Lambda Architecture for Real Time Big Data Analytic

Zirije Hasani* Margita Kon-Popovska** Goran Velinov***

Faculty of Computer Science and Engineering

Ss. Cyril and Methodius University of Skopje

Skopje, Macedonia

zh12796@seeu.edu.mk* margita.kon-popovska@finki.ukim.mk**
goran.velinov@finki.ukim.mk***

Abstract. Defining the environment for analyzing streamed big data in real time is not an easy task. There are many architecture proposals for real time big data analytic, but the most interesting one for our problem is Lambda Architecture. In this paper we are presenting motivation for developing such architecture, how it works and our practical work for implementing it.

Lambda Architecture is comprised by three layers batch, speed and serving layer. Thus far we have implemented the batch layer employing Hadoop framework. We also briefly review the other two layers in order to implement them in the next phase of our work, where for serving and speed layer we conclude that Storm is the best choice. Practical example demonstrates the analytical process in Hadoop for analyzing Wikipedia text data.

Keywords: Hadoop, Lambda Architecture, text data, Storm.

1 Introduction

There is a lot of research for technologies and architectures for computing the arbitrary function on arbitrary dataset in real time, but none of them proposes single tool for this real time Big Data analytic task. Instead variety of tools and techniques are utilized. In this paper we describe Lambda Architecture, proposed by Nathan Marz. This architecture solves the problem of computing arbitrary functions on arbitrary data in real-time by decomposing the problem into three layers: the batch layer, the serving layer, and the speed layer. It blends in the same system Hadoop for the batch layer and Storm[1] for the speed layer. Incoming stream is handled by Kafka, while for the server layer beside one among Storm HBase, Casandra also Oracle, SAP, SploutSL and other technologies can be used (Fig.1). The properties of the system

[1] http://storm-project.net/

This work was partially financed by the Faculty of Computer Science and Engineering at the "Ss.Cyril and Methodius: University

are: robustness and fault tolerance, scalability, generality and extensibility. It allows ad hoc queries, need minimal maintenance and is debug gable.

General motivation for this work is to explore capabilities of this architecture for various datasets and in the future compare various tools used on separate levels, and/or compare Lambda Architecture with other tools and technologies for Big Data stream analytics. Until now we have implemented merely the batch layer, using Hadoop. We conclude that for the speed and serving layer we will examine implementation of the Storm. As an illustration we examine some analytics for Wikipedia text data.

Paper is structured as follows: In the second chapter we explain what Lambda Architecture is, how it works by describing its three layers (batch, speed and serving layer) and its supporting technologies; in the third chapter, we show how to implement this architecture; in the fourth chapter we present some examples for analyzing Big Data with batch layer (Hadoop); in the least chapter we give conclusion for our work done thus far, including also ideas for future work.

2 Architecture for real time Big Data analytic

The two general requirements of big data projects are common: analysis of the (near) real-time information extracted from a continuous inflow of data and persisting analysis of a massive volume of data. Solutions, each addressing these separate cases, have become popular in the recent years: Storm [4], an open source, distributed, real-time computation platform (used by companies like Twitter and Groupon, recently adopted in the Apache foundations incubator) and Hadoop [7], well known open source platform widely adopted for batch processing.

The problem with these approaches is that nowadays business requirements need both approaches - historic and real-time simultaneously. Many organizations recognize the challenge of convergence that happened with time: extraction of real-time data and analysis of massive stored volumes of data. Real-time data (or at least portion of them) accumulates with time and the certain demand for an aggregated with historic view arises, that requires batch processing. The batch processing solution is usually slow, while business users are asking instant or near real-time insight, such as the most recent data updates to react faster to market changes.

One solution to this problem is creation of hybrid architecture. Several such architectures were proposed recently [4], one being from Nathan Marz, the creator of open source projects Storm and Cascalog, named as the Lambda Architecture [2]. Next we are going to describe the Lambda Architecture and discuss the three layers that compose it.

2.1 Introduction to Lambda Architecture

Motivation given by Marz and his team for building hybrid Lambda Architecture system is [2]:

- The need for a robust system that is fault-tolerant, both against hardware failures and human mistakes.
- To serve a wide range of workloads and use cases, in which low-latency reads and updates are required. Related to this point, the system should support ad-hoc queries.
- The system should be linearly scalable, and it should scale out rather than up, meaning that throwing more machines at the problem will do the job.
- The system should be extensible so that features can be added easily, and it should be easily debuggable and require minimal maintenance.

Existing technologies as single tool [4] for Big Data streaming analytics do not offer possibility of computing arbitrary functions on an arbitrary dataset in real time. Instead, to solve this daunting problem we should use a variety of tools and techniques to build a complete Big Data system. The Lambda Architecture decomposes the problem into three layers: the batch layer, the serving layer, and the speed layer [2]. Batch layer contains constantly growing master dataset stored on a distributed file system like HDFS and produce batch views. For processing the batch data MapReduce is used, which is the Hadoop programming model. Serving layer load and expose the batch views in a data store from where they can be queried. This serving layer data store does not require random writes, but must support batch updates and random reads and therefore can be extraordinarily simple [3]. The last layer is speed layer that deals only with new data and compensates for the high latency updates of the serving layer. At this layer stream processing system like Storm is used that deals with recent data only to compute the real time views. These computed views remain valid until the data have found their way through the batch and serving layer [3].

2.2 Description of Lambda Architecture layers

Lambda Architecture three major components interact with new incoming data and respond to queries. Figure 1 gives an overview of components, processes, and responsibilities that compose every it's layer.

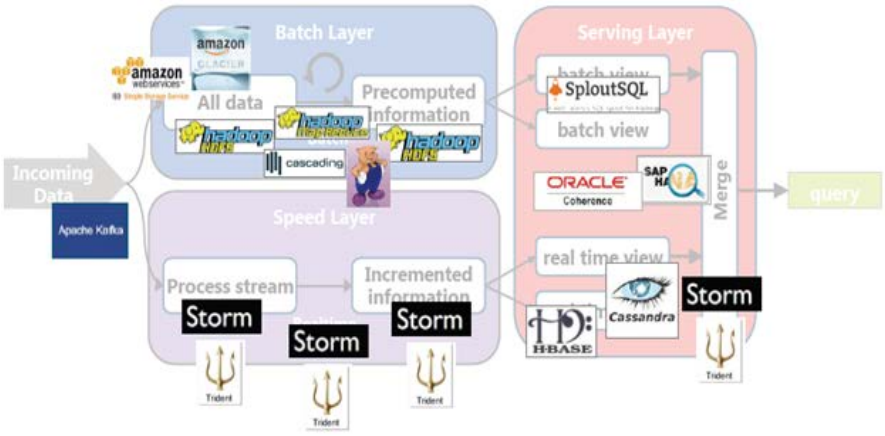

Fig. 1. Overview of the Lambda Architecture [5].

Incoming data are dispatched to both the batch layer and the speed layer for processing. Apache Kafka is the technology for implementing this process.

The batch layer has two functions: managing the master dataset, an immutable, append-only set of raw data, and to pre-compute arbitrary query functions, called batch views. Hadoop's HDFS is typically used to store the master dataset and perform the computation of the batch views using MapReduce. This process is continuous operation, so when new data arrive they will be aggregated into the views during the next MapReduce re-computing iteration. Its output is always outdated by the time it is available since new data has been received in the meantime. The views are computed from the entire dataset and therefore the batch layer is not expected to update the views frequently. Depending on the size of the dataset and cluster, each iteration could take hours [10].

The serving layer indexes the batch views (usually flat files) so that they can be queried with low latency in ad hoc manner. To implement the serving layer, usually technologies such as SploutSQL, Oracle, HBase, Cassandra and Storm are utilized.

Speed layer compensates for the high latency of updates to the serving layer, due to the batch layer. The speed layer deals with the most recent data only in real-time [2] by computing real-time views and Storm is often used to implement this layer.

Opposite to the batch layer that is designed to continuously re-compute the batch views from scratch, the speed layer uses an incremental model whereby the real-time views are incremented as and when new data is received. As soon as the data propagates through the batch and serving layers the corresponding results in the real-time views can be discarded

Any query against the data is answered by querying both the speed and the batch layers, and the result is merged to give a near real-time view on the complete dataset. For our case we are going to use Storm because it is open source and is also used by many large companies as Groupon, Alibaba, and The Weather Channel.

3 Implementation of Lambda Architecture

There are many and various technologies that can compose Lambda Architecture. In this paper we are demonstrating the Hadoop implementation of the Batch Layer and illustrate its work for Wikipedia text data. Our research in the future will be to implement also the other two layers (speed and serving layers) in order to test how adequately we can analyze data stored in Hadoop and data that came in real time.

3.1 Hadoop and how it works

Hadoop is open-source software project developed by Apache. It presents a framework that enables distributed processing of large data sets across clusters of computers using programming models. It is designed to function from a server and thousands of machines, each offering his capacity to process and store information. Rather than rely on hardware to deliver high performance, the library is designed to deal with shortcomings on the application level by offering high performance cluster computers. It is licensed under the Apache v2 license. Hadoop comes from Google's MapReduce and Google File System, [7].

Platform was designed to solve problems having large collections of mixture of complex unstructured, semi-structured and structured data not fitting well into tables, sources including log files, social media feeds, internal data stores and other. It's good for running analytics that are deep and computationally extensive, like clustering and targeting.

Data are broken into parts and then loaded into file system made up of multiple nodes running on commodity hardware. The default file store in Hadoop is the Hadoop Distributed File System, or HDFS. HDFS is adopted to store large volumes of data not requiring them to be organized into relational rows and columns, [7].

Every part of data is replicated more times and loaded into the file system. In case of the node failure another node has a copy. The role of NameNode is to act as facilitator, communicating back to the client information such as which nodes are available, where in the cluster certain data resides and which nodes have failed.

After the data is loaded into the cluster, it is ready to be analyzed via the MapReduce framework. The client submits the job to one of the nodes in the cluster known as the Job Tracker. The Job Tracker refers to the Name Node to determine which data it needs to access to complete the job and where in the cluster that data is located. Once determined, the Job Tracker submits the query to the relevant nodes. Rather than bringing all the data back into a central location for processing, processing then occurs at each node simultaneously, or in parallel [7].

After completed the job each node stores its own result. The client initiates a "Reduce" job through the Job Tracker after which results of the map phase stored locally on the individual nodes are aggregated to determine the "answer" to the original query and are loaded to another node in the cluster. Client can access these results, which can then be loaded into one of number of analytic environments for analysis [7]. The main components of Hadoop are:

- Hadoop Distributed File System (HDFS): The default storage layer in any given Hadoop cluster;
- Name Node: The node in a Hadoop cluster that provides the client information on where in the cluster particular data is stored and if any nodes fail;
- Secondary Node: A backup to the Name Node, it periodically replicates and stores data from the Name Node should it fail;
- Job Tracker: The node in a Hadoop cluster that initiates and coordinates MapReduce jobs, or the processing of the data.
- Slave Nodes: The grunts of any Hadoop cluster, slave nodes store data and take direction to process it from the Job Tracker.

After the completing the MapReduce phase, this processed data are ready for analyses in the next phase where are combined data from speed layer and batch layer of Lambda Architecture in order to produce answer to a given query.

3.2 Hadoop Implementation of Batch Layer in Lambda Architecture

The Batch layer stores the master copy of the dataset and pre-computes batch views (arbitrary functions) on that master dataset (fig.2). Hadoop is the typical example of a batch processing system.

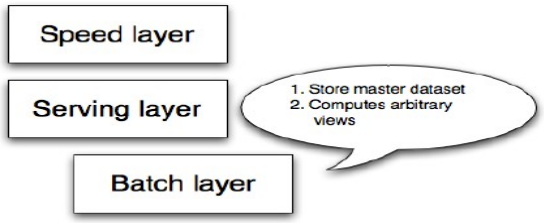

Fig. 2. Batch layer jobs [2]

Figure 3 illustrates the main principle of the Hadoop implementation by utilizing MapReduce. It divides the large problem into a sub-problems (mapping), performs the same function on each sub-problems and finally combines (reduce) the output from all sub-problems.

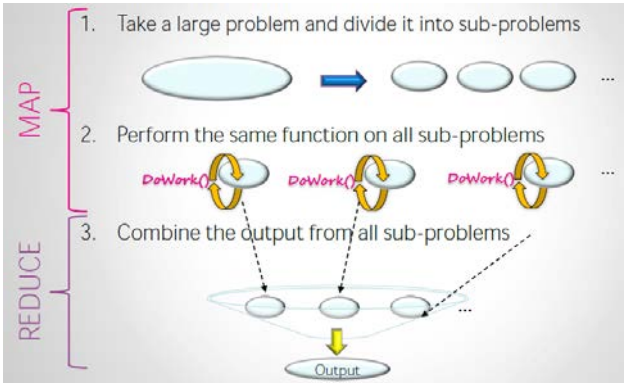

Fig. 3. MapReduce works, [8].

The batch layer can be represented in simple form in pseudo-code [2] showing that it continuously re-computes the batch views from scratch:

```
function runBatchLayer():
while(true):
recomputeBatchViews()
```

Although more performing would be implementation of the incremental MapReduce algorithms to increase the frequency of the batch views, this approach trade performance for human fault-tolerance, enabling easier fixing any problem in the views eventually caused by human errors or other factors. Other implementation characteristics are unrestrained computation, no need of de-normalization, horizontal scalability. Also though written as single-threaded program it automatically parallelizes across a cluster of machines, thus enabling analysis of datasets of any size. Following example illustrates batch layer computation [2].

```
Pipe pipe = new Pipe("counter");
pipe = new GroupBy(pipe, new Fields("url"));
pipe = new Every(
  pipe,
  new Count(new Fields("count")),
  new Fields("url", "count"));
Flow flow = new FlowConnector().connect(
  new Hfs(new TextLine(new Fields("url")),srcDir),
  new StdoutTap(),
  pipe);
flow.complete();
```

The job of this code is to compute the number of pageviews for every URL, given an input dataset of raw pageviews. It can be automatically distributed on a MapReduce cluster, scaling to number of available nodes. E.g., if having n nodes in MapReduce cluster, the computation will finish about n times faster than using only one

node! At the end of the computation, the output will directory contain a number of files with the results [2].

3.3 Hadoop implementation at FINKI working environment

Hadoop is implemented at FINKI [6] in 2013. HDFS have 2 NameNode and six DataNodes at HDFS. Installed are one MapReduce JobTracker and six TaskTracker, and in HBase we have one HBase Master and six RegionalServers. This architecture we adopt as our batch layer in Lambda Architecture. In this architecture we analyze Wikipedia text data show by the example below.

3.4. Implementing Lambda Architecture with Amazon Web Services

In order to use the Amazon Web Services we need to create one account[2] authorized to use Amazon EC2 with Storm-deploy which is a Clojure project, based on Pallet. The storm-deploy project works by contacting Amazon EC2 and requesting new instances. It then installs the needed software, such as Storm, Zookeeper and needed libraries. After configuring the new instances, the deployed system will include your local public ssh key in the authorized keys on the new instances, such that only you can connect to them. Your system is "attached" to the recently deployed cluster, such that the storm/bin/storm script knows which cluster to execute commands on [9].

Our further research will be utilization of this infrastructure for analyzing data in order to have some statistics how this architecture work with Amazon web services and to compare also with the other architecture where Storm and Kafka will be installed and conclude which of approaches is better choice.

The reason why we need to implement this architecture is that we have to analyze and compare transactions done in one defined period in the past, with the transactions that came in real time.

4 Wordcount with pig Hadoop for Wikipedia text data analyze

In working environment installed at FINKI we have executed several jobs. In this paper we analyze Wikipedia text data by using Pig. Apache Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs.

In the WordCount pig code, example for word counting, the first line of code load the data from the text file (enwiki-latest-pages-articles-multistream-index1.txt) stored at HDFS. After loading the data from the file, it generates all the character data as word and then cluster data by words. Code retrieves the list of words in the document and calculates the number of occurrences for each word in the document. The result is

[2] aws.amazon.com

saved in the “wordcount2” file. With the cat command we can see the result from “wordcount2” file. Figure 4 shows the result of execution of the WordCount job.

```
A = load './enwiki-latest-pages-articles-multistream-
index1.txt';
B = foreach A generate flatten(TOKENIZE((chararray)$0))
as word;
C = group B by word;
D = foreach C generate COUNT(B), group;
store D into './wordcount22';
```

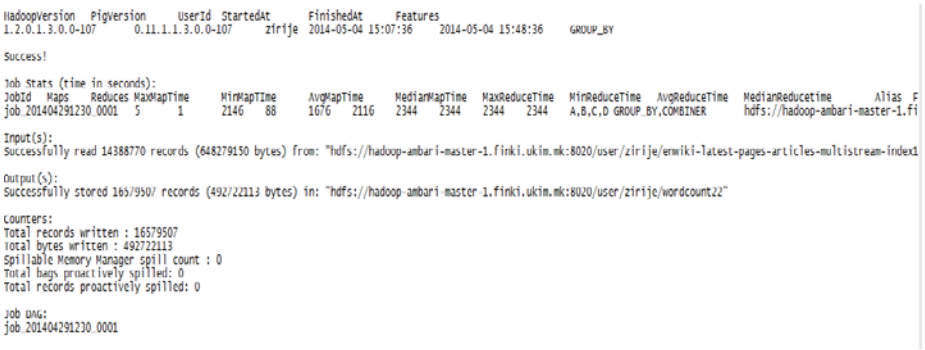

Fig. 4. Result from the completed job of WordCount example

From the examples above we can also conclude that for programming other tools, beside Java, can be exploited. Although Hadoop MapReduce is based on Java and native approach would be Java, other tools as Apache Pig are available. The reason why we use Pig is that it is easier to learn and we have to write less code comparing with Java. This example is illustrated with small amount of data (618 Mb) and the wordcount job for this amount of data was executed in 41 minute as can be seen in figure 4. The job was executed also with large amount of data but takes longer time (8 hours for 6% of completed job for 10 GB data). In figure 5 a small part from the result (wordcount22) of wordcount example for Wikipedia text data is shown.

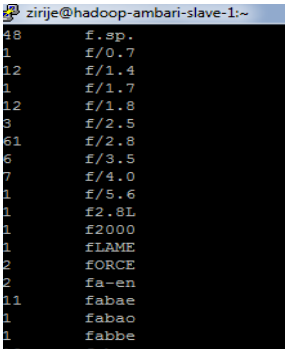

Fig. 5. A part of result from wordcount example

The data analyzed here are pre-stored data (not real time). We use this example to test implementation of the batch layer. In the next phase we are going to analyze transactional data that arrive in real time and transactional data stored before in Hadoop in order to exploit the behavior of the system and its growing by time.

5 Conclusion and future work

The Lambda Architecture is the first approach that handles the complexity of Big Data systems by defining a clear set of principles. Specifically immutability, human fault-tolerance and re-computation are principles that can be easily adopted with the Hadoop platform. Depending on real-time requirements, in many cases the speed layer is not even needed. If omitted, it makes the whole system less complex, but the beauty of the Lambda Architecture is that the speed layer can be integrated later on without a huge hassle.

The idea of coping with real time Big Data stream analytic system is to combine different technologies and to blend the process of batch analytic and real time stream analytic if and when needed. Lambda Architecture is implementing this idea which brings together technologies like Hadoop, Storm and Kafka.

In the future research phase we are going to implement other components of the Lambda Architecture in order to analyze both batch data and real time data. We will test the speed layer with Amazon web services (as an easier task) and also implement Storm in our experimental environment for the speed layer.

References

1. Michael Hausenblas. Applying the Big Data Lambda Architecture, (November 12, 2013). Retrieved April 06, 2014, from http://www.drdobbs.com/database/applying-the-big-data-lambda-architectur/240162604.
2. Nathan Marz, James Warren. Big Data: Principles and best practices of scalable realtime data systems. Manning Publications, 1 edition (October 1, 2013)
3. Christian Gügi. Lambda Architecture, part 1, (8. March 2013). Retrieved April 06, 2014, from http://www.ymc.ch/en/lambda-architecture-part-1

4. Zirije Hasani, Margita Kon-Popovska and Goran Velinov. *Survey of Technologies for Real Time Big Data Streams Analytic*. 11th International Conference on Informatics and Information Technologies, CIIT 11-13 April 2014, Bitola Macedonia.
5. Guido Schmutz. *Kafka and Storm – event processing in realtime*. International Conference for the software community, JAZOON 2013.
6. Bojan Ilioski. Hadoop на FINKI. Skopje, Macedonia. January 2014.
7. Big Data: Hadoop, Business Analytics and Beyond. Retrieved April 16, 2014, from http://wikibon.org/wiki/v/Big_Data:_Hadoop,_Business_Analytics_and_Beyond
8. Nathan Bijnens, Geert Van Landeghem. A real-time architecture using Hadoop and Storm. DataCrunchers, 2013.
9. Kasper Grud, Skat Madsen. Deploying to Amazon EC2 Using storm-deploy (December 27, 2013). Retrieved April 19, 2014, from http://blog.safaribooksonline.com/2013/12/27/storm-deploy-amazon-ec2/.
10. James Kinley, The Lambda Architecture: principles for architecting realtime Big Data systems. Retrieved April 19, 2014, from http://jameskinley.tumblr.com/post/37398560534/the-lambda-architecture-principles-for architecting
11. Storm. Retrieved April 19, 2014, from http://storm-project.net/