SDP Memo: Can SDP use existing big-data systems?

Document number..…….. SKA-TEL-SDP-0000072
Document Type……..…… REP
Revision……..…….…… 01C
Author…….…… R. Simmonds, S. Gounden
Release Date…….…….. 2016-04-07
Document Classification……. Unrestricted
Status……..... Draft

Lead Author	Designation	Affiliation
Rob Simmonds	DELIV Team Lead	University of Cape Town
Signature & Date:	 RWJ Simmonds (Apr 7, 2016) robert.simmonds@uct.ac.za	

Released by	Designation	Affiliation
Paul Alexander	SDP Project Lead	University of Cambridge
Signature & Date:	 Paul Alexander (Apr 7, 2016) pa@mrao.cam.ac.uk	

Version	Date of Issue	Prepared by	Comments
1	2015-07-31	R. Simmonds	M12 submission
01C	2016-04-07	R. Simmonds	SDP delta-PDR submission

ORGANISATION DETAILS

Name	Science Data Processor Consortium

Table of Contents

List of Figures

N/A

List of Tables

N/A

1. Summary

A number of systems designed for “big-data” processing are available and under active development. Given this we need to assess if any of these systems could meet the requirements for SKA processing and could therefore remove the need for some software development within the project. This document gives an overview of the most popular systems, discusses risks of trying to adopt them for the project and provides next steps we feel should be taken in this assessment.

The next steps suggested include a more in depth analysis of how the various processing steps could be partitioned and how well the big processing systems meet the requirements identified in these steps. That will then enable us to identify what systems should be tested and prototyped to determine how well they perform for our needs. From our initial investigation the Apache Spark system has been identified as one that is of particular interest for further investigation.

2. References

2.1. Applicable Documents

The following documents are applicable to the extent stated herein. In the event of conflict between the contents of the applicable documents and this document, **the applicable documents** shall take precedence.

Reference Number	**Reference**
AD01	SKA-TEL-SDP-0000033 Requirements Analysis and Allocations
AD02	SKA-TEL-SDP-0000063 Product Tree Diagram
AD03	SKA-TEL-SDP-0000013 SDP Element Architecture Design

2.2. Reference Documents

The following documents are referenced in this document. In the event of conflict between the contents of the referenced documents and this document, **this document** shall take precedence.

Reference Number	Reference
RD01	Cloud-based big data: http://radar.oreilly.com/2012/02/big-data-in-the-cloud-microsoft-amazon-google.html
RD02	Big data processing systems: http://www.informationweek.com/big-data/big-data-analytics/16-top-big-data-analytics-platforms/d/d-id/1113609
RD03	Apache Spark: https://en.wikipedia.org/wiki/Apache_Spark
RD04	Amazon Big Data: http://aws.amazon.com/big-data/
RD05	Hadoop: http://www.tutorialspoint.com/hadoop/
RD06	MapReduce: http://research.google.com/archive/mapreduce.html
RD07	Large Hadoop Clusters: http://www.hadoopwizard.com/which-big-data-company-has-the-worlds-biggest-hadoop-cluster/
RD08	Shuffle Agorithm: http://analyticsindiamag.com/shuffle-operation-hadoop-spark/
RD09	SKA-TEL-SDP-0000066 Dataflow Choice
RD10	Apache Flink: https://flink.apache.org/

3. Introduction

The purpose of this document is to consider if any of the “big-data” processing systems that are currently available could be used to satisfy some of the requirements of the the Science Data Processor element in the SKA. This mostly considers the software solutions for

performing processing of very large amounts of data, but could include complete solutions with hardware, as provided by commercial cloud computing vendors. This document focuses on technical issues, though there could be other considerations in a final technology choice.

We believe that key criteria for the selection of a big data processing system are:

- Needs to support the range of processing algorithms required by SDP, which includes fast imaging and iterative processing for high-dynamic range imaging
- How computing is parallelised and how data is distributed
- Fault tolerance
- Resource allocation (in our case how resources are allocated to different activities)
- Memory management
- Maintenance
- Ease of debugging

We do not believe that any one system will satisfy all of the requirements being considered by the SDP. For example, the requirements for the pipeline processing are quite different than those of data visualisation. This document focuses on the requirements (as described in AD01) for processing data that arrives from the CSP element and is delivered into the SDP preservation system. This is the Data Processor, C.1 in the SDP product tree [AD02]. Even constraining the problem to this part of the work it is quite possible that no single system will address all the processing requirements. The data flow approach has been proposed to solve this problem and some of the big data processing systems do support this paradigm [RD09]. However, we will not limit this investigation to only those systems.

The processing in the Data Processor can be described as follows:

- Data arrives from the CSP
- Some initial stream processing is performed to handle tasks such as flagging
- Then the data is placed into a buffer (described at the Fast Buffer) with different regions of the buffer being written to by CSP and read from by the Data Processor (DP) at different times
- Initial data products are processed by applications that are described in a task graph
- Intermediate products could be written to temporary buffer space or streamed directly from one application to the next. The implementation of the buffer space is not defined, but could be either nonvolatile or volatile depending on cost considerations and the risk of losing data

- Final products (referred to as Science Products) will be written to a buffer from where they will be moved to the preservation system

This document sets out to determine if a single existing data processing system might satisfy all of the tasks above, or if they could satisfy a large enough part of this to make them worth adopting for that part of the work. It is clear that substantial testing and prototyping of any of these systems with our SKA use cases will be required to fully determine their applicability.

It should be noted that one of the design criteria considered up to now is that not all data is precious, so some data could be lost without having a major impact on the science performed [AD03]. This may not be easy to support with a general use big data processing system since it is unusual to allow this. However, if a fault tolerant system could provide a solution to the processing problems without massive overhead, this criterion could be retired.

The rest of this document is laid out as follows. Section 2 presents some of the different solutions that could be considered. Section 3 provides commentary on the decision of what we should investigate further. Section 4 lists some related decisions. Section 5 presents risks associated with choosing to use one of these systems and Section 6 presents future work suggested in this area.

4. Alternatives

Various big data technologies are offered by vendors such as IBM and Amazon. There are also open source solutions that are becoming widely used. Two classes of technologies have been categorised as follows [RD01]:

1. Operational Big Data: Provides operation capabilities for real-time, interactive workloads where data is primarily captured and stored.
2. Analytical Big Data: Provide analytical capabilities for retrospective and complex analysis that may be applied to some or all of the data.

For the purposes of the Data Processor, the properties of the analytical big data systems are of most interest. However, there is also some real-time processing, such as the Transient Pipeline, which makes it clear that a single system may not satisfy all of the Data Processor requirements. Details of specific systems are provided in the following sections.

4.1. Hadoop

Hadoop is an Apache open source framework written in Java that allows distributed processing of large datasets across a compute cluster using simple programming models based on the MapReduce algorithm. This was developed by and is used by Google to handle large-scale search problems. MapReduce complements the capabilities offered by SQL, which is good for fast searching of small data sets. Most big data processing systems are based on ideas developed in Hadoop, using extensions to the MapReduce programming model. Hadoop now refers to a set of base modules as well as additional software packages that can be installed on top of or alongside Hadoop such as Apache Pig, Apache Hive and Apache HBase [RD05]. These are described below:

- Apache Pig: Platform that allows one to write complex MapReduce transformations using a simple scripting language.
- Apache Hive: Originally developed by Facebook, Hive is a data warehouse infrastructure.
- Apache HBase: HBase is an open-source, distributed, versioned, non-relational database.

Systems based on Hadoop/MapReduce can be scaled up to thousands of compute nodes; Yahoo was running a 42,000 node cluster in 2011 [RD07].

Advantages of Hadoop:

- Hadoop does not rely on hardware to provide fault-tolerance and reliability.
- Servers can be added or removed from the cluster dynamically without incurring interruption of service.
- Hadoop is open source and compatible with all platforms as it is Java-based.

Disadvantages of Hadoop:

- Does not support complex task graphs
- Streaming not supported

Given the limitations of Hadoop, in its current form it does not represent a good solution for SDP to explore for SKA use.

4.2. Apache Spark

Spark [RD03] is now an Apache Top-Level Project, signifying that the project's community and projects are well-governed and that it is being widely adopted. Spark was developed to

support distributed queries on large data sets, with a primary focus on supporting machine learning and supports interactive use. Results have been presented showing that it is 100 times faster than Hadoop for some queries on big data. It is not clear that the distributed queries would be useful to SKA, but the data management provided by Spark might be. Spark supports more general task graphs than are available in Hadoop. It uses Directed Acyclic Graphs (DAGs) to track the generation of data products and can use these to reconstruct data if a computing element fails.

Spark supports the following languages: Scala, Java and Python. In the case of Python the interpreter used can be specified in the Spark configuration files, however Spark documentation states that it has only been tested with CPython. Data items are stored either as Java objects or in a serialized form. The default serializer is not efficient, but you can select an alternative one in the configuration file. Data is stored in immutable regions called Resilient Distributed Datasets. It is the use of these, that are linked in DAGs, that allows data to be reconstructed if a node fails.

According to the main Spark website, it currently works with diverse data sources including HDFS, Cassandra, HBase, and S3. Other data sources including MongoDB are mentioned on the DataBricks website; this company provides access to Spark as a cloud based offering and is run by the creators of Spark.
From initial investigation the biggest reason that Spark may not be appropriate for use in the Data Processor is the way data objects are represented and held in memory. This would need some investigation by the SDP pipelines team.

There are a number of extensions to Spark. In particular there is a module to support streaming of data. This needs some further investigation; it may be that data still needs to be buffered for a number of seconds between applications in a streaming task graph.

It has not been possible to find good information on the scale of the largest Spark deployments. Yahoo use it, and they did report using 42,000 nodes in Hadoop clusters previously, but as with other large scale users, they seldom report the size of their systems. The Spark web site says it has been deployed on clusters with over 8,000 nodes, but does not give any specific reference.

Advantages of Spark:

- Has flexible task graphs
- Handles fault tolerance
- Has multiple language bindings
- Supports iterative algorithms
- Has support for stream processing

Disadvantages of Spark:

- Memory held in Java objects by default
    - Memory kept in serialised form but need to provide an alternative to the slow default serializer
- Does not have concept of non-precious data

Spark does look interesting. While some of the interactive query support is likely not useful in the DP, the fault tolerance features could be useful. The main questions are if it would need too many more hardware resources or if it would be too slow compared to a more optimised solution built using a lower level framework. We also need to look more at how data is distributed [RD08].

4.3. Apache Flink

Flink [RD10] is another Apache project providing a streaming dataflow based big data processing system. It too has static processing API bindings for the Java, Scala and Python languages. It also has a stream processing API and SQL-link table processing API in Java and Scala. Additions to Fink include CEP, the Complex Event Processing library, a machine learning library and Gelly, a graph processing API and library.
There is currently less information available about Fink than there is about Spark, given that it is a more recent addition to the Apache code library. It appears to have many similarities to Spark, but further investigation would be required to determine the advantages of each system for our application.

4.4. Cloud-based systems

Big data technologies are available for processing in the cloud. Examples are Amazon Web Services (AWS), Google's BigQuery data analytics service and IBM's Bluemix.

Characteristics of cloud-based systems:

- Removes overhead of configuring clusters/infrastructure

- Clusters scale with demand
- Data locality is the major issue - for most systems, the shipping/importing of data is slow and expensive, though this can be mitigated when prioritised access is provided

4.4.1. Amazon

Amazon EC2 (Elastic Compute Cloud) for Hadoop is a popular big data solution. In addition, Amazon has launched Elastic Map Reduce which provides a hosted, scalable Hadoop service [RD01][RD04]. Amazon provides high performance computing solutions for the specialist (processing that requires specialised analytics) end of big data processing [RD04]. Amazon EC2 C4 [RD04] was designed for compute-bound workloads, such as transcoding, Massively Multiplayer Online (MMO) gaming and high performance computing applications. AWS Data Pipeline is a service aimed at creating complex data processing workloads that are fault-tolerant, repeatable and highly-available.

Amazon's big data storage and database services are Amazon S3 (Simple Storage Service), Amazon Redshift, Amazon DynamoDB and Amazon RDS (Relational Database Service). Redshift and RDS allow for MySQL, Oracle and SQL Server transactions. RDS is typically used for online-transaction processing and reporting and analysis. Redshift harnesses the scale and resources of multiple nodes and employs optimizations that provide improvements over traditional databases for analysis and reporting against very large datasets (petabyte-scale) [RD04]. DynamoDB is a NoSQL database that allows for the creation and storage of database tables that can store and retrieve data and serve any level of traffic requests. S3 stores data as objects within resources called 'buckets'. Objects can be up to 5 TB in size and access to the bucket can be controlled. Amazon's cloud-based analytics services (Elastic MapReduce, Data Pipeline) can be coupled with a big data storage warehouse such as Redshift to provide a complete, albeit proprietary, big data framework.

4.4.2. Google

Google's cloud platform is distinct in that it offers an application container with defined APIs and containers as opposed to virtualization. The concept of machines is absent - instead, applications execute in the cloud and gain access to processing power as needed. AppEngine is Google's cloud application hosting service and offers a MapReduce facility for parallel computation. BigQuery and Prediction API are Google's core big data offering. BigQuery is an analytical database that suits the interactive analysis of datasets of the order of 1TB. BigQuery offers a SQL interface to data and is comparable to Apache Hive although

typical performance is faster. BigQuery is ideal for exploratory data analysis. Getting large amounts of data into BigQuery is a challenge - data must be directly uploaded or imported from Google's Cloud Storage system. Streaming data into BigQuery is not viable, making regular imports necessary for constantly updating data. Furthermore, BigQuery only accepts data formatted as comma-separated value (CSV) files and external methods are required to format data beforehand. Google's Prediction API is employed for machine learning applications.

4.4.3. Microsoft

Hadoop is the basis of Microsoft's big data offering. Microsoft offers its data platforms on cloud based Windows Azure platform as well as on the OS Windows Server.

It would be a risk to rely on any proprietary system unless there was very great assurance that they would be available and maintainable for the lifetime of the project. However, we do feel it is worth exploring alternatives that are currently available so the risks and benefits can be assessed.

5. Decision

At this time it is not possible to say if one of the existing big data processing systems would satisfy the needs of SDP for the processing of data on the path between the CSP and the preservation system. No one system represents a perfect solution - trade-offs, customisation and integration of different technologies will likely be necessary.

There are certainly systems that look like they could be used for a major part of the SKA processing. That is not to say they should be used however.

One system that looks particularly interesting is Apache Spark. This could be used if we can work with the way that memory is represented and how data is distributed by the system. In the case of memory representation, this would require comment from the pipelines team to determine if using this would be feasible. Apache Flink looks similar to Spark and would also need similar evaluation from the pipelines team.

6. Related Decisions

The decision of what processing model is used could be influenced by the availability of tools that provide solutions to the big data processing problem.

Other issues:

- Data storage within the Data Processor and the implications to what would be needed by the particular processing systems
- Languages used for developing pipelines; if they can use the language bindings provided with these big data systems

7. Risks

- Using proprietary systems that may become unavailable at some point in the project.
- Using a system that is optimised for different use cases could lead to excessive resource requirements
- Not having enough control of an Open Source platform and having the development focus on features that are not of interest to us at the expense of the performance of features that are of interest.

8. Future Work

Additional work is required to ensure that the different SKA processing steps that existing big data processing systems could be used for are clearly identified. This will enable us to check what requirements they could satisfy and enable us to score the systems against how well they meet these requirements. After that scoring we will be able to determine how to prototype and test the most appropriate systems against these requirements.
With the work done so far we believe that the Apache Spark system is the most likely to meet a large number of Data Processor processing requirements, so this will be the first system we evaluate in this way. The Apache Flink system appears similar to Spark and it will also need further evaluation to determine where it may have advantages over Spark. The fact that both of these systems lack a C or C++ binding looks like a limitation. They do Python bindings, but these bindings are not as well developed as their Java and Scala bindings.