PoC - Big Data Health Analytics - Technology Selection

# Introduction

This document has been written to support Task #1, Deliverable #1 of the Big Data Health Analytics proof of concept(PoC) project, which are as follows:

**Task #1**: Selection of the tools and technology for the design of the Big-Data processing platform.

**Deliverable #1**: The specific deliverables are as follows:

1. A *list of software tools and development and deployment environments within the context of the design of the Platform.*
2. *An analysis of ‘why’ these tools were selected and*
3. *A comprehensive and comparative analysis of ‘how’ the selected tools contrast with other choices available.*

# Structure

This document addresses the deliverable described in the introduction in the following manner

|  |  |  |
| --- | --- | --- |
| Section | Synopsis | Deliverable Addressed |
| Section 3: Technology | States the technologies selected for this PoC including specific features provided by them. Presents the deployment environment for this PoC | #1.1 |
| Section 4: Requirements Landscape | Description of the benefits of using the technology described in Section #3 to meet the demands of health care data | #1.2 |
| Section 5: Alternatives Comparison | Comparing the technology stack in Section #3 to the closest alternative in the Big Data Analytics space. This section will also show how this Big Data solution compares to traditional business intelligence solution models | #1.3 |
| Section 6: Conclusion | Closing remarks | NA |

# Technology

## Overview

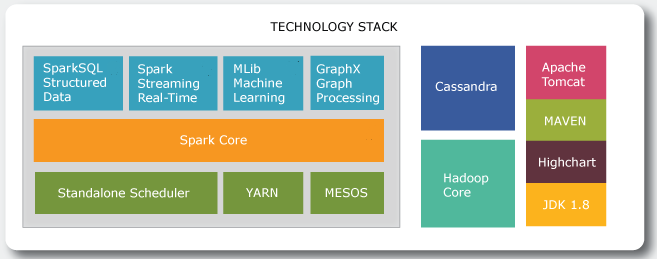
The technology stack that forms the basis of this PoC is presented here. When selecting the technologies for the stack care was taken to bring together components that would scale up to future related PoCs and eventual commercialization.

This PoC is centered around developing a core set of technologies for health care big data analytics. To this end we looked at the most promising industry leading technologies of the day.

Apache Spark is our core engine for this PoC. We will go into more details concerning the history of Apache Spark and as to why this was chosen, but here are a couple of compelling reasons to start with:

* Lightning Fast Processing
* Support for Sophisticated Analytics
* Real Time Stream Processing
* Integration with both NoSQL and RDBMS
* Ability to Integrate with Hadoop
* Active and Expanding Community

## High Level Component Diagram



Ref: Apurba Technology – Original Content

The diagram above describes the technology stack that will form the backbone of the biomedical big data analytics platform.

|  |  |  |
| --- | --- | --- |
| **Spark Framework** | **Component** | **Description** |
| Spark Core | Contains basic Spark functionality. Sparks fundamental programming abstraction, Resilient Distributed Data Set (RDD) represents a collection of items spread across parallel computing nodes. Spark provided an API for creating and managing RDDs. This API also takes care of parallel processing and management of RDDs. |
| SparkSQL Structured Data | Package for handling structured data. Querying via SQL as well as Apache Hive. SparkSQL supports interface with RDBMS, NoSQL Databases, HIVE Tables, Parquet and JSON |
| Spark Streaming Real-Time | Supports streaming of live data. Spark makes use of RDD creation, processing and management subsystem |
| MLib Machine Learning | Common library containing machine learning algorithms including classification, regression, clustering etc. |
| GraphX Graph Processing | Framework for manipulating graphs |
| Standalone Scheduler | Cluster Management Services |
| YARN |
| MESOS |

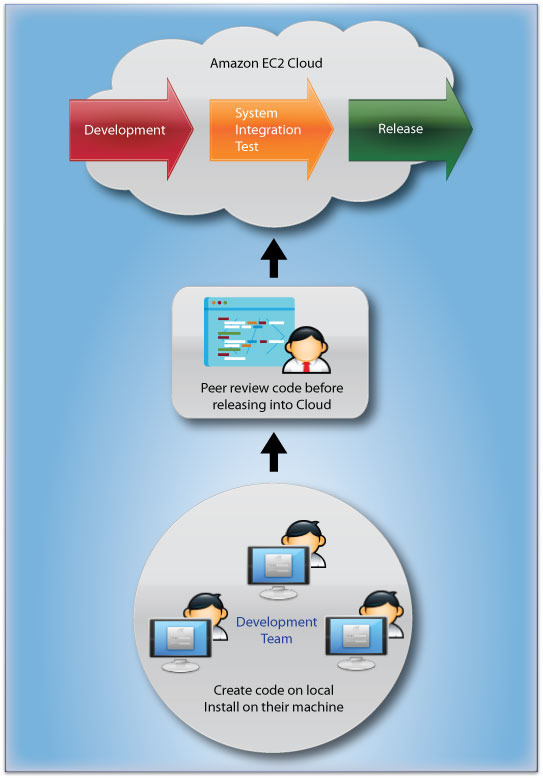
|  |  |  |
| --- | --- | --- |
| Supporting Technology Stack | **Component** | **Description** |
| Hadoop Core | Hadoops file system management services. Called directly from within Spark Core |
| Cassandra | Industry leading NoSQL database, providing these benefits: High Performance, Elastic Scalability, Open-Source, High Availability and Fault Tolerance, Column Orientated, Tunable Consistency, Schema Free |
| JDK 1.8 | Java SDK – Programming Language |
| MAVEN | Utilized to manage code dependencies |
| Apache Tomcat | Web Application Server |
| Highcharts | Java Script based charting library for data visualization |

## Deployment cycle

The technology stack used for this PoC has the significant advantage of being scalable. It can be installed on Laptop or Mac so that developers can build locally, peer review before uploading to the cloud. In the Amazon Cloud there will be three environments:

* Development
  + Code will undergo build and unit testing
* System Integration Test
  + Application will undergo regression testing.
* Release
  + Stable releases will be deployed to this environment for evaluation

The diagram below depicts this lifecycle.



Ref: Apurba Technology – Original Content

### Source code and document management

The system GitHub will be used by the developers as a repository for all source code management. All code as part of the PoC will be documented fully. This documentation will be in the form of both comments and formal implementation documents.

# Requirements landscape

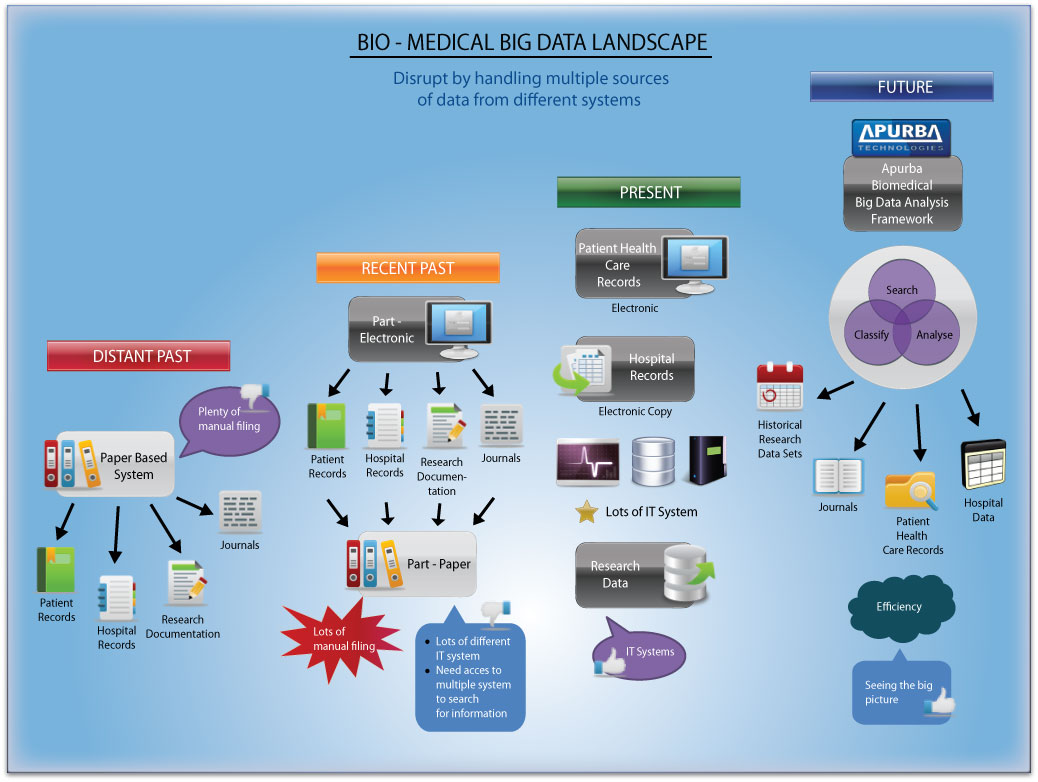
## Introduction

This section has been written to explain to the ready why we have selected the technology stack outlined in section #3, organized as follows:

* Overview of the biomedical big data landscape
* Relate Health Care Big Data to Apache Spark Features

## Overview

Health care data has always been growing. Since the advent of electronic health care records the rise as been exponential to where we now have a mountain of data. The diagram below shows the transition from paper based to electronic systems.



Ref: Apurba Technology - Original

The challenge now is to leverage insight from this data and thereby improve patient care by helping physicians use this information in their day to day work. To this end our systems need to be able to search, analyze and classify information. To do this we must look at the unique facets of this information.

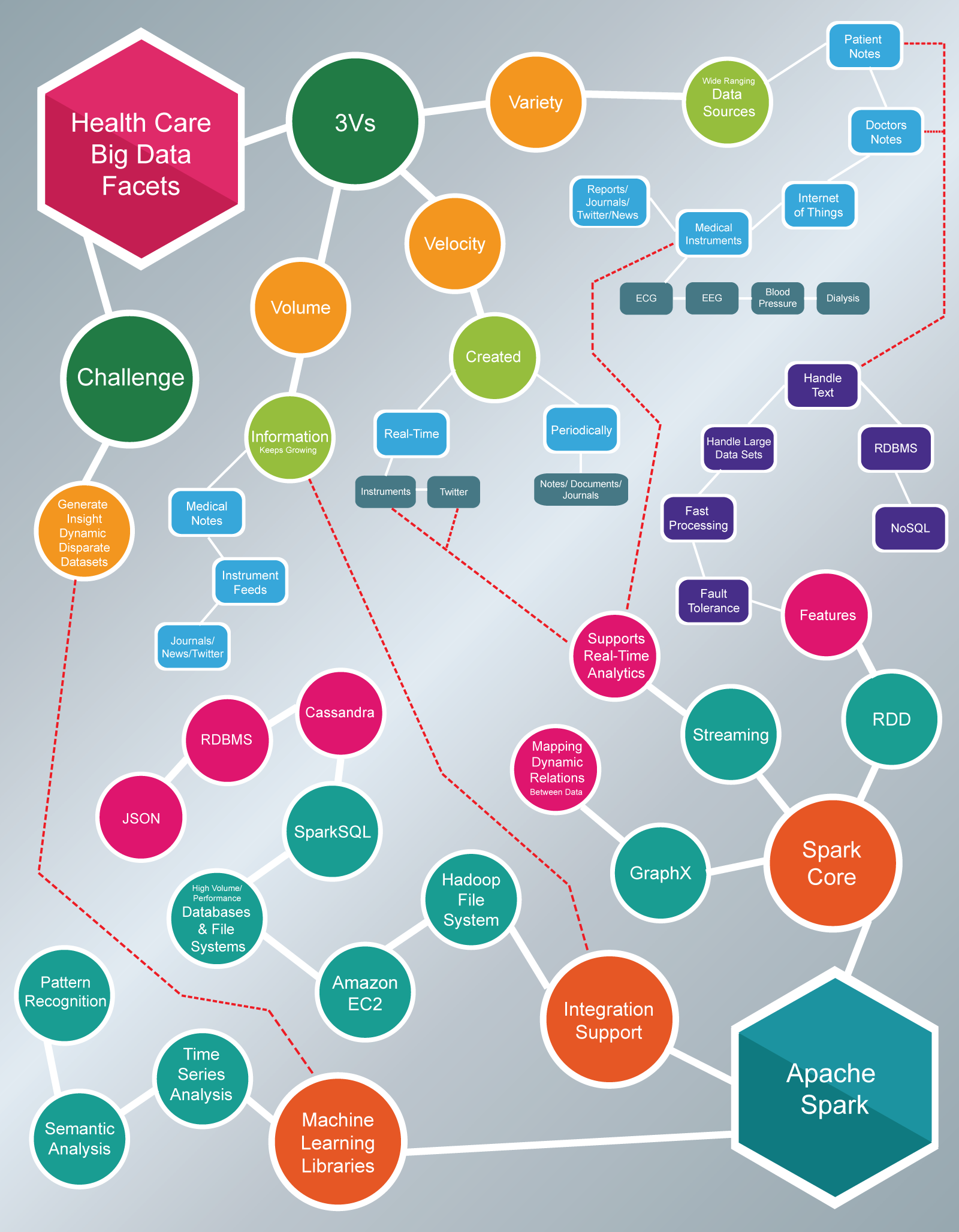
## Variety, volume, velocity

According to the Garner definition, “Big data is high-volume, -velocity and -variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making”.

* Volume – Information accumulating in multiple digital systems in a hospital
* Velocity – Assets being created at different times and speeds. For example, a quarterly performance report, a doctor’s medical note or vital signs monitoring in an ICU
* Variety – Information is being contributed from many sources and there isn’t a single format or standard.

## Health Care Big Data to Apache Spark Features

What does that mean in terms of health care data and how does the technology stack proposed help? The diagram below depicts health care big data’s 3Vs and expands them with health care examples. It also shows how Apache Spark meets those challenges.



Ref: Apurba Technology – Original Content

Apache Spark is being used as a big data analytics framework. To meet the challenges described above the components in the supporting technologies, such as Cassandra, Hadoop Core, JDK will be required too.

# Alternatives comparison

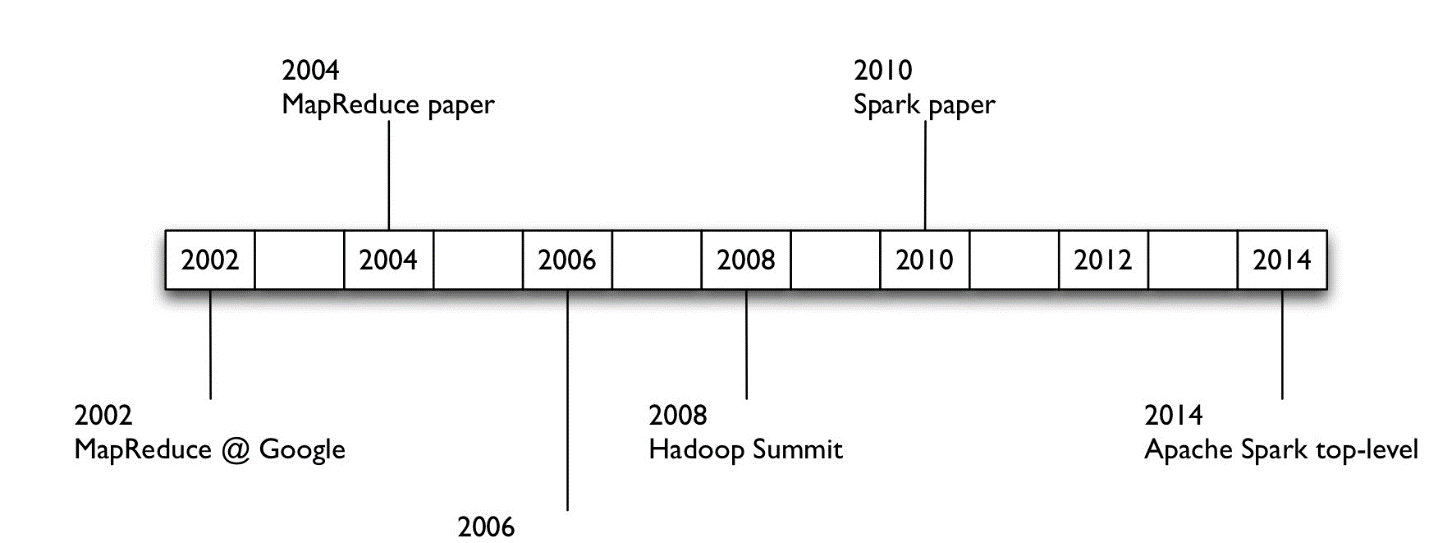
## Selection of Data Processing Engines

For this PoC we have selected Apache Spark as the core data processing engine and surrounding it are a NoSQL database, Hadoop and J2EE technologies.

Apache Spark is one of the most powerful technologies to have been contributed to the big data ecosystem in recent years.

This section is about comparing our chosen stack of technologies with closest alternatives. The closest alternative to Apache Spark is the earlier Hadoop Map/Reduce algorithm.

The diagram below shows a timeline:



Ref: [Sparkcamp](http://www.slideshare.net/databricks/sparkcamp-strata-ca-intro-to-apache-spark-with-handson-tutorials)

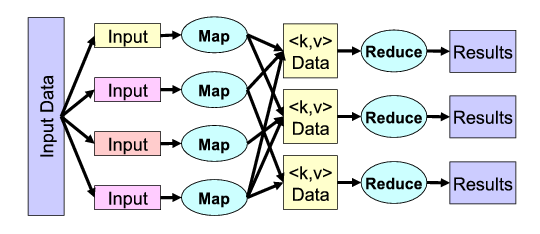
As per the above timeline, MapReduce was developed at Google and was later adopted by the Hadoop platform. In 2010 a research paper contributed by UC Berkeley entitled, “Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing” was made. This programming paradigm was adopted by the Apache Spark project in 2014.

The following subsections summarize these two paradigms. We will begin with the earlier Map/Reduce and then go on to Apache Spark’s Resilient Distributed Datasets (RDD). These sections have been written to provide simple explanation to paradigms that are significantly complex. The explanations are there to set the scene for a comparison of Apache Spark to MapReduce later on.

### Map/Reduce algorithm

Adopted by Hadoop, Map/Reduce was developed to address issues in processing large datasets. Designed to facilitate fast, fault tolerant, and reliable processing of data using a strategy that utilizes parallel processing across a large number of computing nodes.

The name MapReduce came from how the algorithm managed this process. With MapReduce, queries are split and distributed across parallel nodes and processed in parallel (the Map step). The results are then gathered and delivered (the Reduce step).

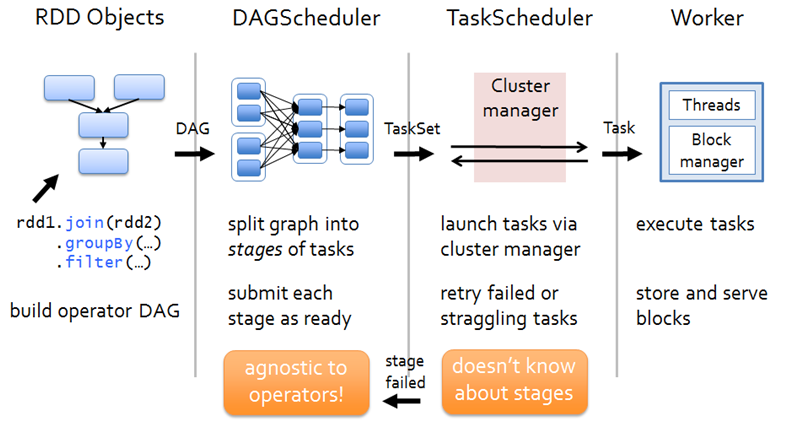


Ref: [MISCO](http://alumni.cs.ucr.edu/~jdou/misco/)

### Resilient Distributed Data set (RDD)

Resilient Distributed Data Sets (RDD) are the principle programming abstraction in Apache Spark. Within Apache Spark’s application interface, RDD’s appear to the programmer simply as a collection of distributed elements. What happens beneath the hood however is profound.

Once an RDD is created, Spark automatically distributes the data contained in the RDD across a cluster of computing nodes and parallelizes any operation that is required to run on them.



Ref: [Spark Blog](https://hxquangnhat.com/tag/spark-submission/)

As shown above, once the RDD is created it moves to a Scheduler which splits up the data. The Task scheduler than launches a cluster manager which in turn, manages a team of worker threads to process the data.

### Apache Spark and MapReduce Performance Comparison

|  |  |  |
| --- | --- | --- |
| Criteria | Apache Spark | Hadoop Map/Reduce |
| Installation | Leverages Hadoop as required | Bound to Hadoop |
| Ease of Use | Easy to program. The RDD model takes the strain of managing the slicing and processing of the data in a cluster | Difficult to program. Will need significant amount of abstractions. |
| Speed | Spark Enables applications to run up to 100x faster in memory and 10x faster even when running on disk. | Utilizes Hadoop storage over In-Memory data storage. Which can offer less chance of degradation where levels of In-Memory are an issue. |
| Data Processing | Can do more than just data processing. It can handle graphs, real-time data as well as batch processing. Also in-built facilities to handle data from NoSQL and RDBMS data sources | Hadoop is primarily batch processing |
| Fault Tolerance | Spark will retry a task. If a process crashes, Spark will need to restart. Spark has great fault tolerance due to the fact that data can be routed to other nodes. | If a process crashes MapReduce can begin where it left off because it uses the hard disk to store temporary data.  This could give a MapReduce a slight edge in fault tolerance. |

Ref: [Transitioning Compute Models: Hadoop MapReduce to Spark](http://www.slideshare.net/sbaltagi/chug-2-1215slimbaltagi)

### Why Apache Spark over Hadoop MapReduce

It is the nature of the biomedical data, ease of use, performance and flexibility of Apache Spark that it was chosen as the data processing engine for this PoC and others like it.

The diagram in section 4.4 was created to convey how many of the features in Apache Spark directly address the issues expected in biomedical data.

To use Hadoop MapReduce instead would have added considerable effort and would duplicate the efforts of the Apache Spark project. For this reason, Hadoop MapReduce was not considered an option.

## Use of NoSQL over RDBMS

The variety of information assets that we are expecting from the biomedical ecosystem has influenced our selection of NoSQL database architecture over RDMBS.

The term NoSQL in itself encompasses a wide variety of database technologies that have been developed in response to the demands of Big Data Analytics. Here are some of the compelling reasons to used them

* Adapted to work with Massive Volumes of Data
* Not tied to specific schema’s
  + Support combinations of Structured/Semi Structured/Unstructured information
* Need to get code out faster
  + 12-18 month waterfalls are now gone
  + Facilitates agile implementations
* Architecture needs to be able to scale out easily
* Relational databases were not designed to cope with this scale and need for this kind of agility

Ref: Ascent: [Top 5 Considerations when evaluating NoSQL databases](https://www.ascent.tech/wp-content/uploads/documents/mongodb/10gen-top-5-nosql-considerations.pdf)

### Kinds of NoSQL Databases

* **Document databases** pair each key with a complex data structure known as a document. Documents can contain many different key-value pairs, or key-array pairs, or even nested

documents. **Apache Spark’s RDD facilitate use of Key/Value passed**

* **Graph stores** are used to store information about networks of data, such as social connections. Graph stores include Neo4J and Giraph. **Apache Spark has a program abstraction called GraphX built work with Graph Stores**
* **Key-value stores** are the simplest NoSQL databases. Every single item in the database is stored as an attribute name (or "key"), together with its value. Examples of key-value stores are Riak and Berkeley DB. Some key-value stores, such as Redis, allow each value to have a type, such as "integer", which adds functionality.
* **Wide-column stores** such as Cassandra and HBase are optimized for queries over large datasets, and store columns of data together, instead of rows.

Ref: [MongoDB NoSQL Explained](https://www.mongodb.com/nosql-explained)

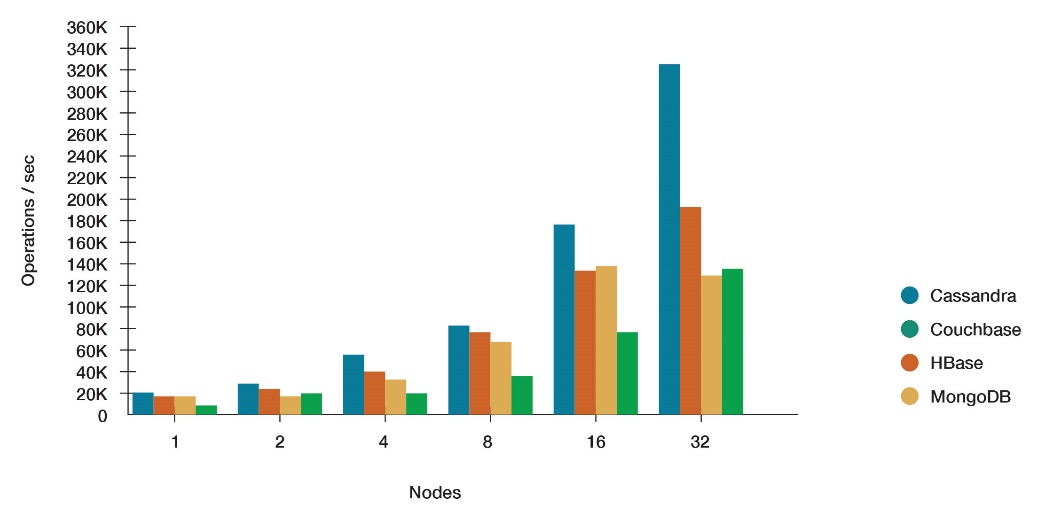
### NoSQL Database selection

The NoSQL Database selected for this PoC is Cassandra. This database was selected primarily because of the availability of a connector from Apache Spark to Cassandra, widespread adoption and the results of a comparative study by the University of Toronto.

This study carried out a performance comparison of the following NoSQL databases:

**Apache Cassandra:** Highly scalable, high performance distributed database designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure.   
**Apache HBase:** Open source, non-relational, distributed database modeled after Google’s BigTable and is written in Java. It is developed as part of Apache Software Foundation’s Apache Hadoop project and runs on top of HDFS (Hadoop Distributed File System), providing BigTable-like capabilities for Hadoop.   
**MongoDB:** Cross-platform document-oriented database system that eschews the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas making the integration of data in certain types of applications easier and faster.   
**Couchbase:** Distributed NoSQL document-oriented database that is optimized for interactive applications.

What took our interest in this study was performance under high load/throughput conditions. The graph below charts the number of operations per second when conditions force additional nodes to be employed for each database.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nodes | Cassandra | Crouchbase | HBase | MongoDB |
| 1 | 18,683.43 | 13,761.12 | 15,617.98 | 8,368.44 |
| 2 | 31,144.24 | 26,140.82 | 23,373.93 | 13,462.51 |
| 4 | 53,067.62 | 40,063.34 | 38,991.82 | 18,038.49 |
| 8 | 86,924.94 | 76,504.40 | 74,405.64 | 34,305.30 |
| 16 | 173,001.20 | 131,887.99 | 143,553.41 | 73,335.62 |
| 32 | 326,427.07 | 192,204.94 | 296,857.36 | 134,968.87 |

The summary of the study was that Cassandra performed best under these conditions. Selecting Cassandra as the NoSQL database based on a single study would be flawed. When taking into consideration other factors such as support for Cassandra within Apache Spark, size of user community and the fact its open source, makes Cassandra the obvious choice for this PoC.

Ref: [Datastax NoSQL Comparison](http://www.datastax.com/nosql-databases/benchmarks-cassandra-vs-mongodb-vs-Hbase)

## Web Application Servers

We decided to use Apache Tomcat as the web application server for the PoC. The criteria used for selection was:

* Open Source
* Wide adoption
* Ease of use

Here were the other candidates.

* Glassfish
* JBOSS Enterprise Application Platform
* Wildfly
* Apache Geronimo
* Jetty

## Other Technologies

The following Technologies were absolute necessities to the PoC because there needed for Spark Development:

* Hadoop Core
  + Required by Apache Spark to interface with Hadoop Filesystem Management System (HDFS)
* MAVEN
  + Required in Apache Spark Development to managed dependencies
* JDK 1.8
  + Apache Spark also supports Python and Scala. As this PoC is intended to produce a framework that can allow inclusion of other devices and computing nodes, as Java is ubiquitous it was decided that it should be used as the programming language of choice

The PoC will need some form of data visualization. In effect this involves building dashboards that represent the results to the user. To this end it was decided to use Highcharts on evaluation license. It’s easy to use, purely JavaScript, produces interfaces that can run on laptops, macs, iPad and Android devices, furthermore the development team have had extensive experience with it.

## Deciding against commercially off the shelf(COTS) alternatives

The principle reasons for deciding against use of commercially off the shelf alternatives were as follows:

* PoC Objectives achievable through Open Source Technologies
  + Apache Spark fulfils
    - Data Processing Demands for health care data
* COTS solutions tie the platform to specific vendors
* Cost
  + PoC objectives can be met via Open Source Technologies
  + Use of COTS would be prohibitively expensive
* Development and Commercialization of Intellectual Property from PoC
  + Dependency on COTS will make this complicated

# Conclusion

In this document the technology stack for the Health Care Big Data Analytics proof of concept was presented. The technologies presented here have been included as they represent the best starting point for this investigation. All the technologies presented have seen rigorous use in other projects with comparative demands on performance, flexibility and ease of use.