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

Martin Hilgendorf
Vincenzo Gulisano
Marina Papatriantafilou
martin.hilgendorf@chalmers.se
vincenzo.gulisano@chalmers.se
ptrianta@chalmers.se
Chalmers University of Technology, CSE Dept.
Gothenburg, Sweden

Jan Engström
Binay Mishra
jan.engstrom.4@consultant.volvo.com
binay.mishra@volvo.com
Volvo Group Trucks Technology
Gothenburg, Sweden

## ABSTRACT

In the age of big data and growing product complexity, it is common to monitor many aspects of a product or system, in order to extract well-founded intelligence and draw conclusions, to continue driving innovation. Automating and scaling processes in data-pipelines becomes essential to keep pace with increasing rates of data generated by such practices, while meeting security, governance, scalability and resource-efficiency demands.

We present FORTE, an extensible framework for robustness and transfer-efficiency in data pipelines. We identify sources of potential bottlenecks and explore the design space of approaches to deal with the challenges they pose. We study and evaluate synergetic effects of data compression and in-memory processing as well as task scheduling, in association with pipeline performance.

A prototype implementation of FORTE is implemented and studied in a use-case at Volvo Trucks for high-volume production-level data sets, in the order of magnitude of hundreds of gigabytes to terabytes per burst. Various general-purpose lossless data compression algorithms are evaluated, in order to balance compression effectiveness and time in the pipeline.

All in all, FORTE enables to deal with trade-offs and achieve benefits in latency and sustainable rate (up to 1.8 times better), effectiveness in resource utilisation, all while also enabling additional features such as integrity verification, logging, monitoring and traceability, as well as cataloguing of transferred data. We also note that the resource efficiency improvements achievable with FORTE, and its extensibility, can imply further benefits regarding scheduling, orchestration and energy-efficiency in such pipelines.

## CCS CONCEPTS

• **Information systems** → **Data management systems**; • **Applied computing** → **Enterprise data management**; • **Computer systems organization** → *Distributed architectures.*


*DEBS '23, June 27–30, 2023, Neuchatel, Switzerland*

ACM ISBN 979-8-4007-0122-1/23/06…$15.00
https://doi.org/10.1145/3583678.3596892

## KEYWORDS

Data pipelines, Internet of Things, distributed processing, edge computing, data transfer efficiency, resource utilization.

**ACM Reference Format:**
Martin Hilgendorf, Vincenzo Gulisano, Marina Papatriantafilou, Jan Engström, and Binay Mishra. 2023. FORTE: An Extensible Framework for Robustness and Efficiency in Data Transfer Pipelines. In *The 17th ACM International Conference on Distributed and Event-based Systems (DEBS '23), June 27–30, 2023, Neuchatel, Switzerland.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3583678.3596892

## 1 INTRODUCTION

Complex systems with advanced sensing capabilities produce large quantities of data at increasingly higher rates. This data needs to be transferred and analysed, often with certain time constraints, through *data pipelines* [7, 19]. Often, data is generated at remote locations by edge devices with constraints in processing capacity, far away from data centres with abundant resources, while to enable processing and analytics for a holistic view of a system, larger volumes of data need to be consolidated in the latter. To this end, *data transfer pipelines* are required to automate the reliable and efficient transfer of large data volumes. For the latter, besides data transmission, some amount of processing becomes necessary for *automating* the work, as well as for managing data aspects such as integrity and confidentiality.

Scaling to meet the aforementioned demands requires efficient use of available resources [6], to minimise the latency of the pipeline. Hence, data transfer pipelines should be deployed spanning the whole system, from where data is sensed/generated (edge) to where (centralised) processing/decision-making takes place. Limited resources at the edge should therefore also be leveraged, to e.g., perform data compression in preparation for a transfer with limited throughput to gain an overall improvement in pipeline latency.

*Challenges in the problem area.* Let us consider an example for context, a real use-case scenario from Volvo Trucks, illustrating key aspects of the challenges [8]. Data from a fleet of development vehicles is sensed at distant locations, such as test tracks and customer sites, to support data-driven product development. Capture rates on the order of 1 GB/min per vehicle are common during such tests. To analyse data in a timely manner, the data must be made available to engineers at a remote data centre, providing them

with storage systems and processing clusters. These require performant and scalable automated data transfer pipelines, to support growing data volumes within existing hardware systems. *Latency*, *throughput* and *sustainable rates* are performance metrics used to demonstrate and argue about efficiency, particularly with respect to constrained resources [6, 7]. A challenge is that contention for such resources increases with growing data volumes and additional processing steps required to both manage automation as well as complement with associated data management qualities, such as integrity, confidentiality, logging, traceability. There is therefore need to capitalise on available concurrent computational power, avoiding slowdowns from dependencies, essentially exploiting data and task parallelism possibilities. Due to the complexity of such a system, a large number of design choices must be made, resulting in multiplicative growing combinations of parameter choices to optimise *pipeline performance and resource utilisation*, as well as balance trade-offs.

*Contributions.* To address these challenges, this industry experience article proposes FORTE, an extensible framework for robustness and transfer-efficiency in data pipelines. We identify sources of potential bottlenecks (e.g., network bandwidth and node-bandwidth) and explore the design space of approaches to deal with several of the challenges they pose. The framework alleviates transmission bandwidth bottlenecks by transferring compressed data and balances the trade-offs between compression time (which adds latency to the overall pipeline) and transfer-effectiveness (which reduces latency, increasing throughput and sustainable rates for transfers). Furthermore, we explore combined effects of such lossless data compression with in-memory processing, as well as task scheduling, in association with pipeline performance.

An implementation of a pipeline based on FORTE is carried out on Apache Airflow [2], a workflow management platform, and is studied using high-volume production-level data sets available through a large-scale use-case of Volvo Trucks. The data amounts to the order of magnitude of hundreds of gigabytes to terabytes per burst (corresponding to the aforementioned capture rates). The detailed experiment study focuses on latency, throughput and sustainable rates, scalability, as well as resource utilisation monitoring, to expose imbalances that are addressed. FORTE enables to deal with trade-offs and to achieve a significant uplift in sustainable rate over the existing baseline, while also providing previously missing desired functionality, such as integrity verification, logging, monitoring and traceability, as well as cataloguing of handled data.

Building on the extensibility properties of the framework, the work also shows:

(1) a variation of the pipeline design using in-memory processing, alleviating an identified hardware bottleneck, and supporting real data rates exceeding 1.4 TB per period with bursty data arrival. This is up to 1.8 times higher than the baseline in representative highly demanding data loads, using identical infrastructure and hardware resources.
(2) an integration with an alternative task scheduling approach, shedding light on trade-offs between average batch throughput and pipeline parallelism.

Additionally, we show resource utilisation metrics for the diverse pipeline designs and resources used by pipeline, motivating the design decisions in FORTE as well as getting insights on their effects on balancing resource usage, alleviating bottlenecks, and improving the overall pipeline performance.

In this article, § 2 introduces preliminaries and describes the problem in some more detail, § 3 motivates and overviews the design decisions in FORTE, § 4 presents the main use-case and subsequent evaluation, while § 5 discusses related work, and § 6 concludes the paper.

## 2 PRELIMINARIES AND PROBLEM DESCRIPTION

### 2.1 System model and general problem

As mentioned in the previous section, we focus on data transfer pipelines; for simplicity, we omit the term "transfer" and simply refer to "(data) pipelines".

A straightforward approach for transferring the data from a point A to a point B is to use existing tools for copying data over networks. This direct approach can be considered as a point of reference, particularly in an industrial setup in which practitioners do not have access to dedicated frameworks and platforms to develop and execute pipelines — instead opting for a more basic solution using well-known and established tools. As such, the baseline focuses purely on the essential requirement to mirror all arriving data to the destination, without supporting additional desired properties of an industrial data pipeline, such as reliability, security, data governance, and observability.

An *automated data transfer pipeline* consists of a sequence of *tasks*, which data passes through in order, towards a given destination. Data items are pushed through the tasks of a pipeline either individually or in *batches*. To simplify the discussion, we use the term batch throughout this work, although the batch may just be a single data item, or a set of items, e.g. files. Each pipeline task has a *service rate*, measured in MB/s, describing the sustained rate at which it can operate on data. Some tasks are purely transmitting data, while other user-defined ones can be responsible for functionality such as creating batches, computing and comparing checksums for data integrity verification, or encrypting data before transfers over unprotected channels. Each pipeline task operates on a single batch at once, but different tasks may operate on different batches in parallel. Figure 2 illustrates the processing of three data batches by a pipeline consisting of three tasks. Batches may experience *queuing delay* when a preceding data batch is still being processed by a task.

As the data volumes grow, resource requirements for operating on them grow as well. Network bandwidth is a common limiting factor in big data transfers, so introducing compression and decompression tasks, while not serving the ultimate goal of transferring data or meeting industrial requirements, can be seen as yet another kind of task for the purpose of improving overall pipeline performance. By having further tasks, the overall resource requirements of such a pipeline change, thus leading to more parameters to consider in the problem.

Figure 1 illustrates an industrial setup target example, derived from the motivating use-case and running example application for this work. Large volumes of data from a fleet of field test vehicles

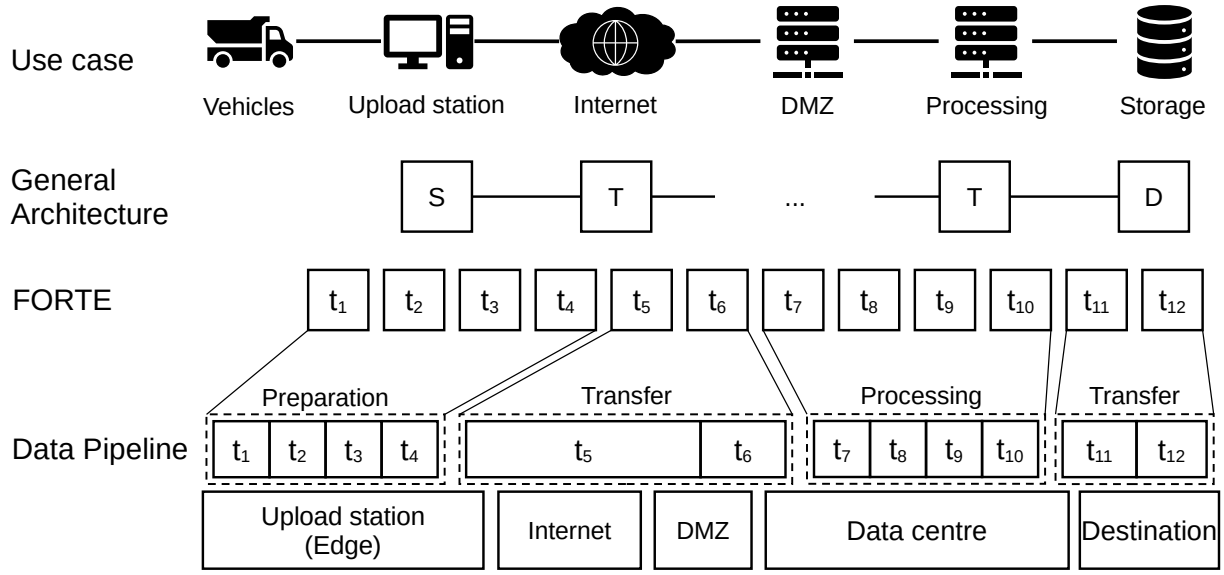



**Figure 1: Overview of a typical data flow for which a data pipeline is desired. The industrial use-case is illustrated at the top, and a general architecture for a data pipeline for this application is shown, using S to denote the data source, T for pipeline tasks, and D as the destination. FORTE is used to structure a sequence of abstract processing tasks and allocate them to the system components to instantiate a functional data pipeline.**

at remote test sites are recorded and placed on a local upload station, from where they are transferred to a final destination, the production zone; there, they are made available to engineers to analyse them, after integrity verification and data quality checks are performed.

The aforementioned baseline would handle all data files as a single batch, excluding possibilities both for embedding these checking tasks in the process, as well as for exploiting data parallelism in the system. The approach is performance-limited by bandwidth and sequential task bottlenecks when transferring data across the Internet and cannot scale with increasing data volumes.

To summarise, the requirements for an automated industrial data pipeline include:

(1) Timely delivery of data — desire for short lead times between data capture and analysis;
(2) Scaling with increasing load — this requires studying various trade-offs in the system, to e.g. balance processing and transmission times and identify ways to do latency hiding.

## 2.2 Metrics of interest

In order to evaluate pipeline transfer timeliness, we identify key metrics of interest, also referred to as Key Performance Indicators (KPIs). These include *batch latency* of a batch $b$, measured as the time for the batch to traverse the complete pipeline; e.g., see the "Batch $b_2$ start" and "Batch $b_2$ end" markers in Figure 2. Batch latency, denoted $l_b$, is thus equivalent to the sum of queuing delay and processing time for all tasks in the pipeline:

$$l_b = \text{Batch}_{\text{End}} - \text{Batch}_{\text{Start}} = \sum_{i=1}^{N} q_i(b) + \sum_{i=1}^{N} t_i(b)$$

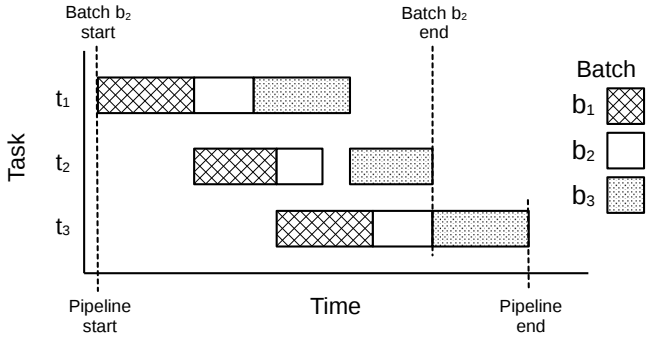



**Figure 2: Execution schedule of a pipeline with three tasks, processing three batches from the same burst.**

where $q_i(b)$ is the queuing delay experienced by batch $b$ before beginning processing by task $i$, of a pipeline consisting of $N$ tasks. Similarly, $t_i(b)$ is the time taken for batch $b$ to be handled by task $i$ (recall that the task can involve processing and/or transmission).

Batch latency in a transfer pipeline is directly related to the *batch size*, the total size of data in a batch. As batches may vary in size, we define *batch throughput* $R_b$ of a batch $b$ (measured in MB/s), as:

$$R_b = \frac{\text{Size}(b)}{l_b}$$

which is used as a normalised metric to enable comparison between batches of different sizes.

Similarly, *pipeline throughput*, denoted $R_P$, is a KPI that describes the average rate at which the pipeline transfers data during a specific time interval. Consider a period of continuous pipeline operation of duration $T$ during which $B$ batches transferred. The real time duration $T$ starts as the first batch begins processing by the first pipeline task, and ends when the final batch completes processing by the final pipeline task (Figure 2). The total data volume is the sum of the sizes of all $B$ data batches handled by the pipeline during the interval $T$. Pipeline throughput for this period is then defined as:

$$R_P = \frac{\text{Total data volume}}{\text{Pipeline}_{\text{End}} - \text{Pipeline}_{\text{Start}}} = \frac{\sum_{i=1}^{B} \text{Size}(b_i)}{T}$$

To evaluate whether the performance of a pipeline is sufficient for a given application, we also consider the *nominal data rate*, the rate at which input data is generated (measured in MB/s) and compare it to the *real data rate* (also in MB/s), the rate at which the pipeline can accomplish transferring data, i.e. the pipeline throughput.

If data batches arrive in several bursts, for example at the end of each of two 8-hour working shifts on a customer site, the effective transfer rate for each batch is $R_P$, as in the preceding formula. Note that deriving the real data rate of the pipeline over a larger interval (e.g. a 24 h one) from this $R_P$ provides a pessimistic estimate of the real data rate for automated pipelines (as this calculation does not account for actual latency hiding that can happen across bursts of batches). In the baseline, the latter is the actual real data rate, due to the way that the data is transferred.

In order to systematically identify and address pipeline performance bottlenecks, the underlying causes for bottlenecks need to be identified. *Resource utilisation* measurements of the various hardware resources used by the pipeline give important insight when diagnosing performance limitations that may lead to bottlenecks. Hence, additional relevant metrics include *utilisation* levels of CPU cores, system memory, disk bus bandwidth, and network bandwidth.

# 3 OVERVIEW OF THE PROPOSED APPROACH

## 3.1 Problem insights

Data pipelines place high demands on system performance. For example, in the context of the case study that we also use as a running example, the baseline approach poses hard limitations on performance in terms of how much data per day it can handle, given that more field test vehicles are introduced and produce data.

Commonly, there are two complementary approaches to improving system performance: *hardware scaling* and *performance tuning*. Hardware scaling addresses a lack of resources by procuring and deploying more systems and resources for a process to use. In the context of data transfer processes, this could entail increasing network bandwidth by upgrading hardware or provider services. Performance tuning, on the other hand, aims to reduce resource requirements of an existing system for performing the same amount of work. By optimising resource utilisation by a system to be more efficient, the system can perform more work using the same quantity of resources. This requires a detailed understanding of the system and the intricate inter-dependencies between its components. Identifying and addressing bottlenecks requires studying the various *trade-offs* that arise in complex systems.

In face of limited bandwidth, especially when performing long-distance transfers across the Internet, hardware upgrades such as high-bandwidth fibre-optic connections can be economically infeasible. Instead, we consider the possibility of compressing data at the source before transferring it across such a bottleneck link. However, adding such auxiliary tasks in an attempt to improve performance places even higher demand on the limited computing resources at the edge, which are already loaded with the encryption and checksum computation tasks necessary for the automated pipeline.

This *trade-off* requires the preparatory processing at the source to be performed in a particularly efficient manner, to utilise the resources well while avoiding introducing another, potentially more severe, throughput bottleneck to the pipeline. If a data pipeline is bottlenecked by hardware limitations, such as Internet upload bandwidth, this limited resource should be utilised to the highest possible extent — as long as there is data yet to pass this bottleneck in a pipeline, this limited resource should not be idle. To facilitate this, all preceding processing must be fast enough, which becomes a major challenge due to the limited processing resources available.

To summarise, extra tasks such as encryption and integrity verification — which do not directly support the primary goal of transferring the data but are valuable nonetheless — need to be performed carefully to avoid increasing pipeline latency further. Besides, processing and transmission times need balancing, and options for latency hiding are desirable; i.e., a large number of design choices must be made, resulting in multiplicative growing combinations of parameter choices, for such optimization and balancing. With the challenges and these thoughts in mind, we propose the key ideas in this work, described in the following subsections.

## 3.2 The proposed framework

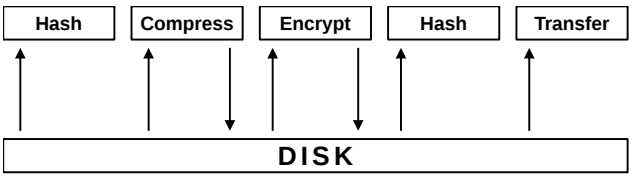
The architecture of the proposed framework FORTE for big data transfer pipelines is illustrated in the lower half of Figure 1. Three main sections can be identified, which consist of sets of corresponding tasks:

**Preparation:** Data to be processed by the pipeline arrives on the remote upload station at the edge of the network. To support observability and traceability, all data files are indexed into a metadata catalogue so that they can be tracked. Then, data is compressed and encrypted to prepare it for secure transfer across the Internet.

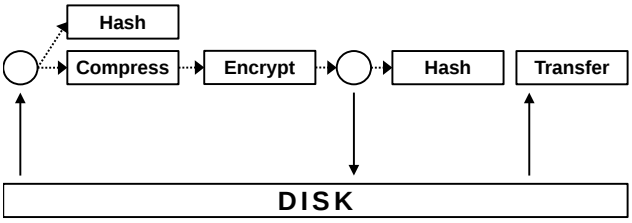
**Transfer:** The compressed and encrypted data files are transferred across the Internet from the upload station to the data centre. This pipeline section faces performance challenges due to bandwidth limitations during the network transfer.

**Delivery & Storage:** The arriving data is unpacked, recovering the original data files. Checksums are calculated and compared to guarantee data integrity. Then, data files are transferred to storage servers for access by various downstream consumers. Compute resources are more abundant than at the edge, and these tasks may be performed in parallel for the files contained in the batches.

Each batch of data passes through dedicated instances of this pipeline — there are no data dependencies or communication requirements between steps of different batches, and as such, they may run in parallel. The degree of parallelism that is achievable in practice depends on the available resources, which are limited




**(a) Read and write operations when intermediate results are written to and read from disk by each task.**




**(b) Read and write operations performed with in-memory processing. Dashed arrows indicate in-memory data transfers between tasks, which do not involve the disk.**

**Figure 3: In-memory processing for reducing the amount of disk I/O operations. Vertical arrows indicate data travelling between disk and various processing tasks in a pipeline.**

on thin edge devices. Further, performance of data transfers may suffer with too high parallelism as contention and collisions on the communication link increase. The framework therefore needs to provide control over the degree of parallelism at various identified points in the pipeline to allow tuning for efficient operation in various environments. Further, scheduling of the tasks is significant to avoid bottlenecks resulting from convoy effects.

A subsequent consideration is the following: When processing large volumes of data, local *disk I/O throughput* on a host can quickly become a limiting factor when reading and writing large volumes in parallel. This problem is exacerbated when spinning storage media devices are used for cost-effective bulk storage of big data, as concurrent access to various physical locations incurs significant overhead. The main option for counteracting these effects is to reduce the number of accesses to the data stored on the disk. To address this, we propose the use of *in-memory* processing to avoid the potential bottleneck. In-memory processing aims to avoid reading and writing intermediate results to and from the disk, and instead passes data between parallel processing steps via shared memory. A significant amount of disk I/O operations can be avoided, as illustrated in Figure 3.

## 3.3 Implementation aspects

Following these ideas, a pipeline is implemented to transfer data from a remote upload station host with limited hardware resources, across the Internet, to a data centre. As shown in Figure 1 and Table 1, the pipeline architecture is composed of a sequence of steps. This type of workflow can effectively be modelled as a directed acyclic graph (DAG). In this workflow DAG, tasks correspond to nodes and dependencies between tasks induce the edges and an execution schedule for the workflow can be determined. To this end, our implementation uses Apache Airflow [2], an open-source orchestration platform for implementing and executing workflows. Various workflow tools exist in the open-source software ecosystem, such as Apache Oozie[1], Azkaban[2], Luigi[3], or Argo Workflows[4]. However, Oozie, Azkaban, and Luigi all focus purely on orchestrating data processing workflows within Apache Hadoop clusters, limiting their applications outside this environment. Similarly, the Argo suite of tools, which includes Argo Workflows, specialises in orchestration of containerised workloads in Kubernetes environments [1]. As such, Airflow is selected for the implementation of the pipeline due to its flexibility (it integrates with a wide variety of systems and protocols), and scalability due to its modular architecture. Airflow further facilitates the operation of the pipeline via a graphical user interface, monitoring of workloads, and traceability via detailed task logs.

Regarding scheduling, the implementation platform does not provide explicit support for priorities. However, by explicitly maintaining the tasks in priority queues/heaps, it is possible to enforce other policies too. The next section explores aspects of the scheduling influences, by studying such an example, following a Shortest Task/Job First approach, known to optimise the system throughput and the average waiting time per task in common scheduling systems. However as this system is a multi-stage pipeline, the total effect may differ. The next section gives more insights on these aspects.

# 4 EMPIRICAL STUDY

In this section, we present the case-study used for the experimental evaluation of FORTE. First, we describe the evaluation environment in which the measurements were carried out, including infrastructure systems and the evaluation data set. Then, we describe the process of selecting a suitable data compression technique, which may vary by the type of data to be processed by the pipeline. This is followed by a detailed study of the various performance metrics of a transfer pipeline.

## 4.1 Use-case and experiment setup

The evaluation is performed in a target environment at Volvo Trucks, using the same components that will later be used by a production deployment of FORTE for the data transfer needs of the case-study. Here, we describe the various components of this system and how the previously identified pipeline tasks map to theses. Then, a suitable evaluation data set for use in performance benchmarks is sampled from real development data.

*Systems and infrastructure.* The evaluation environment consists of multiple systems and network links, geographically and logically spread out over a variety of regions. Figure 4 illustrates the geographical layout of the full system.

The pipeline begins at the remote upload station at the customer site 850 km away. This system is equipped with an Intel® Xeon® E-2236 12-core CPU operating at 3.40 GHz, 64 GB RAM, a 1 Gbit/s network link to its gateway router, running Ubuntu 20.04.4 LTS.

[1] https://oozie.apache.org/
[2] https://azkaban.github.io/
[3] https://github.com/spotify/luigi
[4] https://argoproj.github.io/workflows/

| Pipeline task | Description |
|---|---|
| $t_1$ | Calculate the SHA256 checksum of each file in the batch. |
| $t_2$ | Using the selected compression strategy, compress each data file in the batch. |
| $t_3$ | Encrypt all data files in the batch. |
| $t_4$ | Calculate the SHA256 checksum of each encrypted file in the batch. |
| $t_5$ | Transfer the batch from the edge system to the DMZ. |
| $t_6$ | Transfer from the DMZ relay to the internal network. |
| $t_7$ | Verify integrity of inbound data files by comparing SHA256 checksums before any processing begins. |
| $t_8$ | Decrypt all data files in the batch. |
| $t_9$ | Decompress data files. |
| $t_{10}$ | Compare SHA256 checksum to verify integrity after processing is completed. |
| $t_{11}$ | Transfer data to a network attached storage for archiving. |
| $t_{12}$ | Transfer data to a network attached storage for usage by analysts and engineers. |

**Table 1: The tasks comprising the pipeline in the case-study implementation.**

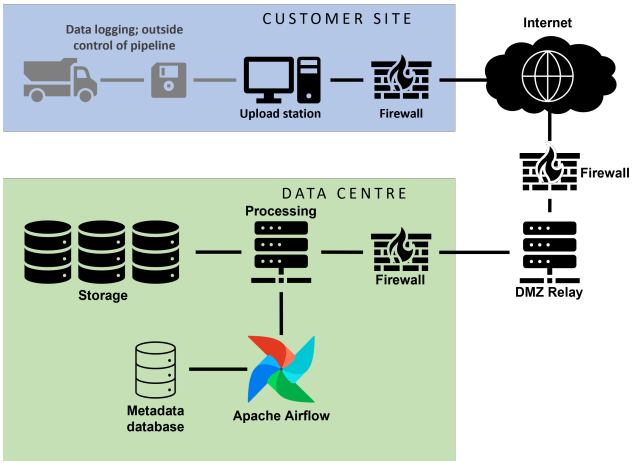



**Figure 4: Systems present in the evaluation environment.**

From the upload station, data is transferred over the Internet to Volvo networks. Data protection during this transfer is guaranteed by common network security techniques, such as VPN tunnelling and SSH for remote authentication and access. These protocols introduce additional, but unavoidable, communication and processing overhead, which further reduces data rates.

The next host in the pipeline sequence is the relay host in the demilitarised zone. This is a virtualised host which functions as a buffer for inbound data on the way to hosts on the internal network, and does not perform any processing tasks itself. Disk capacity (at over 20 TB) is abundant for the expected data volumes. As such, network bandwidth is the most significant resource at this node in the pipeline, though with a connection speed of 10 Gbit/s, the pipeline will not be bottlenecked here.

A dedicated physical server is used for any processing tasks after data has been pulled from the relay host to the internal network. This processing node has an Intel® Xeon® E5-2690 28-core CPU running at 2.60 GHz, 256 GB of RAM, and runs Red Hat Enterprise Linux Server 7.9. Data is stored on a network-attached storage cluster to enable access from anywhere within the data centre with high speed over a 10 Gbit/s network link. This design simplifies scaling out the processing steps to multiple servers in the future.

To implement and operate FORTE on this infrastructure, Apache Airflow version 2.1.2 and Python 3.8 are used. The Airflow deployment consists of the scheduler, web server, and a single worker instance, alongside a dedicated PostgreSQL database for persistent metadata storage and a Redis® instance for message brokering between the scheduler and worker processes. Further, a PostgreSQL 12.9 instance is used for storing FORTEs file metadata catalogue.

*Evaluation data set.* The controlled test data set consists of 337 GB authentic development data from the project in which the case-study is performed. The availability of such data allows for more accurate evaluation and conclusions, as it reflects attributes of the real data that could be missing in a synthetic data set.

In the case-study at hand, each start-up of a vehicle generates a new unique directory into which data for the following test run is placed. This directory structure provides the separation of data into batches required by FORTE without further pre-processing.

Batches in this application contain two types of data files: a constant set of plain-text log files, and a series of large data files, referred to as "bag files", storing sequences of timestamped signals. These bag files are continuously written while data is recorded, and each bag file contains data for a tumbling window of 60 s of vehicle operation, producing files between 500 MB to 1 000 MB in size. The log files are considerably smaller, with an average size around 350 kB each.

The benchmark data set contains 14 batches of such data. Batch sizes in the data set vary, proportional to the time the vehicle was running, which reflects a variation in size of data batches seen in the project. This allows examining pipeline performance and behaviour under realistic loading conditions. Figure 5 illustrates the distribution of batch sizes in the input data set.

We measure the overall performance of four designs for the data transfer pipeline:


**Figure 5: Evaluation data set batch sizes (labelled A–N). Several very small batches (A, B, E, L, M, N) are seen, as well as a particularly large batch I.**

**Baseline** The baseline approach as described in § 2, used as a point of reference for comparisons. Recall that this solution purely focuses on moving data, without any auxiliary processing.

***Compression-based*** (CB) The basic batched streaming pipeline in FORTE, as implemented on the Airflow platform, including additional functionality (e.g., encryption, integrity verification, traceability) and data compression.

***Compression-based with in-memory processing*** (CB-IM) The enhancement of the CB pipeline, utilising in-memory processing.

***Integration with shortest-task-first scheduling*** (CB-IM-SJF) Shortest-job-first scheduled adaptation of the CB-IM pipeline[5].

Before discussing the outcomes for the empirical study, in the following subsection we describe an overview of the compression benchmarking, that led to the choice adopted in the experiments.

## 4.2 Compression benchmarks

| Type | Count | Size | Proportion by count (%) | Proportion by size (%) |
|---|---|---|---|---|
| Log | 252 | 89.59 MB | 27.85 | 0.03 |
| Bag | 653 | 336.92 GB | 72.15 | 99.97 |
| Total: | 905 | 337.01 GB | 100.00 | 100.00 |

**Table 2: Evaluation data set composition by data file type.**

The distribution of file types in the test data set is shown in Table 2. Based on these statistics, a compression algorithm that is effective for bag files can act on 99 % of the overall data. This observation is at the basis of our first experiment, which aims to find a suitable compression strategy.

[5]Note that in this case the results are derived from the execution of CB-IM.

For this, four compression algorithms (Bzip2[6], LZ4[7], Gzip[8], and Zstandard[9]) are evaluated at each available *compression level* setting on a subset of 100 bag data files (54 GB) from the evaluation data set. The compression level is a tunable parameter, expressed as an integer, which allows tuning the trade-off between compression time and resource demand vs compression effectiveness to match the requirements of the application. To quantify compression effectiveness, *compression ratio* is defined as the ratio of original size against the compressed size; e.g., a compression ratio of 2 means that the compressed output is half the size of the input.

The sampled data files are compressed sequentially using every combination of algorithm and its possible compression level settings. To determine compression speed and effectiveness, the elapsed time for compression of all files and the total size of the compressed output were recorded, shown in Figure 6a and Figure 6b, respectively.

A distinct cluster of high-throughput compression performance is seen with the newer LZ4 and Zstandard algorithms at low compression levels. While these LZ4 strategies are particularly fast, the achieved compression ratio of 1.84 (Figure 6b) is the lowest of any strategy. Both Gzip and Bzip2 achieve a compression ratio between 2.2 to 2.3, which is consistently higher than LZ4 at any level. Zstandard is the only algorithm able to exceed a compression ratio of 2.5, but later plateaus until extremely high compression levels with low throughput are used. Lower compression levels of Zstandard achieve compression speeds in excess of 200 MB/s and compress the test data set in 3 to 5 minutes, while the highest levels slow down to levels similar to Bzip2 at less than 10 MB/s and require over 2 hours to complete.

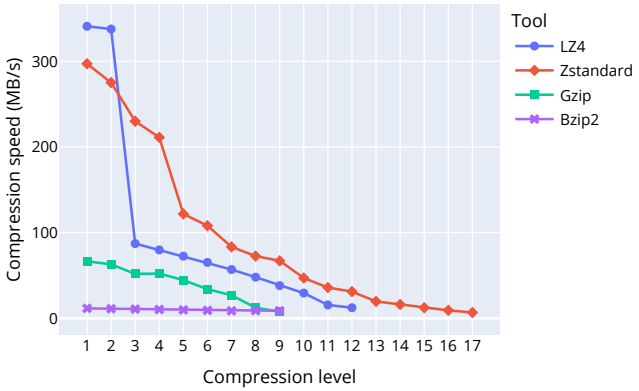
As the compression speed of Zstandard decreases significantly beyond level 4 while achieving only minor gains in compression effectiveness despite the significant additional time investment, Zstandard at low compression levels appears to be a promising strategy. Detailed performance data for the 4 lowest compression levels is shown in Table 3, along with relative performance compared to the highest speed and ratio amongst these strategies.

| Strategy | Speed (MB/s) | Delta from best speed (%) | Compression ratio | Delta from best ratio (%) |
|---|---|---|---|---|
| `zstd -1` | 297.1 | | 2.330 | −28.35 |
| `zstd -2` | 275.1 | −7.41 | 2.648 | −18.58 |
| `zstd -3` | 230.0 | −22.59 | 3.054 | −6.10 |
| `zstd -4` | 211.2 | −28.94 | 3.252 | |

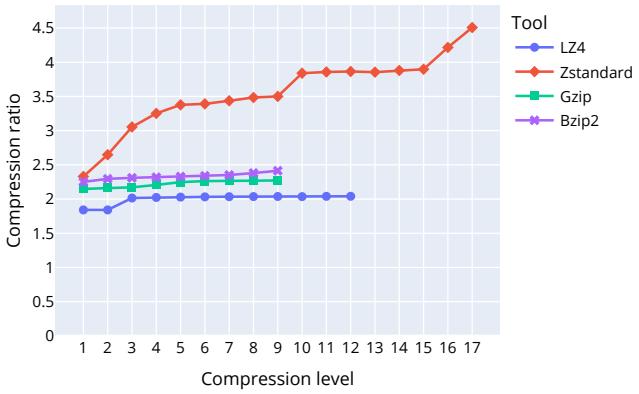
**Table 3: The compression strategies with best expected performance in the transfer pipeline application. The final decision is to select a balance point between speed and compression ratio.**

For implementation of the proposed transfer pipeline, Zstandard with compression level 3, the default, is selected as it exhibits a good balance between compression speed and effectiveness.

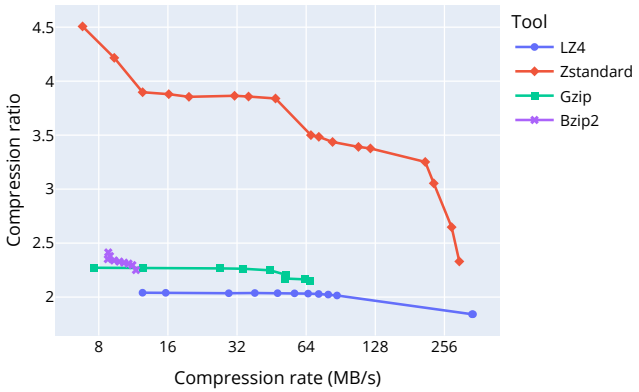
[6]https://sourceware.org/bzip2/
[7]https://lz4.github.io/lz4/
[8]https://www.gnu.org/software/gzip/
[9]https://facebook.github.io/zstd/




**(a) Compression speed of every strategy. The rate at which uncompressed input data is processed by the compression tool at the given compression level setting.**




**(b) Compression ratio achieved on the test data set by each strategy. Higher compression ratio implies better compression effectiveness.**




**(c) Compression rate versus ratio for all strategies.**

**Figure 6: Comparing compression effectiveness and processing rate.**

| Transfer step | Duration (s) |
|---|---|
| To DMZ | 5 779 |
| To landing zone | 3 272 |
| To storage | 3 491 |
| Full pipeline | 12 542 |

**Table 4: Latency for transfer of 337 GB test data set using baseline solution.**

| Pipeline design | $l_P$ (s) | $R_P$ (MB/s) | Real data rate (TB per 24h) |
|---|---|---|---|
| Baseline | 12 542 | 26.87 | 2.32 |
| CB | 9 002 | 37.44 | 3.23 |
| CB-IM | 6 842 | 49.26 | 4.26 |
| CB-IM-SJF | 7 465 | 45.15 | 3.90 |

**Table 5: Performance for transfer of 337 GB data set with the various pipeline designs. Nominal data rate in the application is 3.36 TB per 24 h.**

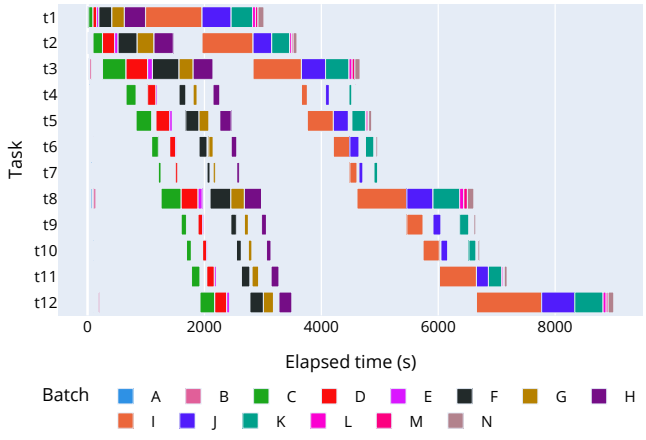
## 4.3 Evaluation results

*Pipeline performance.* Using each pipeline design, the full evaluation data set is transferred. For the baseline, which operates on the full data set as a single batch and processes files sequentially in each step, we record the duration of each of the three transfer steps. The execution of the parallelised pipelines is visualised as timeline diagrams in Figure 7a and Figure 7b. Based on the CB-IM execution timeline, we also simulate the execution of a shortest-job-first scheduling policy (shown in Figure 7c), as opposed to the FIFO ordering used by Airflow.

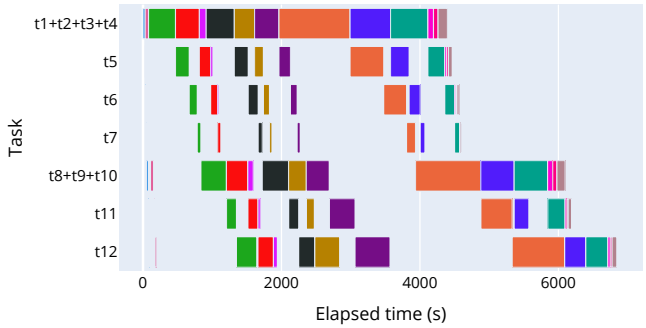
*Pipeline latency.* Table 5 lists the measured pipeline latency $l_P$ for each pipeline design to process the data set, and the corresponding pipeline rate $R_P$. From this, we make a pessimistic estimate of the real data rate of each design, which can the be compared to the nominal data rate in the application (up to 3.4 TB per 24 h for 7 vehicles operating for up to two 8-hour shifts each day). Both the baseline and CB do not sustain the required data rates to keep up with the data volumes, while the further adjustments made to CB-IM and CB-IM-SJF allow them to achieve significant performance uplifts over the baseline (+83 % and +68 %, respectively).

*Per-batch performance.* As data batches vary in size, large batches create a *convoy effect* as subsequent batches need to queue and wait under the FIFO scheduling policy. This is clearly visible in the execution timeline diagrams (Figure 7a and Figure 7b), where batch I significantly delays the processing of batches J–N. This leads to inefficient use of the following pipeline tasks and the associated system resources, which are idle for extended periods of time while the pipeline is blocked further upstream, and degrades per-batch performance metrics for the subsequent smaller batches.

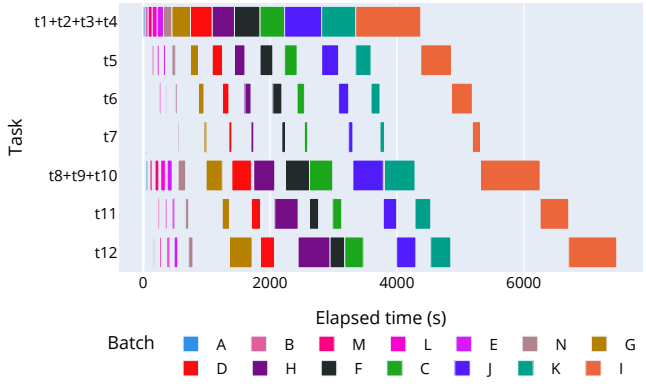
Due to FIFO scheduling in Airflow, smaller batches which arrive for processing after a large batch will experience significantly




**(a) Execution timeline of CB pipeline for all batches (A–N) of the test data set. Note the queuing (convoy effect) following batches C and I.**




**(b) Execution timeline of CB-IM pipeline for all batches (A–N) of the test data set. The preparation steps, which appeared as separate tasks in the CB pipeline, have been merged into a single task. Similarly, decryption, decompression, and integrity verification on the processing side have been grouped into a single task. The convoy effect, particularly following batch I, is still present.**




**(c) Execution timeline of CB-IM-SJF, a simulated shortest-job-first reordering of the in-memory pipeline execution seen in Figure 7b. Batches have the same label and colour as in the figures above, but are now processed ordered by size. The convoy effect is no longer present.**

**Figure 7: Execution timelines for transferring the evaluation data set using various pipeline designs.**

higher batch latency relative to their size. Figure 8a and Figure 8b show batches J–N spending 60 % to 97 % of their total batch latency queuing rather than being processed.

CB-IM alleviates the impact of this by improving processing latency and data rates in the preparation and processing stages of the pipeline. However, the relative waiting times for the last batches remain the same, and the suboptimal resource utilisation patterns with later pipeline tasks being idle for extended periods of time are still present.

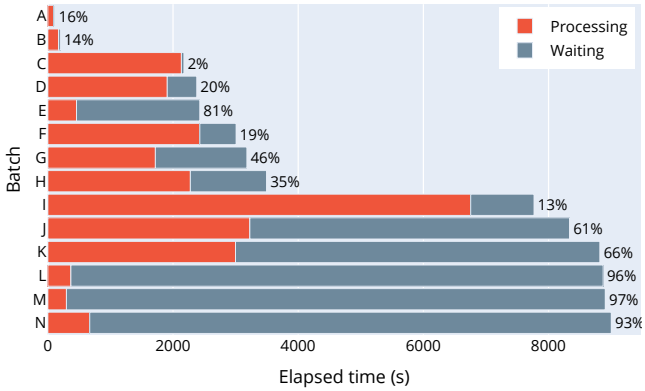
To alleviate this further, we explore how shortest-job-first (SJF) as an alternative scheduling strategy impacts pipeline performance in CB-IM-SJF based on the measurements collected with the CB-IM pipeline. Batch size is now used to determine the processing order of batches, and the resulting execution schedule, effectively a reordering of Figure 7b, is shown in Figure 7c.

Although the total makespan (at 7 465 s) is now longer than for CB-IM[10] and sustained pipeline rate $R_P$ is lower at 45.15 MB/s, the per-batch latency is now much more consistent and predictable, illustrated in Figure 9. By processing smaller batches early sooner, they suffer less queuing due to preceding large batches (the convoy effect), which can also be seen in Figure 8c. Larger batches, on the other hand, are subject to compulsory queuing behind smaller batches, but this does not penalise them as heavily relative to their processing time in comparison to small batches. In particular, no batch has to wait for the slowest (biggest) batch to be processed. This behaviour is visible as the large spread of $R_b$ for CB and CB-IM in Figure 9, compared to the more consistent values for CB-IM-SJF.

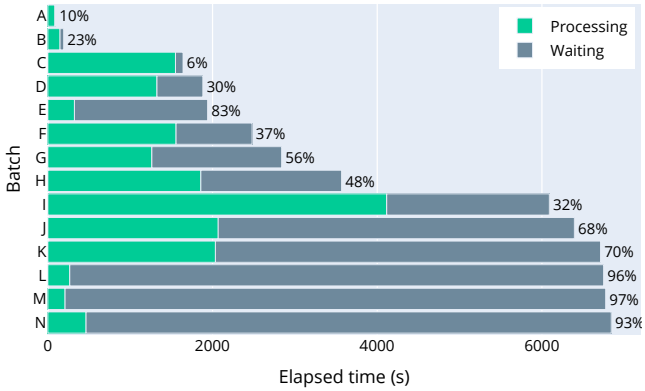
*Resource utilisation on the edge.* The study here is to quantify the motivating arguments and validate the design choices in FORTE, as well as to illustrate the effect of the possibilities and constraints when involving edge devices in the pipeline steps. To complete the evaluation of the explored design parameters, we visualise utilisation metrics for a selection of hardware resources at the edge system during each pipeline run in Figure 10.

In the baseline, the single transfer workload task on the edge system consistently utilises disk and network bandwidth to a high degree, while leaving CPU largely unused. Attempting to utilise the available CPU resources for processing tasks in the *Compression-based* pipeline design leads to higher CPU utilisation and better pipeline performance, but fully saturates the disk bandwidth due to the previously described access patterns (Figure 3a). The impact of this on pipeline performance becomes clear when the CB-IM pipeline reduces the amount of data I/O operations in favour of in-memory processing to perform identical processing work significantly faster (improving $R_P$ by 31 % over CB). CPU utilisation during this work is more consistent with CB-IM than in CB, while memory usage has increased minimally. The fluctuating behaviour seen in the network bandwidth utilisation is more frequently operating at a higher performance state with the CB-IM pipeline, as the (identical) preceding preparatory tasks now complete faster and supply ready-for-transfer data at a faster rate.

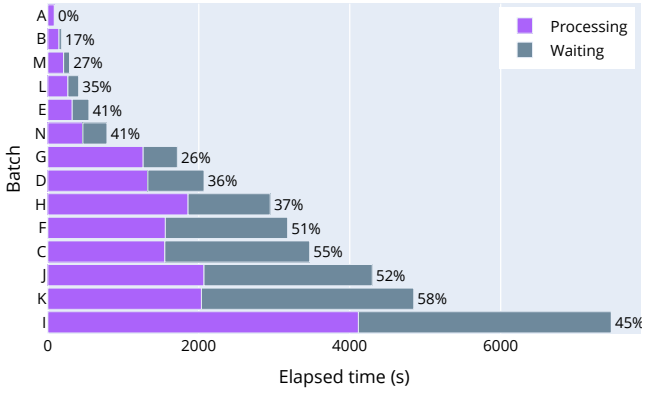
[10] In fact, the makespan is now maximally long for this set of tasks and pipeline configuration, assuming that tasks immediately begin execution once their dependencies are met and they are not blocked.




**(a) CB: Time spent executing tasks or queuing for each batch. Queuing times are collapsed into a single block for clarity.**




**(b) CB-IM: Time spent executing tasks or queuing for each batch. Even though the pipeline processes data much faster than the CB pipeline, the last 5 smaller batches are still queuing for significant portions of their total batch latency.**




**(c) CB-IM-SJF: Batches are now processed in order of increasing size, so smaller batches never wait for larger batches. Queuing is only experienced before the first task of the pipeline, and is overall less extreme.**

**Figure 8: Processing vs queuing times across designs.**

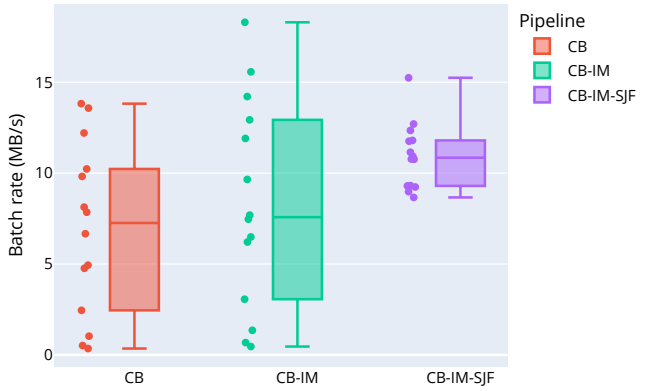



**Figure 9: Distribution of batch rate $R_b$ for CB, CB-IM, CB-IM-SJF pipelines.**

More details about the use-case, the pipeline implementation, and additional data, particularly regarding the service rate of each task in a pipeline, are contained in the master's thesis report [8].

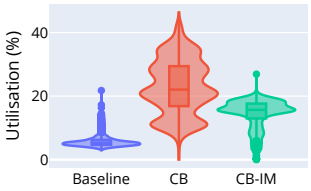
## 4.4 Discussion

The presented results for FORTE show clear improvements to pipeline latency and sustainable data rates, providing more timely delivery as well as increased pipeline data transfer capacity. By benchmarking compression strategies, a suitable balance of speed and effectiveness is found to address the network bandwidth bottleneck of a purely transfer-focused baseline. Further processing, such as integrity verification and data encryption, is also introduced, and, by exploiting available data independence between batches, can be pipelined to better hide the additional latency from these tasks. Additionally, modern x86 CPU hardware, as used in the evaluation here, includes instruction sets to accelerate workloads such as SHA256 computation or AES encryption, giving important efficiency improvements for a big data pipeline.

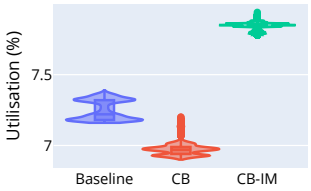
Depending on the requirements of the application, and the underlying scheduling framework used for the pipeline implementation, the trade-off of using FORTE in combination with SJF scheduling improves the average performance for batches compared to FIFO scheduling. SJF scheduling also allows the pipeline to exhibit more predictable delivery times relative to batch size, where smaller batches complete in less time than larger ones. Although this comes at the cost of somewhat higher $l_P$, i.e. time until the biggest batch of a burst is completed, the benefit for smaller batches is substantial; both in terms of absolute and relative batch latency while also being more consistent. Additionally, SJF scheduling introduces a risk of starving particularly large data batches from being transferred if there is insufficient time between data bursts containing large amounts of new smaller batches.

# 5 RELATED WORK

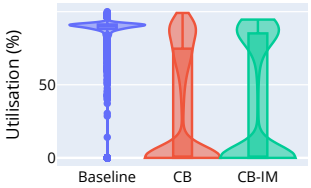
Efficient transfer of large data volumes is key for modern cyber-physical systems, and various techniques have been developed to enhance performance by reducing data volumes through compression [6, 20]. These techniques are sometimes discussed within the context of prototypes of general-purpose frameworks for managing,



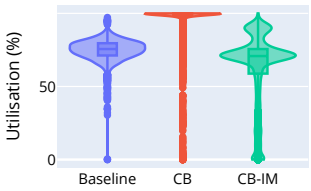

**(a) Distribution of CPU utilisation on the upload station with each pipeline.**




**(b) Distribution of RAM usage on the upload station with each pipeline.**




**(c) Distribution of network bandwidth utilisation on the upload station while transferring data using each pipeline. 100 % corresponds to the maximum achievable bandwidth on the route between the remote site and the DMZ (68 MB/s). The median for both CB and CB-IM is 0.01, but note the higher frequency of high utilisation by the CB-IM pipeline.**




**(d) Distribution of disk bus utilisation. Corresponds to the proportion of time that the disk is busy servicing requests. At 100 % utilisation, the disk is never idle and additional incoming requests beyond the maximum service rate of the disk will need to wait in queue.**

**Figure 10: Distribution of utilisation level for hardware resources on the upload station during transfer of the test data set for each pipeline. The box within each violin illustrates the span of Q1 and Q3 quantiles ($25^{th}$ and $75^{th}$ percentile) of the data; the line bisecting the box designates the median value.**

aggregating, and processing data in applications such as vehicular networks or the Internet of Things (IoT) [6, 9, 10]. In many of these applications, raw data or summarisation of preprocessed data is continuously transmitted in a streaming or micro-batch fashion, whereas bulk transfer of raw data is less common as it hinders the potential for real-time analysis.

Existing compression techniques can be categorised as either lossless or lossy. In our specific setup, lossy compression is not feasible, as the raw data cannot undergo approximations in relevant applications. However, it is worth noting related work that has employed Piecewise Linear Approximation (PLA), which approximates time series as sequences of segments, and achieves a significantly higher reduction in data volumes compared to lossless techniques such as ZIP, while still offering a guaranteed and configurable bounded error [6]. In order to apply PLA in fog/edge distributed systems, where data is processed in a streaming fashion, efficient online generation of the segments is crucial, while considering trade-offs between achieved compression, latency, and approximation error as studied by Duvignau et al. [4].

Other relevant research has focused on the communication aspects of data transfer pipelines in the context of 5G technology [15, 17], where opportunities for data-intensive applications have increased significantly. For example, the ERAIA framework [7] establishes a flexible and scalable basis for implementing data pipelines in the IoT domain. It is important to note that these aspects are complementary to those discussed in our paper: despite the improved latency and bandwidth provided by 5G, effective utilisation of computational power at the edge remains critical for reducing latency in Cloud-Edge pipelines [13]. One such case is discussed in connection with autonomous vehicles [5], where the timeliness of task execution is crucial for safe operation.

Efforts to optimise high-volume data transfer also highlight the challenge of dealing with the large parameter space and propose various techniques to address this issue. Yildirim et al. [18] identify pipelining, parallelism, and concurrency as important parameters for improving data transfer throughput by hiding latency. They develop models of the complete data transfer system, including data set characteristics, and optimisation algorithms to determine optimal parameter values. Similarly, Arslan et al. [3] study parameters related to I/O throughput and parallelism to improve transfer throughput, as well as propose adaptive tuning to adjust parameters in real time. Liu et al. [12] explore the possibility of more efficient integrity verification in high-volume transfers, achieved through parallelising and overlapping transfer and integrity verification to better hide the introduced latency. Furthermore, work on pipelines that involve processing interleaved with data transfers, is contributed through the AMESoS framework [16]; the framework facilitates pipelines to meet latency requirements, through resource provisioning based on predictive load-estimations and elastic scaling in the presence of overloads, facilitated by methodologies as in [19].

## 6 CONCLUSIONS

This industry experience article discusses challenges and possibilities in conjunction with automating data transfer pipelines and trade-offs that arise within the context of such pipelines. The leveraging of composable data handling tasks within data transfer pipelines alleviates bottlenecks in network bandwidth but also introduces trade-offs with respect to e.g., latency, throughput, and resource efficiency. In addition, such tasks can introduce in-node bottlenecks, especially in resource-constrained environments. Moreover, scheduling possibilities need to balance local, task/batch-latency, and throughput optimisation with end-to-end metrics for the whole pipeline. The work proposes the extensible FORTE framework and instantiates the study with a real-world demanding use case. The discussed trade-offs advocate for holistic, cross-layer approaches in the software stack, as proposed in FORTE, showing the interplay between latency and resource efficiency. For continued work in such contexts, instance-based scheduling and adaptation in orchestration in general, such as in [11, 14], can form the basis for more advanced orchestration methods toward balancing the aforementioned trade-offs and implications in energy efficiency. Further, by identifying data of higher importance already at the edge and source of the pipeline, transfers can be prioritised and

scheduled accordingly to yield lower latencies for such important data.

## ACKNOWLEDGMENTS

This work is supported by the Swedish Research Council (Vetenskapsrådet) project "EPITOME" 2021-05424, by the Marie Skłodowska-Curie Doctoral Network project RELAX-DN, funded by the European Union under Horizon Europe 2021-2027 Framework Programme Grant Agreement number 101072456 (www.relax-dn.eu/) and by Chalmers Un. AoA frameworks Energy and Production, proj. INDEED, and WP "Scalability, Big Data and AI", respectively. The implementation of this work was conducted at Volvo Group Trucks Technology as part of the master's thesis of the first author.

## REFERENCES

[1] 2022. Kubernetes - Production-Grade Container Orchestration.
[2] Apache Software Foundation. 2021. Apache Airflow.
[3] Engin Arslan, Bahadir A. Pehlivan, and Tevfik Kosar. 2018. Big Data Transfer Optimization through Adaptive Parameter Tuning. *J. Parallel and Distrib. Comput.* 120 (Oct. 2018), 89–100. https://doi.org/10.1016/j.jpdc.2018.05.003
[4] Romaric Duvignau, Vincenzo Gulisano, Marina Papatriantafilou, and Vladimir Savic. 2019. Streaming Piecewise Linear Approximation for Efficient Data Management in Edge Computing. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. Association for Computing Machinery, New York, NY, USA, 593–596.
[5] Ionel Gog, Sukrit Kalra, Peter Schafhalter, Joseph E. Gonzalez, and Ion Stoica. 2022. D3: A Dynamic Deadline-Driven Approach for Building Autonomous Vehicles. In *Proceedings of the Seventeenth European Conference on Computer Systems*. ACM, Rennes France, 453–471. https://doi.org/10.1145/3492321.3519576
[6] Bastian Havers, Romaric Duvignau, Hannaneh Najdataei, Vincenzo Gulisano, Marina Papatriantafilou, and Ashok Chaitanya Koppisetty. 2020. DRIVEN: A Framework for Efficient Data Retrieval and Clustering in Vehicular Networks. *Future Generation Computer Systems* 107 (June 2020), 1–17. https://doi.org/10.1016/j.future.2020.01.050
[7] Aitor Hernandez, Bin Xiao, and Valentin Tudor. 2020. ERAIA - Enabling Intelligence Data Pipelines for IoT-based Application Systems. In *2020 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, Austin, TX, USA, 1–9. https://doi.org/10.1109/PerCom45495.2020.9127385
[8] Martin Hilgendorf. 2022. *Efficient Industrial Big Data Pipeline for Lossless Transfer of Vehicular Data.* Master's thesis. Chalmers University of Technology, Gothenburg, Sweden.
[9] Amir Keramatian, Vincenzo Gulisano, Marina Papatriantafilou, and Philippas Tsigas. 2021. Mad-c: Multi-stage Approximate Distributed Cluster-Combining for Obstacle Detection and Localization. *J. Parallel and Distrib. Comput.* 147 (2021), 248–267.
[10] Amir Keramatian, Vincenzo Gulisano, Marina Papatriantafilou, and Philippas Tsigas. 2023. PARMA-CC: A Family of Parallel Multiphase Approximate Cluster Combining Algorithms. *J. Parallel and Distrib. Comput.* 177 (2023), 68–88.
[11] Tim Kraska. 2021. Towards Instance-Optimized Data Systems. *Proceedings of the VLDB Endowment* 14, 12 (July 2021), 3222–3232. https://doi.org/10.14778/3476311.3476392
[12] Si Liu, Eun-Sung Jung, Rajkumar Kettimuthu, Xian-He Sun, and Michael Papka. 2016. Towards Optimizing Large-Scale Data Transfers with End-to-End Integrity Verification. In *2016 IEEE International Conference on Big Data (Big Data)*. 3002–3007. https://doi.org/10.1109/BigData.2016.7840953
[13] Shaoshan Liu, Liangkai Liu, Jie Tang, Bo Yu, Yifan Wang, and Weisong Shi. 2019. Edge Computing for Autonomous Driving: Opportunities and Challenges. *Proc. IEEE* 107, 8 (2019), 1697–1716.
[14] Dimitris Palyvos-Giannas, Vincenzo Gulisano, and Marina Papatriantafilou. 2019. Haren: A Framework for Ad-Hoc Thread Scheduling Policies for Data Streaming Applications. In *Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems (DEBS '19)*. Association for Computing Machinery, New York, NY, USA, 19–30. https://doi.org/10.1145/3328905.3329505
[15] Murtaza Ahmed Siddiqi, Heejung Yu, and Jingon Joung. 2019. 5G Ultra-Reliable Low-Latency Communication Implementation Challenges and Operational Issues with IoT Devices. *Electronics* 8, 9 (Sept. 2019), 981. https://doi.org/10.3390/electronics8090981
[16] Michail Tsenos, Aristotelis Peri, and Vana Kalogeraki. 2022. AMESoS: A Scalable and Elastic Framework for Latency Sensitive Streaming Pipelines. In *Proceedings of the 16th ACM International Conference on Distributed and Event-Based Systems (DEBS '22)*. Association for Computing Machinery, New York, NY, USA, 103–114. https://doi.org/10.1145/3524860.3539642
[17] Gorka Velez, Edoardo Bonetto, Daniele Brevi, Angel Martin, Gianluca Rizzi, Oscar Castañeda, Arslane Hamza Cherif, Marcos Nieto, and Oihana Otaegui. 2022. 5G Features and Standards for Vehicle Data Exploitation. https://doi.org/10.48550/arXiv.2204.06211 arXiv:2204.06211 [cs]
[18] Esma Yildirim, Engin Arslan, Jangyoung Kim, and Tevfik Kosar. 2016. Application-Level Optimization of Big Data Transfers through Pipelining, Parallelism and Concurrency. *IEEE Transactions on Cloud Computing* 4, 1 (Jan. 2016), 63–75. https://doi.org/10.1109/TCC.2015.2415804
[19] Nikos Zacheilas, Vana Kalogeraki, Yiannis Nikolakopoulos, Vincenzo Gulisano, Marina Papatriantafilou, and Philippas Tsigas. 2017. Maximizing Determinism in Stream Processing under Latency Constraints. In *Proceedings of the 11th ACM International Conference on Distributed and Event-Based Systems (DEBS '17)*. Association for Computing Machinery, New York, NY, USA, 112–123. https://doi.org/10.1145/3093742.3093921
[20] Hongbo Zou, Yongen Yu, Wei Tang, and Hsuanwei Michelle Chen. 2014. Improving I/O Performance with Adaptive Data Compression for Big Data Applications. In *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*. 1228–1237. https://doi.org/10.1109/IPDPSW.2014.138