Clayton School of Information Technology
Monash University

Honours Thesis — Semester 1, 2015

A study of data stream processing systems for use with railway

Jonathan Poltak Samosir

[2271 3603]

Supervisors: Dr Maria Indrawan-Santiago
Dr Pari Delir Haghighi

Contents

1 Introduction 1

1.1 Research Context . . . 1

1.1.1 Big Data . . . 1

1.1.2 Batch data processing . . . 2

1.1.3 Realtime data processing . . . 2

1.1.4 Monash IRT Railway Project . . . 3

1.2 Research Objectives . . . 3

1.3 Research Questions . . . 4

1.4 Thesis Structure . . . 4

1.5 Summary . . . 4

2 Literature Review 6

2.1 Big data processing background . . . 6

2.1.1 Batch data processing . . . 6

2.1.2 Realtime data processing . . . 8

2.2 Discussion and analysis . . . 12

2.3 Conclusion . . . 15

3 DSPS Technology Overview 16

3.1 DSPS Technology Choices . . . 16

3.2 Testing Parameters . . . 16

3.3 Evaluation Method & Approach . . . 16

4 Implementation 17

4.1 Testing Environment Details . . . 17

4.2 Setting Up of DSPS Technologies . . . 17

4.3 Implementation of Pipelines in DSPS Technologies . . . 17

5 Discussion and Evaluation 18

5.1 Quantitative Evaluations . . . 18

5.1.1 Outcomes of Tests Performed . . . 18

5.2 Qualitative Evaluations . . . 18

5.3 DSPS Technology Recommendations . . . 18

6 Conclusions 19

6.1 Research Contribution . . . 19

6.2 Future Work . . . 19

Abstract

1 Introduction

Currently, as a society, we are generating very large amounts of data from a large range of different sources. These sources include scientific experiments, such as the Australian Synchrotron [5] and The Large Hadron Collider [8], companies, such as Amazon [2], and also data generated by end users of products, such as social networks. The rate of data that is being generated is constantly increasing, presenting major challenges when it comes to the storage and processing of that data [14]. This is what is often referred to now as "Big Data". A further big data project, that will be looked into as the basis of the project outlined in this thesis, is the automated monitoring of railway tracks and cars by the Institute of Railway Technology at Monash University (IRT) [7].

Out of all of these data that are faced in such projects, often only specific parts of the data are of particular use for given purposes. Hence, rather than attempting to store all the new data that is being generated, an increasingly popular method of dealing with such data, in both academia and industry associated with big data, is the processing and analysis of data in realtime as it is received.

There are currently numerous realtime data processing frameworks that are in development and in production use, both in industry and academia. Examples of these realtime data processing frameworks include the widely used Storm project [1], developed at BackType and Twitter, Inc., and also the up-and-coming Spark Streaming project [10], developed at UC Berkeley's AMPLab [3], both of which are open-source projects. While there are a growing number of these projects being developed, often these projects are designed with a particular type of data in mind, or to facilitate a particular type of data processing. For example, the before mentioned Spark Streaming project, along with its mother project, Spark [4], was originally designed for highly parallelisable data with the use-case in mind of processing data in-memory using highly iterative machine learning algorithms related to data analytics [40].

What is proposed in this chapter is a realtime big data processing system for the aforementioned railway monitoring system by the IRT at Monash University given the possibility that data is able to be streamed in realtime from the railway in realtime.

This chapter will be structured as follows:
A brief outline of the domain on big data processing, both batch and realtime processing, will be given in §1.1. Furthermore, an overview of the Monash University Institute of Railway Technology's project, upon which this project is based on, will be covered in the aforementioned chapter. In §1.2, the goals and aims of the project will be given, as well as more specific goals and aims relating to the subproject covered in this thesis. Research questions for this subproject will be given in §1.3. Finally, the structure of this overall thesis will be given in §1.4, before concluding this chapter in §1.5.

1.1 Research Context

1.1.1 Big Data

Big data, as explained previously, is becoming commonplace in both industry and academia. Everyday companies are finding that they are generating too much data and that their traditional relational database management system (RDMBS) solutions cannot scale to the epic proportions needed to handle this data in an efficient and robust manner [41]. Hence, companies and academics alike have started looking at alternative solutions designed with the goal of handling these massive datasets.

The most popular solution for this problem, up until recently, has been the MapReduce model of programming along with some type of scalable distributed storage system [12]. The MapReduce model was started at Google, Inc. with their own proprietary implementation along with their proprietary distributed file system, known as the Google File System (GFS) [25]. Without going into the low-level details of MapReduce and GFS, the use of this solution at Google allowed the company to easily handle all the data that was coming into their servers, including that related to Google Search, and perform the necessary processing operations that was needed at the time [25] [20].

1.1.2 Batch data processing

From the success of MapReduce usage combined with GFS at Google, the open-source community responded swiftly with the development of the Apache Hadoop framework. Hadoop originally offered an open-source implementation of MapReduce and their own open-source distributed file system known as the Hadoop Distributed File System (HDFS) [50].

Hadoop soon became the subject of mass-adoption in both industry and academia, being deployed at a fast rate. Development of the Hadoop framework also grew at a fast rate, with new applications related to HDFS and MapReduce being built on top of Hadoop, greatly benefiting the ecosystem as a whole. Some of these applications grew into widely adopted systems in their own right. For example, Hadoop applications such as Apache Pig [23] and Hive [59] allow for easy querying and manipulation of data stored on HDFS, both coming with the addition of their own query languages [46].

Additionally, as further non-MapReduce model applications became of interest to the Hadoop community, Hadoop soon developed a further abstraction on top of the underlying resources (in most cases, HDFS). The goal of this was to facilitate the development and deployment of many different applications, varying in use-case, which could be run on the Hadoop ecosystem, without forcing developers to fit their application into the MapReduce model. This development was known as Apache Hadoop YARN: Yet Another Resource Negotiator, which can be thought of as an operating system-like abstraction sitting atop of the available Hadoop resources [66]. The abstraction YARN provides facilitated the development of much more advanced, and non-MapReduce technologies which have since become widely used parts of the Hadoop ecosystem [31].

1.1.3 Realtime data processing

One of the major limitations of Hadoop, and the MapReduce model in general, soon became obvious: MapReduce was designed with the goal of being able to process batches of data, hence, given Hadoop's dominance, batched data processing was the focal point of the entire distributed data processing domain [36]. Essentially, batched data processing is where data gets collected first into large enough batches before being processed all-at-once. The point of processing in such a way is so there would be less overheads than attempting to process each individual datum as it arrives. For a lot of use-cases this was, and still is, fine as there were no other drawbacks apart from a high level of latency between the stages of when the data arrives and when it gets processed. However, for other applications, such as stock trading, sensor monitoring, and web traffic processing, a more low-latency, realtime solution was needed [36].

Soon, many solutions, with different use-cases and design goals, were developed in the area of distributed stream processing systems (DSPS). Given the Hadoop ecosystem that was already widely adopted, most of these DSPSs were built upon the still new YARN layer, ensuring overall compatibility with the Hadoop ecosystem, and the underlying HDFS. Some examples of such projects include the beforementioned Apache Storm, currently being used at Twitter, Inc. [62], among many other companies. Also up-and-coming projects, such as Apache Samza which is a recently open-sourced project, currently being used in production at LinkedIn Corporation [9].

1.1.4 Monash IRT Railway Project

The railway project that has been developed at Monash University's Institute of Railway Technology uses numerous sensor technologies on certain train cars, such as the Track Geometry Recording Car (TGRC) and the Instrumented Ore Car (IOC), to monitor railway track conditions and detect track abnormalities [17] [18]. These train cars operate in the Pilbara region of Western Australia, continously performing round trips from a port to a given loading point, where they are loaded with recently mined minerals and ores.

As is currently the case, data is received and processed using batch data processing technologies. Due to limited coverage of cellular networks in the Pilbara region of Western Australia, in which the trains currently operate, sensor data from a given trip is automatically transmitted in large batches to be received by remote servers once a train has concluded a round trip and arrives back in port from a loading point [58]. Given the current cellular network infrastructure in the region, this is the only feasible option for transmission of data, however Monash IRT have indicated that given future improvements in cellular network infrastructure, streaming the data back to remote servers in realtime is a likely possibility. This leads to the potential possibility of this research project's outcomes outlined in this thesis.

The form of batch handling and processing performed on the railway data currently leads to a number of limitations and problems. The data is currently stored in a relational database management system (RDBMS), and with the current implementation of the sensors, the data received is not consistently structured. In fact, the only sensor data guaranteed to be received in each batch is geographic location and time. Due to the highly structured nature of RDBMS technology, certain work-arounds need to be performed on the data so that it is compliant to RDBMS schemas, such as the insertions of default values in in the case of missing attributes. Furthermore, low query performance has been noted as a problem plaguing the IRT team working with the railway data. They wish to resolve these problems by looking into non-relational models for their data storage and processing systems.

1.2 Research Objectives

The main aim or goal of this research project is to develop a fully automated, non-relational big data system to manage the data received from railway car sensors as a part of Monash University's Institute of Railway Technology project. The intention is to replace their current relational solution, offering at least the same capabilities at a larger scale, and with higher performance. The scope of this project is relatively large, and the project, or subproject, outlined in this thesis only covers a portion of the greater aforementioned project.

The main aim of this subproject is to look at the hypothetical possibility of dealing with the railway sensor data being streamed in realtime. This will also involve non-relational big data systems, however the details of these such systems in the scope of the greater project will be left up to other individuals working on those subprojects.

To allow the possibility of realtime data streaming from the railway sensors, appropriate data stream processing system (DSPS) technology needs to be looked at, tested, and evaluated. This makes up the core part of this subproject's work. These DSPS technologies need to appropriately take, or accept, the data from some specified source, *e.g.* the sensors, apply any realtime processing logic that is required, *e.g.* pre-processing, then forward the data on for handling at another source, *e.g.* long term storage, such as HDFS.

This DSPS system will act as a sort of processing "pipeline" through which the sensor data flows. By the end of this subproject, we aim to have proof-of-concept implementations of the pipeline working on the National eResearch Collaboration Tools and Resources (NeCTAR) cloud services [6], making use of certain DSPS technologies.

1.3 Research Questions

The following research questions are the main focus points of this subproject:

1. How can existing DSPS technologies be used to fill this realtime processing gap in the Monash University's IRT project?
2. What qualitative and quantitative tests can be performed to recommend a particular DSPS technology for building the pipeline?
3. How can we design the DSPS processing pipeline to be extensible, allowing for the addition of future realtime processing requirements?

Additionally, after answering each of these preliminary research questions, we will want to properly implement the theoretical discoveries from each stage. We do this with the goal of achieving some deployable pipeline that can be then be used in the testing and overall evaluation stages.

1.4 Thesis Structure

The proposed structure of the remainder of the thesis is as follows. §2 looks at prior research and work into the area of big data stream processing, performed both in industry and academia. A clear overview of the DSPS technologies chosen for this project will be given in §3, along with detailing the experiments that will be performed and evaluated. §4 will detail how the experimental systems built for this project were implemented using the DSPS technologies highlighted in §3, with an evaluation of the experiments being detailed in §5. Finally, §6 will conclude the thesis, along with highlighting any further research gaps that have been made apparent as the result of this project.

1.5 Summary

This chapter has introduced the overall project, and in-turn the subproject upon which this thesis will be based. It has described the current state of the Monash University Institute of Railway Technology's project and the current problems that they are faced with, such as the need to deal with non-consistently structured data and low query performance.

These problems are exacerbated given the non-realtime nature of the current data, where issues with sensors are only discovered much later after-the-fact. To overcome these issues, and look forward into the hypothetical, but likely, possibility of having the technological infrastructure to support realtime data processing, this research aims to develop a realtime processing pipeline, taking the data straight from the railway sensors and perform any needed realtime processing.

This chapter also outlined a brief context of the research project and big data as a whole, the scope of this subproject, this subproject's research questions. Finally it was concluded with an overview of this thesis' entire structure. The following chapter will look into previous research that has been done on realtime big data processing and handling, along with going into more detail of the railway project's needs.

2 Literature Review

The realtime processing of big data is of great importance to both academia and industry. Advancements and progress in modern society can be directly attributed back to data. The value of data has become more apparent, and data has become a sort of currency for the information economy [52]. Hence, those in society who realised the value of data early hold immense power over the entire economy, and in turn society, overall [39]. From seemingly inconsequential gains at the macro level, such as the ability to more accurately predict the rise and fall of airline tickets [19], to those of utmost importance for society as a whole, such as predicting and tracking the spread of the Swine Flu Pandemic in 2009 more accurately than the United States Centers for Disease Control and Prevention could [48, 42]. It is applications of big data processing like these that have been recognised by academics and organisations in industry alike, with the last decade seeing a major shift in research and development into new methods for the handling and processing of big data.

This review will give a background on the types and classes of big data, as well as the various methods employed to process those given classes of data. We will more specifically be focusing on the methods that are involved with the analysis and processing of realtime data streams, as opposed to the batch processing of big data. This review will look into detail at previous work that has been done in the field of big data, specifically those works that have had a greater influence on the field as a whole. This includes both works looking specifically at the processing of streaming data, and works involving processed big data in batch mode, given that batch mode processing arguably led onto the current hot-topic of realtime stream processing.

This review will be structured in two main sections. In §2.1, an overview will be given of the major open-source big data processing systems in the scope of the Hadoop ecosystem. A special emphasis will be given to realtime data processing systems, otherwise known as data stream processing systems (DSPSs), given that the main area of this research is focusing on realtime data processing. This section will also be concluded with a brief comparison of the presented systems. In §6 an analysis and discussion will be given on the content covered from the relevant literature, as well as identifying how the research contributions of this project will address the gaps found in the relevant literature.

2.1 Big data processing background

Much more work has been done in the area of data processing than the area related to classification of data; both in the areas of big data and traditional data processing. Unlike data classification, which was more aimed at the classifying of data in general, when looking at data processing, we are more interested in the relatively newer technologies which enable the processing of big data, both in batch mode and realtime. Note that in this review, we will refer to the processing of data in realtime as simply "realtime data processing". This term should be assumed to encompass the meaning that is also often represented as "data stream processing", "realtime stream processing", and "stream processing".

2.1.1 Batch data processing

Over the last decade, the main "go-to" solution for any sort of processing needed on datasets falling under the umbrella of big data has been the MapReduce programming model on top of some sort of scalable distributed storage system [12]. From a very simplified functionality standpoint, the MapReduce programming model essentially combines the common **Map** and **Reduce** functions (among others), found in the standard libraries of many functional programming languages, such as Haskell [38] or even Java 8 [56], to apply a specified type of processing in a highly parallelised and distributed fashion [69].

The MapReduce data processing model specialises in batch mode processing. Batch data processing can be thought of where data needed to be processed is first queued up in batches before processing begins. Once ready, those batches get fed into the processing system and handled accordingly.

2.1.1.1 MapReduce and GFS

Dean and Ghemawat, in [20], originally presented MapReduce as a technology that had been developed internally at Google, Inc. to be an abstraction to simplify the various computations that engineers were trying to perform on their large datasets. The implementations of these computations, while not complicated functions themselves, were obscured by the fact of having to manually parallelise the computations, distribute the data, and handle faults all in an effective manner. The MapReduce model then enabled these computations to be expressed in a simple, high-level manner without the programmer needing to worry about optimising for available resources. Furthermore, the MapReduce abstraction provided high scalability to differently sized clusters.

As previously stated, the MapReduce programming model is generally used on top of some sort of distributed storage system. In the previous case at Google, Inc., in the original MapReduce implementation, it was implemented on top of their own proprietary distributed file system, known as Google File System (GFS). Ghemawat et al., in [25], define GFS to be a "scalable distributed file system for large distributed data-intensive applications", noting that can be run on "inexpensive commodity hardware". Note that GFS was designed and in-use at Google, Inc. years before they managed to develop their MapReduce abstraction, and the original paper on MapReduce from Dean and Ghemawat state that GFS was used to manage data and store data from MapReduce [20]. Furthermore, McKusick and Quinlan, in [43], state that, as of 2009, the majority of Google's data relating to their many web-oriented applications rely on GFS.

2.1.1.2 Hadoop MapReduce and HDFS

While MapReduce paired with GFS proved to be very successful solution for big data processing at Google, Inc., and there was notable research published on the technology, it was proprietary in-house software unique to Google, and availability elsewhere was often not an option [28]. Hence, the open-source software community responded in turn with their own implementation of MapReduce and a distributed file system analogous to GFS, known as the Hadoop Distributed File System (HDFS). Both of these projects, along with others to date, make up the Apache Hadoop big data ecosystem [1]. The Apache Hadoop ecosystem, being a top level Apache Software Foundation open source project, has been developed by a number of joint contributors from organisations and institutions such as Yahoo!, Inc., Intel, IBM, UC Berkeley, among others [30].

While Hadoop's MapReduce implementation very much was designed to be a functional replacements for Google's MapReduce, HDFS is an entirely separate project in its own

[1] https://hadoop.apache.org

right. In the original paper from Yahoo! [51], Inc., Shvachko et al. present HDFS as "the file system component of Hadoop" with the intention of being similar to the UNIX file system, however they also state that "faithfulness to standards was sacrificed in favour of improved performance".

While HDFS was designed with replicating GFS' functionality in mind, several low-level architectural and design decisions were made that substantially differ to those documented in GFS. For example, in [15], Borthakur documents the method HDFS uses when it comes to file deletion. Borthakur talks about how when a file is deleted in HDFS, it essentially gets moved to a `/trash` directory, much like what happens in a lot of modern operating systems. This `/trash` directory is then purged after a configurable amount of time, the default of which being six hours. To contrast with this, GFS is documented to have more primitive way of managing deleted files. Ghemawat, et al., in [25], document GFS' garbage collection implementation. Instead of having a centralised `/trash` storage, deleted files get renamed to a hidden name. The GFS master then, during a regularly scheduled scan, will delete any of these hidden files that have remained deleted for a configurable amount of time, the default being three days. This is by far not the only difference between the two file systems, this is simply an example of a less low-level technical difference.

2.1.1.3 Pig and Hive

Given the popularity of Hadoop, there were several early attempts at building further abstractions on top of the MapReduce model, which were met with a high level of success. As highlighted earlier, MapReduce was originally designed to be a nice abstraction on top of the underlying hardware, however according to Thusoo et al., in [60], MapReduce was still too low level resulting in programmers writing programs that are "are hard to maintain and reuse". Thus, Thusoo et al. built the Hive abstraction on top of MapReduce. Hive allows programmers to write queries in a similarly declarative language to SQL — known affectionately as *HiveQL* — which then get compiled down into MapReduce jobs to run on Hadoop [61].

Another common abstraction that was developed prior to Hive was what is known simply as Pig. Like Hive, Pig attempts to be a further higher level abstraction on top of MapReduce, which ultimately compiles down into MapReduce jobs, although what differentiates it from Hive is that instead of being a solely declarative SQL-like language, it is more of a mix of procedural programming languages while allowing for SQL-like constraints to be specified on the data set to define the result [47]. Olston et al. describe Pig's language — known as *Pig Latin* — to be what they define as a "dataflow language", rather than a strictly procedural or declarative language.

Furthermore, note that Pig and Hive, being high level abstractions on top of MapReduce, also enable many of their own optimisations to be applied to the underlying MapReduce jobs during the compilation stage [24, 61] as well as having the benefit of being susceptible to manual query optimisations, familiar to programmers familiar with query optimisations from SQL [29].

2.1.2 Realtime data processing

With HDFS being an open source project with a large range of users [68] and code contributors [30], it has grown as a project in the last few years for uses beyond what it was originally intended for; a backend storage system for Hadoop MapReduce. HDFS is now not only used with Hadoop's MapReduce but also with a variety of other technologies, a lot of which run as a part of the Hadoop ecosystem. Big data processing has moved on from the more "traditional" method of processing, involving MapReduce jobs, which were most suitable for batch processing of data, to those methods which specialise in the realtime processing of data. The main difference of which is that rather than waiting for all the data before processing can be started, in realtime data processing the data can be streamed into the processing system in realtime at any time in the whole process.

Comparing batched data processing to realtime data processing, it is useful to relate back to the four V's identified in §**??**. Velocity of data is often inconsistent with realtime processing, while in batch mode processing, where you are processing the data that has already arrived and is waiting in batches to be processed, the velocity can be considered consistent. Veracity of data is often not expected to be as consistent in realtime, as sometimes there might be times where data does not arrive or only certain parts of the data arrive at certain times. A realtime processing system, often called a data stream processing system (DSPS) in other literature, needs to be able to deal with these timeliness issues, while a batch data processing system may expect everything that needs to be there to be available.

2.1.2.1 Hadoop YARN

As previously looked at, the focus of the MapReduce model was performing distributed and highly parallel computations on distributed batches of data. This suited a lot of the big data audience, and hence Hadoop became the dominant method of big data processing [40]. However for some more specialised applications, such as the realtime monitoring of sensors, stock trading, and realtime web traffic analytics, the high latency between the data arriving and actual results being generated from the computations was not satisfactory [36].

A recent (2013) industry survey on European company use of big data technology by Bange, Grosser, and Janoschek, noted in [11], shows that over 70% of responders show a need for realtime processing. In that time, there has certainly been a response from the open-source software community, responding with extensions to more traditional batch systems, such as Hadoop, along with complete standalone DSPS solutions.

On the Hadoop front, the limitations of the MapReduce model were recognised, and a large effort was made in developing the "next generation" of Hadoop so that it could be extensible and used with other programming models, not locked into the rigidity of MapReduce. This became known officially known as YARN (Yet Another Resource Negotiator). According to the original developers of YARN, Vavilapalli et al. state that YARN enables Hadoop to become more modular, decoupling the resource management functionality of Hadoop from the programming model (traditionally, MapReduce) [65]. This decoupling essentially allowed for non-MapReduce technologies to be built on top of Hadoop, still interacting with the overall ecosystem, allowing for much more flexible applications of big data processing on top of the existing robust framework Hadoop provides.

Examples of such systems now built, or in some cases ported, to run on top of Hadoop, providing alternative processing applications and use cases include:

- Dryad, a general-purpose distributed execution system from Microsoft Research [34]. Dryad is aimed at being high level enough to make it "easy" for developers to write highly distributed and parallel applications.

- Spark, a data processing system, from researchers at UC Berkeley, that focuses on computations that reuse the same working data set over multiple parallel operations [71]. Spark, and in particular Spark Streaming, will be looked at further in §2.1.2.3.

- Storm, a realtime stream processing system [44, p. 244]. Performs specified processing on an incoming stream of data indefinitely, until stopped. Storm will be looked at further in §2.1.2.2.
- Tez, an extensible framework which allows for the building of batch and interactive Hadoop applications [57].
- REEF, a YARN-based runtime environment framework [16]. REEF is essentially a further abstraction on top of YARN, with the intention of making a unified big data application server.
- Samza, a relatively new realtime data processing framework from LinkedIn. Discussed further in §2.1.2.4.

These are just some of the more popular examples of applications built to interact with the Hadoop ecosystem via YARN.

2.1.2.2 Storm

One very notable DSPS technology developed independently of Hadoop, and that is gaining immense popularity and growth in its user base, is the Storm project. Storm was originally developed by a team of engineers lead by Nathan Marz at BackType [54]. BackType has since been acquired by Twitter, Inc. where development has continued. Toshniwal et al. [63] describe Storm, in the context of its use at Twitter, as "a realtime distributed stream data processing engine" that "powers the real-time stream data management tasks that are crucial to provide Twitter services" [63, p. 147]. Since the project's inception, Storm has seen mass adoption in industry, including among some of the biggest names, such as Twitter, Yahoo!, Alibaba, and Baidu [55].

While Storm does not run on top of YARN, there is currently a large effort from engineers at Yahoo!, Inc. being put into a YARN port for Storm, named "storm-yarn" [26]. This YARN port will allow applications written for Storm to take advantage of the resources managed in a Hadoop cluster by YARN. While still in early stages of development, "storm-yarn" has begun to gain attention in the developer community, through focus from channels such as the Yahoo Developer Network [67] and Hortonworks [21].

2.1.2.3 Spark Streaming

Spark is another popular big data distributed processing framework, offering of both realtime data processing and more traditional batch mode processing, running on top of YARN [71]. Spark was developed at UC Berkeley, and is notable for its novel approach to in-memory computation, through Spark's main data abstraction which is termed a *resilient distributed dataset* (RDD). An RDD is a set of data on which computations will be performed, which can be specified to be cached in the memory across multiple machines. What this then allows is multiple distributed operations being performed on this same dataset in parallel. A further benefit from the design of Spark is the reduce of overhead from IO operations. Spark is designed with highly iterative computations in-mind, where the intermediate data at each iteration stays in memory without being written and read to the underlying storage system (*e.g.* HDFS).

As stated earlier, Spark allows the processing of data in realtime and batch mode. Originally, Spark was released as a project that simply focused on batch processing, however after the need for realtime processing became apparent, an extension project, Spark Streaming, was initiated. Spark Streaming uses a different programming model that involves what is labelled as "D-Streams" (discretised streams), which essentially lets a series of deterministic batch computations be treated as a realtime data stream [72]. The D-Stream model is specific to the Spark Streaming system — the original batch mode Spark system continues to use the previously mentioned RDD abstraction — and the creators claim performance improvements of being $2-5\times$ faster than other realtime data processing systems, such as S4 and Storm [73]. However, this has since been disputed [27].

Both Spark and Spark Streaming have started to gain notable usage in both industry and research projects in academia in the last few years. Online video distribution company, Conviva Inc., report to be using Spark for the processing of analytics reports, such as viewer geographical distribution reports [35, 70]. The Mobile Millennium project at UC Berkeley [64], a traffic monitoring system that uses GPS through users' cellular phones for traffic monitoring in the San Francisco Bay Area, has been using Spark for scaling the main algorithm in use for the project: an expectation maximisation (EM) algorithm that has been parallelised by being run on Spark [33].

2.1.2.4 Samza

Samza is a relatively new realtime big data processing framework originally developed at LinkedIn, which has since been open-sourced at the Apache Software Foundation [49]. Samza offers much similar functionality to that of Storm, however instead the running of Samza is highly coupled with the Kafka message broker, which handles the input and output of data streams. Essentially, Kafka is a highly distributed messaging system that focusses on the handling of log data [37], integrating with the Hadoop ecosystem.

While Samza is lacking in maturity and adoption rates, as compared to projects such as Storm, it is built on mature components, such as YARN and Kafka, and thus a lot of crucial features are offloaded onto these platforms. For example, the archiving of data, stream persistence, and imperfection handling is offloaded to Kafka [13]. Likewise, YARN is used for ensuring fault tolerance through the handling of restarting machines that have failed in a cluster [13].

2.1.2.5 S4

S4 (Simple Scalable Streaming System) is another realtime big data processing framework that originated at Yahoo!, Inc. that has since been open-sourced [45]. It is a relatively old project compared to the before-mentioned projects, with development becoming less of a priority in the last few years. S4 was highly influenced by the MapReduce programming model that was discussed in §2.1.1.1.

Much like what was said about Samza in §2.1.2.4, S4 attempts offload several lower level tasks to more mature and established systems specialising in those areas. The logical architecture of S4 lays out its jobs in a network of processing elements (PEs) which are arranged as a directed acyclic graph. Each of these PEs entail the type of processing to be done on the data at that point in the network. Each of the PEs are assigned to a processing node, a logical host in the cluster. The management and coordination of these processing nodes is offloaded by S4 to ZooKeeper [36]. Much like the before-mentioned Kafka, ZooKeeper in itself is its own complex service used as a part of many different big data infrastructures, including Samza. ZooKeeper specialises in the high-performance coordination of distributed processes inside distributed applications [32].

2.2 Discussion and analysis

From the previously covered literature, it is rather difficult to provide a reasonable comparison for all the different realtime data processing projects. A lot of the claims made in original literature relating to the projects cannot be quantified fairly, as comparisons or tests have not been carried out relating to other projects. Instead, Stonebraker, Çẽntintemel, and Zdonik proposed what they claim to be the 8 requirements for realtime data processing systems [53], which can be used to given an impartial comparison of the previously covered projects. The requirements were defined a number of years prior to the creation of the four main realtime data processing systems that were covered (2005), however are highly cited as being the defining features that the current generation of realtime data processing systems have strived to meet. The requirements put forward by Stonebraker et al. are summarised as follows:

1: Keep the data moving - This requirement relates to the high mobility of data and importance of low latency in the overall processing. Hence, processing should happen as data moves, rather than storing it first, then processing what is stored.

2: SQL on streams - This requirement states that a high-level SQL-like query language should be available for performing on-the-fly queries on data streams. SQL is given as an example, however it is noted the language's operators should be more oriented to data streams.

3: Handle stream imperfections - Given the high degree of imperfections in data streams, including factors such as missing and out-of-order data, this requirement states that processing systems need to be able to handle these issues. Simply waiting for all data to arrive if some is missing is not acceptable.

4: Generate predictable outcomes - This requirement relates to the determinism associated with the outcomes of specified processes to be applied to data. A realtime processing system should have predictable and repeatable outcomes. Note that this requirement is rather hard to satisfy as in practice data streams are, by character, rather unpredictable. However, the operations performed on given data are required to be predictable.

5: Integrate stored and streamed data - This requirement states that a realtime processing system should provide the capabilities to be able to process both data that is already stored and data that is being delivered in realtime. This should happen seamlessly and provide the same programming interface for either source of data.

6: Guarantee data safety and availability - This requirement states that realtime processing systems should ensure that they have a high level of availability for processing data, and in any cases of failures, the integrity of data should remain consistent.

7: Partition and scale applications automatically - This requirement states that the partitioning of and processing of data should be performed transparently and automatically over the hardware on which it is running on. It should also scale to more different levels of hardware without user intervention.

8: Process and respond instantaneously - This requirement relates to delivering highly responsive feedback to end-users, even for high-volume applications.

The previously covered literature has been used to determine whether or not the beforementioned realtime data processing systems adhere to these requirements. The outcome of this is shown in Table 1.

Note that in Table 1, those cells with “N/A” as the value simply mean that the literature is inconclusive on whether or not they adhere to the particular requirement. Further investigation is required.

Table 1: **Realtime data processing systems compared**

	Storm	**Spark Streaming**	**S4**	**Samza**
1: Data mobility	Fetch	Micro-batch	Push	Fetch (Kafka)
2: SQL on streams	Extension (Trident)	No	No	No
3: Handling stream imperfections	User responsiblity	N/A	No	Yes (Kafka)
4: Deterministic outcomes	N/A	N/A	Depends on operation	N/A
5: Stored and streaming data	Yes (Lambda architecture)	Yes (Spark)	N/A	No
6: High availability	Yes (rollback recovery)	Yes (checkpoint recovery)	Yes (checkpoint recovery)	Yes (rollback recovery)
7: Partition and scaling	User responsiblity	N/A	Yes (KVP)	Yes (Kafka topics)
8: Instant response	N/A	N/A	N/A	N/A

2.3 Conclusion

The choosing of appropriate realtime data processing frameworks for the processing of a given application and dataset is an important, and often confusing, problem. Most frameworks compare themselves in terms of performance with other frameworks, often which are disputed by members in other framework "camps" [27, 22]. Hence, providing an informed recommendation based on processing requirements and the class into which the dataset falls is a much needed and important contribution to address the gap shown to exist in current systems.

This taxonomy of realtime data processing technologies will be based off the literature relating to the realtime data processing frameworks, as presented in §2.1.2. This taxonomy will be used, along with the created taxonomy of data classes, to develop the previously stated processing recommendations.

This research will further the field of realtime big data processing in addressing the shown gaps, and making the decision process far more streamlined for researchers and developers alike.

3 DSPS Technology Overview

3.1 DSPS Technology Choices

3.2 Testing Parameters

3.3 Evaluation Method & Approach

4 Implementation

4.1 Testing Environment Details

4.2 Setting Up of DSPS Technologies

4.3 Implementation of Pipelines in DSPS Technologies

5 Discussion and Evaluation

5.1 Quantitative Evaluations

5.1.1 Outcomes of Tests Performed

5.2 Qualitative Evaluations

5.3 DSPS Technology Recommendations

6 Conclusions

6.1 Research Contribution

6.2 Future Work

References

[1] Storm, distributed and fault-tolerant realtime computation. http://storm-project.net, November 2013.

[2] Amazon.com: Online Shopping for Electronics, Apparel, Computers, Books, DVDs & more. http://www.amazon.com, August 2014.

[3] Amplab - UC Berkeley — Algorithms, Machines and People Lab. https://amplab.cs.berkeley.edu, 2014.

[4] Apache Spark$^{\text{TM}}$– Lightning-Fast Cluster Computing. https://spark.apache.org, 2014.

[5] Australian Synchrotron. https://www.synchrotron.org.au, August 2014.

[6] home — NeCTAR. http://www.nectar.org.au, August 2014.

[7] Institute of railway technology. https://platforms.monash.edu/irt/, March 2014. (Visited on 2015-05-09).

[8] The Large Hadron Collider — CERN. http://home.web.cern.ch/topics/large-hadron-collider, August 2014.

[9] Samza. http://samza.incubator.apache.org, 2014.

[10] Spark Streaming — Apache Spark. https://spark.apache.org/streaming/, 2014.

[11] Bange, C., Grosser, T., and Janoschek, N. Big data survey europe - usage, technology and budgets in european best-practice companies. http://www.pmone.com/fileadmin/user_upload/doc/study/BARC_BIG_DATA_SURVEY_EN_final.pdf, 2013.

[12] Bifet, A. Mining big data in real time. *Informatica 37*, 1 (Mar. 2013), 15+.

[13] Bockermann, C. A survey of the stream processing landscape.

[14] Bohlouli, M., Schulz, F., Angelis, L., Pahor, D., Brandic, I., Atlan, D., and Tate, R. Towards an integrated platform for big data analysis. In *Integration of Practice-Oriented Knowledge Technology: Trends and Prospectives*. Springer, 2013, pp. 47–56.

[15] Borthakur, D. The hadoop distributed file system: Architecture and design. *Hadoop Project Website 11* (2007), 21.

[16] Chun, B.-G., Condie, T., Curino, C., Douglas, C., Matusevych, S., Myers, B., Narayanamurthy, S., Ramakrishnan, R., Rao, S., Rosen, J., et al. Reef: Retainable evaluator execution framework. *Proceedings of the VLDB Endowment 6*, 12 (2013), 1370–1373.

[17] Darby, M., Alvarez, E., McLeod, J., Tew, G., and Crew, G. The development of an instrumented wagon for continuously monitoring track condition. In *AusRAIL PLUS 2003, 17-19 November 2003, Sydney, NSW, Australia* (2003).

[18] Darby, M., Alvarez, E., McLeod, J., Tew, G., Crew, G., et al. Track condition monitoring: the next generation. In *Proceedings of 9th International Heavy Haul Association Conference* (2005), vol. 1, pp. 1–1.

[19] Darlin, D. Airfares made easy (or easier). *The New York Times 1* (2006), B1.

[20] Dean, J., and Ghemawat, S. MapReduce: Simplified data processing on large clusters. *Commun. ACM 51*, 1 (Jan. 2008), 107–113.

[21] Evans, B., and Feng, A. Storm-YARN Released as Open Source. `https://developer.yahoo.com/blogs/ydn/storm-yarn-released-open-source-143745133.html`, 2013.

[22] Evans, B., and Graves, T. Yahoo compares Storm and Spark. `http://www.slideshare.net/ChicagoHUG/yahoo-compares-storm-and-spark`, 2014.

[23] Gates, A. F., Natkovich, O., Chopra, S., Kamath, P., Narayanamurthy, S. M., Olston, C., Reed, B., Srinivasan, S., and Srivastava, U. Building a high-level dataflow system on top of map-reduce: The pig experience. *Proc. VLDB Endow. 2*, 2 (Aug. 2009), 1414–1425.

[24] Gates, A. F., Natkovich, O., Chopra, S., Kamath, P., Narayanamurthy, S. M., Olston, C., Reed, B., Srinivasan, S., and Srivastava, U. Building a high-level dataflow system on top of Map-Reduce: the Pig experience. *Proceedings of the VLDB Endowment 2*, 2 (2009), 1414–1425.

[25] Ghemawat, S., Gobioff, H., and Leung, S.-T. The google file system. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2003), SOSP '03, ACM, pp. 29–43.

[26] GitHub. yahoo/storm-yarn. `https://github.com/yahoo/storm-yarn`, 2014.

[27] Goetz, P. T. Apache Storm Vs. Spark Streaming. `http://www.slideshare.net/ptgoetz/apache-storm-vs-spark-streaming`, 2014.

[28] Grossman, R. L., and Gu, Y. On the varieties of clouds for data intensive computing. *IEEE Data Eng. Bull. 32*, 1 (2009), 44–50.

[29] Gruenheid, A., Omiecinski, E., and Mark, L. Query optimization using column statistics in hive. In *Proceedings of the 15th Symposium on International Database Engineering & Applications* (2011), ACM, pp. 97–105.

[30] Hadoop.apache.org. Who we are. `https://hadoop.apache.org/who.html#Hadoop+Committers`, 2014.

[31] Harrison, G. Hadoop's next-generation YARN. *Database Trends and Applications 26*, 4 (Dec. 2012), 39.

[32] Hunt, P., Konar, M., Junqueira, F. P., and Reed, B. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX Annual Technical Conference* (2010), vol. 8, p. 9.

[33] Hunter, T., Moldovan, T., Zaharia, M., Merzgui, S., Ma, J., Franklin, M. J., Abbeel, P., and Bayen, A. M. Scaling the mobile millennium system in the cloud. In *Proceedings of the 2nd ACM Symposium on Cloud Computing* (2011), ACM, p. 28.

[34] Isard, M., Budiu, M., Yu, Y., Birrell, A., and Fetterly, D. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS Operating Systems Review* (2007), vol. 41, ACM, pp. 59–72.

[35] Joseph, D. Using Spark and Hive to process BigData at Conviva. `http://www.conviva.com/using-spark-and-hive-to-process-bigdata-at-conviva/`, 2011.

[36] Kamburugamuve, S., Fox, G., Leake, D., and Qiu, J. Survey of distributed stream processing for large stream sources.

[37] Kreps, J., Narkhede, N., Rao, J., et al. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB* (2011).

[38] Lämmel, R. Google's mapreduce programming model—revisited. *Science of computer programming 70*, 1 (2008), 1–30.

[39] Lievesley, D. Increasing the value of data. *BLRD REPORTS 6122* (1993), 205–205.

[40] Liu, X., Iftikhar, N., and Xie, X. Survey of real-time processing systems for big data. In *Proceedings of the 18th International Database Engineering &#38; Applications Symposium* (New York, NY, USA, 2014), IDEAS '14, ACM, pp. 356–361.

[41] Marz, N. *Big data : principles and best practices of scalable realtime data systems.* O'Reilly Media, [S.l.], 2013.

[42] Mayer-Schönberger, V., and Cukier, K. *Big Data: A Revolution that Will Transform how We Live, Work, and Think.* An Eamon Dolan book. Houghton Mifflin Harcourt, 2013.

[43] McKusick, M. K., and Quinlan, S. GFS: Evolution on Fast-forward. *ACM Queue 7*, 7 (2009), 10.

[44] Murthy, A., Vavilapalli, V. K., Eadline, D., Markham, J., and Niemiec, J. *Apache Hadoop YARN: Moving Beyond MapReduce and Batch Processing with Apache Hadoop 2.* Pearson Education, 2013.

[45] Neumeyer, L., Robbins, B., Nair, A., and Kesari, A. S4: Distributed stream computing platform. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on* (2010), IEEE, pp. 170–177.

[46] Olston, C., Reed, B., Srivastava, U., Kumar, R., and Tomkins, A. Pig latin: A not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2008), SIGMOD '08, ACM, pp. 1099–1110.

[47] Olston, C., Reed, B., Srivastava, U., Kumar, R., and Tomkins, A. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (2008), ACM, pp. 1099–1110.

[48] Ritterman, J., Osborne, M., and Klein, E. Using prediction markets and twitter to predict a swine flu pandemic. In *1st international workshop on mining social media* (2009), vol. 9.

[49] Samza.incubator.apache.org. Samza. `https://samza.incubator.apache.org/`, 2014.

[50] Shvachko, K., Kuang, H., Radia, S., and Chansler, R. The hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)* (May 2010), pp. 1–10.

[51] Shvachko, K., Kuang, H., Radia, S., and Chansler, R. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on* (2010), IEEE, pp. 1–10.

[52] St Amant, K., and Ulijn, J. M. Examining the information economy: Exploring the overlap between professional communication activities and information-management practices. *Professional Communication, IEEE Transactions on 52*, 3 (2009), 225–228.

[53] Stonebraker, M., Çetintemel, U., and Zdonik, S. The 8 requirements of real-time stream processing. *ACM SIGMOD Record 34*, 4 (2005), 42–47.

[54] Storm.apache.org. Storm, distributed and fault-tolerant realtime computation. `https://storm.apache.org/`, 2014.

[55] Storm.apache.org. Storm documentation. `https://storm.apache.org/documentation/Powered-By.html`, 2014.

[56] Su, X., Swart, G., Goetz, B., Oliver, B., and Sandoz, P. Changing Engines in Midstream: A Java Stream Computational Model for Big Data Processing. *Proceedings of the VLDB Endowment 7*, 13 (2014).

[57] Tez.apache.org. Apache Tez – Welcome to Apache Tez. `http://tez.apache.org`, 2014.

[58] Thomas, S., Hardie, G., and Thompson, C. Taking the guesswork out of speed restriction. In *CORE 2012: Global Perspectives; Conference on railway engineering, 10-12 September 2012, Brisbane, Australia* (2012), Engineers Australia, p. 707.

[59] Thusoo, A., Sarma, J., Jain, N., Shao, Z., Chakka, P., Zhang, N., Antony, S., Liu, H., and Murthy, R. Hive - a petabyte scale data warehouse using hadoop. In *2010 IEEE 26th International Conference on Data Engineering (ICDE)* (Mar. 2010), pp. 996–1005.

[60] Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., and Murthy, R. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment 2*, 2 (2009), 1626–1629.

[61] Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Zhang, N., Antony, S., Liu, H., and Murthy, R. Hive-a petabyte scale data warehouse using hadoop. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on* (2010), IEEE, pp. 996–1005.

[62] Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J. M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, J., Bhagat, N., Mittal, S., and Ryaboy, D. Storm@Twitter. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2014), SIGMOD '14, ACM, p. 147–156.

[63] Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J. M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, J., et al. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data* (2014), ACM, pp. 147–156.

[64] Traffic.berkeley.edu. History of the Project — Mobile Millennium. `http://traffic.berkeley.edu/project`, 2014.

[65] Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., et al. Apache Hadoop YARN: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing* (2013), ACM, p. 5.

[66] Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., Saha, B., Curino, C., O'Malley, O., Radia, S., Reed, B., and Baldeschwieler, E. Apache hadoop YARN: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing* (New York, NY, USA, 2013), SOCC '13, ACM, pp. 5:1–5:16.

[67] Walker, J. Streaming IN Hadoop: Yahoo! release Storm-YARN - Hortonworks. `http://hortonworks.com/blog/streaming-in-hadoop-yahoo-release-storm-yarn/`, 2013.

[68] Wiki.apache.org. PoweredBy - Hadoop Wiki. `https://wiki.apache.org/hadoop/PoweredBy`, 2014.

[69] Yang, H.-c., Dasdan, A., Hsiao, R.-L., and Parker, D. S. Map-reduce-merge: simplified relational data processing on large clusters. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data* (2007), ACM, pp. 1029–1040.

[70] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., Mccauley, M., Franklin, M., Shenker, S., and Stoica, I. Fast and interactive analytics over hadoop data with spark. USENIX.

[71] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing* (2010), pp. 10–10.

[72] Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., and Stoica, I. Discretized streams: A fault-tolerant model for scalable stream processing. Tech. rep., DTIC Document, 2012.

[73] Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., and Stoica, I. Discretized streams: Fault-tolerant streaming computation at scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (2013), ACM, pp. 423–438.