PhUSE 2013

Paper DH07

Elephants and Storms - using Big Data techniques for Analysis of Large and Changing Datasets

Geoff Low, Medidata Solutions, London, United Kingdom

ABSTRACT

As an industry we are data-led. We gather large quantities of data, both patient and otherwise, often in a variety of formats. Standardisation goes some way towards the marshalling of similar data items, ready for tabulation and comparison. The use of standard data representations has already shown advantages for the collection and storage of large volumes of mostly static data. Big Data techniques can aid organisations in the processing of large volumes of data, but in general these are limited to batch operations for sets of data. In many cases the data itself may change, but the traditional Big Data approaches require a re-processing of all the data for any changes. We are going to talk about an approach that bridges the requirement between the efficiency of batch operations and the requirements to process new data, all in a timely fashion.

INTRODUCTION

We are living in a data-driven economy. Much of the progress made in new directions is around the ability to process data; but what is the scale of the problem? In a word, mammoth - IBM estimated that there were 2.5 quintillion (2.5 x 10^18)[1] bytes of data created every day during 2012, and the rate of generation is predicted to double by 2015. If we take Google Search as an example of a problem that exists today, finding results for each search term requires the parsing of billions of publicly available web pages to extract search terms such that when we look up a term, we get a list of all the sites that hold that particular term. In addition, Google (courtesy of it's famous PageRank system) goes deeper, to also rank a search result for a term against all other search results - each step requiring analysis of more and more data to get the desired result.

Similarly to the wider community, the pharmaceutical industry is continuing to accumulate data; nearly 19,500 new studies were received by clinicaltrials.gov during 2012. Each of these studies will be capturing data that ends up stored in a Clinical Data Warehouse. This new data goes on top of results from the multitude of previously conducted studies. Each of these datasets have tremendous value for us; subjects have participated in the studies and their contribution can contribute to further studies. Setting up virtual studies using existing study data is a fascinating concept - why go through the cost and risk of executing a new study when much of the data to test a hypothesis may already have been captured?

As the amount of data increases, we need different approaches to storing and processing the data. The agreed term dealing with the large amounts of data that industries need to work with is “Big Data”.

WHAT IS BIG DATA?

Big Data is one of those loaded terms that have become very popular with people trying to sell products and solutions. Wikipedia uses the following definition: [2]

> “the term for a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications”.

An alternative (and interesting) definition is offered by Werner Vogels (CTO of Amazon): [3]

> “Big data is when your data sets become so large that you have to start innovating how to collect, store, organize, analyze, and share it”.

The latter definition focuses more on the opportunities that Big Data provides for companies – the data can drive value and being able to deal with this data necessitates innovation. Using either definition, we are clearly at the point where we will benefit from addressing data in different ways to get the best results for those people depending on our

analyses.

There are two main approaches to working with Big Data; scalable batch-based processing and stream-based processing. Batch-based processing is where all data is pooled and processed to give results. Stream-based processing looks at delivering results from a continual stream of data.

Scalable batch-based processing is the technique made popular by Google's MapReduce programming model - Apache Hadoop[4] is the open-source implementation. Implementations of stream-based processing include Storm[5] and Apache S4[6].

SCALABLE BATCH-BASED PROCESSING (MAPREDUCE)

The processing system called MapReduce arose from a paper written by Google about a computing system they are using internally[7]. Dean and Ghemawat laid out their programming model used for processing massive parallelizable datasets using large numbers of commodity computers.

The basis of their approach is to break the computation down to two steps; "a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key"[7].

MapReduce uses a cluster of computing nodes, each of which is a 'commodity computer'. A master node splits the source dataset up into *M* partitions that are dispatched to the *M* mapper nodes. Each mapper node executes a map function on the partition of data it is allocated to output a set of values grouped by key and writes the outcome of the result to the local file system as an intermediate dataset. To speed up the overall calculation time, the mapper may also aggregate results *in situ* (using a combiner). It notifies the master node that has completed, and sends the location of the results to the master node.

When all the map tasks are complete, then the master node partitions the key space into *R* blocks (where *R* corresponds to the number of nodes allocated as reducers). It assigns one block to each reducer node. The reducer node retrieves the intermediate datasets from the mapper nodes and then carries out the reduce function on aggregated datasets. When complete, the reducers write out the final results and the master node aggregates them.

Failure is an important consideration - consider that a MapReduce cluster may be made up of hundreds of thousands of machines and manufacturers accepted failure rate could equate to 1000s of nodes or failures of system interconnects can bring down multitudes of nodes. It is important that the failure of a single or multiple nodes don't prevent the completion of a computation or lead to loss of source data. There are many levels of health monitoring to reassign any jobs that fail and ensure that data integrity is maintained.

An important component of a Big Data system is a distributed file system. Distributed file systems both solve the problem of the volume of data being too large for a single file system and fault tolerance by partitioning across many nodes in a series of small uniform sized blocks. The blocks are replicated across multiple nodes to provide failure-recovery. Small blocks are used as it makes it simpler to move them between machines for calculation and it aids in for indexing.

To illustrate the MapReduce approach, we use the example of a word count. The problem we seek to resolve is we have a large number of documents from which we want to get a count of the incidence of the words across all documents. As an example, this would be useful in the case of building training datasets for Machine Learning.

Using MapReduce approach the following steps are applied:

1. The Master node partitions each document into a number of smaller blocks (e.g. lines) and send these off to mapper nodes.
2. Each mapper node applies the map function. In the word count example the map consists of normalizing the text by unifying the case, removing non-alphanumeric characters case and tokenizing. The map phase emits a tuple (e.g. (word, 1)) for each word in the block. In the emitted tuple the first element is the key. The key is used to define the key space to partition intermediate datasets for the reduce step.
3. Each Reducer node sums the total counts for the each word it has been assigned and returns the results to the operator.

A representation of the word count example is show in Figure 1

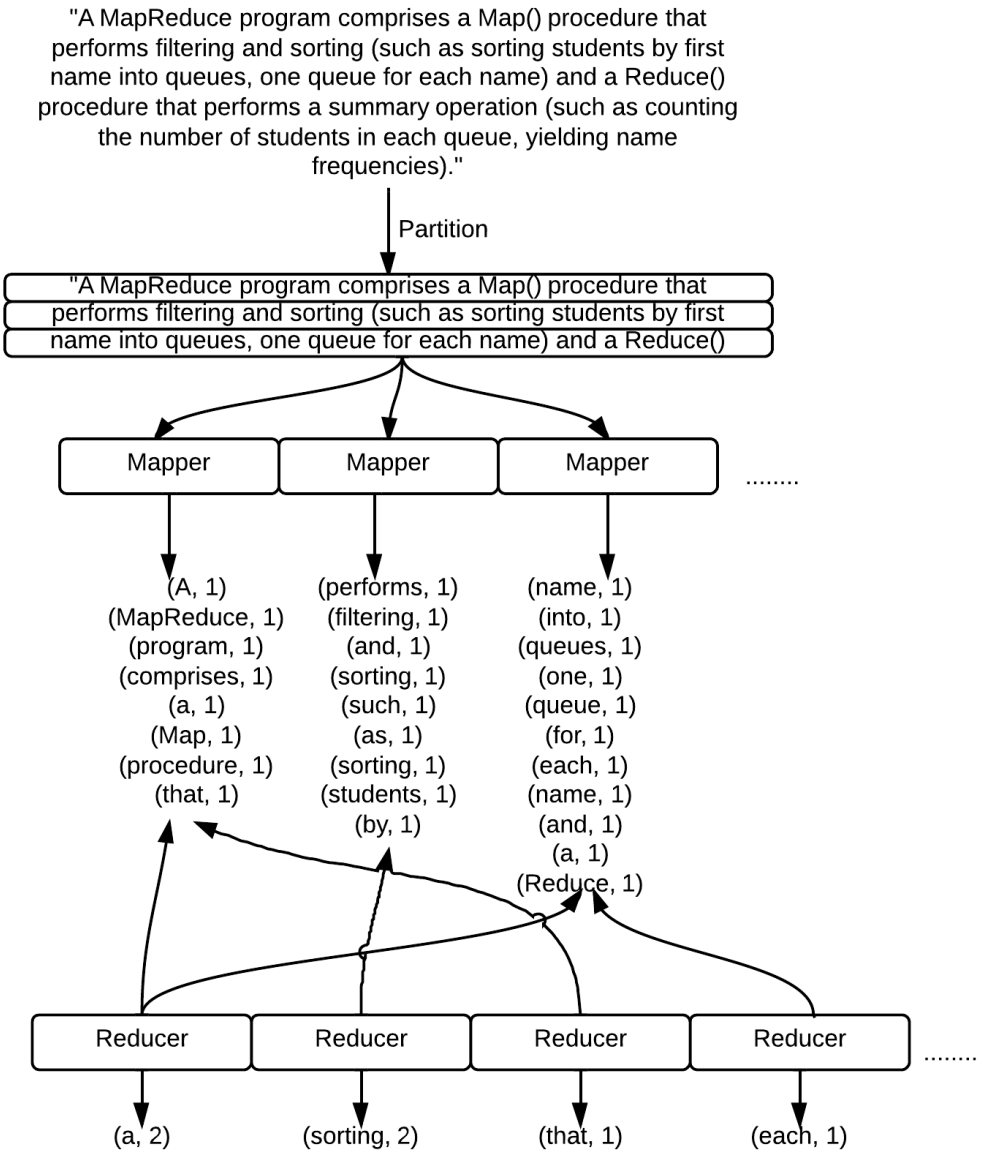

Figure 1 Word Count Example using MapReduce

This 'simple' approach performs exceedingly well. Dean and Ghemawat show a couple of examples; one of which is the MapReduce application to the TeraSort benchmark (sorting 10^{100} 100 byte records, approximately 1 Tb of data).[8] The authors reported a completion time of 832 seconds, representing a 22% improvement over the previous best time[7]. Subsequently, organizations have continued to push the calculations using MapReduce, with the 2013 record producing a time of 54 seconds using Hadoop.[9]

The use of commodity computing is a huge advantage, giving a lower cost entry point as well as a more tunable runtime – machines can be added and removed from MapReduce clusters with ease. A cluster can be tuned for specific job requirements; say by increasing the number of nodes available or increasing the resources available to each node.

Apache Hadoop is an open-source implementation of the MapReduce programming model. Hadoop uses a distributed file system called HDFS[10]. Having an open source implementation has encouraged the development of other projects to augment the Hadoop ecosystem. Some examples include:

- Apache Hive, a Data Warehouse tool on top of HDFS.[11]
- Apache HBase, a non-relational distributed database on top of the HDFS.[12]
- Cloudera Impala, a real time query engine for HDFS.[13]

Each product adds more capability to the Hadoop platform and many are implementations of publications by Google detailing closed products used internally. The development ecosystem is very active and capabilities are being added or refined continually, the Apache Foundation is a vibrant community for development. Other organisations such as cloud-computing providers have embraced Hadoop and customized it for their platforms.[14] The MapReduce approach is a good fit for the Cloud and Hadoop forms an important component of business analytics in the cloud. Having Hadoop as a presence in the Cloud takes away some of the adoption overhead for companies wishing to experiment, as the requirement for significant outlay on hardware for a proof of concept in negligible.

One possible concern for an organization seeking to adopt is that MapReduce does not natively use SQL. This may present a problem for existing analysts who need to learn some programming skills to define the map and reduce functions. However, some of the already mentioned projects go some way towards allowing analysts to get further using their existing skills. Apache Hive uses an SQL-like language, which is under continual development to add more ANSI-SQL features to it (as well as performance optimizations). Cascading Lingual is another example of an application framework that layers an ANSI-SQL compatible interface (and can be used like JDBC) to Hadoop.[15] The development of tools like these brings the cost of adoption down with reuse of existing skills becoming easier.

The MapReduce programming model has inherent latency. A MapReduce computation has to execute on **all** data as it can't be assumed that data is necessarily clustered – all data must have completed the map phase prior to the reduce phase commencing. Other factors that contribute to latency include; coordinating large numbers of machines, needing to move data to and from nodes, relying on network connectivity. The latency leads to some delay in getting results to queries. In most cases we don't expect to query the data directly, rather waiting for a batch-process to generate the analytical datasets from which we derive our results. The batch-based approach presents a gap when trying to analyze continually updating data.

Like any technological approach, it is not necessarily the solution for all problems. For best return on investment it is important to use it for applicable problems. We have looked at scalable batch-based processing and now consider stream-based processing.

STREAM-BASED PROCESSING

A stream-based processing processes data as it comes into a system, not using a persistent data store. Storm is an example of a stream processing engine[5]. It takes an unbounded stream of data from a source (such as a Twitter) as input and operates directly on it. The processing of data is done in real or near-real time; Storm describes itself as a real-time computation engine.[5]

As a computation engine, Storm operates using a few novel key concepts:

- Stream - an unbounded sequence of data
- Spouts - source of the Stream
- Bolts - a node for manipulation of the Stream

These components are assembled into a Topology and the Topology is deployed to the Storm infrastructure. An example Topology is illustrated in Figure 2

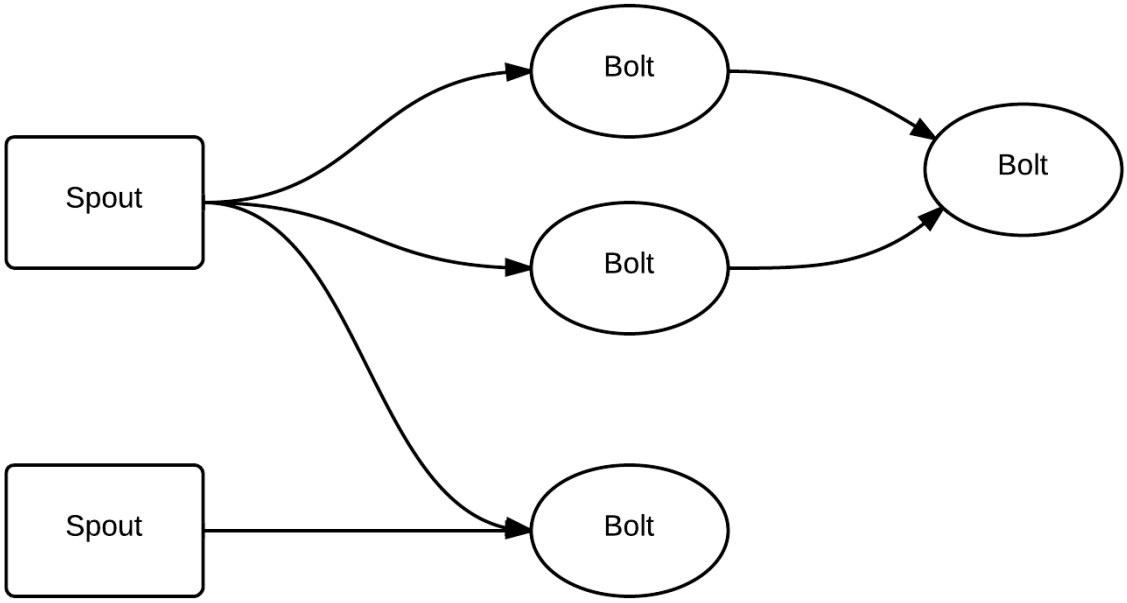

Figure 2 Storm Topology

As shown in the Figure 2, the data (or Stream) is diverted down execution paths by code in the Bolts. Bolts can transform, split or merge the Stream. Bolts can be added or updated on an already executing Topology without taking the whole platform down. It presents as a very adaptable processing engine.

An example of the use of a Storm-based processing engine is categorizing a Twitter stream. A pharmaceutical company may want to carry out some tracking and categorization of mentions of a drug on the market. They can subscribe to Tweets (acting as a Spout) and then push as a Stream through a Storm Topology. As an example the Topology emits a count each time the drug is mentioned. The analytics can be further refined to carry out some gender and age-classification using Social networking analysis techniques to emit counts to a series of buckets for overlaps of the different criteria (e.g. Female + Teenage, Male + Teenage).

Stream-based processing will give an organization information about what is happening right now and facilitate business decisions with the information available in real time. However, stream-based processing only applies to data while it is operating; a Stream enters the Topology, some actions are taken and then it leaves. After the Stream has left the Topology there is no further opportunity to glean any more insights from it. There is no historical data in a stream-based processing engine.

BIG CHANGING DATA?

We now have described the two methodologies for processing Big Data. Each has been shown to have both strengths and weaknesses from the perspective of the queries we might wish to raise. The ideal solution would make it possible to query both historical and the real-time datasets. A solution has been proposed to use MapReduce for the batch-based and the stream-based processing to give datasets accommodating these requirements. This design is called the Lambda Architecture.

THE LAMBDA ARCHITECTURE

The Lambda Architecture was suggested by Nathan Marz (the creator of Storm) as a way to merge the real-time advantages of stream-processing systems with the batch-processing abilities.[16] His central idea is that the most general purpose definition of a query is this:

query = function(all data)

That is, the purpose of a data system is to compute arbitrary functions on arbitrary data. We need the capability to be able to carry out any calculation on all data, at any point of time.

For this to be possible the authors defined some requirements that must be fulfilled. The data system should support:[16]

- Fault-tolerance – tolerant of system and human errors
- Low latency – returning results with minimal wait
- Scalability – maintain performance with increased datasets
- General – support a wide range of applications
- Extensibility – minimal effort to add new computations
- Ad-hoc queries – support the ability to take arbitrary 'looks' at the data at any point

The Lambda Architecture was proposed to satisfy these criteria. A representation of the Lambda Architecture is shown in Figure 3

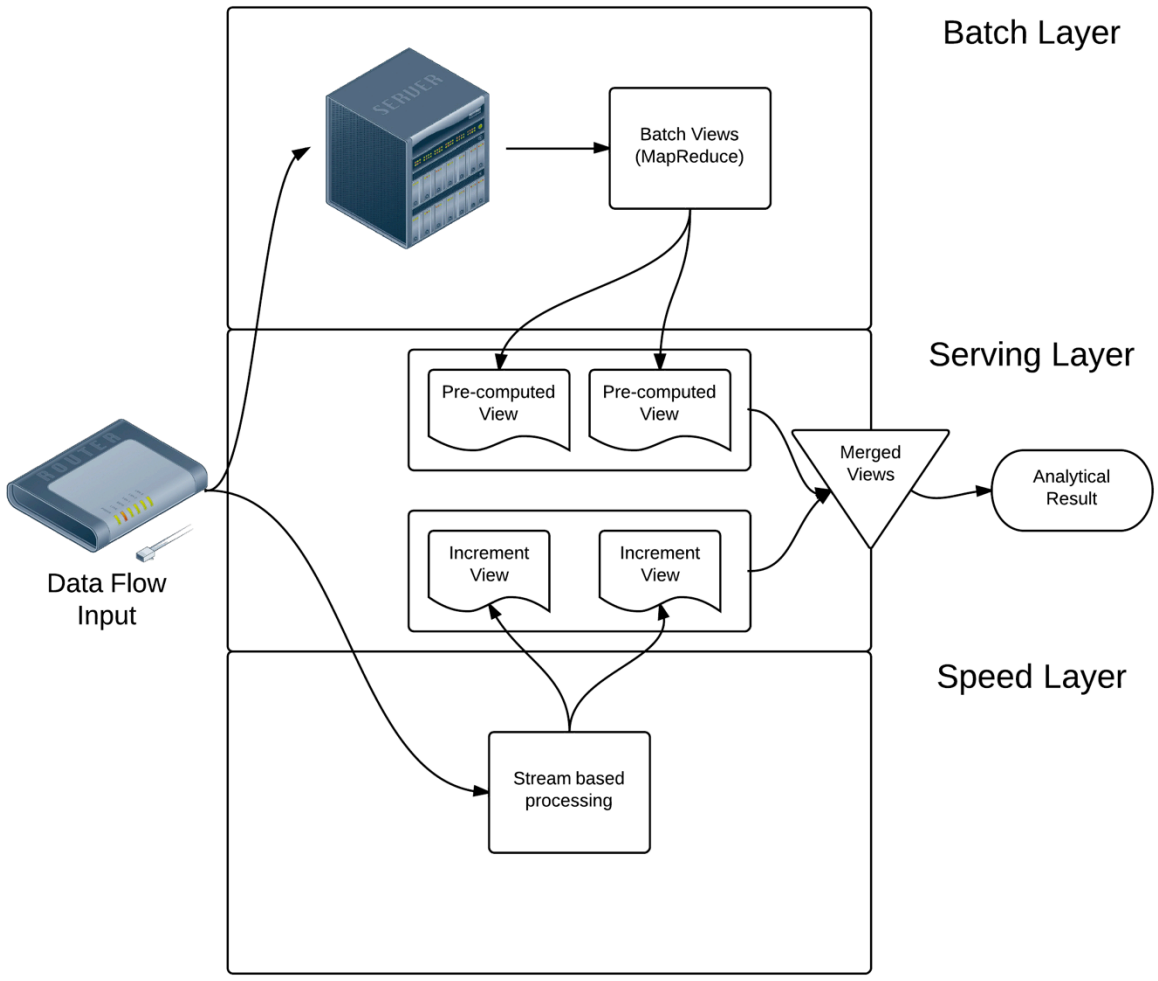

Figure 3 The Lambda Architecture

There are three layers involved:

- The batch layer contains the master dataset in a fault-tolerant distributed file system. The batch layer continually generates the pre-computed views using Hadoop.
- The serving layer indexes and loads the pre-computed views. The serving layer is updated each time the batch layer completes generation of the pre-computed views.
- The speed layer is responsible for generating the incremental views as data is received using Storm. The incremental views include the data entering the system while the pre-computed views are being generated.

Queries are resolved by merging the pre-computed and incremental views. The batch and serving layers satisfy all criteria except for low latency. The speed layer satisfies the remaining criterion with the incremental views including data generated while the pre-computed views are being built. An immutable append-only source dataset is a

fundamental requirement. The immutable append-only data source enables fault-tolerance (any human errors can be flushed by disposing of the views which has no penalty apart from a temporary increase in latency), ad-hoc queries (all the source data remains for future evaluations) and extensibility (new views can be built that include emergent and historical data).

The Lambda Architecture provides a powerful analytics engine providing analysts with a fault-tolerant environment capable of answering queries that combine the historical and the real time views. The architecture combines on the strengths of each technique to given the best solution.

BIG DATA USE CASES IN PHARMA

We consider some possible use cases for Big Data techniques in the domain of Clinical Research.

REPORTING

The more informed a company is on the conduct of its operations, the better placed it is to realize efficiencies. Large amounts of data can become an impediment to good reporting, as the time required to provide results can scale terribly. Big Data techniques make it simpler to generate stratified reports quickly. Using the map function to partition a transactional feed by date make summarizing transactions by hour, day, week, month or quarter simple and using the Lambda Architecture provides the capability for continual real time reporting. Maintaining the source data for a long period leads to the ability of continually adapting to changing reporting requirements.

INTELLIGENT USE OF SOCIAL NETWORKS

Market intelligence can pull from an ever-expanding selection of data sources when looking at the impact of a drug. One of the most valuable is social media, with consumers often referencing a drug or indication. Using tools like the Lambda Architecture we can mine the large feeds of information to pull out interesting messages that can form the basis of impression analyses. The large amounts of information that people share could mean these techniques may serve in trial recruitment. A recent study regarding cancer trials showed that clinical trials were offered to patients only 20% of the time, but of those 75% accepted[17]. Further investigation identified that 32% said they would be very willing to participate in a clinical trial if asked. If, using these techniques, it is possible to identify candidates for inclusion then studies can be bought to subjects; the statistics suggest that there is a good probability of a successful match benefitting the patients.

DISTRIBUTED ETL

Simple transformations can be achieved using standard libraries with MapReduce. Examples include mapping a raw date string to an ISO8601 format or populating standard and raw variables given the datatype. Assuming that rows are independent, then it is possible to parallelize mapping processes across many workers. By minimizing the time required to generate the transformed dataset then there the requirement for holding many copies of intermediate datasets can be mitigated, as it becomes possible to regenerate the required datasets from the source data on demand (or even continuously). In addition, the suitability of MapReduce for sorting of large datasets has previously been shown.

The MapReduce approach can assist ETL processes; mapper nodes can be configured to execute SQL queries against a database making it possible to parallelize extracting data from databases. An open source tool called Apache Sqoop uses this technique to extract data from a RDBMS into HDFS.[18] If an enterprise service bus or equivalent messaging layer is attached as a Spout, then real time transformations are possible feeding data directly into standardized analysis platforms.

STUDY STATUS REPORTS

Much of the data in clinical studies needs to be processed to provide overall metrics for the study; for example, how many CRFs pages need reviewing? These status metrics need to be processed from the point at which changes are made (the CRF level) up to the subject, site or study level. The status at any level is constituted of the status of its children (and their children, etc.). Viewing the hierarchy as a tree, it is possible to split the overall calculations as a series of sub-calculations based on bifurcation points, which can be processed independently. The child calculations are recursively computed upwards, with the result of each level giving the status of its parent. This is a non-conventional MapReduce calculation, as multiple ‘cycles’ contribute to the overall calculation.

CONCLUSION

Big Data is a valuable approach and as an industry we should have a strategy for incorporating it into our data practices. The advantages that can be gained by enabling rapid processing of large amounts of data and being able

to use the results to make informed decisions are significant. As we see with the Apache Hadoop project, many tools are built up around the central platform that lower the cost of adoption for organizations seeking to embrace these new technologies. The definition of Big Data being a problem that necessitates continual innovation presents many opportunities for improvements in what can be achieved with data generated by clinical studies, benefitting the industry and those depending on what it produces. Data analysis becomes less bound by data logistics; both in terms of storage considerations and time-to-generate considerations; this will open the organization to wider ranges of approaches to data processing and the types of analytics that can be attempted. The process becomes data driven, which is where it should be – subjects have contributed their data and we should be able to get maximum value from this for their sake.

We have provided an overview of some core Big Data techniques and shown how the techniques can be used to deliver data analytics for clinical datasets, both static and changing.

REFERENCES

1. "What is Big Data", IBM, (http://www-01.ibm.com/software/data/bigdata/) - Retrieved 2013-08-31
2. "Big Data", Wikipedia, (http://en.wikipedia.org/wiki/Big_Data) - Retrieved 2013-09-03
3. "Werner Vogels on the definition of big data, and the bandwidth of FedEx boxes", Alex Wilheim, (http://thenextweb.com/insider/2012/08/23/amazons-cto-here-two-definitions-big-data/) - Retrieved 2013-09-01
4. Apache Hadoop, Apache Foundation, (http://hadoop.apache.org) - Retrieved 2013-09-01
5. Storm, Storm Project (currently under evaluation for joining the Apache Foundation), (http://storm-project.net/) - Retrieved 2013-09-01
6. S4 distributed stream computing platform, Apache Foundation, (http://incubator.apache.org/s4/) - Retrieved 2013-09-01
7. Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (January 2008), 107-113. DOI=10.1145/1327452.1327492 http://doi.acm.org/10.1145/1327452.1327492
8. "Spsort: How to sort a terabyte quickly.", Jim Wyllie. (http://alme1.almaden.ibm.com/cs/spsort.pdf.)
9. "MapR Sets New TeraSort Benchmark Record", MapR, (http://www.hpcinthecloud.com/hpccloud/2012-10-24/mapr_sets_new_terasort_benchmark_record.html) - Retrieved 2013-09-02
10. HDFS Architecture Guide (http://hadoop.apache.org/docs/stable/hdfs_design.html)
11. Apache Hive, Apache Foundation, (http://hive.apache.org) - Retrieved 2013-09-04
12. Apache HBase, Apache Foundation, (http://hbase.apache.org) - Retrieved 2013-09-04
13. "Cloudera Impala: Real-Time Queries in Apache Hadoop, For Real", Cloudera, (http://blog.cloudera.com/blog/2012/10/cloudera-impala-real-time-queries-in-apache-hadoop-for-real/) - Retrieved 2013-09-09
14. "Amazon Elastic MapReduce (Amazon EMR)", Amazon, (http://aws.amazon.com/elasticmapreduce/) - Retrieved 2012-09-08
15. Cascading Lingual, Cascading, (http://www.cascading.org/lingual/) - Retrieved 2013-09-04
16. Nathan Marz and James Warren, "Big Data - Principles and best practices of scalable realtime data systems", Manning Publications Co., (2014)
17. "Why Bother Offering That Clinical Trial? An Internal Dialogue", William M. Sikov, (http://connection.asco.org/Commentary/Article/ID/3650/Why-Bother-Offering-That-Clinical-Trial-An-Internal-Dialogue.aspx) - Retrieved 2013-09-09
18. Apache Sqoop, Apache Foundation, (http://sqoop.apache.org/) - Retrieved 2013-09-09

ACKNOWLEDGMENTS

I would like to thank the management at Medidata Solutions for giving me the time to research these techniques and the opportunity to try them out!

RECOMMENDED READING

Nathan Marz and James Warren, "Big Data - Principles and best practices of scalable realtime data systems", Manning Publications Co., (2014)

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Author Name: Geoff Low
Company: Medidata Solutions
Address: Metro Building, 7th floor, 1 Butterwick, Hammersmith
City / Postcode: London W6 8DL.
Work Phone: +44 (0) 208 6006504
Email: glow@mdsol.com
Web: http://www.mdsol.com