WLCG Analytics Platform: XRootD Monitoring Use Case Results using a Hadoop and MapReduce Framework

J Andreeva,[1] A Khan,[2] L Magnoni,[1] D R Smith[2] and U Suthakar[2]

[1] CERN, European Organisation for Nuclear Research, Switzerland

[2] Brunel University London, College of Engineering, Design and Physical Sciences, UK

ABSTRACT

The Worldwide LHC Computing Grid (WLCG) serves over 150 computer centres and 8,000 physicists who executes over 2 million jobs daily, resulting in the movement of petabytes of data over the heterogeneous infrastructure [1]. It is becoming troublesome to monitor the infrastructure as the volume and velocity of the data is expected to increase when the second period of data taking starts in 2015 [2]. The existing monitoring infrastructure uses Oracle RDBMS for storing and analysing logged data, however, this has clear limitations with scalability [1]. Therefore, a new analytic platform is required so it was decided to investigate the Lambda approach, which is a generic, scalable and fault-tolerant data processing architecture [3]. This paper will evaluate the batch layer from the Lambda architecture, and prototype the layer with a well-known use case XRootD monitoring system using Hadoop and its MapReduce paradigm for processing input data in Comma-Separated Value (CSV), that stores comma separated data into plain text file and Avro format, which is a data serialisation framework that serialises the data into a compact binary format, so that it can be transferred efficiently across the network. Finally, the execution time of both jobs will be evaluated. This investigation is purely in order to evaluate the Hadoop based batch layer for the Experiment Dashboard.

1. INTRODUCTION

The Worldwide LHC Computing Grid (WLCG) provides the computing resources to store, distribute and analyse the ~25 petabytes of data annually generated by the Large Hadron Collider (LHC), serving a community of more than 8,000 physicists distributed among 150 computing centres around the world [1].

Daily, thousands of files are transferred and hundreds of thousands of processing jobs are executed on the distributed infrastructure in order to perform scientific analysis of LHC data [1]. Monitoring such a distributed, geographically sparse and data intensive infrastructure is a core functionality and one of the greatest challenges to provide a reliable scientific platform. The development and the operation of the WLCG monitoring infrastructure is the responsibility of the Support for Distributed Computing (SDC) group, within the CERN IT department.

The current monitoring architecture uses a relational database system to store, process and serve monitoring events; however, this has clear limitations, especially when the next data collection period is due to start in 2015, which is expected to produce a substantial increase in the amount of data [2]. Servers can easily transfer logs at 1 kHz so the database cannot keep up with the load hence scalability is difficult to achieve with the current architecture. Moreover, effective monitoring requires low-latency read access to real-time data, while built-in database procedures impose constraints on the format, granularity and timeliness of monitoring information. Therefore, it was decided to adopt the Lambda approach to create an analytic platform that would cope with the high volume, velocity and variety of data. The architecture consists of three layers; the batch layer, to store a constantly growing dataset providing the ability to compute arbitrary functions on it, the second layer is the serving layer, to store the batch processed views, the third layer offers real-time processing and is able to perform analysis of fresh data with incremental algorithms to compensate for batch-processing latency [3].

In this paper the batch layer will be evaluated, which was prototyped using Hadoop and its MapReduce framework in order to analyse a well-known use case: XRootD Transfer Monitoring System.

The XRootD protocol extends flexible combinations of topologies, which is efficient for wide area network direct file access and client-side caching [2]. All four LHC experiments, ATLAS, CMS, ALICE and LHCb, utilise the XRootD protocol for data transfers and data access [2, 4, 5].

In this paper, the monitoring of two different storage systems that use the XRootD protocol for data transfers will be assessed:

1. Federated storage system using XRootD, which clusters Tier-1, Tier-2 and Tier-3 storage resources together into a common namespace and allows remote accessibility from geographically separated sites [2]. The XRootD based federation storage systems will be referred to as FED from now.
2. The EOS Storage Service, which is a disk only storage system without RAID array. It is a plug-in based architecture that allows remote file access through the XRootD protocol [5]. Raw data are sent from Tier-0 to CERN Advance Storage Manager (CASTOR), a storage management system for handling disk and tape tiers, and then copied into EOS for analysis activities [6].

The current XRootD monitoring data flow architecture as shown in Fig.1, heavily relies on Oracle. Each data transfer and file access is logged at the storage system; logs are then collected by the dedicated collectors and then propagated into message queues [7]. There is a consumer at the other side of the message queue that reads those messages and persistently stores them in a devoted raw table. There is a PL/SQL procedure that aggregates raw data into statistics with different time period granularities and stores them into a statistics table in order to serve the Experiment Dashboard [7].

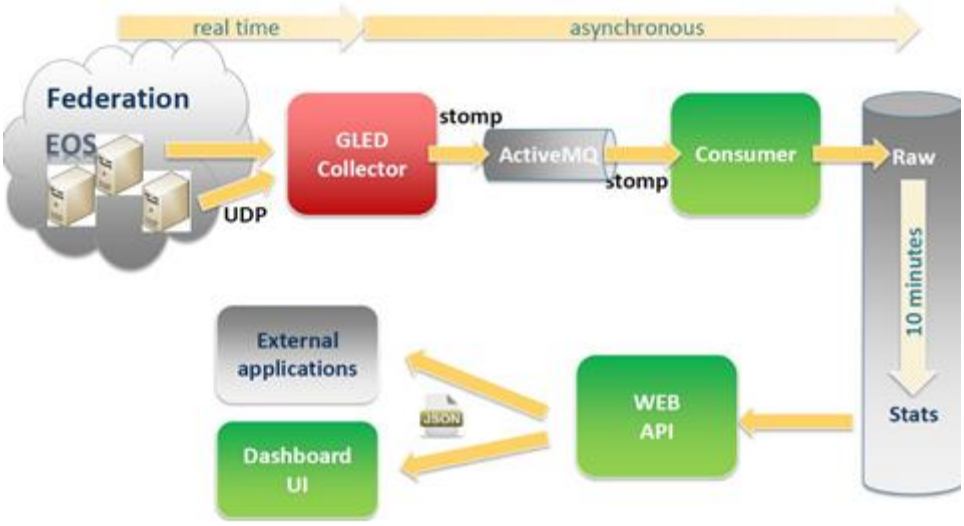

Figure 1. XRootD monitoring data flow [7].

The current monitoring system has proven to be a solid and reliable solution to support WLCG functions and operations during the LHC data-taking years. However, it is no longer efficient in terms of the current and expected trend of dataflow [2]. This is due to the increase in velocity and volume of the data that is being monitored and logged. Storage servers can easily transfer logs at 1 kHz, with such a rate the PL/SQL procedure cannot cope with the overwhelming amount of data, which takes over ten minutes to process every ten minutes worth of data. The latency of the current system is limited by the data input rate and scalability issues.

It was decided to explore the Hadoop system and its MapReduce paradigm for data analytics. MapReduce is a programming model that was designed to remove the complexity of processing data that are geographically scattered around a distributed infrastructure [8]. It hides the complexity of computing in parallel, load balancing and fault tolerance over a large range of inter-connected machines from developers [8]. There are two simple parallel methods; map and reduce are predefined in the MapReduce programming model and are user-specified methods that are used to develop the analytics platform. The MapReduce framework is built on the Hadoop Distributed File System (HDFS) and executes I/O operations on it [9]. The HDFS guarantees: scalability on commodity hardware, fault tolerance, high throughput, load balance, data integrity and portability [9].

2. DESIGN/METHODOLOGY/APPROACH

There were a few things that had to be considered before prototyping:

1. Firstly, how are the data going to be pipelined from message queues to HDFS?
2. How are the data going to be partitioned and stored in HDFS so that they are ready for processing by MapReduce?
3. How and where will the processed statistics data be stored and served?

It was decided that the data pipeline will be discarded or minimised as per the first phase as shown in Fig.2. Instead, the decision was made to export data from the Oracle raw table to HDFS using a tool called Sqoop, a tool for efficiently transferring

data between structured datastores and HDFS [10]. This job will be scheduled to transfer daily data at the end of each transaction day.

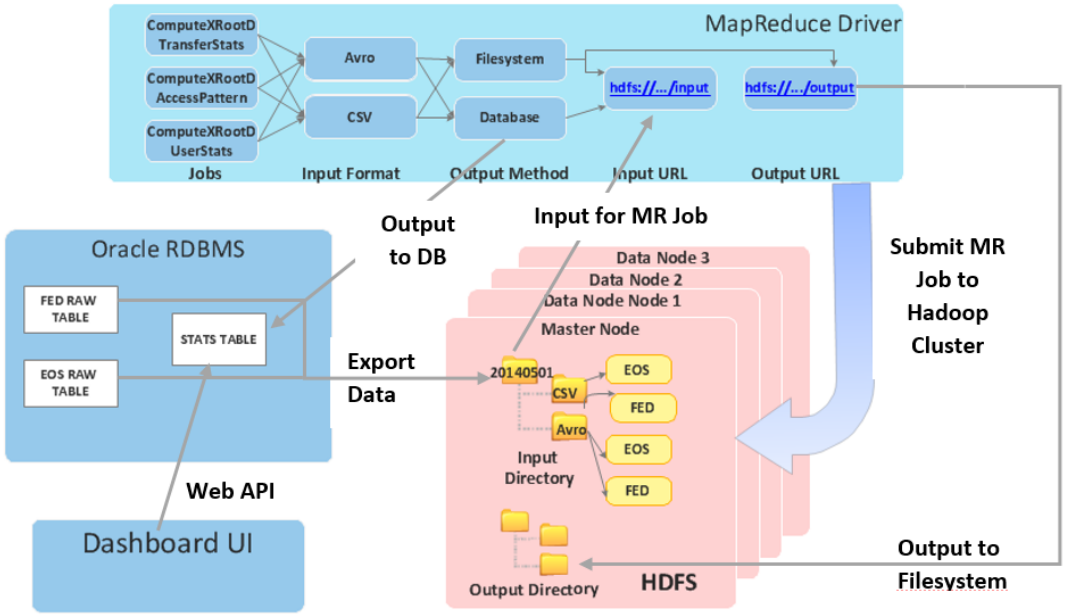

Figure 2. Phase 1 - Overview of Hadoop based XRootD Transfer Architecture.

For efficient data processing the data will be partitioned by date as shown in Fig.3, as this method will support processing data by date ranges. There will be a separate folder for each date; within this folder there will be two discrete folders consecutively assigned for storing EOS and FED data. This method will enable both datasets to be processed together or individually, which is lacking in the current system as there are separate PL/SQL procedures for processing each dataset. The computed statistics will have the option to be stored in HDFS or the Oracle table.

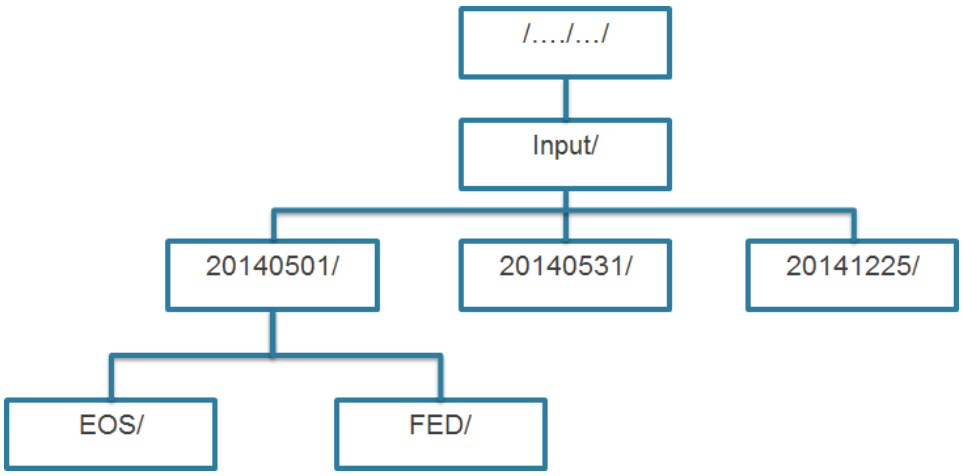

Figure 3. Data structure by date.

In order to measure the performance of log analytics, two different types of MapReduce jobs were implemented. One of these reads a comma-separated CSV file, computes and produces a CSV output. The other job will read the file in Avro format, de-serialise, compute, serialise and then writes the output back to HDFS in Avro format [11].

The CSV-MapReduce job consists of two chained MapReduce jobs; the first job will read, compute, aggregate and write the output to HDFS, whereas the second job will read the output of the first job and sort the data. The Avro-MapReduce job consists of only a single MapRecude job as Avro has inbuilt sorting mechanisms; the sorting parameter could be set with the corresponding column in schema, therefore a single job is sufficient to carry out the task [11].

The testing infrastructure consists of four nodes as shown in Fig.4; a master node is responsible for sending the job to where the data resides with the help of Yet Another Resource Negotiator (YARN). It also holds the metadata of the data blocks that are replicated and stored in data nodes. It also performs the role of data node to store data and to execute jobs. The other three nodes store data and these nodes are also referred to as worker nodes as they are responsible for computing [9]. One of these three nodes is also configured to act as the secondary master node, in case of failure of the master node, when it will take over its duty.

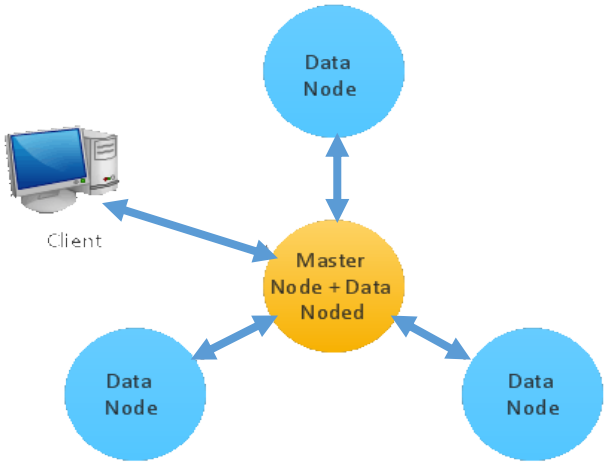

Figure 4. IT/DBA Hadoop cluster.

All four nodes share the same specifications as shown in Table 1. The cluster consists of small but very powerful physical machines. The cluster was not busy when the testing was carried out.

Table 1.Cluster specifications.

Caption	Capacity
Model	Intel(R) Xeon(R) CPU L5520 @ 2.27GHz
CPU MHz	1596
CPU cores	4
Memory	25GB
Diskspace	500GB

x4

The test was done on one month (May 2014) of historical raw data of both EOS and FED data type. The full EOS dataset was ~58GB, while the FED dataset was ~11.8GB. These data were partitioned by date and stored as CSV format into HDFS. However, Sqoop does not support the Avro format [10]. Therefore an Avro schema was created and a MapReduce job was implemented to take the CSV file as an input, and serialise it into Avro format using the schema and output the data into HDFS.

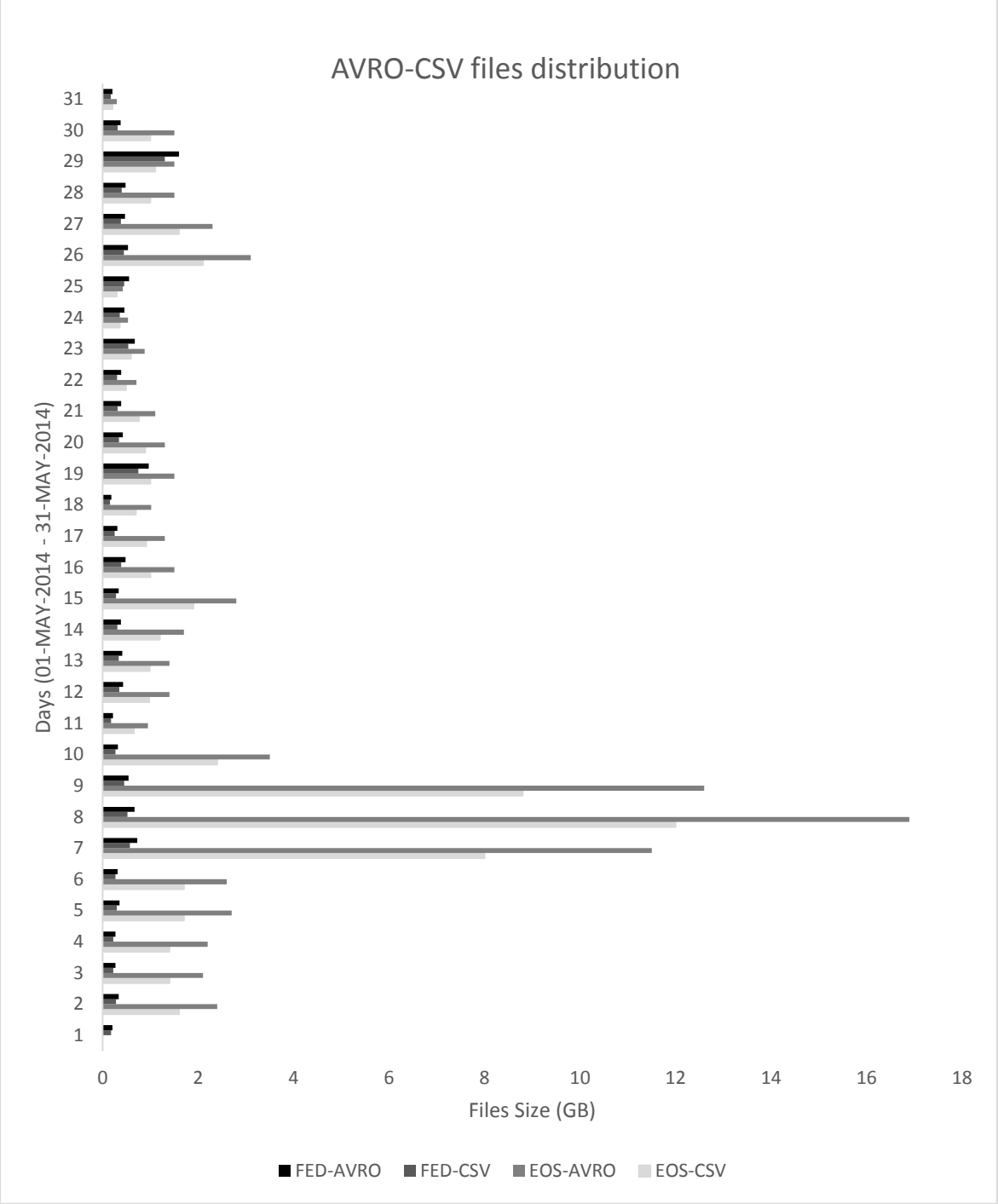

Figure 5. AVRO-CSV files distribution.

As shown in Fig.5, the EOS logs dataset is huge compared with the FED logs dataset as it deals with transferring a large amount of raw data from/to Tier-0 [6]. On the other hand, when file formats were scrutinised it can be seen that Avro files are very much larger when compared with CSV files. This is due to the fact that the Avro file contains data as well as the schema coupled to it. For example, each record will have the column's name attached as shown in Fig.6, whereas the CSV file only

contains data that are comma separated as shown in Fig.7. This has resulted in a 45.13% increase in volume for EOS-AVRO data compared with EOS-CSV and a 23.26% increase in volume for FED-AVRO data compared with FED-CSV data.


```
{
  "unique_id":"d88ac72c-c666-11e3-9717-57c84f86beef-8dddbc",
  "file_lfn":"/dpm/ph.liv.ac.uk/home/atlas/atlasproddisk/rucio/data12_8TeV/eb/5b/DESD_TAUMUH.01464491._007108.pool.root.1",
  "file_size":20269934,
  "client_domain":"ph.liv.ac.uk",
  "client_host":"192.168.22.11",
  "server_domain":"ph.liv.ac.uk",
  "server_host":"hepraid9",
  "read_bytes_at_close":20269934,
  "read_bytes":20269934,
  "read_operations":4,
  "read_average":5067483.5,
  "read_min":3492718,
  "read_max":8388608,
  "read_sigma":1938726.334667,
  "read_single_bytes":20269934,
  "read_single_operations":4,
  "read_single_average":5067483.5,
  "read_single_min":3492718,
  "read_single_max":8388608,
  "read_single_sigma":1938726.334667,
  "read_vector_bytes":"0",
  "read_vector_operations":"0",
  "read_vector_average":"0",
  "read_vector_min":"0",
  "read_vector_max":"0",
  "read_vector_sigma":"0",
  "read_vector_count_average":"0",
  "read_vector_count_min":"0",
  "read_vector_count_max":"0",
  "read_vector_count_sigma":"0",
  "write_bytes_at_close":"0",
  "write_bytes":"0",
  "write_operations":"0",
  "write_min":"0",
  "write_max":"0",
  "write_average":"0",
  "write_sigma":"0",
  "server_username":"",
  "user_dn":"/DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=atlpilo2/CN=531497/CN=Robot: ATLAS Pilot2",
  "user_fqan":"",
  "user_role":"",
  "user_vo":"atlas",
  "app_info":"",
  "start_time":1399075196,
  "end_time":1399075200,
  "start_date":"02-MAY-14",
  "end_date":"03-MAY-14",
  "insert_date":"02-MAY-14",
  "server_site":"UKI-NORTHGRID-L",
  "user_protocol":"xrootd"
 }
```


Figure 6. Data in JSON format after de-serialising the Avro file.

d88ac72c-c666-11e3-9717-57c84f86beef-8dddba,/dpm/ph.liv.ac.uk/home/atlas/atlasproddisk/rucio/data12_8TeV/ee/9a/DESD_TAUMUH.01464491._008347.pool.root.1,17374961,ph.liv.ac.uk,192.168.23.13,ph.liv.ac.uk,hepraid15,17374961,17374961,4,4343740.25,597745,8388608,2758536.655259,17374961,4,4343740.25,597745,8388608,2758536.655259,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,,/DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=atlpilo2/CN=531497/CN=Robot: ATLAS Pilot2,,,atlas,,1399075196,1399075200,02-MAY-14,03-MAY-14,02-MAY-14,UKI-NORTHGRID-L,us,xrootd

Figure 7. Data in CSV format.

3. FINDINGS/RESULTS

In order to evaluate the execution time, it was decided to compute the dataset by days, ranging from: 1 day, 3 days, 7 days and 21 days. Jobs were submitted a few times for each range in order to capture an average performance time. In Fig.8a, the comparison between the execution time and data input size of Avro jobs and CSV jobs can be seen. The Avro jobs are much faster than the CSV jobs, although Avro input data size is larger than with CSV as shown in Fig.8b. In Fig.8c, the data process rate in Kilobyte per second can be seen. It can be observed that an average of 67.55% increase in processing Avro data over CSV. CSV jobs are slower than Avro jobs because they add an extra overhead for parsing data into appropriate data types for computations, whereas Avro is not required to parse data due to the fact that data are accompanied with the schema. It should be noted that input data conversion from CSV to Avro format was not recorded within the execution time, even though this task requires an extra resource.

It was expected that writing the aggregated statistics data by the Avro job into HDFS would add an additional overhead to the disk I/O, therefore the execution time will be increased because the job will have to write the schema with it. However, that was not the case as it did not make any difference in execution time whether the output was unstructured, semi-structured or structured. At one point, writing to HDFS was disabled to monitor the performance but there was no different in the execution time. Also, the input data were compressed to observe the effect on execution time, which increased slightly, although the input data size were greatly reduced. This was expected as Hadoop is not efficient at processing small files, which increase the data block seeking time at disk level. Compression may work well if the data size is very large; however, the monitoring data that are propagated are not that large.

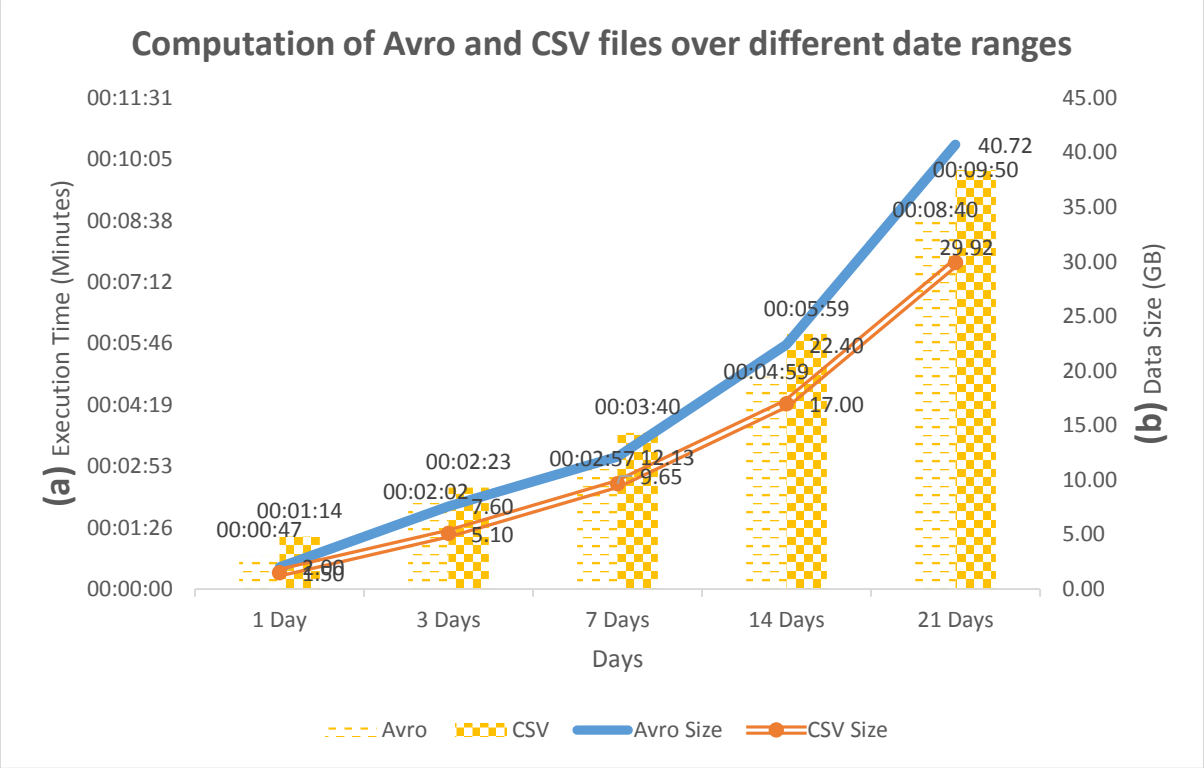

Figure 8a. Computation of Avro and CSV files over different date ranges. The primary axis (a) shows the execution time that is being represented by bars, whereas the secondary axis (b) represents the input data size in Gigabyte (GB) which is being represented by lines.

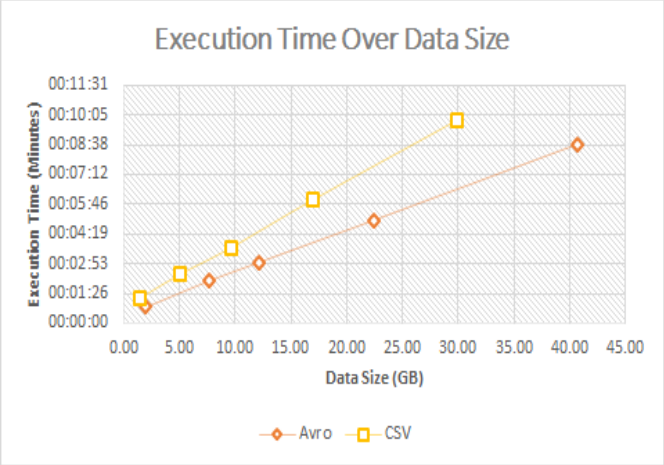

Figure 8b. Execution Time over Data Size. It represents the execution time over both Avro and CSV file data size.

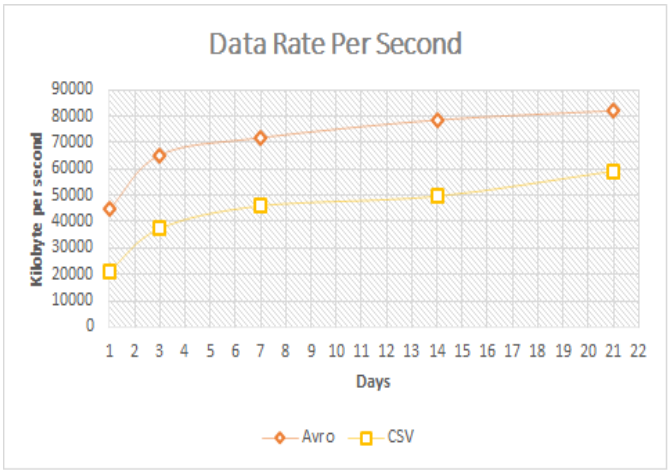

Figure 8c. Data Rate per Second. It represents the data process rate in Kilobyte per second over different date ranges for Avro and CSV files.

Even though the job is split into multiple map tasks and sent to data nodes where the data reside in order to reduce the movement of large data files over the network, the intermediate results of these tasks still need to be shuffled and shifted around to

reducers (most likely to different nodes) [8]. This could create a bottleneck in the network, so in order to minimise this issue, it was decided to compress the intermediate results before transferring them to the reducer nodes. When this approach was used the performance was improved by ~2 seconds but the assumption is that this would have worked better if the cluster was larger and the data more geographically distributed across the network.

4. CONCLUSION

Taking into account the result presented of Hadoop and its MapReduce framework prototype, it is an optimal fit for the XRootD monitoring computation. The performance results are more than acceptable, even with non-optimal configuration compared with the Oracle PL/SQL script. It is also clear that the Avro job appears to be performing better when compared with CSV. The trade-off with Avro is that it takes a somewhat larger storage space but it is something that could be compromised for better performance, structured data and platform independency. A big dataset is not an issue as Hadoop supports commodity hardware and it scales horizontally, therefore the storage issue can be quickly resolved by plugging a node into a cluster that not only increases the disk space but also the performance as the tasks will be parallelised and distributed evenly [9]. Even if storage is an issue then the data could be compressed with a minor degradation to performance as experienced at the testing stage.

5. FUTURE PLAN/DIRECTIONS

The prototype architecture presented is still undergoing further development to meet the goal, which is to design and implement an architecture that would support real-time processing on fresh data as they are consumed and synchronised with the batch processing for computing on historical data. The next phase of the project will be to consume data from the message queue directly and write it into HDFS as shown in Fig.9. Apache Flume will be investigated for consuming the stream of logs from message queues as it is designed for distributed and fault tolerant service for collecting a large amount of data and moving them to HDFS [12]. In this phase it is

also expected to introduce real-time processing using Esper, an event driven real-time processing technology that also supports SQL like commands [13].

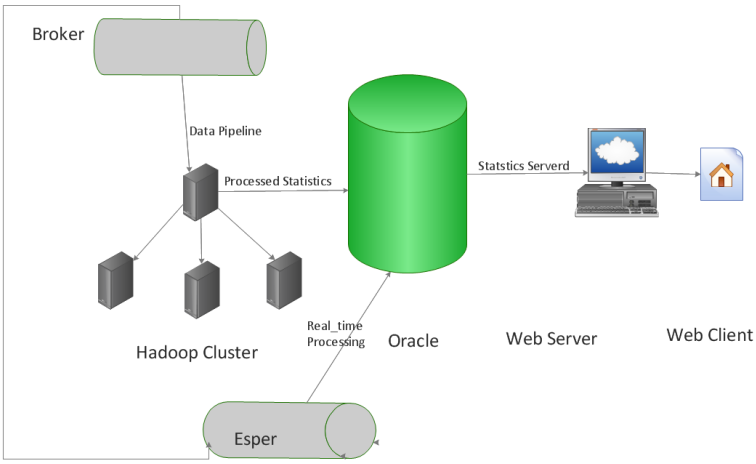

Figure 9. Phase 2 – introducing real-time analytics.

Thereafter the serving layer will be introduced and the Oracle backed database will be decommissioned from the architecture. Elasticsearch will be evaluated for serving the batch and real-time processed statistics as shown in Fig.10.

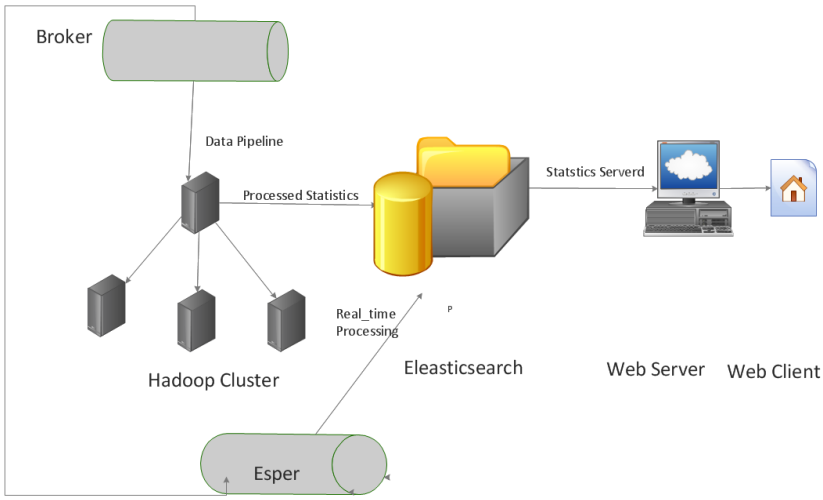

Figure 10. Phase 3 – introducing the serving layer.

ACKNOWLEDGEMENTS

The authors would like to thank the Database Administrators (DBA) at CERN for providing a small but powerful cluster for this investigation and also for the mutual interest in collaboration for studying different ways to export data from Oracle to Hadoop.

References

1. Andreeva J, Beche A, Belov S, Kadochnikov I, Saiz P, Tuckett D. WLCG transfers dashboard: A unified monitoring tool for heterogeneous data transfers. International Conference on Computing in High Energy and Nuclear Physics. ; 2013.

2. Gardner R, Campana S, Duckeck G, Elmsheuser J, Hanushevsky A, Hönig FG, et al. Data federation strategies for ATLAS using XRootD. International Conference on Computing in High Energy and Nuclear Physics. ; 2013.

3. Marz NaW,James. Big data: Principles and best practices of scalable realtime data systems. ; 2014.

4. Andreeva J, Beche A, Belov S, Arias D,D Giordano, Oleynik D, Petrosyan A, et al. Monitoring of large-scale federated data storage:XRootD and beyond. International Conference on Computing in High Energy and Nuclear Physics. ; 2013.

5. Peters A, Sindrilaru Alin E, Zigann P. Evaluation of software based redundancy algorithms for the EOS storage system at CERN. International Conference on Computing in High Energy and Nuclear Physics 2012. ; 2013.

6. Espinal X, Adde G, Chan B, Iven J, Lo Presti G, Lamanna M, Mascetti L, Pace A, Peters A, Ponce S, Sindrilaru E. Disk storage at CERN: Handling LHC data and beyond. International Conference on Computing in High Energy and Nuclear Physics. ; 2013.

7. WLCG data transfer monitoring [Internet, Last Accessed August 2014]. Available from: https://twiki.cern.ch/twiki/bin/view/LCG/WLCGDataTransferMonitoring.

8. Mackey G, Sehrish S, Bent J, Lopez J, Habib S, Jun Wang. Introducing map-reduce to high end computing. Petascale data storage workshop, 2008. PDSW '08. 3rd; ; 2008.

9. Attebury G, Baranovski A, Bloom K, Bockelman B, Kcira D, Letts J, et al. Hadoop distributed file system for the grid. Nuclear science symposium conference record (NSS/MIC), 2009 IEEE; ; 2009.

10. Sqoop documentation [Internet, Last Accessed June 2014]. Available from: http://sqoop.apache.org/docs/.

11. Avro documentation [Internet, Last Accessed July 2014]. Available from: http://avro.apache.org/docs/.

12. Apache flume documentation [Internet, Last Accessed August 2014]. Available from: https://flume.apache.org/documentation.html.

13. Esper - complex event processing [Internet, Last Accessed May 2014]. Available from: http://esper.codehaus.org/esper/documentation/documentation.html.