International Conference on Innovation and Advances in Interdisciplinary Research
Mewar University, NH - 48 Gangrar, Chittorgarh, Rajasthan
Date: 28th - 29th November, 2022
ISBN : 978-93-91535-29-2

A Review on Lambda Architecture for Big Data Analytic

Tasiu Ado Salisu[1], Dr Ravi Khatwal[2], Abubakar Ibrahim[3]

[1]*Research Scholar Sangam University,* [2]*Asistant Professor Sangam University,* [3]*M.Sc Vivekananda Universty*

1*adosalisutasiu@yahoo.co.uk*, 2*ravi.khatwal@sangamuniversity.ac.in*,
3*abubakarlabaran7@gmail.com*

Abstract.
Every second, smart phone technologies provide potential for data explosion, streaming, and collection from heterogeneous devices. Analyzing these enormous datasets can reveal new unexplored habits and help optimise methods to city-wide applications or societal use cases. However, acquiring and handling these huge datasets offers issues in how to do optimum online data analysis 'on-the-fly,' as current approaches are frequently constrained by capacity, expenditure, and resources. Defining the environment for processing streamed huge data in real time is a difficult issue. There are numerous architecture proposals for real-time big data analytics, but the most interesting one for our review is Lambda Architecture. Lambda Architecture contained three layers,batch, speed and serving layer.

Keywords:*Big data processing, Hadoop, Lambda Architecture, text data, Storm.*

1 Introduction

Real-time processing data and batch (static) data are both handled by a single Lambda architecture. It helps to find a solution to the issue of computing arbitrary functions. With this deployment technique, accuracy is maintained while latency is decreased and tiny errors are retained.There has been a lot of study on technologies and architectures for computing any function on any dataset in real time, but none of them propose a single tool for this real time Big Data analytic work. Instead, a wide range of instruments and approaches are employed. In this work, we describe Nathan Marz's Lambda Architecture. This design decomposes the challenge of computing arbitrary functions on arbitrary data in real-time into three layers: the batch layer, the serving layer, and the speed layer. It combines Hadoop for the batch layer and Storm1 for the speed layer in the same system. Kafka handles the incoming stream, while for the server layer, in addition to Storm HBase and Casandra, Oracle, SAP, SploutSL, and other technologies can be used (Fig.1). The system's qualities include resilience and fault tolerance, scalability, generality, and extension. It supports ad hoc searches, requires little maintenance, and is debug gable. The overall goal of this effort is to investigate the capabilities of this architecture for diverse datasets and, in the future, evaluate various tools employed on distinct levels, as well as compare Lambda Architecture to other tools and technologies for Big Data stream analytics. We have just implemented the batch layer thus far, using Hadoop. We conclude that we will investigate Storm implementation for the speed and serving layer. As an example, we look at some analytics for Wikipedia text data.The following is how the paper is organized: In the second chapter, we explain what Lambda Architecture is and how it works by describing its three layers (batch, speed, and serving layer) and supporting technologies; in the third chapter, we demonstrate how to implement this architecture; in the fourth chapter, we present some examples for analyzing Big Data with batch layer (Hadoop); and in the final chapter, we provide a conclusion for our work thus far, as well as ideas for future work.

1.1 Introduction to LambdaArchitecture

Motivation team for building hybrid Lambda Architecture system is [2]:

❖ The requirement for a fault-tolerant system that can handle both hardware failures and human errors. • The ability to service a wide range of workloads and use cases that require low-latency reads and updates. In relation to this, the system should support ad hoc inquiries.

❖ The system should be linearly scalable and scale out rather than up, which means that throwing more machines at the problem will suffice.

❖ The system should be expandable so that new features may be simply added, as well as easily debug gable and low-maintenance.

Existing solutions, such as a single tool [4] for Big Data streaming analytics, do not allow for the computation of arbitrary functions on arbitrary datasets in real time. Instead, to tackle this big task, we should employ a combination of tools and methodologies to construct a holistic Big Data system. The Lambda Architecture divides the problem into three layers: batch layer, serving layer, and speed layer [2]. The batch layer stores a constantly expanding master dataset on a distributed file system such as HDFS and generates batch views. MapReduce, the Hadoop programming model, is used to process batch data. The serving layer loads and stores batch views in a data store where they may be queried. This service layer data store, which does not require random writes but must enable batch updates and random reads, can be quite basic [3]. The final layer is the speed layer, which deals only with new data and compensates for the serving layer's high latency updates. At this layer, a stream processing system like Storm is utilized to compute real-time views using only recent data. These computed views are valid until the data has passed through the batch and serving layers [3].

1.2 Lambda Architecturelayers

The three key components of the Lambda Architecture interact with new incoming data and reply to requests. Figure 1 depicts the components, procedures, and responsibilities that comprise each tier.

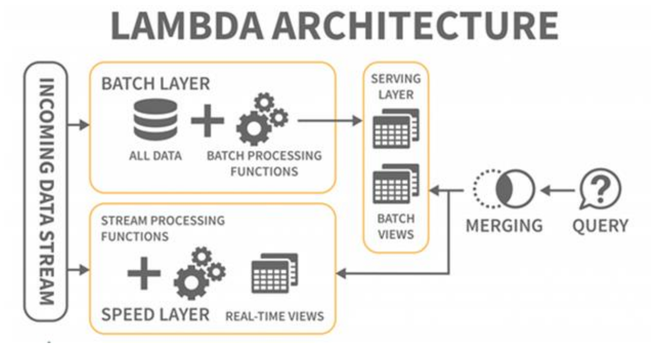

Figure 1. Overview of the Lambda Architecture [5].

Incoming data is routed to the batch and speed layers for processing. The technology used to implement this procedure is Apache Kafka.

The batch layer has two purposes: it manages the master dataset, which is an immutable, append-only set of raw data, and it pre-computes arbitrary query functions known as batch views. Hadoop's HDFS is often used to store the master dataset and MapReduce is utilised to compute the batch views. Because this is a continuous process, fresh data will be collected into the views on the next MapReduce re-computing iteration. Its output is always out of date by the time it becomes available since fresh data has been received in the interim.

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 tothebatchlayer.Thespeedlayerdealswiththemostrecentdataonlyinreal-time[2] by computing real-time views and Storm is often used to implement this layer.

Because the views are calculated from the complete dataset, the batch layer is not anticipated to update the views regularly. Each loop could take hours depending on the size of the dataset and cluster [10].
The serving layer indexes batch views (often flat files) so that they can be queried ad hoc and with low latency. Typically, technologies such as SploutSQL, Oracle, HBase, Cassandra, and Storm are used to create the serving layer.
The speed layer compensates for the batch layer's excessive latency in updating the serving layer. Only the most current data is dealt with by the speed layer in real-time.

Storm is frequently used to implement this layer [2] by computing real-time views.

In contrast to the batch layer, which is meant to constantly recompute the batch views from scratch, the speed layer employs an incremental model in which the real-time views are incremented as new data is received. Once the data has passed through the batch and serving layers, the related results in real-time views can be removed. Any query against the data is answered by querying both the speed and batch layers, and the results are combined to provide a near real-time view of the entire dataset. Storm will be employed in our scenario because it is open source and is used by a number of significant firms such as Groupon, Alibaba, and The Weather Channel.

2 Implementation of Lambda Architecture

Lambda Architecture can be composed of a wide range of technologies. In this paper, we demonstrate the Hadoop Batch Layer implementation and show how it works with Wikipedia text data. Our future research will include implementing the other two layers (speed and serving layers) to see how well we can analyses Hadoop data as well as real-time data.

2.1 Hadoop and how it works

Apache Hadoop is an open-source software project. It describes a paradigm for distributed processing of big data sets across computer clusters utilising programming models. It is intended to run on a server and thousands of machines, each with its own capability to process and store data. Rather than relying on hardware to deliver high performance, the library is intended to address application deficiencies by providing high performance cluster computers. The Apache v2 licence applies to it. Hadoop is based on Google MapReduce and the Google File System [7].

The platform was created to address issues involving enormous collections of complicated unstructured, semi-structured, and structured data that do not fit well into tables, as well as sources such as log files, social media feeds, internal data stores, and others. It's ideal for executing deep and computationally intensive analytics like clustering and targeting.
Data is partitioned and then loaded into a file system comprised of several nodes running on commodity hardware. The Hadoop Distributed File System, or HDFS, is the default file storage in Hadoop. HDFS is used to store enormous amounts of data that do not need to be arranged into relational rows and columns [7].

Every piece of data is copied multiple times before being loaded into the file system. In the event of a node failure, another node has a copy. NameNode's job is to act as a facilitator, communicating back to the client information such as which nodes are available, where certain data exists in the cluster, and which nodes have failed.

After loading the data into the cluster, it is ready for analysis using the MapReduce framework. The client sends the job to one of the cluster nodes known as the Job Tracker. The Job Tracker consults the Name Node to determine the data it needs to access to finish the job and where that data is located in the cluster. When this is determined, the Job Tracker sends the query to the appropriate nodes. Rather of bringing all of the data back to a central point for processing, processing now takes place at each node concurrently or in parallel [7].

After completed the job each node stores its own result. The client initiates a "Re- duce" 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].
Each node stores its own result after the work is completed. The client launches a "Reduce" job via the Job

Tracker, following which the map phase results saved locally on the individual nodes are aggregated to derive the "response" to the original query and loaded to another node in the cluster. These results can be accessed by the client and then imported into one of several analytic environments for study [7].

Hadoop's key components are:

❖ Hadoop Distributed File System (HDFS): The Hadoop cluster's default storage layer.

❖ Name Node: The node in a Hadoop cluster that informs the client about where certain data is stored in the cluster and if any nodes fail.

❖ Secondary Node: A backup to the Name Node, it copies and saves data from the Name Node on a regular basis in case it fails.

❖ Job Tracker: A Hadoop cluster node that launches and coordinates MapReduce tasks, or data processing.

❖ Slave Nodes: The grunts of every Hadoop cluster, slave nodes store data and receive processing instructions from the Job Tracker.

After the MapReduce phase is completed, the processed data are ready for analysis in the next step, which combines data from the speed layer and batch layer of the Lambda Architecture to create a solution to a given query.

2.2 Hadoop Implementation of Batch Layer in LambdaArchitecture

The Batch layer holds the master dataset copy and performs batch views (arbitrary functions) on that master dataset (fig.2). A prominent example of a batch processing system is Hadoop.

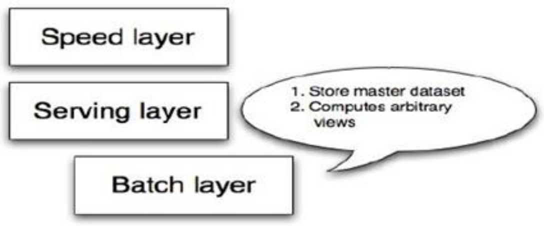

Fig. 2. Batch layer jobs [2]

Figure 3 depicts the core idea of Hadoop implementation using MapReduce. It separates the major problem into subproblems (mapping), applies the same function to each subproblem, and then combines (reduces) the output from all subproblems.

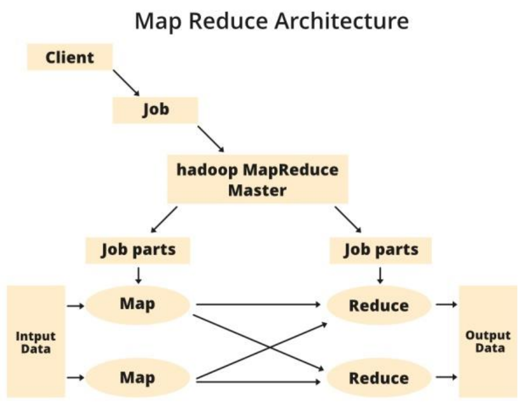

Fig. 3. MapReduce works, [8].

The batch layer can be represented in pseudo-code [2] in a simple way that shows how it constantly recomputes the batch views from scratch:

```
functionrunBatchLayer():
while(true): recomputeBatchViews()
```

Although using incremental MapReduce methods to increase the frequency of batch views would be more performant, this strategy trades performance for human fault-tolerance, making it easier to correct any problems in the views that are eventually caused by human errors or other reasons. Other implementation characteristics include unrestricted computation, the absence of the necessity for de-normalization, and horizontal scalability. Moreover, although being implemented as a single-threaded application, it automatically parallelizes across a cluster of servers, allowing for the study of datasets of any size. The following example shows 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(
newHfs(new TextLine(new Fields("url")),srcDir), new StdoutTap(),
pipe); flow.complete();
```

Given an input dataset of raw pageviews, this code's job is to compute the number of page views for each URL. It may be distributed automatically on a MapReduce cluster, scalable to the number of available nodes. For example, if there are n nodes in the MapReduce cluster, the calculation will be completed approximately n times faster than if only one node is used! The output directory will contain a number of files containing the results at the end of the computation [2].

3 Conclusion

The Lambda Architecture is the first method to dealing with the complexity of Big Data systems by establishing a defined set of rules. Immutability, human fault-tolerance, and re-computation are three principles that can be easily implemented using the Hadoop platform. Depending on the real-time needs, the speed layer may not even be required. If it is left out, the system becomes less complex, but the beauty of the Lambda Architecture is that the speed layer can be added later without much difficulty.

The idea behind dealing with real-time Big Data stream analytic systems is to combine diverse technologies and blend batch and real-time stream analytic processes as needed. This concept is being implemented by Lambda Architecture, which combines technologies such as Hadoop, Storm, and Kafka.
In the future research phase, we will develop additional Lambda Architecture components to evaluate both batch and real-time data. We will test the speed layer with Amazon web services (as an easier effort) and also integrate Storm in our speed layer testing environment.

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, fromhttp://www.ymc.ch/en/lambda-architecture-part-1

[4] ZirijeHasani, MargitaKon-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, BitolaMacedonia.

[5] Guido Schmutz. *Kafka and Storm – event processing in realtime.* International Con- ference for the software community, JAZOON2013.

[6] BojanIlioski. Hadoop на FINKI. Skopje, Macedonia. January2014.

[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 Da-ta systems. Retrieved April 19, 2014, fromhttp://jameskinley.tumblr.com/post/37398560534/the-lambda-architecture-principles-for architecting

[11] Storm. Retrieved April 19, 2014, fromhttp://storm-project.net/

[12] B. Ellis, Real-Time Analytics: Techniques to Analyze and Visualize Streaming Data. Hoboken, NJ, USA: John Wiley & Sons, 2014.

[13] P. Basanta-Val, N. Fernndez-Garca, A. J. Wellings and N. C. Audsley, “Improving the predictability of distributed stream processors,” Future Generation Comput. Syst., vol. 52, pp. 22–36, 2015.

[14] P. Basanta-Val, N. C. Audsley, A. J. Wellings, I. Gray and N. Fernndez-Garca, “Architecting time-critical big-data systems,” IEEE Trans. Big Data, vol. 2, no. 4, pp. 310–324, Dec. 2016.

[15] Apache Spark, [Online]. Available: http://spark.apache.org, Accessed on: 20-Apr.-2016.

[16] T. White, Hadoop: The Definitive Guide, 3rd ed. Sunnyvale, CA, USA: Yahoo Press, 2012.

[17] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker and I. Stoica, “Spark: Cluster Computing with Working Sets,” in Proc. 2nd USENIX Conf. Hot Topics Cloud Comput., 2010, pp. 10–10.

[18] M. Zaharia, M. Chowdhury, T. Das and A. Dave, “Fast and Interactive Analytics over Hadoop Data with Spark,” Usenix, vol. 37, no. 4, pp. 45–51, 2012.