Suthakar *et al. J Big Data (2016) 3:21*
DOI 10.1186/s40537-016-0056-1

Journal of Big Data

RESEARCH Open Access

An efficient strategy for the collection and storage of large volumes of data for computation

Uthayanath Suthakar[1,2*], Luca Magnoni[2], David Ryan Smith[1], Akram Khan[1] and Julia Andreeva[2]

*Correspondence: Uthayanath.Suthakar@brunel.ac.uk
[1] College of Engineering, Design and Physical Sciences, Brunel University London, Uxbridge, Middlesex UB8 3PH, UK
Full list of author information is available at the end of the article

Abstract

In recent years, there has been an increasing amount of data being produced and stored, which is known as Big Data. The social networks, internet of things, scientific experiments and commercial services play a significant role in generating a vast amount of data. Three main factors are important in Big Data; Volume, Velocity and Variety. One needs to consider all three factors when designing a platform to support Big Data. The Large Hadron Collider (LHC) particle accelerator at CERN consists of a number of data-intensive experiments, which are estimated to produce a volume of about 30 PB of data, annually. The velocity of these data that are propagated will be extremely fast. Traditional methods of collecting, storing and analysing data have become insufficient in managing the rapidly growing volume of data. Therefore, it is essential to have an efficient strategy to capture these data as they are produced. In this paper, a number of models are explored to understand what should be the best approach for collecting and storing Big Data for analytics. An evaluation of the performance of full execution cycles of these approaches on the monitoring of the World-wide LHC Computing Grid (WLCG) infrastructure for collecting, storing and analysing data is presented. Moreover, the models discussed are applied to a community driven software solution, Apache Flume, to show how they can be integrated, seamlessly.

Keywords: Big Data, Data pipeline, Hadoop, Apache Flume, MapReduce, HDFS

Introduction

The field of data science has become a widely discussed topic in recent years due to a data explosion, especially with scientific experiments such as those that are part of the Large Hadron Collider (LHC) at CERN and commercial businesses keen to enhance their competitiveness by learning about their customers to provide tailor made products and services, dramatically increasing the usage of sensor devices. Traditional techniques of collecting (e.g. lightweight Python framework), storing (e.g. Oracle) and analysing (e.g. PL/SQL) data are no longer optimal with the overwhelming amount of data that are being generated. The challenge of handling big volumes of data has been taken on by many companies, particularly those in the internet domain, leading to a full paradigm shift in methods of data archiving, processing and visualisation. A number of new technologies have appeared, each one targeting specific aspects of large-scale

distributed data-processing. All these technologies, such as batch computation systems (e.g. Hadoop) and non-structured databases (e.g. MongoDB), can handle very large data volumes with little financial cost. Hence, it becomes necessary to have a good understanding of the currently available technologies to develop a framework which can support efficient data collection, storage and analytics.

The core aims of the presented study were the following:

- To propose and design efficient approaches for collecting and storing data for analytics that can also be integrated with other data pipelines seamlessly.
- To implement and test the performance of the approaches to evaluate their design.

Background

Over the past several years there has been a tremendous increase in the amount of data being transferred between Internet users. Escalating usage of streaming multimedia and other Internet based applications has contributed to this surge in data transmissions. Another facet of the increase is due to the expansion of Big Data, which refers to data sets that are many orders of magnitude larger than the standard files transmitted via the Internet. Big Data can range in size from hundreds of gigabytes to petabytes [1].

Within the past decade, everything from banking transactions to medical history has migrated to digital storage. This change from physical documents to digital files has necessitated the creation of large data sets and consequently the transfer of large amounts of data. There is no sign that the continued increase in the amount of data being stored and transmitted by users is slowing down. Every year Internet users are moving more and more data through their Internet connections. With the growth of internet based applications, cloud computing, and data mining, the amount of data being stored in distributed systems around the world is growing rapidly. Depending on the connection bandwidth available and the size of the data sets being transmitted, the duration of data transfers can be measured in days or even weeks. There exists a need for an efficient transfer technique that can move large amounts of data quickly and easily without impacting other users or applications [1].

In addition to corporate and commercial data sets, academic data are also being produced in similarly large quantities [2]. To give an example of the size of the data sets utilised by some scientific research experiments, a recent study observed a particle physics experiment (DZero) taking place at the Fermi Lab research center. While observing the DZero experiment between January 2013 and May 2015, Aamnitchi et al. [2] analysed the data usage patterns of users. They found that 561 users processed more than 5 PB of data with 13 million file accesses to more than 1.3 million distinct data files. An individual file was requested by at most 45 different users during the entire analysed time period.

In the DZero experiment, and many like it, scientists are generating datasets with an extremely large number of data files. Use of entire datasets is quite popular amongst users, however, the individual data files in these sets are rarely used concurrently since they are so numerous.

There are many scientific research facilities that have similar data demands. The most popular and well known example today is the LHC at CERN where thousands of researchers in the fields of physics and computer science are involved with the various

experiments based there. The experiments being conducted at the LHC generate petabytes of data annually [3, 4]. One experiment, ALICE, can generate data at the rate of 1.25 GB/s. Figure 1 illustrates the growth in the size of data sets being created and stored by CERN. This graph shows the total amount of storage (both disk and tape) utilised by all of the top-level servers in the CERN organisation. The amount of data stored in the system has grown at a steady pace over the past 3 years and is expected to grow faster now that the intensity of their experiments is increasing, which will result in more data collected per second [5].

Geographically dispersed researchers eagerly await access to the newest datasets as they become available. The task of providing and maintaining fast and efficient data access to these users is a major undertaking. Also, monitoring computing behaviours in the Worldwide LHC Computing Grid (WLCG), such as data transfer, data access, and job processing, is crucial for efficient resource allocation. This requires the gathering of metadata which describes the data (e.g. transfer time) from geographically distributed sources and the processing of such information to extract the relevant information for the WLCG group [6]. Since the LHC experiments are so well known and many studies have been conducted on their demands and requirements, one can use the LHC experiments as a suitable case study for this research.

To meet the computing demands of experiments like those at the LHC, a specialised distributed computing environment is needed. Grid computing fits the needs of the LHC experiments and other similar research initiatives.

The WLCG was created by CERN in 2002 in order to facilitate the access and dissemination of experimental data. The goal of the WLCG is to develop, build, and maintain a distributed computing infrastructure for the storage and analysis of data from LHC experiments [7]. The WLCG is composed of over a hundred physical computing centers with more than 100,000 processors [8]. Since the datasets produced by the LHC are extremely large and highly desired, the WLCG utilises replication to help meet the demands of users. Copies of raw, processed, and simulated data are made at several locations throughout the grid.

The WLCG utilises a four-tiered model for data dissemination. The original raw data is acquired and stored in the Tier-0 center at CERN. This data is then forwarded in a highly

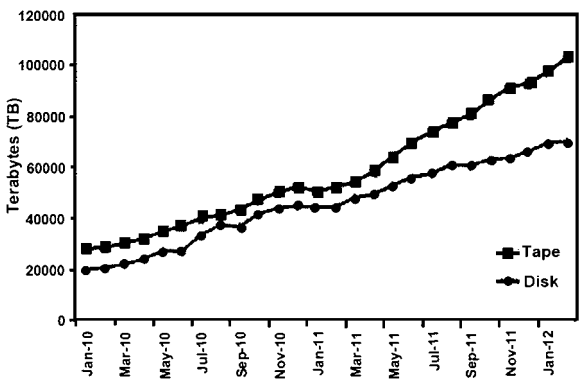

Fig. 1 Size of CERN LHC experimental data sets over the past years. The total disk and tape storage amounts aggregated for all Tier-1 locations in the CERN grid (adapted from [5])

controlled fashion on dedicated network connections to all Tier-1 sites. The Tier-1 sites are located in Canada, Germany, Spain, France, Italy, Nordic countries, Netherlands, Republic of Korea, Russian Federation, Taipei, United Kingdom and USA (Fermilab-CMS and BNL ATLAS).

The role of the Tier-1 sites varies according to the particular experiment, but in general they are responsible for managing permanent data storage (of raw, simulated, and processed data) and providing computational capacity for processing and analysis [7]. The Tier-1 centers are connected with CERN through dedicated links (Fig. 2) to ensure high reliability and high-bandwidth data exchange, but they are also connected to many research networks and to the Internet [8]. The underlying components of a Tier-1 site consist of online (disk) storage, archival (tape) storage, computing (process farms), and structured information (database) storage. Tier-1 sites are independently managed and have pledged specific levels of service to CERN. It is left to a given site's administrators to guarantee that these services are reliably provided.

The Tier-2 sites are used for Monte Carlo event simulation and for end-user analysis. Any data generated at Tier-2 sites is forwarded back to Tier-1 centers for archival storage.

Other computing facilities in universities and research laboratories are able to retrieve data from Tier-2 sites for processing and analysis. These sites constitute the Tier-3 centers, which are outside the scope of the controlled WLCG project and are individually maintained and governed. Tier-3 sites allow researchers to retrieve, host, and analyse specific datasets of interest. Freed from the reprocessing and simulation responsibilities

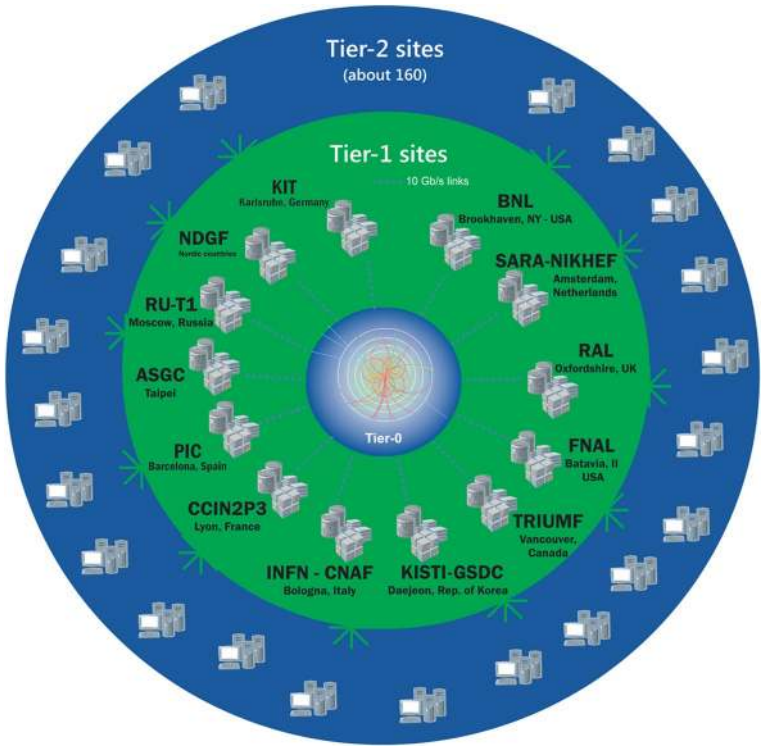

Fig. 2 WLCG Tier-1 and Tier-2 connections [8]

of Tier-1 and Tier-2 centers, these Tier-3 sites can devote their resources to their own desired analyses and are allowed more flexibility with fewer constraints [9]. As there are thousands of researchers eagerly waiting for new data to analyse, many users will find less competition for time and resources at Tier-3 sites than at the Tier-2 sites.

It is important to note that users connecting to either Tier-2 or Tier-3 sites will use public, shared network connections, including the Internet. Grid traffic and normal world wide web traffic will both be present on these shared links. A user will also be sharing the site that they access with multiple other users. These factors can affect the performance of the data transfer between the selected retrieval site and the user. Retrieving these large data files also places a burden on shared resources and impacts other grid and non-grid users. When it comes to retrieving data in the WLCG, a normal user (depending on their security credentials) can access data on either Tier-2 or Tier-3 sites. The user would select a desired site and issue a request for a specific data file. Selecting a site to utilise can be a complicated task, with the performance a user obtains being dependent on the location chosen.

Grid computing has emerged as a framework for aggregating geographically distributed, heterogeneous resources that enables secure and unified access to computing, storage and networking resources for Big Data [10]. Grid applications have vast datasets and/or carry out complex computations that require secure resource sharing among geographically distributed systems.

Grids offer coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organisations [11]. A virtual organisation (VO) comprises a set of individuals and/or institutions having access to computers, software, data, and other resources for collaborative problem-solving or other purposes [12]. A grid can also be defined as a system that coordinates resources that are not subject to centralised control, using standard, open, general-purpose protocols and interfaces in order to deliver non-trivial qualities of service [13].

A number of new technologies have emerged for handling big-scale distributed data-processing, (e.g. Hadoop), where the belief is that moving computation to where data reside is less time consuming than moving data to a different location for computation when dealing with Big Data. This is certainly true when the volume of data is very large because this approach will reduce network congestion and improve the overall performance of the system. However, a key grid principle contradicts with this as in the grid approach computing elements (CE) and storage elements (SE) should be isolated, although this is changing in modern grid systems. Currently, a lot of scientific experiments are beginning to adopt the "new" Big Data technologies, in particular for metadata analytics at the LHC, hence the reason for the presented study.

Parallelising is used in order to enhance computations of Big Data. The well known MapReduce [14] framework that has been used in this paper has been well developed in the area of Big Data science and has the parallelization feature. Its other key features are its inherent data management and fault tolerant capabilities.

The Hadoop framework has also been employed in this paper. It is an open-source MapReduce software framework. For its functions it relies on the Hadoop Distributed File System (HDFS) [15], which is a derivative of the Google File System (GFS) [16]. In its function as a fault-tolerance and data management system, as the user provides data to

the framework, the HDFS splits and replicates the input data across a number of cluster nodes.

The approaches for collecting and storing Big Data for analytics described in this paper were implemented on a community-driven software solution, Apache Flume, in order to understand how the approaches can be integrated seamlessly the data pipeline. Apache Flume is used for effectively gathering, aggregating, and transporting large amounts of data. It has a flexible and simple architecture, which makes it fault tolerant and robust with tunable reliability and data recovery mechanisms. It uses a simple extensible data model that allows online analytic applications [17].

Design and methodology

When data messages are consumed from a data transport layer and written into storage, there will most likely be some sort of data transformation carried out before storage in the storage layer. Such a transformation could be extracting the body from the message and removing the header as it is not required, or serialisation or compression of the data. The WLCG uses a Python agent, the Dashboard consumer, to collect infrastructure status updates, transform them, and store them in the data repository, which is implemented in Oracle. It uses Procedural Language/Structured Query Language (PL/SQL) procedures for analytics. This is an example of a traditional approach that is commonly used. However, these technologies and methods are no longer optimal for data collection, storage and analytics as they are not primarily designed for handling Big Data. There needs to be a strategy in place to carry out the required transformation as this will play a significant role in improving the performance of subsequent computations. In this paper three different approaches were explored:

1. Implement the data transformation logic within the data pipeline. Therefore, the messages, M, will be read by the consumer, to apply the transformation $<T>$ and to write the results into the storage layer, S, for analytics $<A>$:

$$M \overset{<T>}{\rightarrow} S \rightarrow < A > \tag{1}$$

2. Write the raw messages, M, directly into the storage layer, S, without any modification. Then there is another intermediate transformation $<iT>$ that reads the raw data from storage, transforms the data and writes the results into a new path but to the same storage layer for analytics $<A>$:

$$M \rightarrow S \overset{<iT>}{\rightarrow} S \rightarrow < A > \tag{2}$$

3. Write the raw messages, M, into the storage layer, S, without any modification. Let the analytics $<A>$ jobs carry out the transformation $<T>$:

$$M \rightarrow S \rightarrow \ll T > + < A \gg \tag{3}$$

The first approach is the traditional way of transforming, storing and computing data as has been already described for the WLCG use case. However, this method relies too much on the data pipeline. If the data pipeline is replaced then the transformation logic would need to be re-implemented. Therefore, it is an inefficient design. Nevertheless, this method needs to be tested on the technology that supports Big Data.

The second approach has two benefits as the transformation logic is moved to a centralised location and untampered raw data are stored as well as the transformed data. Therefore, in the case of any inaccuracy in the transformed data, the correct transformed data can be recreated from the raw data. This is not possible with the first method because as soon as the data are transformed the raw data are discarded. Nevertheless, the second approach is very complex as there is a requirement for a job to transform the data, rather than the consumer carrying out the transformation, and it raises the question of when and how this job should be scheduled. This approach also requires increased data storage as both raw and transformed data will be kept. A transformation job could be used here to compress the raw data and archive it to reduce the amount of storage required.

The third option is very simple and straight forward, as the raw data will be written into the storage layer without any modification. The transformation will only take place at the data analytics time. The transformation logic can be implemented in a shared library, which can be imported into any analytics jobs. Therefore, the transformation will take place as and when it is required. This way, the untempered raw data is still kept in the storage layer and no additional job or storage is needed for data transformation. This approach does add an extra execution time overhead to the analytics jobs and will repeat the data transformation every time an analytics job is carried out. This should, however, not be too much of a problem as Big Data technologies are built to enhance computation speed by parallelising jobs. Hence, this arrangement should not significantly affect the execution time. A summary of the advantages and disadvantages of the proposed three approaches is given in Table 1.

Table 1 Summary of advantages and disadvantages of the proposed approaches

	Advantage	Disadvantage
Approach 1 Data transformation occurs within the data pipeline	Well tested approach: typical scenario in most data analytics platforms	Complex: transformation logic is kept in the data pipeline so in the case of data pipeline replacement the transformation logic needs to be re-implemented Lost data authenticity: the data is transformed by the data pipeline so the raw data is lost
Approach 2 Data transformation occurs within the storage layer	Easy to migrate/replace: the transformation logic is moved to a centralised location so it is easier to migrate or replace the data pipeline Raw data is intact: meets regulatory standards of storing the raw data both before and after transformation	Complex: an intermediate job is required for transformation Large storage needed: both raw and transformed data are stored
Approach 3 Data transformation occurs within the analytics jobs	Clean and simple: no complexity added to the data pipeline Less storage needed: only raw data is stored Easy to migrate or replace: the transformation logic is moved to a centralised location	Increased execution overhead: the analytics job will transform the data Repetition: transformation will take place every time an analytics job is executed

Implementation

The data pipeline presented in this paper uses Dirq library that offers a queue system, using the underlying file system for storage for consuming messages, which allows concurrent read and write operations [18]. Therefore, it can support a variety of heterogeneous applications and services that can write messages and have multiple readers reading the messages simultaneously. The data pipeline was developed using the Hadoop native library that reads messages from the Dirq library and writes them into HDFS using an appending mechanism. The Hadoop software framework was originally designed as a create-once-read-many system [19]. Therefore, appending was not available in the initial software release but later versions, 2.0 onwards, supported this mechanism. Hadoop also has the benefit of working well with a few large files but is not as efficient when working with a large number of small files. The appending method is convenient as it allows for the creation of a single large file.

For the first approach, the data will be consumed from the Dirq, transformed and written into HDFS. The implementation of the second approach is similar to the first with the exception of no transformation being carried out in the pipeline. However, it requires chained MapReduce jobs in a centralised Hadoop cluster in order to take the raw data that has not previously been processed, and apply the appropriate data transformation, merge the transformed data with previously transformed data, delete the old transformed data, update the raw data as processed and merge and compress the raw data. An issue was encountered during testing of this second approach where it was found that data that were not processed by the transformation job were not then available for analytics. The third approach is again like the second approach in that no transformation is carried out in the data pipeline, but the transformation logic is implemented in a common library and is available to be imported into any analytics jobs. Therefore, the transformation can be carried out as and when it is required. This approach does not have the issue of data unavailability as present in the other two approaches as all written data will be picked up by the analytics jobs and the transformation will be done as and when required. All three approaches were implemented as a daemon that continuously ran on the WLCG test infrastructure checking for data every 5 min.

In order to decrease the data aggregation delay from the data pipeline and to evaluate how easy it is to migrate these approaches to a different data pipeline, Apache Flume was used. Apache Flume is a community-driven software solution that receives messages from the transport layer and writes them into HDFS. There are three ways to flush consumed data into HDFS: periodically based on the elapsed time, the size of data or the number of events [17].

As expected, the first approach was complex as all the transformation logic was in the custom data pipeline so the transformation logic had to be re-implemented into Apache Flume. The second and third approach made the migration to Apache Flume extremely simple, as all the transformation logic was implemented within the storage layer. But, as noted before, the second approach added complexity to the storage layer, as it required a chain of actions for data transformation. The third approach was the simplest to implement, as no transformation was carried out on the Apache Flume side and no transformation was carried out in the storage layer, keeping the complexity low.

All three approaches did encounter a common problem: Apache Flume pushes the events but does not flush the file until the configured file roll time is met (e.g. every hour)

resulting in the data being unavailable for computation between these times. While HDFS supports appending functionality, and the custom data pipeline, Apache Flume does not support it. The analytics jobs were able to read the data that were written by the custom data pipeline but not those written by Apache Flume. Therefore, the appending functionality was taken from the custom pipeline and implemented into Apache Flume, making it a custom library (see Algorithm 1). With this amendment, Apache Flume was then able to write a single file and append it while at the same time analytics jobs were able to read the data while the data were being written into HDFS.

Algorithm 1 File appending algorithm for Apache Flume: adding a close and reopen at every push to get the required append behaviour.

```
1:  procedure create-global-data-file-writer
2:          declare a global DataFileWriter object
3:          create a file in HDFS
4:          initialise the file to global DataFileWriter

1:  procedure consume-messages-and-sync-flush
2:          create a temp DataFileWriter refelecting(reopen) the global DataFileWriter
3:          consume all messages
4:          append messages using temp DataFileWriter
5:          close temp DataFileWriter WHEN messages <= 0

1:  procedure roll-files
2:          close the global DataFileWriter
```

Results and discussion

The three approaches developed for the collection, storage and analytics of Big Data described in this paper were evaluated on the WLCG infrastructure that provides the computing resources to store, distribute and analyse the 30 petabytes of data generated annually by the LHC and distributed to 170 computing centres around the world [20]. Furthermore, the current method used by the WLCG group for the collection and storage of data for analytics was evaluated for benchmarking the new approaches.

It was very complicated to carry out performance measurements on the proposed approaches and the current approach, as they employ different methods for consuming, writing and transforming the data in each case. Therefore, in order to get a meaningful performance measurement, a full computation cycle was carried out, including: consuming messages, writing to HDFS and carrying out a simple analytics job on those data. The full cycle comprised three segments:

1. Data ingestion with data transformation and without data transformation.
2. Intermediate data transformation using a MapReduce job.
3. A simple statistical analytic computation using a MapReduce job and a PL/SQL procedure with and without data transformation.

The configurations of the current and proposed data pipelines in the WLCG are shown in Fig. 3a and b respectively. For both configurations, the monitoring events are pushed as JavaScript Object Notation (JSON) records through the STOMP protocol to the ActiveMQ message broker. However, the configuration varies from the consumers

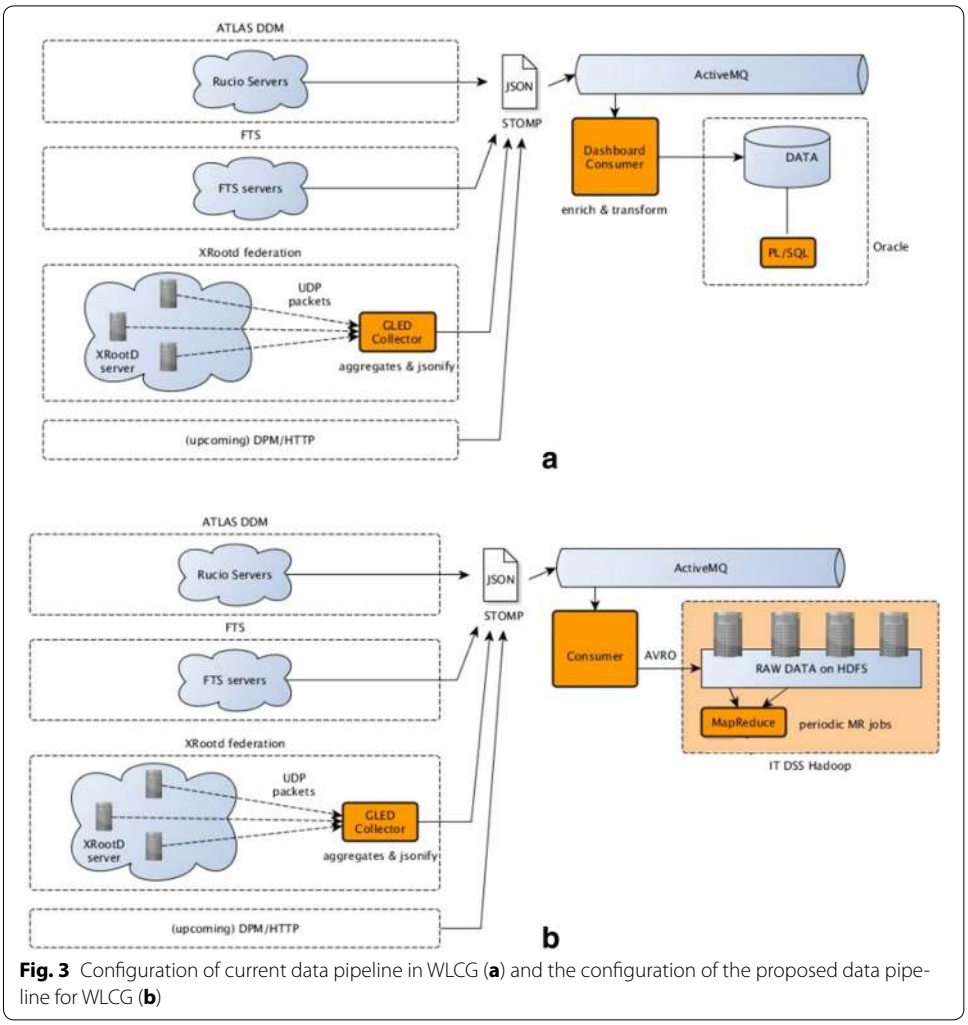

Fig. 3 Configuration of current data pipeline in WLCG (**a**) and the configuration of the proposed data pipeline for WLCG (**b**)

in both data pipelines. The current configuration uses Python collectors for reading the monitoring events, transforming and writing them into an Oracle storage database. On the other hand, the proposed configuration uses a custom data pipeline daemon, as explained in "Implementation" section that reads monitoring events and writes them into a Hadoop cluster. This configuration can be modified to support the three proposed approaches, i.e. transform and serialise the messages into Avro format.

In order to evaluate the proposed approaches, it was decided to push messages from the broker in batch sizes ranging from 10,000 to 100,000 messages. Data ingestion and analytics were conducted ten times for each batch of messages in order to capture an average performance time. The performance measurements were carried out on a heterogeneous Hadoop cluster that consisted of 15 nodes (8 nodes: 32 cores/64 GB, 7 nodes: 4 cores/8 GB).

Performance results of data ingestion with and without data transformation

The first approach had to consume all messages from Dirq, apply a simple data transformation, which involved taking the source and destination IP address from the message

and using a topology mapping file to determine the domain address and replace the IP address with the domain, and finally, convert the data file into Avro format, which is a data serialisation framework that serialises the data into a compact binary format and writes the file into HDFS. As shown in Fig. 4, this approach (pre-trans-avro) is slower than the second approach (raw-json), which just reads the raw messages in JSON, an easy-to-read format, and writes them into HDFS. The second approach is the fastest compared with the first and third approaches (raw-avro), which read raw data, convert them into Avro format and write them into HDFS. The third approach was faster than the first approach because it does not do any transformation.

The current approach (pyth-traditional-plsql) used by the WLCG is similar to the proposed first approach (pre-trans-avro) but the difference is that it uses the Python agent for collection and the Oracle database for storing the transformed data, so no serialisation is involved. Although the current approach is similar to the first of the three proposed approaches the performance of the current approach was slower than all three of the newly proposed approaches. This is due to the connection and communication limitations that occurs between the database and collectors.

Figure 5 shows data representing unprocessed messages from the broker, raw JSON messages, a pre-transformed Avro and a raw Avro file written into HDFS by the custom

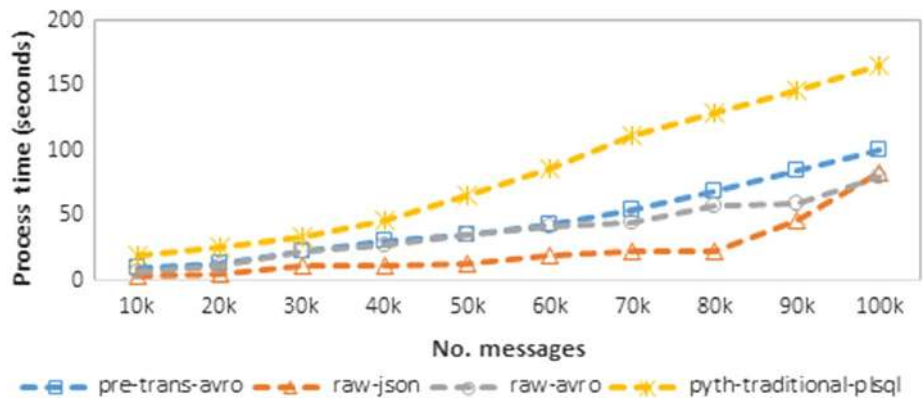

Fig. 4 Data ingestion from message queue to HDFS with and without data transformation

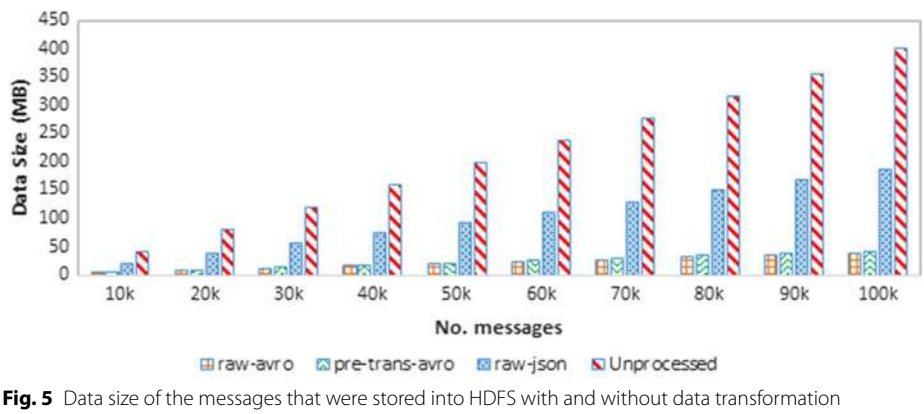

Fig. 5 Data size of the messages that were stored into HDFS with and without data transformation

data pipeline. The Avro files are smaller than the JSON file and contain unprocessed data because they are serialised into binary format. However, the pre-transformed Avro file is larger than the raw Avro file because transformation was applied.

Performance results of intermediate data transformation using a MapReduce job

A test was designed to measure the performance of an intermediate MapReduce transformation done on a centralised Hadoop cluster. As shown in Fig. 6, only the raw JSON data will go through this transformation, as the pre-transformed Avro file has already been transformed at the data pipeline level and the raw Avro data will be transformed at the analytic time when it is required. Also, the data stored in the database by the Python agent does not require an intermediate transformation as it has already been performed at the data pipeline. Transforming the data using an intermediate job is very expensive in terms of execution time, as the process is carried out by chained MapReduce jobs that will transform, aggregate and merge the data. The majority of the execution time overhead was used for finding resources and submitting the chained jobs to the Hadoop cluster.

Performance results of a simple analytic computation with and without data transformation

The final step of the evaluation cycle was to carry out a simple computation on the 100,000 messages dataset and measure the performance. Two sets of analytics jobs were implemented to compute a summary view of the XRootD operations, performed by the different users for each WLCG site belonging to the XRootD federation [20]. An analytics job was modified to include the data transformation prior to the computation. The modified job was executed on the raw Avro data. As shown in Fig. 7, an extra execution time overhead was added to the modified analytics job when compared with unmodified

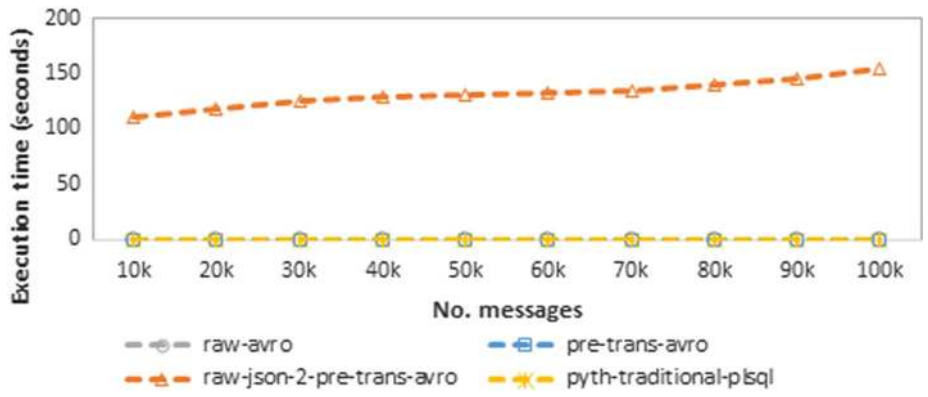

Fig. 6 Intermediate MapReduce job for data transformation. Only the raw JSON messages are transformed with the MapReduce job

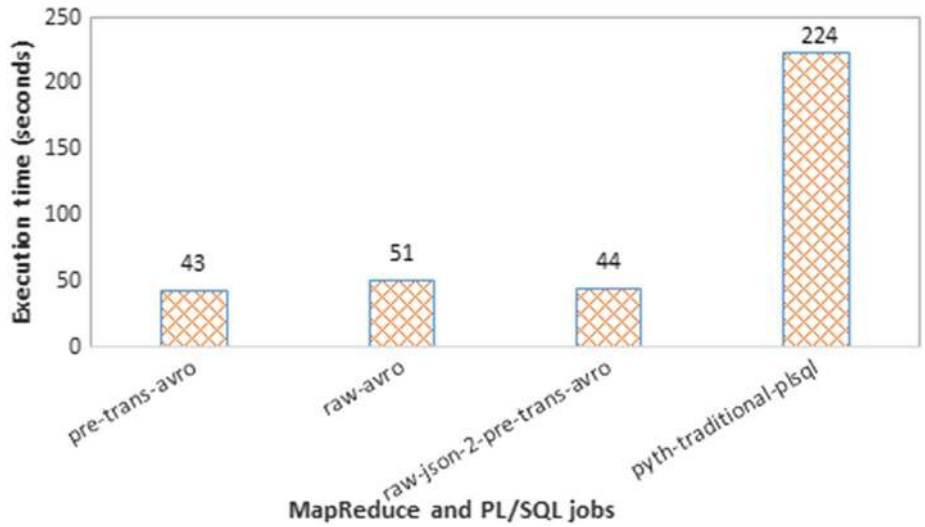

Fig. 7 Performance measurements of the statistic computation were done on pre-transformed and the raw 100,000 messages dataset

job that computed pre-transformed data, but the computation was seamless, as the MapReduce framework adopts a parallel programming model. Therefore, the jobs will be split into multiple tasks and will be sent to data nodes where the data reside. The current approach used by the WLCG (pyth-traditional-plsql) for analytics was very slow compared with the proposed approaches due to the constraints imposed by the database being used and its lack of scalability.

Summary of the performance results

In order to understand which approach performed better, the execution time of the largest dataset of 100,000 messages was selected from "Performance results of data ingestion with and without data transformation", "Performance results of intermediate data transformation using a MapReduce job" and "Performance results of a simple analytic computation with and without data transformation" sections and the total is presented in Table 2. It is clear that writing the raw Avro data into HDFS and letting the analytics do the transformation outperforms the other two proposed approaches. The slowest of the proposed approaches is the second approach where there is an intermediate job for transformation. This is understandable as the transformation is carried out by chained MapReduce jobs, which add extra execution time overhead. The first approach is comparable in terms of performance to the second approach but it will be beneficial to keep

Table 2 Total sum of execution time for 100,000 messages dataset from "Performance results of data ingestion with and without data transformation", "Performance results of intermediate data transformation using a MapReduce job" and "Performance results of a simple analytic computation with and without data transformation" sections

Data transformation	"Performance results of data ingestion with and without data transformation" section execution time (s)	"Performance results of intermediate data transformation using a MapReduce job" section execution time (s)	"Performance results of a simple analytic computation with and without data transformation" section execution time (s)	Total execution time (s)
pre-trans-avro-mr	100	0	43	143
raw-json-2-pre-trans-avro-mr	83	155	44	282
raw-avro-mr	80	0	51	131
pyth-traditional	166	0	224	390

a copy of the untempered raw data file in HDFS and let the analytics job do the transformation, which is better than carrying out transformation in the data pipeline as the authenticity is lost once the transformation is done and stored in HDFS. Although the current approach used by the WLCG employs the same pre-transformation approach, it performs inadequately compared with the new approaches presented in this paper, primarily due to database communication and scalability constraints as the current approach cannot handle the increasing data and workload.

Evaluation of Apache Flume

During the evaluation of all three proposed approaches there was still a 5 min delay in polling data from the message queue. In order to eliminate this polling latency, custom-made Apache Flume data collectors (as explained in "Implementation" section) that utilise an appending mechanism were put in place of the consumer shown in Fig. 3b. The performance test results showed that the third approach is optimal. Therefore, Apache Flume agents were configured to consume messages and flush them into HDFS directly. Figure 8 shows spikes in the total number of messages propagated with a rate >1 kHz, and it can be seen that Apache Flume seamlessly absorbs the load on its single virtual machine. Meanwhile, the current Python-Oracle based consumers used by the WLCG, running on two production virtual machines, were struggling to keep up, causing a backlog of message stored in the broker.

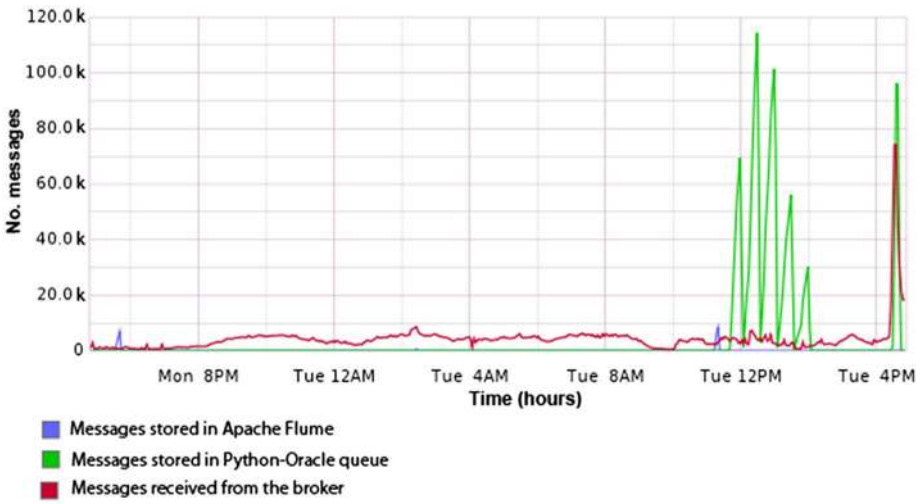

Fig. 8 Spikes of messages with a rate >1 kHz. The *red line* is the messages received from the broker, *green denotes* the messages stored in old consumers, and *blue denotes* the messages stored in Apache Flume

Conclusion

The proposed approaches for collecting and storing Big Data for analytics presented in this paper show how important it is to select the correct model for efficient performance and technology migration. It is clear from the study that keeping the main logic in a centralised location will simplify technological and architectural migration. The performance test results show that eliminating any transformation at the data ingestion level and moving it to the analytics job is beneficial as the overall process time is reduced, untempered raw data are kept in the storage level for fault-tolerance, and the required transformation can be done as and when required using a framework such as MapReduce. The presented results show that this proposed approach outperformed the approach employed at the WLCG and following this work the new approach has been adopted by the WLCG and it has been used for collecting, storing, and analysing metadata at CERN since April 2015 [6]. This approach can be easily applied to other use cases (e.g. in commercial businesses for collecting customer interest datasets) and is not restricted to scientific applications. Future work will include looking at how the data pipeline in the new approach will perform if the MapReduce framework were to be replaced by the Spark ecosystem which supports in-memory processing [21].

Authors' contributions

US is the primary researcher for this study. His contributions include the original idea, literature review, implementation and initial drafting of the article. LM guided the initial research concept and played a crucial role in the design of the analytics approaches presented. DRS discussed the results with the primary author to aid writing of the evaluation and conclusion sections and played an essential role in editing the paper. AK and JA helped organise the structure of the manuscript and edit the article. All authors read and approved the final manuscript.

Author details

[1] College of Engineering, Design and Physical Sciences, Brunel University London, Uxbridge, Middlesex UB8 3PH, UK.
[2] European Organisation for Nuclear Research, CERN, Geneva, Switzerland.

Acknowledgements

The work by Uthayanath Suthakar was supported by a Brunel University London College of Engineering, Design and Physical Sciences Thomas Gerald Gray postgraduate research scholarship.

Competing interests

The authors declare that they have no competing interests.

Received: 2 August 2016 Accepted: 14 October 2016
Published online: 24 October 2016

References

1. Snijders C, Matzat U, Reips U-D. Big Data: big gaps of knowledge in the field of internet science. Int J Internet Sci. 2012;7(1):1–5.
2. Aamnitchi A, Doraimani S, Garzoglio G. Filecules in high energy physics: characteristics and impact on resource management. In: High performance distributed computing. 2016. p. 69–80.
3. Minoli D. A networking approach to grid computing. Hoboken: Wiley; 2004.
4. Nicholson C, et al. Dynamic data replication in LCG 2008. Concurr Comput Pract Exp. 2008;20(11):1259–71.
5. CERN. LHC physics data taking gets underway at new record collision energy of 8TeV. http://press.web.cern.ch. Accessed 18 Dec 2015.
6. Magnoni L, Suthakar U, Cordeiro C, Georgiou M, Andreeva J, Khan A, Smith DR. Monitoring WLCG with lambda-architecture: a new scalable data store and analytics platform for monitoring at petabyte scale. J Phys Conf Ser. 2015;664(5):052023.
7. Knobloch J, Robertson L. LHC computing grid: technical design report-LCG-TDR-001. CERN; 2015.
8. WLCG: The worldwide LHC computing grid infrastructure. http://wlcg.web.cern.ch. Accessed 20 Dec 2015.
9. Grim, K. Tier-3 computing centers expand options for physicists, International Science Grid This Week (iSGTW). ISGTW. 2009.
10. Foster I, Kesselman C. The grid 2: blueprint for a new computing infrastructure. San Francisco: Morgan Kaufmann Publishers Inc.; 2013.
11. Foster I, Kesselman C, Tuecke S. The anatomy of the grid: enabling scalable virtual organizations. Int J High Perform Comput Appl. 2011;15(3):200–22.

12. Laure E, Fisher SM, Frohner A, Grandi C, Kunszt PZ, Krenek A, Mulmo O, Pacini F, Prelz F, White J, Barroso M, Buncic P, Hemmer F, Meglio AD, Edlund A. Programming the grid with gLite. Comput Methods Sci Technol. 2006;12(1):33–45.
13. Foster I. What is the grid? A three point checklist, GRIDToday. GRIDToday; 2011.
14. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Commun ACM. 2008;51(1):107–13.
15. Shvachko K, Kuang H, Radia S, Chansler R. The Hadoop distributed file system. IEEE 26th symposium on mass storage systems and technologies (MSST); 2010. p. 1–10.
16. Ghemawat S, Gobioff H, Leung ST. The google file system. ACM SIGOPS Oper Syst Rev. 2003;375:29–43.
17. Apache Flume project. https://flume.apache.org. Accessed 02 Jan 2016.
18. Skaburska K. The Dirq project. http://dirq.readthedocs.org. Accessed 27 Dec 2015.
19. White T. Hadoop: the definitive guide. O'Really Media. Sunnyvale: Yahoo Press; 2010.
20. Gardner R, Campana S, Duckeck G, Elmsheuser J, Hanushevsky A, Honig FG, Iven J, Legger F, Vukotic I, Yang W. The Atlas collaboration: data federation strategies for ATLAS using XRootD. J Phys Conf Ser. 2014;513(4):042049.
21. Zaharia, M et al. In: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. NSDI'12. Berkeley: Proceedings of the 9th USENIX conference on networked systems design and implementation-USENIX association; 2012. p. 2.