EGI-Engage

Analysis on Techniques to Manage Big Data on the EGI Accounting System

D3.6

Date	31 March 2016
Activity	WP3
Lead Partner	STFC
Document Status	FINAL
Document Link	https://documents.egi.eu/document/2667

Abstract

This deliverable presents an analysis on techniques to manage big data on the EGI accounting system. The EGI accounting system receives accounting data from sites providing compute, cloud and storage services to EGI. The data is aggregated and the totals across a number of aggregation parameters are sent to an accounting portal for visualisation. The central processing stage for the CPU accounting data takes many hours to complete and operates in a single processor thread. Recent advances in big data tools provide an opportunity to address these limitations: improving the performance and resilience of the central repository. Additionally the Accounting Repository should evolve to support new types of data and new communities that will make use of the EGI infrastructure over the coming years. In this document we review the available technologies, discuss how they can be applied to the accounting service and propose a methodology for testing and comparing them.

COPYRIGHT NOTICE

DELIVERY SLIP

	Name	*Partner/Activity*	*Date*
From:	S. Pullinger, A. Coveney	STFC/JRA1	2016-02-10
Moderated by:	Małgorzata Krakowian	NA1/EGI.eu	
Reviewed by	E. Yen G. Sipos C. Kanellopoulos	TW ASGC/SA2 EGI.eu/SA2 GRNET/JRA1	2016-02-22 2016-02-12 2016-02-16
Approved by:	AMB and PMB		2016-03-04

DOCUMENT LOG

Issue	*Date*	*Comment*	*Author/Partner*
v1	2016-01-22	First version of document	S. Pullinger / STFC A. Coveney / STFC
v2	2016-02-08	Revisions following internal review	A. Coveney / STFC S. Pullinger / STFC
v3	2016-02-25	Revisions following external review	A. Coveney / STFC
FINAL	2016-03-03	Final version after external and PMB review	A. Coveney / STFC S. Pullinger / STFC

TERMINOLOGY

A complete project glossary is provided at the following page: http://www.egi.eu/about/glossary/

Contents

Executive summary

This deliverable presents an analysis on techniques to manage big data on the EGI accounting system. The EGI accounting system receives accounting data from sites providing compute, cloud and storage services to EGI. The data is aggregated and the totals across a number of aggregation parameters are sent to an accounting portal for visualisation. The central processing stage for the CPU accounting data takes many hours to complete and operates in a single processor thread. Recent advances in big data tools provide an opportunity to address these limitations: improving the performance and resilience of the central repository. Additionally the Accounting Repository should evolve to support new types of data and new communities that will make use of the EGI infrastructure over the coming years.

There are a number of technologies that could be used to manage big data in the Accounting Repository. Two options would involve optimising the use of the database backend that it currently uses, either by using a performance oriented version of MySQL, or by parallelising the processing stage. Beyond this, there are a large number of tools that make use of Apache Hadoop and the Hadoop Distributed File System. This would require more radical changes to the way the Accounting Repository operates, but could bring much increased performance and future proofing.

A number of metrics should be used to evaluate the technologies within the context of the EGI accounting system. These include the time it takes to summarise and transfer the data, the amount of storage space required, the infrastructure it requires to operate, the ease of use and installation, and whether it has interfaces similar to those already in use and if can be controlled using similar languages.

There are a number of resources available to the APEL project within STFC that can help enable the testing and evaluation of the different big data technologies. The first is a self-service cloud, which provides an internal IaaS cloud resource for STFC users. The second is a small Hadoop cluster of ten nodes that has been used for evaluating cloud storage and compute, and for some development work.

Most of the technologies introduced in this report meet the minimum requirements needed for integrating into the Accounting Repository. It is intended to test six different configurations that make use of these technologies using the resources available and evaluating them against the relevant metrics.

1 Introduction

The EGI Accounting Service receives accounting data from sites providing compute, cloud and storage services to the EGI Federation. Sites send the data to an external message bus. The data is downloaded at a central repository and stored in a MySQL database. The data is aggregated and the totals across a number of aggregation parameters are sent to an accounting portal for visualisation.

The central processing stage for the CPU accounting data takes many hours to complete and operates in a single processor thread. Although the data is backed up regularly, the data is not distributed across multiple hosts to provide greater resilience and processing power. Recent advances in Big Data tools provide an opportunity to address these limitations: improving the performance and resilience of the central repository.

Additionally the Accounting Repository should evolve to support new types of data and new communities that will make use of the EGI infrastructure over the coming years such as the Research Infrastructures currently involved in the EGI-Engage Competence Centres[1].

In this document, we review the available technologies, discuss how they can be applied to the accounting service and propose a methodology for testing and comparing them. A future addendum will address the results of these tests and comparisons.

The outline of this document is as follows: first we provide a short introduction to the EGI accounting service and the limitations of its current implementation. Then there is an overview of a variety of big data tools that may be useful in addressing these limitations. In the next section, metrics that can be used to evaluate the different tools are discussed. A review of the different tools and how suitable they might be follows. Lastly, the resources available for testing are shown and a proposal is made for different configurations of the accounting repository that should be tested.

1.1 Current APEL architecture

Figure 1 shows how the APEL client, central Accounting Repository (APEL server) and EGI Accounting Portals interact. The flow of accounting data goes through the following stages:

1. APEL clients can run an APEL parser to extract data from a batch system and place it in their client database, or they can use third-party tools to extract batch or cloud data. This data is then unloaded into a message format suitable for transmission.

[1] https://wiki.egi.eu/wiki/EGI_Distributed_Competence_Centre

2. APEL clients run a sending Secure Stomp Messenger[2] (SSM) to send these messages containing records via the EGI Message Brokers the central APEL server. The messages can contain either Job Records or Summary records. This is configurable in the APEL client.
3. The central APEL server runs an instance of the SSM, which receives these messages and a "loader" processes the records in the messages and loads them into a MySQL database.
4. A "summariser" process runs to create summaries of any Job Records received and load them in a "SuperSummaries" table along with any Summary records. This summariser runs as a cron job approximately once a day.
5. A database "unloader" process unloads the summary records into the message format to be sent on by the sending SSM via the EGI Message Brokers to the EGI Accounting Portal.

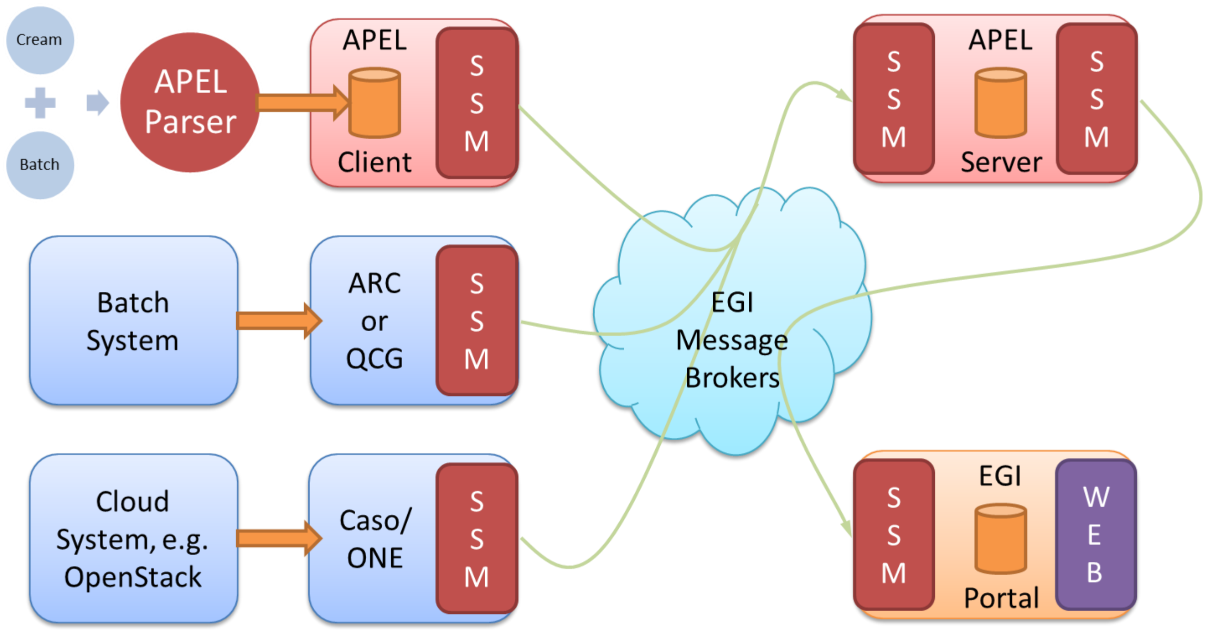

Figure 1 - APEL components and their interactions. Components in red are provided by the APEL project.

The database for the central Accounting Repository currently contains around 750 million records and is over 500 gigabytes in size. The repository receives approximately three million records every day and these can be single batch job records, or aggregated summary records.

1.2 Motivations for changing the APEL technical architecture

As mentioned earlier in this section, the accounting system pulls sites' data from a message broker and stores it in a central repository. The data is aggregated over a number of fields to create

[2] https://github.com/apel/ssm

summaries. These summary totals are then sent on to the accounting portal for display to the users.

The summarising process runs as a query against a large MySQL database of around 500 gigabytes. The process runs in a single thread and takes 8 to 10 hours to complete during which time no data is loaded into the database. It is therefore only practical to run this once per day.

One motivator for investigating alternative tools is to reduce this latency in the system so that summaries arrive at the portal with a shorter delay. Another motivator is the possibility to use multiple cores for the processing thus making better use of the hardware.

Finally, the accounting team have several new types of accounting in development (storage, data-sets and GPGPUs) which, should they all go into production, would increase the volume of data sent to the central accounting repository. Combined with this is the expected increase in volume of accounting data as more researchers make use of the high performance computing resources as a result of the engagement work that EGI-Engage is doing, particularly with the Competence Centres (CCs) and the related research infrastructures (ELIXIR; EPOS, DARIAH, EISCAT-3D, BBMRI, Lifewatch). Therefore, the accounting service will expect to receive greater and greater volumes of data for processing and needs to be prepared to handle larger amounts of data.

2 Technologies to manage big data

2.1 MySQL optimisations

2.1.1 Parallelised processing

A technology that could be applied to the Accounting Repository with little infrastructure change is parallelisation. Changes to the APEL software could be made to support running the summarising process across a number of parallel threads using separate connections to the existing MySQL database backend.

2.1.2 Percona Server

Percona Server[3] is a fork of the MySQL relational database management system created by Percona. It aims to retain close compatibility to the official MySQL releases, while focusing on the performance of operations.

It is a drop-in replacement for MySQL, designed to work with applications that would be too demanding for MySQL itself to support. Percona freely includes a number of scalability, availability, security, and backup features that are usually only available in MySQL's commercial Enterprise Edition.

2.2 InfluxDB

InfluxDB[4] is an open source distributed time series database with no external dependencies. Its traditional use case is recording metrics, events, and performing analytics, such as continuous sensor data, with readings in the order of 10 a second. It is most efficient when handling an insert/append workload, with very few updates[5] and aims to answer aggregation queries in real-time[6].

2.3 Apache Hadoop

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage[7]. The two main parts of

[3] https://www.percona.com/software/mysql-database/percona-server
[4] https://influxdata.com/time-series-platform/influxdb/
[5] https://docs.influxdata.com/influxdb/v0.9/concepts/storage_engine/
[6] https://github.com/influxdata/influxdb
[7] https://hadoop.apache.org/

Apache Hadoop are the Hadoop Distributed File System (HDFS) for storage and the MapReduce programming model.

The HDFS splits files into large blocks and distributes them across nodes in a cluster. To process data, MapReduce transfers packaged code to nodes in order to process the data on that node, taking advantage of data locality[8].

Hadoop can be downloaded from their website[9].

2.3.1 Hadoop Distributed File System

Many of the tools covered in this report build upon the HDFS. Therefore the technical benefits and limitations of the HDFS must be understood.

The HDFS is a Java-based file system that provides scalable and reliable data storage. It was designed to span large clusters of commodity servers and has demonstrated production scalability of up to 200 PB[10]. The entire size of the APEL database is approximately 0.5 TB, so this should be more than sufficient even allowing for replication and possible expansion of the data once loaded into the HDFS.

The HDFS stores metadata and application data separately. As in other distributed file systems, like PVFS[11], Lustre[12], and Google File System, the HDFS stores metadata on a dedicated server, called the NameNode. Application data are stored on other servers called DataNodes[13]. All servers are fully connected and communicate with each other.

Data on DataNodes are broken down into smaller blocks. These blocks are then distributed and replicated throughout the cluster by using TCP-based protocols. Unlike Lustre and PVFS, the DataNodes in the HDFS do not rely on data protection mechanisms such as RAID to make the data contained durable. Instead, the HDFS relies on the file content being replicated on multiple DataNodes for reliability. The default replication setting is to store two additional copies of each block to increase fault tolerance, but this can be changed globally or per file[14]. This replication has the added advantage that there are more opportunities for locating computation near the needed data[15].

Version 2.7.1 of the HDFS introduced transparent, end-to-end encryption. Once configured, data read from and written to special HDFS directories is transparently encrypted and decrypted without requiring changes to user application code. This encryption is also end-to-end, which

[8] http://www-01.ibm.com/software/data/infosphere/hadoop/hdfs/
[9] https://hadoop.apache.org/releases.html
[10] http://hortonworks.com/hadoop/hdfs/
[11] http://www.pvfs.org/
[12] http://www.lustre.org
[13] http://www.aosabook.org/en/hdfs.html
[14] https://www-01.ibm.com/software/data/infosphere/hadoop/hdfs/
[15] http://www.aosabook.org/en/hdfs.html

means the data can only be encrypted and decrypted by the client. The HDFS never stores or has access to unencrypted data or unencrypted data encryption keys. This satisfies two typical requirements for encryption: at-rest encryption (meaning data on persistent media, such as a disk) as well as in-transit encryption (e.g. when data is travelling over the network[16]).

2.3.2 MapReduce

The MapReduce programming model is composed of Map and Reduce procedures. A Map method performs filtering and sorting; then a Reduce method performs a summary operation. An example MapReduce program could contain a Map method for sorting students by first name into queues, one queue for each name; then a Reduce method for counting the number of students in each queue, yielding name frequencies.

The MapReduce System manages all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance.

2.3.3 Enabling Real-Time MySQL to HDFS Integration

Hadoop Applier reads from the MySQL binary log and inserts data into the HDFS in real time, applying the events as they happen on the MySQL server[17].

The Hadoop Applier uses an API provided by *libhdfs*, a C library to manipulate files in the HDFS which comes precompiled with Hadoop distributions. Databases are mapped as separate directories, with their tables mapped as sub-directories. A tool, such as Scoop[18], may be needed to transfer the existing APEL data[19].

MySQL Applier can be downloaded from the MySQL website; however it comes with a warning that it is “not fit for production” and is “provided solely for testing purposes”[20].

[16] https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/TransparentEncryption.html
[17] http://innovating-technology.blogspot.co.uk/2013/04/mysql-hadoop-applier-part-1.html
[18] http://sqoop.apache.org/
[19] https://www.percona.com/blog/2013/07/11/mysql-and-hadoop/
[20] http://labs.mysql.com/

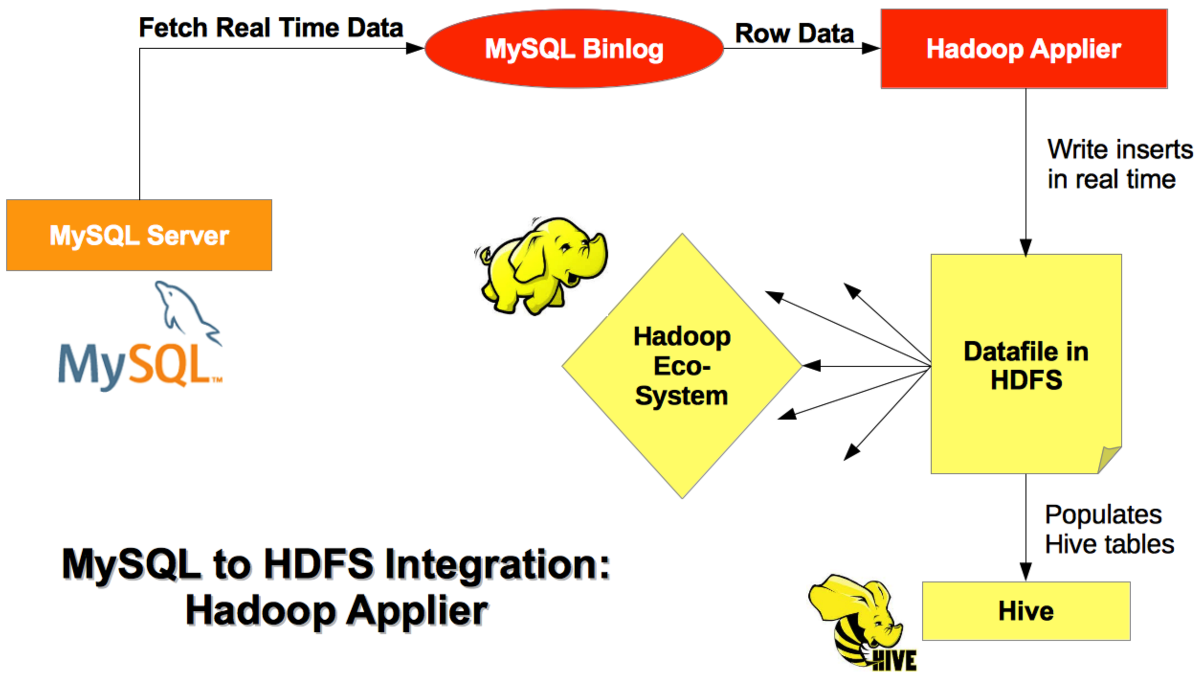

Figure 2 - MySQL to HDFS Integration[21]

2.3.4 Writing a Hadoop MapReduce program

Hadoop MapReduce programs are typically written in Java. However, Apache have released a JAR file that takes non Java programs as inputs and uses them as the Map and Reduce methods. These programs need to take their input from STDIN and put their output to STDOUT. Using this, MapReduce operations using the HDFS can be written in Python[22] or any language capable of reading from STDIN and writing to STDOUT.

2.3.5 Tools built on the HDFS

As well as the MySQL Applier and the ability to write MapReduce programs in any language, the Apache Foundation provide many tools that build on the HDFS, providing additional functionality. Many of these tools aim to conceal the complexity of the MapReduce model, by being “database-like” and providing a “SQL-like” interface to the stored data. A number of these tools are described in the following sections.

[21] https://dev.mysql.com/tech-resources/articles/mysql-hadoop-applier.html
[22] http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/

2.4 Datastores

2.4.1 Hive

Apache Hive is a data warehouse infrastructure built on top of Hadoop for providing data summarisation, query, and analysis[23] of the data stored in the HDFS via an SQL-like interface[24]. It has been designed to perform full-table scans across petabyte-scale data sets.

2.4.2 HBase

Apache HBase is the Hadoop database, a distributed, scalable, big data store providing random, real-time read/write access to data. Apache HBase is an open-source, distributed, versioned, non-relational database modelled after Google's Bigtable[25]. Just as Bigtable uses the distributed data storage provided by the Google File System, Apache HBase provides Bigtable-like capabilities on top of Hadoop and HDFS.

2.4.3 Cassandra

Apache Cassandra is a NoSQL database providing linear scalability and fault-tolerance on commodity hardware. Cassandra supports replicating across multiple datacentres, providing lower latency for users and the peace of mind of knowing that you can survive regional outages.

2.5 Data Transfer Tools

2.5.1 Apache Flume

Apache Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of streaming data into the HDFS. It has a simple and flexible architecture based on streaming data flows; and is robust and fault tolerant with tuneable reliability mechanisms for failover and recovery[26]. It uses a simple extensible data model that allows for online analytic application[27].

2.5.2 Apache Sqoop

Apache Sqoop is a tool designed for efficiently transferring bulk data between Hadoop and structured data stores such as relational databases. Relational databases are examples of structured data sources with well defined-schema for the data they store. Cassandra, Hbase are examples of semi-structured data sources and HDFS is an example of unstructured data source that Sqoop can support.

[23] Venner, Jason (2009). Pro Hadoop. Apress. ISBN 978-1-4302-1942-2
[24] http://hortonworks.com/hadoop-tutorial/how-to-process-data-with-apache-hive/
[25] http://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf
[26] https://kzhendev.wordpress.com/2014/04/06/apache-flume-get-logs-out-of-rabbitmq-and-into-hdfs/
[27] https://flume.apache.org/

2.6 Data flow/stream processing tools

2.6.1 Apache Accumulo

Accumulo is a sparse, distributed, sorted, multi-dimensional map[28] based on Google's BigTable[29] design and is built on top of Apache Hadoop, Zookeeper[30], and Thrift[31]. It is designed to scale to trillions of records and tens of petabytes. Apache Accumulo features a few novel improvements on the BigTable design in the form of cell-based access control and a server-side programming mechanism that can modify key/value pairs at various points in the data management process.

2.6.2 Apache Camel

Apache Camel uses URIs to work directly with any kind of transport or messaging model, such as HTTP, as well as pluggable components and data format options. Apache Camel is a small library with minimal dependencies for easy embedding in any Java application[32].

2.6.3 Apache Samza

Samza is a distributed stream processing framework. It uses Apache Kafka[33] for messaging, and Apache Hadoop to provide fault tolerance, processor isolation, security, and resource management[34]. The processing that Samza enables is often called stream processing. The expected time to get output from a stream process is usually much lower than batch processing, frequently in the sub-second range[35].

2.6.4 Cascading

Cascading is a software abstraction layer for Apache Hadoop. Cascading is used to create and execute complex data processing workflows on a Hadoop cluster using any JVM-based language, hiding the underlying complexity of MapReduce jobs. Cascading also comes with an extension called Lingual. Lingual simplifies application development and integration by providing an ANSI SQL interface for Apache Hadoop. This interface can connect existing SQL codes with Hadoop and accelerate application development with Hadoop[36].

[28] https://www.youtube.com/watch?v=nk4yhqHjxOU
[29] http://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf
[30] http://zookeeper.apache.org/
[31] http://thrift.apache.org/
[32] http://camel.apache.org/
[33] http://kafka.apache.org/
[34] http://samza.apache.org/
[35] http://samza.incubator.apache.org/learn/documentation/0.7.0/container/state-management.html
[36] http://www.cascading.org/projects/lingual/

2.6.5 Apache Storm

Storm reliably processes unbounded streams of data, doing for real-time processing what Hadoop did for batch processing. Storm can be used with many programming languages. It is scalable, fault-tolerant, guarantees data processing, and claims to be easy to set up and operate. Storm uses Thrift, an interface definition language and binary communication protocol[37] that is used to define and create services for numerous languages. Apache Avro also does a similar job, but does not require running a code-generation program when a schema changes[38].

2.6.6 Apache Spark

Spark is built on the concept of *distributed datasets*, which contain arbitrary Java or Python objects. You create a dataset from external data, and then apply parallel operations to it. There are two types of operations: *transformations*, which define a new dataset based on previous ones, and *actions*, which kick off a job to execute on a cluster[39].

2.6.7 Apache Pig

Apache Pig[40] is a high-level platform for creating MapReduce programs used with Hadoop. The language for this platform is called Pig Latin. Pig Latin abstracts the programming from the Java MapReduce idiom into a notation which makes MapReduce programming high level, similar to that of SQL for RDBMS systems. Pig Latin can be extended using User Defined Functions which the user can write in Java, Python, JavaScript, Ruby, or Groovy and then call directly from a Pig Latin program.

2.7 Elasticsearch

Elasticsearch for Apache Hadoop (ES-Hadoop) is a two-way connector that provides real-time search on top of Hadoop. While the Hadoop ecosystem offers a multitude of analytics capabilities, it is less appropriate for fast search. ES-Hadoop allows for combining Hadoop's big data analytics and the real-time search of Elasticsearch[41].

[37] http://jnb.ociweb.com/jnb/jnbJun2009.html
[38] https://avro.apache.org/
[39] http://spark.apache.org/examples.html
[40] http://pig.apache.org/
[41] https://www.elastic.co/products/hadoop

3 Evaluation of technologies

For one of the Big Data tools listed above to be considered a viable avenue of investigation it must meet certain criteria. First, as APEL is built in Python, the Big Data tool should ideally have a Python API. Secondly, the chosen tool should have a SQL-like query interface for debugging purposes. Thirdly, the order the data processed in must be the same as the data received, as APEL requires this to produce correct summaries.

The requirement for a Python API excludes Camel and Samza. The requirement for an SQL-like interface excludes Accumulo.

A table summarising the tools covered in this section can be found in Appendix I.

3.1 MySQL optimisations

3.1.1 Parallelised processing

Parallelising the processing of the summaries would use the existing software and interfaces, so no new languages would need to be used, but there would need to be modifications to the APEL software.

3.1.2 Percona Server

Percona Server is written in C and C++. Migrating from MySQL to Percona requires replacing the MySQL binaries with Percona replacements. This allows Python software to interact with Percona Server using the same libraries as it interacts with MySQL (MySQL-Python), however an additional Percona library is needed be installed as well[42]. It does not require a transfer of data as other tools do (e.g. transfer to the HDFS). Percona server is also able to manage a much larger number of concurrent threads compared to the standard version of MySQL.

Percona XtraDB Cluster[43] can also provide multi-master replication, allowing writing to any node in a Percona cluster[44].

Python API	***SQL-like interface***	***In-order processing***
Yes	Yes	Yes

[42] http://stackoverflow.com/questions/21481985/mysql-python-install-with-percona
[43] https://www.percona.com/software/mysql-database/percona-xtradb-cluster
[44] https://www.percona.com/doc/percona-xtradb-cluster/5.6/features/multimaster-replication.html

3.2 InfluxDB

InfluxDB is written in Go and has a built-in HTTP API. A Python client has been developed by the same developers as InfluxDB itself[45]. It also supports an SQL-like query language[46].

Accounting data arrives at the Accounting Repository in batches and is loaded in batches, so it may be necessary to import this data into InfluxDB, rather than insert it. A one off bulk import would be required to store historical data. Clustering is supported out of the box, so data can be replicated over multiple nodes. However there are no map-reduce-style operations to take advantage of the clustering[47].

InfluxDB supports continuous queries, meaning aggregations could be updated on the fly.

Python API	***SQL-like interface***	***In-order processing***
Yes	Yes	Yes

3.3 Datastores

3.3.1 Hive

Hive is written in Java and has a client API for many languages including Python. An SQL-like (HiveQL) interface for querying data in the HDFS. Such queries have traditionally had high latency, and even small queries could take some time to run because they were transformed into map-reduce jobs and submitted to the cluster to be run in batch mode. Newer versions of Hive have improved this performance by using the Tez execution framework, by using the Optimized Row Columnar (ORC) file format or by enabling cost-based query optimisation[48]. From the top down, Hive looks much like any other relational database.

Python API	***SQL-like interface***	***In-order processing***
Yes	Yes	Yes

3.3.2 HBase

HBase has a Python API and, unlike Hive, allows write operations into existing tables and the HDFS. Its query language is not SQL-like and appears relatively simple. However, an extension, Apache Phoenix[49], provides an SQL layer allowing HBase to be a replacement for a MySQL database. HBase is designed to handle large volumes of data (over a few hundred gigabytes) and large number of

[45] https://github.com/influxdata/influxdb-python
[46] https://docs.influxdata.com/influxdb/v0.8/api/query_language/
[47] http://www.shift8creative.com/posts/influxdb/
[48] http://hortonworks.com/hadoop-tutorial/real-time-data-ingestion-hbase-hive-using-storm-bolt/
[49] https://phoenix.apache.org/

concurrent clients. As HBase is column oriented, aggregation operations might be quicker than row based databases.

Python API	***SQL-like interface***	***In-order processing***
Yes	Yes, with Phoenix	Yes

3.3.3 Cassandra

Cassandra's data model offers the convenience of column indexes with the performance of log-structured updates, strong support for denormalisation and materialised views, and powerful built-in caching[50]. It has a Python API[51] and an SQL -like query language, which supports inserts and updates. Newer versions support aggregation operations[52], but this may have to be on columns which are part of the primary key.

Cassandra uses its own version of the HDFS, the Cassandra File System (CFS). In contrast to the master-slave architecture of HDFS, the architecture of the CFS is peer-to-peer and so does not have a master node. This simplifies the operational overhead of Hadoop by removing the single points of failure in the HDFS. A user is able to create a cluster that seamlessly stores real-time data in Cassandra, performs analytic operations on that same data, and also handles enterprise search operations[53].

Python API	***SQL-like interface***	***In-order processing***
Yes	Yes	Yes

3.4 Data Transfer Tools

3.4.1 Apache Flume

Flume is not a method of querying the data already in the HDFS but rather a method of getting new data into the HDFS. As such, it has not SQL-like query language and would have to be combined with another tool, such as Hive or HBase. It does not provided any ordering guarantees, it only guarantees a single message will be processed exactly once.

3.4.2 Apache Sqoop

Sqoop allows for transfers from relational databases such as MySQL to Hadoop data stores such as HDFS and Hive and vice versa. The tool could be useful for one-off transfers to create a corpus of test data but also for integration in a system where data exists in a database and Hadoop tool in parallel – allowing for both tools to be used to their best advantage.

[50] http://cassandra.apache.org/
[51] https://github.com/datastax/python-driver
[52] https://issues.apache.org/jira/browse/CASSANDRA-4914
[53] https://www.datastax.com/wp-content/uploads/2012/09/WP-DataStax-HDFSvsCFS.pdf

3.5 Data flow/stream processing tools

3.5.1 Cascading

Cascading has both a Python API and a SQL interface via Lingual[54]. It can query existing data and insert data into the HDFS. One benefit of Cascading is the ability to use Lingual to insert query data using actual SQL, thus requiring less change to the APEL software.

Python API	***SQL-like interface***	***In-order processing***
Yes	Yes	Yes

3.5.2 Apache Storm

Storm has many use cases: real time analytics, online machine learning, continuous computation, distributed RPC, ETL, and more. Storm is fast: a benchmark clocked it at over a million tuples processed per second per node. Storm integrates with the queueing and database technologies you already use. A Storm topology consumes streams of data and processes those streams in arbitrarily complex ways, repartitioning the streams between each stage of the computation however needed. It has a Python wrapper[55] and serves as a means to get data into the HDFS and process it and route.

Python API	***SQL-like interface***	***In-order processing***
Yes	No	Yes

3.5.3 Apache Spark

Spark has both a Python API[56], including examples for non-streaming spark, and a SQL-like for trouble shooting. Spark can also interact either directly with the HDFS, or with other Apache tools, such as Hive[57]. It can be used to write data to the HDFS[58].

Python API	***SQL-like interface***	***In-order processing***
Yes	Yes	Yes

3.5.4 Apache Pig

Pig Latin is a query language for the HDFS that is SQL-like though it diverges from the language more than other SQL-like interfaces listed here. It supports loading directories of data from the

[54] http://www.cascading.org/projects/lingual/
[55] http://github.com/twitter/pycascading/wiki
[56] http://spark.apache.org/examples.html
[57] http://spark.apache.org/sql/
[58] https://spark.apache.org/docs/1.1.1/api/java/org/apache/spark/rdd/RDD.html

HDFS and working with it to get it into the form for querying. Hive is considered friendlier and more familiar to users who are used to using SQL for querying data[59].

Python API	***SQL-like interface***	***In-order processing***
No	Yes, with caveats	Yes

3.6 ElasticSearch

ElasticSearch is a searching method that can be placed on top of the HDFS. It has a Python interface but not a SQL-like query language. ElasticSearch itself cannot input new data into the HDFS, but it does can become aware of new data when it is inserted into the HDFS via other methods, such as cURL.

Python API	***SQL-like interface***	***In-order processing***
Yes	No	Yes

3.7 Summary

Most of the technologies introduced in this report meet the minimum requirements needed for integrating into the Accounting Repository. As mentioned above, the requirement of a Python API excludes Camel and Samza. Of the remaining tools, most seem applicable to the Accounting Repository. However, two of the technologies, InfluxDB and ElasticSearch, are more specialised and have less in common with the other ones, so they are not planned be tested at this stage due to the limited available effort.

[59] http://hortonworks.com/hadoop-tutorial/how-to-process-data-with-apache-hive/

4 Testing the technologies

4.1 Metrics to test the different technologies

In the context of the accounting system, there are a number of metrics that should be measured for candidate technologies.

An important part of the accounting system is the processing that it does to the data that it receives before the data is sent on to the accounting portal for visualisation. This aggregation or “summarising” currently takes a long time, in the order of hours, and means that is impractical to perform the summarising more than once a day. Ideally, this process would be much quicker, allowing data to be sent to the portal on a much more regular basis or even streamed continuously if the technology allows it. This means that the time it takes to process the data is an important metric, measured as the time it takes to perform the summarising process. This can then be compared to the duration of the current summarising process.

As some of the proposed configurations discussed further on would keep the existing database and then use one of the technologies under consideration only for processing, the time it takes to transfer the data through the interface of a candidate technology would be relevant as it may become a regular event. Some technologies may allow for incremental transfers once the initial transfer is done and this would reduce the impact of the transfer time, but it should still be measured in case the transfer of data between the existing database and the interface of the technology needs to be factored into the processing time.

The MySQL database at the heart of the accounting repository currently takes up about half a terabyte of storage space. When this is converted to another format, such as HDFS, this may increase the storage requirements for the repository. This would be due to the change in data format as well as replication in the case of a solution that uses a cluster.

Different technologies may need different resources, be they infrastructure (such as VMs or physical hosts), licences or effort, and this should be noted along with the cost of these resources for comparison between the technologies.

An important part of deploying a new technology is supporting it. This has an impact on the level of effort required to use a technology and in some cases, it may not be possible to use the technology without some minimum expertise being available. With this in mind, there are a few areas that should be evaluated when testing a candidate technology. The first is if there is enough knowledge about the chosen technology within the APEL team and also the EGI collaboration. The second is how easy it is to install and configure the technology; if there is clear and comprehensive documentation available, this can help. Lastly is how easy it is to use the technology; an important aspect of that is if there is an easy way to query the data once it is imported into the tool. A familiar interface, such as a SQL-like query language, would aid this. Additionally, if the new technology has a Python API or SQL API, it will require less effort to integrate it into the existing APEL software.

Table 1 summarises the metrics discussed above for evaluating the different technologies.

Table 1 - Metrics to evaluate the technologies

Metric	***Short description***
Time to summarise data (batch) or latency (streaming)	The time taken to create a complete set of summaries – for batch-like technologies where all of the summaries are calculated at once; or the time taken from a new record arriving in the system to the corresponding summary being updated – for streaming-like technologies where the summaries are updated continuously.
Time to transfer data out of/in to data store	For technologies which require data to be transferred out of the data store for processing, the length of time taken for the data to be transferred out and the results to be returned to the data store.
Storage space	The storage space required to store the test data corpus.
Number of hosts	The number of hosts required for a particular technology e.g. an HDFS installation requires at least 2 hosts for failover.
Ease of installation	A subjective measure reflecting the quality of the documentation and the amount of configuration required.
Ease of use	A subjective measure of how easy the technology is to use reflecting the expertise of the APEL accounting team.
SQL interface	Whether the technology has as an SQL interface or something similar allowing for easier integration with existing accounting software and workflows.
Python API	Whether the technology has a Python API allowing the technology to be interfaced to the existing APEL accounting software written in Python.

4.2 Resources available

There are a number of resources available to the APEL project within STFC that can help enable the testing and evaluation of the different big data technologies.

The first is a self-service cloud, which provides an internal IaaS cloud resource for STFC users. The cloud is based on OpenNebula for the virtualisation and CEPH for the storage. For the hardware, there are:

- 28x Dell R420 for the hypervisors
- 30x Dell R520 for the storage nodes

This gives a total of 3.5TB of memory, 896 processing cores and a storage capacity of approximately 750TB. A variety of OS images is available, mainly consisting of different versions of Scientific Linux and Ubuntu, although other images can be requested if there is a use-case for them. This cloud allows developers to quickly requisition resources for testing and experimentation.

The second is a small Hadoop cluster of ten nodes that has been used for evaluating cloud storage and compute, and for some development work.

Lastly, the existing accounting data is an excellent resource for creating test data from. To create a corpus of test data, a tool like Apache Sqoop[60] can be used to extract the data from the current relational database management system, MySQL, into the Hadoop Distributed File System, transform the data in Hadoop MapReduce, and then export the data back into MySQL.

4.3 Possible configurations

Figure 3 shows the broad categories of technologies and their topology in the APEL system. A message broker operates externally receiving data from sites. The APEL server pulls this data and stores it in a database. Periodically, the data is processed to create summaries. The summaries are then exported from the database and sent on via the same message broker to the EGI Accounting Portal for display to users. The database receives ad-hoc queries for the purpose of investigating problems with sites' data.

[60] http://sqoop.apache.org/

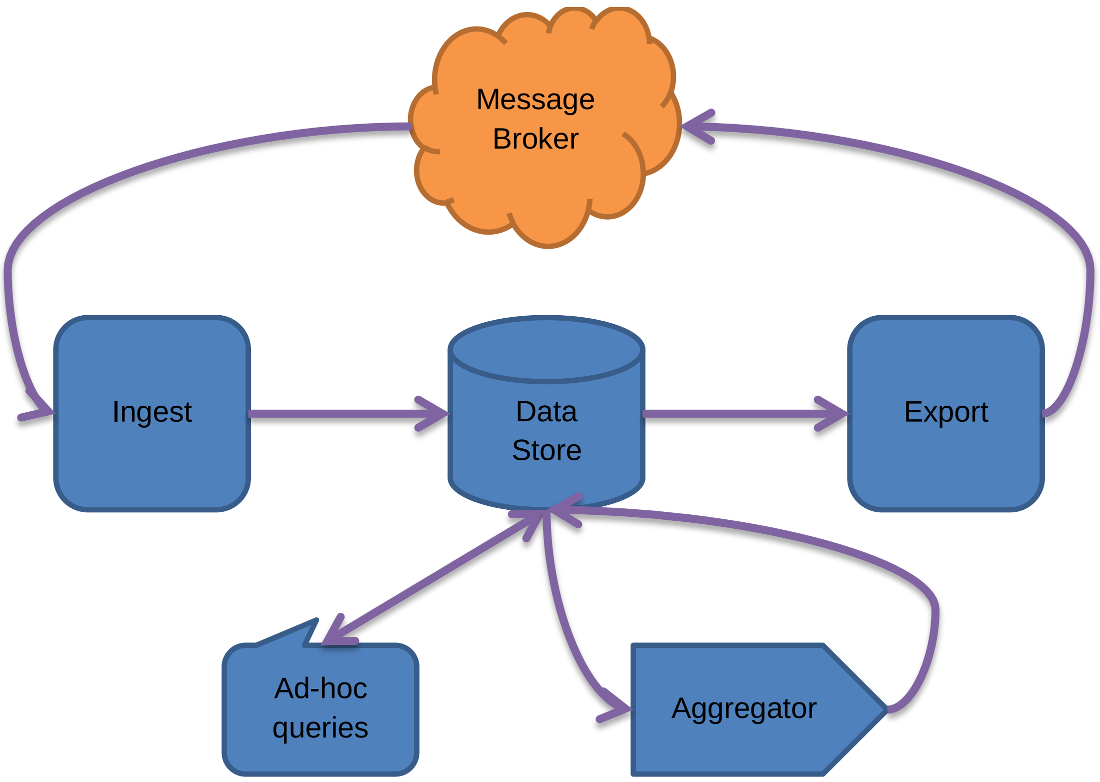

Figure 3 - APEL technology topology

4.3.1 Current APEL configuration

The current APEL configuration consists of:

- Ingest: APEL SSM receiver and APEL DB Loader
  The APEL SSM software pulls messages from the message bus and writes them to the file system. The APEL DB Loader reads the files and loads them into the database.
- Data Store: MySQL
- Export: APEL DB Unloader and APEL SSM Sender
  The APEL DB Unloader and SSM Sender operate like the loader and receiver in reverse.
- Ad-hoc queries: MySQL Client
- Aggregator: APEL Summariser

The APEL Summariser runs a single-threaded query against the MySQL database to produce the summaries. The query takes 8 to 10 hours to complete.

4.3.2 Parallel batch processing

Parallelising the processing of the summaries would use the existing software and interfaces, but with modifications to the APEL software. This would entail re-writing the APEL software to spread the summarising process across multiple database connections.

4.3.3 Replacement APEL backend

The APEL software currently has a MySQL backend. This could be replaced by an alternative SQL-like interface such as Percona Server, Hive+HiveQL or HBase+Phoenix. This would small changes to the APEL software initially, but may require configuration and some changes to the software to make best use of it.

4.3.4 Replace APEL tools with Hadoop/HDFS tools

In this configuration, the separate APEL tools are replaced with Hadoop/HDFS tools which perform the same function. The example below shows the configuration for Hive but the same could be achieved using Flume, HBase and Phoenix.

- Ingest: Apache Flume
  Flume can pull data from a message bus and write directly to Hive.
- Data Store: Apache Hive
- Export: Scripted HiveQL query
  Flume does not extract data from Hive. However, it is possible to extract data from Hive to csv using HiveQL. A small script could convert this to the correct format for the APEL SSM software to send to the accounting portal.
- Ad-hoc queries: Apache Hive Beeline
  Beeline is the interactive shell for the Hive Server.
- Aggregator: pre-iledcompiled Hive query

4.3.5 Combined MySQL and Hadoop

- Ingest: APEL SSM receiver and APEL DB loader
- Data Store: MySQL
- Export: APEL DB unloader and SSM sender
- Ad-hoc queries: MySQL
- Aggregator: Sqoop could be used to export the data from MySQL to HDFS for later processing by a number of Hadoop Map/Reduce or other technologies built on it. Sqoop could also be used to return the summaries to MySQL. This provides the advantage of multi-core processing without the difficulty of making larger changes to the existing service architecture. The disadvantage of this approach is the overhead of data transfers – though incremental updates could reduce this. Further there could be potential for the 2 copies of the data to go out of sync.

4.3.6 Parallel stream processing

The stream processing frameworks provide the opportunity to split the data store and aggregator functions apart so that the aggregation is performed continuously. In this configuration Apache Flume is used to duplicate the data pulled from the message broker: with one copy going to a data store – such as MySQL or Hive – for ad-hoc queries and backup; and the other copy going to a stream processing framework – such as Samza, Spark or Storm – to provide near real-time generation of summaries.

4.4 Testing Schedule

Configuration	***Start of testing period***	***End of testing period***
Parallel batch processing	March 2015	April 2015
Replace APEL tools with Hadoop/HDFS tools	May 2015	June 2015
Replacement APEL Backend	July 2015	August 2015
Combined MySQL and Hadoop	September 2015	October 2015
Parallel stream processing	November 2015	December 2015

Appendix I. Summary of big data tools

Database-like Tools

Tool	***Tool language***	***API language***	***SQL-like interface?***	***Data entry into HDFS?***	***Searching into HDFS?***	***Batch or streaming?***	***Notes***
Percona Server	C/C++	Python	Yes	N/A	N/A	Both	Drop-in replacement for MySQL.
InfluxDB	Go	HTTP, Python	Yes	N/A	N/A	Both	Batch imports less efficient.
Hive	Java	Java, Python, PHP	Yes	Yes	Yes	Batch	Hive only has "INSERT...VALUES" in version 0.14.
HBase	Java	Java, Python	No, see notes	Yes	Yes	Batch	SQL like query language can be added on top of HBase[61].
Cassandra	Java	Java, Python	Yes	Yes, see notes	Yes, see notes	Batch	Cassandra replaces the HDFS with its own file system[62].

[61] https://phoenix.apache.org
[62] https://www.datastax.com/wp-content/uploads/2012/09/WP-DataStax-HDFSvsCFS.pdf

Non Database-like Tools

Tool	***Tool language***	***API language***	***SQL-like interface?***	***Data entry into HDFS?***	***Searching into HDFS?***	***Ordering preserved?***	***Batch or streaming?***	***Notes***
Parallelised processing	Python	Python	Yes	N/A	N/A	N/A	N/A	Does not affect loading of data.
Samza	Java, Scala	Java	No[63]	No	Yes	Yes, per stream[64]	Streaming	Stateful stream processing, could possibly compute aggregate on the fly
Spark	Java, Scala, Python, R	Java, Scala, Python, R[65]	Yes	Yes	Yes	Yes, per stream	Streaming	Spark/Hadoop cluster already set up
Camel	Java	Mostly Java[66], see notes	Yes[67]	Yes[68]	Yes	No[69]	Both	Expressions and Predicates can be written in Python
ElasticSearch	Java	Java, Python	No	No, see notes	Yes	N/A	Batch	Data can be inserted using the Linux tool cURL.

[63] https://samza.apache.org/learn/documentation/0.7.0/container/state-management.html
[64] http://samza.apache.org/learn/documentation/0.7.0/comparisons/spark-streaming.html
[65] http://spark.apache.org/docs/latest/api.html
[66] http://camel.apache.org/scripting-languages.html
[67] http://camel.apache.org/sql-component.html
[68] http://camel.apache.org/hdfs.html
[69] http://camel.apache.org/parallel-processing-and-ordering.html

Accumulo	Java	Java, Python	No	Yes	Yes	N/A	Batch	
Flume	Java	Java, Python, Scala	No	Yes	No	No[70]	Streaming	Not a method of querying the data, but a method of putting it into queries methods.
Cascading	Java	Java, Python, SQL, others[71]	Yes	Yes	Yes	N/A	Batch	SQL API, Lingual[72]
Storm	Clojure, Java	Java, Python	No	Yes	No	Yes, but per stream only[73]	Streaming	
Pig	Pig Latin	Java, Python, JS, Ruby, Groovy	No (Pig Latin)	No	Yes	N/A	Batch	No append to existing files.

[70] https://github.com/cloudera/flume/wiki/FAQ
[71] http://www.cascading.org/extensions/
[72] http://www.cascading.org/projects/lingual/
[73] https://samza.apache.org/learn/documentation/0.10/comparisons/storm.html