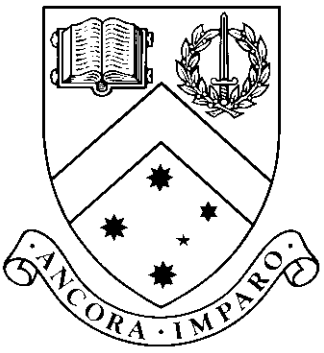
Clayton School of Information Technology
Monash University



Honours Literature Review — Semester 2, 2014

A study of the Hadoop ecosystem for pipelined realtime data stream processing

Jonathan Poltak Samosir

[2271 3603]

Supervisors: Dr Maria Indrawan-Santiago
Dr Pari Delir Haghighi

Contents

1 Introduction

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 [37]. Hence, those in society who realised the value of data early hold immense power over the entire economy, and in turn society, overall [27]. From seemingly inconsequential gains at the macro level, such as the ability to more accurately predict the rise and fall of airline tickets [9], 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 that the United States Centers for Disease Control and Prevention could [34] [29]. It is example 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 focussing 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 three main sections. In §2, an overview of the different classes and types of big data will be presented. This includes an overview of the big data classes presented through others' findings as well as our own proposed classes for big data, based on the criticisms of those prior findings. In §3, an overview will be given of the major open-source big data processing systems. A special emphasis will be given on data stream processing systems (DSPSs), given that the main area of this research is focusing on realtime data processing, or data stream processing. In §4 an analysis and discussion will be given on the content covered from the relevant literature. All of the sections will then be summarised in the conclusion in §5.

2 Data types and characteristics background

2.1 Velocity, variety, volume, and veracity

Data, and more specifically, big data, are often characterised into what is known as the "four V's" [49]. These can be thought of as different "dimensions" of big data, and can be summarised as follows [11]:

- *Velocity:* The rate at which data is being collected and made available to the data consumers.
- *Variety:* The heterogeneity of data. Big data often exhibits substantial variations in both the structural level and the instance level (representations of real-world entities). This is often highlighted by data systems that depend on acquiring of data from a number of non-conforming, and sometimes unrelated, data sources.
- *Volume:* The amount of data that is obtained by the data consumer from the data source/s.
- *Veracity:* The quality, in terms of accuracy, coverage, and timeliness, of data that is consumed from the data source/s. Veracity of data can widely differ between sources.

While the four V's are often described in terms of big data, they can also apply to more traditional data warehousing and processing in general, albeit on a far smaller scale. In the domain of big data processing, data will exhibit signs of high velocity, variety, and volume [3], and hence the veracity of the data may also fluctuate. Meanwhile, in more traditional data processing, the scope may be limited, especially in terms of factors such as variety and, as a consequence, there is less need of an emphasis on veracity due to limited variety in data sources.

As will be made clear in the following sections, a lot of the identified classes and characteristics of data directly relate back to these four V's. These can be considered the underlying features of many characteristics of data, both in the sense of big data and traditional data.

2.2 Classification of data

Data, in general, can be categorised into a number of different classes or types. We define the concept of a data class to mean the same as the terms of "data type", "data category", or "data format", as all terms were often used interchangeably in other literature.

Each class of data can be further defined and categorised via the characteristics they exhibit. Furthermore, these characteristics exhibited by data classes can be exploited and it is often possible to optimise the processing of each class of data by processing it using a specific method depending on those characteristics.

To give an example of this, data that is expected to have highly iterative processing applied to it would benefit from a data processor that does not have to unnecessarily write to disk after every single iteration. The elimination of this I/O overhead is an example of the optimisations that could be applied to the overall process from correctly identifying the data class beforehand, and processing it accordingly.

Furthermore, particular classes of data are generally only found in particular applications or use cases of data processing. As this is the case, it narrows down the amount of

classification needed, depending on the application that is being looked at. This will be elaborated on in later parts of this section.

There is no concrete, universally accepted standard for the classification of data. While the study of big data processing can arguably be considered still in its infancy, data handling and processing in general is relatively mature. From preliminary research on looking at past work and literature in this area, it must be noted that there is a significant lack of research on the classification of data.

The literature that will be reviewed in this section is often not wholly focused on the idea of data classification, hence data classification is presented relative to whatever the overall topic of the literature is on. This is important to note, as one attempt at data classification may not be appropriate under a different context. This also explains the large variation in different classification attempts, although we will also highlight the recurring similarities between different data classification literature.

2.2.1 IBM's classification of big data

This section looks at a classification of big data types, along with the key data characteristics, proposed by IBM Architects Mysore, Khupat, and Jain, published by IBM in 2013 [1]. The content is targeted towards beginners in the area of big data processing; much like the set of recommendations that we intend to produce from this research project. The different data classes, or "formats" as they were labelled, that are commonly encountered in big data are identified. For each of these formats, the underlying characteristics of the data was discussed, and it was noted that the type of processing needed would be dependent on those characteristics.

The characteristics of data, as put forward by Mysore et al., in [1], include the following:

Analysis type - Whether or not the data would be processed/analysed in realtime, or batched for later processing. Often this data class characteristic is dependent on the application of the data, *e.g.* the processing of social media data for the analysis of currently occurring events would want to be processed in realtime, regardless of the type of data that is involved.

Processing methodology - This characteristic involves the approach used when processing the data. Some examples of different processing methodologies include: predictive processing, analytical, ad-hoc queries, and reporting. Often the processing methodology for a particular class is determined by the business requirements or application of the data. Depending on the processing methodology used, many different combinations of big data technologies can be used.

Data frequency and size - The amount of data expected to arrive to the processing system, along with the speed and regularity of the incoming data. Knowing this characteristic beforehand can determine the methods for data storage and preprocessing, if needed. Examples of data frequency includes: on-demand data (social media), continuous/realtime (weather data, transactions), time-series (email). Considering the four V's, the characteristic of data frequency and size directly relates back to velocity and volume.

Content format - This characteristic relates back to the structure of the underlying data. Examples of data content format include: structured (JSON, XML), unstructured (human-readable literature), semi-structured (email).

Data source - This characteristic relates back to where the data originated from. As discussed previously in §2.1, the origin of data can have a great effect on whether or not

that data is usable, as data often varies greatly, especially when many different sources are used which may or may not conform to a specific content format. Another thing that is dependent on the data source is whether or not the data can be trusted. Considering the four V's, the characteristic of data source directly relates back to veracity and variety.

Table 1, highlights the different classes of data put forward by Mysore, et al., in [1]. The table organises each class, along with giving a brief explanation of the class. Furthermore, each class is related back to the previously explained characteristics in an attempt to show the connections between class and underlying characteristics.

The classes and characteristics of data presented by Mysore et al., in [1], are highly oriented towards industry and business users. While this is not an issue as such, as noted earlier in this section, these characteristics and data classes may not be as relevant or appropriate for usage in other non-business domains, or even business domains with a different focus on data.

Table 1: **IBM Data Classes**

Data class	**Explanation**	**Characteristics**
Machine generated data	• Data that is automatically generated as a by-product of some interaction with a machine.	• Structured data (JSON, XML). • Frequency of data varies depending on application.
Web & social data	• Data that is automatically generated through use of the Internet or social media, such as Facebook or Twitter.	• Unstructured text (long: blogs, short: microblogs, Facebook). • Miscellaneous multimedia (video, image, audio). • On-demand frequency. • Can be continuous feed of data in cases such as Twitter.
Transaction data	• Data that is automatically generated as a by-product of transactions, such as money transactions or otherwise.	• Structured text (JSON, XML, logs). • Continuous feed.
Human generated data	• Data that is solely produced by humans. • Examples of human generated data, as it is defined here, include such things as music, literature, recordings, and emails.	• Unstructured text (mail, literature). • Miscellaneous multimedia (audio, video, images). • Semi-structured text (email, online messaging services). • On-demand frequency.
Biometrics data	• Data that relates to human bioinformatics.	• Structured data. • On-demand frequency. • Continuous feeds of data in cases such as persistent health monitoring sensors (*i.e.* hospital patients).

2.2.2 Big data classification in BI&A

Another major contribution to big data classification is a paper from Chen, Chiang, and Storey, focusing on the impact of big data in the field of business intelligence and analytics [7]. Similarly to the paper looked at in §2.2.1, there is an emphasis on data classes and how they relate to the area of business and organisations. However, this paper has more of an explicit focus on business, being in published in the area of business intelligence and analytics (BI&A). BI&A in itself is a highly data driven field, where data is gathered and analysed to help make informed business decisions [50].

In the paper, Chen et al. [7], discuss the evolution of the field of BI&A, while elaborating on each identified stage of the evolution through highlighting of the major BI&A applications present. For each of the BI&A applications presented, they attempt to show the classes of data which are deemed important, and subsequently the characteristics associated which each class.

In Table 2, the classes and characteristics of data, given by Chen et al. [7], are presented. They are presented in terms of the BI&A application of which they are categorised under.

As can be seen from the data classes and characteristics identified by Chen et al. [7], there is a far greater variation to those previously presented by Mysore et al., in [4]. As explained earlier, this is mainly because of the more domain specific content of this piece of literature, while the paper from Mysore et al. [4], while still having underlying tones of business and industry, had a less explicit focus on their particular domain.

Table 2: **BI&A Data Classes**

BI&A Application	**Data types**	**Data characteristics**
E-Commerce & Market Intelligence	• Website logs and analytics data. • User activity logs for e-commerce websites. • User transaction records. • User-generated content, such as reviews, feedback.	• Structured web-based data (transactions records, logs, network information). • Unstructured user-generated content (reviews, feedback).
E-Government & Politics 2.0	• Government information, such as statistics. • Rules and regulations. • Citizen-generated content, such as feedback, comments, and requests.	• Fragmented data sources (think high data variety). • Unstructured data (citizen-generated content). • Rich textual content.
Science & Technology	• Machine-generated data from tools and instruments. • Sensor data. • Network data.	• High velocity data collection from instruments and tools and sensors. • Structured data, often formatted in uncommon structures.
Smart Health & Wellbeing	• Genomics and sequence data (DNA sequences). • Electronic health records. • Health and patient social media.	• Varying, but interrelated, data. • Data specific to individual patients.
Security & Public Safety	• Criminal record data. • Statistical data (crime maps). • Media content relating to crime (news articles). • Cyber-crime data (computer viruses, botnet data).	• Highly sensitive information (identity data). • Incomplete and deceptive content (speculative media content). • Multilingual content.

2.2.3 Big data classification in contemporary organisations

Coming away from the business point-of-view, Géczy [18] attempts to characterise big data in a more generic way. He uses what he labels as "aspects" to determine what he believes to be the deciding characteristics of data, in terms of the way they should be processed and also simply their intrinsic traits.

Géczy uses the following aspects to determine the different intrinsic characteristics of data:

Sensitivity - Relates to whether or not given data contains sensitive information, *i.e.* personally identifiable information, confidential information, etc. The sensitivity of the data determines the requirements relating to how it should be handled. Often it is either a legal requirement, or in the owners' interest, to keep protected the handled data deemed sensitive.

Diversity - Relates to the range of different data elements present within the data. The example given explains the ability of smart phones to produce highly diverse data; *e.g.* audio, video, location data, gyroscopic data, etc. Having high diversity in data can both be beneficial and detrimental; diversity can add factors of complexity, although also makes for a more rich dataset. Note that this data characteristic relates directly back to the *Variety* dimension, of the four V's.

Quality - Quality characteristics of data are defined to be features that affect data quality; *e.g.* completeness, accuracy, timeliness. Often the quality of data may be subject to the qualitative metrics of an organisation, or predefined standards. The quality of data relates back to the *Veracity* dimension, of the four V's.

Volume - Volume refers to the size of data in terms of its basic forms of measurement, bits and bytes. Volume is an important characteristic to take into consideration when it comes to determining the type of processing needed. Volume, as the name suggests, directly relates back to the *Volume* dimension of the four V's.

Speed - Data speed refers to the inflow and outflow speeds; inflow being the data that is being acquired, while outflow being the data leaving the system (often results of computations). Different classes of data often require different data speeds. *e.g.* audio is often streamed at a far lesser speed than video, due to the relatively low amount of data in audio when compared with video.

Structure - Structure relates to whether data are in structured or unstructured formats. Generally unstructured data is more suitable for human consumption, such as literature or music. Structured data is usually structured in such a way that it is easily able to be parsed by an algorithm, often automated by computers. The structure of data directly relates to the difficulty of processing that data, as unstructured data usually will need some pre-processing or artificial intelligence to process.

Géczy later goes on to talk about the aspects of data that relate to data processing, similar to what will be talked about later in §3.

Overall, Géczy looks at data characteristics, not from any particular perspective, but from one that attempts to capture the interests and be relevant to a number of disciplines.

2.3 Discussion and analysis

From the literature presented previously in this section, there are a number of notable points. Firstly, they were all highly varied in the classifications and characteristics of

data given. As previously stated, this can be attributed to the variety in sources for this literature; they were all produced in the context of quite different domains, aiming the content for most relevance in those domains. However, while there is much disconnect between the identified classes and characteristics of data presented in the papers, there are also some similarities which show the connections between data in different research domains.

For example, Biometrics Data, as shown in Table 1 by Mysore et al.[1], can be seen as being related to that of data under the BI&A application of Smart Health & Wellbeing, as presented in Table 2 from Chen et al.[7]. By "related", what is meant is that the characteristics of data underlying these classes will be similar. For example, data relating to a hospital patient's vital signs may fall under both of these identified classes. These relations between data classes extend to Géczy's classifications as well, with certain data that would fall under the class of Transaction Data, from Mysore et al.[1], also falling under the classification of being highly sensitive data, as put forward by Géczy [18].

Given Géczy's contribution of big data classification in terms of characteristics intrinsic to data [18], shown in §2.2.3, a lot of these characteristics can be related back to the big data classifications given by Mysore et al.[1] and Chen et al.[7] in Table 1 and Table 2, respectively. This has been attempted to be shown in Table 3, where the classes of big data from Mysore et al., and Chen et al., are shown in relation to each of Géczy's characteristics, and whether or not they show high or low levels of those characteristics.

Note that certain data classes fall under both high and low variations of a given characteristic, and sometimes are not present under either. This is mainly due to displaying high or low levels of a characteristic under different contexts, or there being inconclusive information relating to the data classes' characteristics.

Also note that the data classes of machine and human generated data, as put forward by Mysore et al.[1], have been emitted. In the classification given by Mysore et al., the classes of machine and human generated data are presented as two discrete classes of data. The distinction between the two is very clear, however it could very much be argued that these two classes should be considered super classes of which other presented data classes could be classified under, rather than being presented as equal classes. The examples given for types of data that fall under the classes of human generated and machine generated data are also very vague in the way that they are presented, and could very easily fall under other identified classes as well.

Table 3: **Data classes compared**

Characteristic	High	Low
Sensitivity	• Smart Health & Wellbeing • Security & Public Safety • Transaction data • Web & Social Data • Biometrics data	• E-Commerce & Market Intelligence • E-Government & Politics 2.0 • Science & Technology • Web & Social Data
Diversity	• E-Commerce & Market Intelligence • E-Government & Politics 2.0 • Science & Technology • Smart Health & Wellbeing • Security & Public Safety	• Web & Social Data • Transaction data • Biometrics data
Quality	• Science & Technology • Smart Health & Wellbeing • Security & Public Safety • Web & Social Data • Transaction data • Biometrics data	
Volume	• E-Commerce & Market Intelligence • E-Government & Politics 2.0 • Science & Technology • Smart Health & Wellbeing • Security & Public Safety • Web & Social Data • Transaction data • Biometrics data	• Science & Technology • Smart Health & Wellbeing • Transaction data • Biometrics data
Speed	• Science & Technology • Web & Social Data • Transaction data	• E-Commerce & Market Intelligence • E-Government & Politics 2.0 • Biometrics data
Structured	• E-Commerce & Market Intelligence • E-Government & Politics 2.0 • Science & Technology • Smart Health & Wellbeing • Security & Public Safety • Transaction data • Biometrics data	• E-Commerce & Market Intelligence • E-Government & Politics 2.0 • Web & Social Data

3 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".

3.1 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 [4]. 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 [26] or even Java 8 [41], to apply a specified type of processing in a highly parallelised and distributed fashion [52].

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.

3.1.1 MapReduce and GFS

Dean and Ghemawat, in [10], 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 [14], 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 [10]. Furthermore, McKusick and Quinlan, in [30], state that, as of 2009, the majority of Google's data relating to their many web-oriented applications are rely on GFS.

3.1.2 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 [16]. 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 framework [1]. The Apache Hadoop framework, 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 [19].

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 right. In the original paper from Yahoo! [36], 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 [6], 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 [14], 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.

3.1.3 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 [43], 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 [44].

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,

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

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 [33]. 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 [13, 44] as well as having the benefit of being susceptible to manual query optimisations, familiar to programmers familiar with query optimisations from SQL [17].

3.2 Realtime data processing

With HDFS being an open source project with a large range of users [51] and code contributors [19], 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 §2.1. 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.

3.2.1 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 [28]. 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 [24].

A recent (2013) industry survey on European company use of big data technology by Bange, Grosser, and Janoschek, noted in [2], 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) [47]. 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 [22]. 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 [54]. Spark, and in particular Spark Streaming, will be looked at further in §3.2.3.
- Storm, a realtime stream processing system [31, p. 244]. Performs specified processing on an incoming stream of data indefinitely, until stopped. Storm will be looked at further inse̊ctrefssub:storm.
- Tez, an extensible framework which allows for the building of batch and interactive Hadoop applications [42].
- REEF, a YARN-based runtime environment framework [8]. 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 §3.2.4.

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

3.2.2 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 [39] BackType has since been acquired by Twitter, Inc. where development has continued. Toshniwal et al. [45] 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" [45, 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 [40].

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" [15]. 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 [48] and Hortonworks [12].

3.2.3 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 [54]. 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 [55].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 [56].

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 [23, 53]. The Mobile Millennium project at UC Berkeley [46], 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 [21].

3.2.4 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 [35]. 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 [25], integrating with the Hadoop ecosystem.

While Samza is lacking in maturity and adoption rates, as compared to project 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, and stream persistence and imperfection handling is offloaded to Kafka [5]. Likewise, YARN is used for ensuring fault tolerance through the handling of restarting machines that have failed in a cluster [5].

3.2.5 S4

S4 (Simple Scalable Streaming System) is another realtime big data processing framework that originated at Yahoo!, Inc. that has since been open-sourced [32]. 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 §3.1.1.

Much like what was said about Samza in §3.2.4, 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 [24]. 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 [20].

3.3 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 [38], which can be used to given an impartial comparison of the previously covered projects. The requirements were defined a number of years prior to (2005) the creation of the four main realtime systems that were covered, 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.

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 4.

Table 4: **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

4 Analysis and discussion

5 Conclusion

References

[1] Big data architecture and patterns, part 1: Introduction to big data classification and architecture, Sept. 2013.

[2] Bange, C., Grosser, T., and Janoschek, N. Big data survey europe - usage, technology and budgets in european best-practice companies, 2013.

[3] Beyer, M. Gartner says solving 'big data' challenge involves more than just managing volumes of data, 2011.

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

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

[6] Borthakur, D. The hadoop distributed file system: Architecture and design.

[7] Chen, H., Chiang, R. H., and Storey, V. C. Business intelligence and analytics: From big data to big impact. *MIS quarterly 36*, 4 (2012), 1165–1188.

[8] 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.

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

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

[11] Dong, X. L., and Srivastava, D. Big data integration. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on* (2013), IEEE, pp. 1245–1248.

[12] Evans, B., and Feng, A. Storm-yarn released as open source, 2013.

[13] 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.

[14] 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.

[15] GitHub. yahoo/storm-yarn, 2014.

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

[17] 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.

[18] Géczy, P. BIG DATA CHARACTERISTICS.

[19] Hadoop.apache.org. Who we are, 2014.

[20] 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.

[21] 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.

[22] 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.

[23] Joseph, D. Using spark and hive to process bigdata at conviva, 2011.

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

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

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

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

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

[29] 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.

[30] McKusick, M. K., and Quinlan, S. Gfs: Evolution on fast-forward. *ACM Queue 7*, 7 (2009), 10.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] Samza.incubator.apache.org. Samza, 2014.

[36] 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.

[37] 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.

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

[39] Storm.apache.org. Storm, distributed and fault-tolerant realtime computation, 2014.

[40] Storm.apache.org. Storm documentation, 2014.

[41] 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).

[42] Tez.apache.org. Apache tez – welcome to apache tez, 2014.

[43] 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.

[44] 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.

[45] 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.

[46] Traffic.berkeley.edu. History of the project — mobile millennium, 2014.

[47] 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.

[48] Walker, J. Streaming in hadoop: Yahoo! release storm-yarn - hortonworks, 2013.

[49] Wang, L., Zhan, J., Luo, C., Zhu, Y., Yang, Q., He, Y., Gao, W., Jia, Z., Shi, Y., Zhang, S., et al. Bigdatabench: A big data benchmark suite from internet services. *arXiv preprint arXiv:1401.1406* (2014).

[50] Watson, H. J. Tutorial: Business intelligence-past, present, and future. *Communications of the Association for Information Systems 25*, 1 (2009), 39.

[51] Wiki.apache.org. Poweredby - hadoop wiki, 2014.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] 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.