provide some insights.
- **API Limitations in Cloud Networks:** Some cloud-based networks, like Fivetran, do not provide APIs to get detailed information about experiments that ran. Therefore, workarounds are needed to gather the maximum possible information from available APIs and perform necessary calculations.

6.2 Future Work

Based on the limitations and findings of this thesis, the following areas are suggested for future research:

- **Scalability Testing:** It is essential to test the scalability of the framework and engine with much larger datasets to understand the performance implications at scale.
- **Cloud-Based Engine Metrics:** Developing methods to gather resource consumption data for cloud-based engines, possibly through logs and other available tools, can provide more detailed insights.
- **Additional Compression Techniques:** Exploring a wider array of compression techniques beyond GZIP and LZ4, such as BZIP2, Zstandard, or Snappy, could yield more comprehensive performance insights.
- **Supporting Additional Migration Engines:** Extending the framework to support other migration engines, such as AWS Database Migration Service and Google Cloud Database Migration Service, would make it more versatile.
- **Running More Experiments:** Conducting additional experiments with the already supported engines to gather more data would enhance the robustness of the findings.
- **Comparative Studies:** Running comparative studies between different migration engines on the same datasets can help identify the best engine for specific scenarios.
- **Predictive Modeling:** Developing a predictive model that utilizes the gathered experimental data to predict the best engine and configuration for a given scenario without the need to run new experiments.

- **Self-Adaptive System:** The ultimate goal is to create a self-adaptive system that, based on predictive models, can dynamically adjust configurations even during the migration experiment itself.

Bibliography

[1] F. Hamouda, M. Fokaefs, and D. Jania, "Dmbench: Load testing and benchmarking tool for data migration," in *Companion of the 15th ACM/SPEC International Conference on Performance Engineering*, ICPE '24 Companion, (New York, NY, USA), p. 47–51, Association for Computing Machinery, 2024.

[2] Datastax, "How apache kafka messages are written," 2022. [Online; accessed 17 May 2024].

[3] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[4] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, pp. 50–58, 04 2010.

[5] "Cloud data warehouse is the future of data storage." `https://www.sigmoid.com/blogs/cloud-data-warehouse-is-the-future-of-data-storage/`. Accessed: 2024-03-1.

[6] S. S. Sarmah, "Data migration," *Science and Technology*, vol. 8, no. 1, pp. 1–10, 2018.

[7] A.-E.-E. Hussien, "Data migration need, strategy, challenges, methodology, categories, risks, uses with cloud computing, and improvements in its using with cloud using suggested proposed model (dmig 1) open access," *Journal of Information Security*, vol. 12 No.1 2021, pp. 79–103, 01 2021.

[8] A. A. Hussein *et al.*, "Data migration need, strategy, challenges, methodology, categories, risks, uses with cloud computing, and improvements in its using with cloud using suggested proposed model (dmig 1)," *Journal of Information Security*, vol. 12, no. 01, p. 79, 2021.

[9] R. Amin, S. Vadlamudi, and M. M. Rahaman, "Opportunities and challenges of data migration in cloud," *Engineering International*, vol. 9, no. 1, pp. 41–50, 2021.

[10] S. Groß and A. Schill, "Towards user centric data governance and control in the cloud," in *Open Problems in Network Security: IFIP WG 11.4 International Workshop, iNetSec 2011, Lucerne, Switzerland, June 9, 2011, Revised Selected Papers*, pp. 132–144, Springer, 2012.

[11] A. A. L. Abdul Rahman, S. Islam, C. Kalloniatis, and S. Gritzalis, "A risk management approach for a sustainable cloud migration," *Journal of Risk and Financial Management*, vol. 10, no. 4, p. 20, 2017.

[12] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, vol. 10, no. 3152676, pp. 10–5555, 2017.

[13] S. Shakya, "An optimal solution for secure data transfer in cloud computing environments," *International Journal of Computer Applications*, vol. 178, no. 42, pp. 12–18, 2019.

[14] A. Iqbal and R. Colomo-Palacios, "Key opportunities and challenges of data migration in cloud: Results from a multivocal literature review," *Procedia Computer Science*, vol. 164, pp. 48–55, 2019.

[15] N. Mohammad, "Data integrity and cost optimization in cloud migration," *International Journal of Information Technology*, 2021.

[16] W. S. Dewi, E. Utami, and B. Sudaryatno, "Database migration using data synchronization and transactional replication," *International Journal of Innovative Technology and Exploring Engineering*, 2019.

[17] A. Astrand, *Re-engineering a database driven software tool: Rebuilding, automating processes and data migration*. PhD thesis, Osuva University of Vaasa, 2020.

[18] "Bringing direction to the indefinability of modern cloud migration." https://www.dbta.com/Editorial/News-Flashes/Bringing-Direction-to-the-Indefinability-of-Modern-Cloud-Migration-164025.aspx. Accessed: 2024-03-1.

[19] "Database migration performance testing." https://huddle.eurostarsoftwaretesting.com/database-migration-performance-testing/. Accessed: 2024-03-1.

[20] DBTA, "2021 Hadoop-to-Cloud Migration Benchmark Report." https://www.dbta.com/DBTA-Downloads/ResearchReports/2021-Hadoop-to-Cloud-Migration-Benchmark-Report-11067.aspx, 2021.

[21] C. Puliafito, D. M. Goncalves, M. M. Lopes, and F. Merlino, "Mobfogsim: Simulation of mobility and migration for fog computing," *Simulation Modelling Practice and Theory*, vol. 101, p. 102023, 2020.

[22] M. H. Shirvani, A. M. Rahmani, and A. Sahafi, "A survey study on virtual machine migration and server consolidation techniques in dvfs-enabled cloud datacenter: taxonomy and challenges," *Journal of King Saud University - Computer and Information Sciences*, 2020.

[23] D. A. Shafiq, N. Z. Jhanjhi, A. Abdullah, and M. A. Alzain, "Load balancing techniques in cloud computing environment: A review," *Journal of King Saud University - Computer and Information Sciences*, 2022.

[24] M. S. Aslanpour, S. S. Gill, and A. N. Toosi, "Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research," *Internet of Things*, 2020.

[25] K. Karthikeyan, R. Sunder, and K. Shankar, "Energy consumption analysis of virtual machine migration in cloud using hybrid swarm optimization (abc–ba)," *The Journal of Supercomputing*, 2020.

[26] J. Morris, *Practical Data Migration*. BCS Learning & Development Ltd., 2nd ed., 2012.

[27] B. Dufrasne, A. Warmuth, J. Appel, *et al.*, *DS8870 Data Migration Techniques*. IBM Redbooks, 2017.

[28] Cloudflare, "What is data migration?."

[29] Qlik, "Data migration."

[30] "Ibm db2 migration service." `https://www.ibm.com/docs/en/db2/11.5?topic=db2-migration-service`. Accessed: 2024-03-4.

[31] "Ibm db2." `https://www.ibm.com/products/db2`. Accessed: 2024-03-4.

[32] "Python." `https://www.python.org/`. Accessed: 2024-03-4.

[33] "Paramiko." `https://www.paramiko.org/`. Accessed: 2024-03-4.

[34] "Ssh." `https://www.cloudflare.com/learning/access-management/what-is-ssh/`. Accessed: 2024-03-4.

[35] "Mysql." `https://www.mysql.com/`. Accessed: 2024-03-4.

[36] IBM, "What is containerization?."

[37] "Kafka." `https://kafka.apache.org/`. Accessed: 2024-03-4.

[38] G. C. Team, "cadvisor: an open-source container monitoring and performance analysis tool," 2014.

[39] Prometheus community, "Node exporter: a software component used in conjunction with prometheus for monitoring linux and unix system."

[40] Julius Volz and Björn Rabenstein and Matt Bostock, "Prometheus : an open-source monitoring and alerting toolkit," 2012.

[41] "Grafana." https://grafana.com/. Accessed: 2024-03-4.

[42] Dwight Merriman, Eliot Horowitz, and Kevin Ryan, "Mongodb: an open-source, document-oriented nosql database," 2007.

[43] IBM, "Db2 migration service," 2023. [Online; accessed 17 May 2024].

[44] Fivetran, "Fivetran." https://www.fivetran.com/. Accessed: 2024-05-09.

[45] IBM, "Ibm db2 overview," 2023. [Online; accessed 17 May 2024].

[46] S. Madyalkar, "Maximizing data migration efficiency with db2move," 2023. [Online; accessed 17 May 2024].

[47] Virtual DBA, "A comprehensive guide to migrating to ibm db2: Benefits, risks," 2023. [Online; accessed 17 May 2024].

[48] IBM, "Scale new applications, analytics, and ai with fully managed databases," 2023. [Online; accessed 17 May 2024].

[49] Jazz.net, "Optimized concurrent database migration," 2023. [Online; accessed 17 May 2024].

[50] Transaction Processing Performance Council, "Tpc-h benchmark," 2023. [Online; accessed 17 May 2024].

[51] SingleStore, "Unlocking performance: Singlestore's tpc-h benchmark on 4th generation intel xeon scalable processors in aws," 2023. [Online; accessed 17 May 2024].

[52] P. Inc., "Which compression algorithm or tool should you use?." https://www.privex.io/articles/which-compression-algorithm-tool/, 2023. Accessed: 2024-08-21.

Appendices

A Appendix 1

```
[targetServer]
host = ***
username = ***
password = ***
port = ***
type = ***

[sourceServer]
host = ***
username = ***
password = ***
port = ***

[KafkaCluster]
host = ***
port = 9092
performanceBenchmarkTopic = performanceBenchmark
migrationEngineTopicName = migrationEngine
frameworktopicname = framework

[migrationEnvironment]
migrationEngineDockerImage = fareshamouda/db2migrationservice:test
loggingId =
numberofexperiments = 6
time_to_wait_beforeExperiment = 10

[experiment]
compress = LZ4, GZIP, NO
sourceDatabasetoTargetDatabase = testdb_testdb
tables = LINEITEM1_ORDERS2_ORDERS10, GIGANTICORDERS,
ORDERS2_ORDERS3_ORDERS4_ORDERS5_ORDERS6_ORDERS7_ORDERS8, ORDERSWIDE2
maxStreams = 10
binary = False
```

B Appendix 2

```
[targetServer]
host = ***
username = ***
password = ***
port = ***
type = ***

[sourceServer]
host = ***
username = ***
password = ***
port = ***

[KafkaCluster]
host = ***
port = 9092
performanceBenchmarkTopic = performanceBenchmark
migrationEngineTopicName = migrationEngine
frameworktopicname = framework

[migrationEnvironment]
migrationEngineDockerImage = fareshamouda/db2migrationservice:test
loggingId =
numberofexperiments = 6
time_to_wait_beforeExperiment = 10

[experiment]
compress = LZ4, GZIP
sourceDatabasetoTargetDatabase = testdb_testdb
tables = LINEITEM1_ORDERS2_ORDERS10, GIGANTICORDERS, ORDERSWIDE2
maxStreams = 5
binary = False, True
```