A Comparative Performance of Real-time Big Data Analytic Architectures

Apisit Sanla
Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang
Bangkok, Thailand
apisit.sa@kmitl.ac.th

Thanisa Numnonda
Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang
Bangkok, Thailand
thanisa@it.kmitl.ac.th

Abstract—**Nowadays, many organizations pay attention to the relevant technologies of Big Data to analyze more accurately, quickly, and efficiently. Real-time Big Data analytics is challenging due to the massive volume of complex data needed to distribute in processing. Therefore, in this research, we investigate two state-of-the-art architectures: Lambda and Kappa. The Kappa architecture is simply the Lambda architecture without the batch layer. To help businesses decide on the right architecture, their processing time, and resource utilization in the same environment need to be found out. Experiments had been carried out with the data size 3 MB, 30 MB, and 300 MB. The results showed that Lambda architecture outperforms Kappa architecture around 9% for the accuracy test when using processing time approximately 2.2 times more than Kappa architecture. Lambda architecture also used more 10-20% of CPU usage and 0.5 GB of RAM usage than Kappa architecture.**

Keywords-Lambda architecture; Kappa architecture; real-time; Big Data

I. Introduction

Big Data is a data source consisting of 4Vs, Volume, Velocity, Variety, and Value [1]. Since Big Data may be in unstructured, semi-structured, and structured platforms, it is not easy to manage and analyze. Therefore, many organizations now pay attention to the related technology of Big Data to help them analyze accurately, quickly, and efficiently. Real-time Big Data analytics means that big data is processed as it arrives. This event is challenging due to the massive volume of complex data needed to distribute in processing. How to handle significant data accuracy and worth for investment is also extremely challenging. Therefore, this article investigates two state-of-art real-time Big Data analytic architectures, Lambda architecture, and Kappa architecture. Lambda architecture consists of three layers: batch layer, speed layer, and serving layer, whereas Kappa architecture consists of only two layers: streaming layer and serving layer.

This paper is structured as follows. Section 2 provides background work on Lambda and Kappa architectures, Apache Hadoop, Spark, and some other relevant technologies. Related work is discussed in Section 3. Methodology and Implementation on Hadoop clusters are demonstrated in Section 4. Then, the results are shown in Section 5, and the conclusion will be discussed in the last section.

II. Background Work

A. Lambda Architecture

Lambda architecture, first introduced by Nathan Marz's [2], is a real-time architecture for Big Data. There are three layers: batch layer, speed layer, and serving layer, as shown in Fig. 1. The batch layer stores and processes the master dataset to generate the batch views. Therefore, this process usually takes high latency. To compensate for the latency problem in the batch layer, speed layer is a real-time processing layer with the latest data to produce real-time views. It updates the views when it receives new data instead of recomputing the views as the batch layer does. Then, the serving layer combines the outcomes of the batch layer and speed layer to provide queries on them.

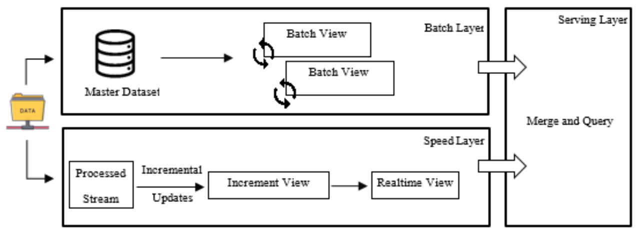


Figure 1. Lambda Architecture

B. Kappa Architecture

Jay-Kreps proposed Kappa architecture in 2014 [3]. The batch layer is eliminated, remaining only the streaming layer to process streaming data. If there is a change of the data in this layer, it will always be replaced with the new data. The serving layer provides queries for the streaming layer, as shown in Fig. 2. Kappa architecture stores data in the way of distribution; therefore, requirement and analysis can be easily changed [4].

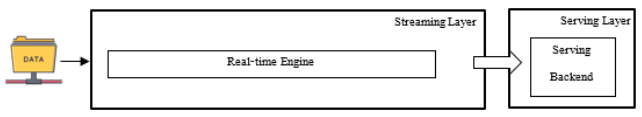


Figure 2. Kappa Architecture

C. *Apache Hadoop*

Apache Hadoop is an open source Big Data platform. It can be used to store and process massively distributed data [5]. Hadoop 1.0 has mainly two parts: Hadoop Distributed File System (HDFS) for data storage and MapReduce for data processing. The MapReduce divides files into large groups and spread across nodes in the cluster. Then the package code will be sent as a node to process parallel data. This event can help the massive data set to be processed faster and more efficiently. However, the MapReduce is still slow and inconvenient to implement using the Java programs directly. Therefore, in Hadoop 2.0, YARN helps other platforms than MapReduce such as Spark or Hive to process data from HDFS more efficiently.

D. *Apache Spark*

Apache Spark is an open-source, general purpose, scalable, and in-memory computing platform, developed by UC Berkeley University in 2009 and donated to the Apache Software Foundation in 2010 [6]. There are many available different analyses modules, such as Spark SQL, Spark Streaming, MLlib, and GraphX. A spark is a powerful tool which reduces the processing time by using memory instead of HDFS. This tool can dramatically improve the performance on a cluster. The nature of the work comes in the Resilient Distributed Dataset (RDD) style, which can be divided into small data sets distributed within Spark and has similar behavior to MapReduce [7]. After the RDD is imported, mapped, and transformed, the new value of RDD is sent to produce the final result.

The Spark streaming is part of the Spark API, which can process not only massively streaming data but also historical data. Besides, it can make streaming applications resistant to errors by continuously sending a streaming message known as DStreams, as shown in Fig. 3. It can be used in both similar components: the speed layer in the Lambda architecture and the streaming layer in the Kappa architecture. After an agent such as Kafka, RabbitMQ, or ActiveMQ receives massively streaming data and distributes that data to the Spark streaming, the Spark streaming will process and send the results back to the agent.

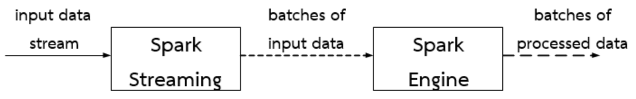


Figure 3. The work of DStream in Spark Streaming

E. *Apache Kafka*

Apache Kafka is a distributed streaming platform that can transfer both asynchronous and synchronous messages. It was developed at LinkedIn in 2010 [8]. It can be applied to record transaction data, where new data is appended only, not overwritten [9]. Besides, Kafka can set the age not to expire meaning that the same data can be read over and over again. The data in Kafka are stored in the disk space continuously acting as files. The advantages of Kafka are that it is fast and can be a data store. It also has very low latency on reading and writing [10].

F. *Apache NiFi*

Apache NiFi [11] is a software development project from the Apache Software Foundation proposing to help design the flow of data between software automatically. It is developed by NSA, which the current source name is NiFi coming from NiagaraFiles. Its software design is based on the programming style according to the work plan of the program developer.

G. *Elasticsearch*

Elasticsearch [12] is an open source, full-text search, and analysis engine. It allows searching enormous data almost real-time with an HTTP web interface and schema-free JSON documents. It uses a calculation method called TF-IDF (Term Frequency-Inverse Document Frequency), developed by Apache Lucene since its launch in 2010.

H. *Kibana*

Kibana [13] is a Web application that displays analytical results. It can run the data from Elasticsearch and enable users to see an overview of real-time data in the different desired report formats.

III. Related Work

Lambda architecture has high accuracy and fast command processing. However, it has to pay the high cost of long-term maintenance for each layer, which is separated. Furthermore, the same data have to keep in both layers; batch and speed layer. Therefore, when there is a change in the data in one layer, the other layer must also change the data as well. Whereas Kappa architecture does not include the batch layer in the architecture, presumably that many applications do not require the massive amount of data. However, there is a segmentation of the streaming data and the number of resources adjusting the size according to segment size [14].

Both Lambda and Kappa architectures provide great flexibility when increasing the size of the data set. However, the size of the data set is a factor that affects performance. When the size of the data set is more massive than 100M, compared with the work time, it is found that Kappa architecture used up to +2209% of work time while Lambda architecture uses only +1940%. Therefore, increasing the amount of imported data, Kappa architecture tends to decrease performance higher than the Lambda architecture [15].

In the Lambda architecture, if the data has to be processed repeatedly, the architecture can retrieve a copy of the data that has been treated recently, reproduce that data, and compare it with the data obtained from the speed layer. Whereas, in Kappa architecture, the data have to be replicated before processing every time. However, the need for the Lambda architecture to prepare areas must support data generated by rewriting or re-processing as well. Therefore, the budget for making a space for supporting lambda architecture data may have to use twice more than the budget of Kappa architecture [16].

Based on the above data, it can be seen that there are differences in both architectures, as concluded in Table 1.

TABLE I. COMPARISON BETWEEN LAMBDA AND KAPPA ARCHITECTURE

	Lambda Architecture	**Kappa Architecture**
Layer	Batch layer Speed layer Serving layer	Streaming layer Serving layer
Error (In case of sudden data changes)	Low Risk	High Risk
Reliability	High	Medium
Change of Structure	Hard	Flexible
Cost	High	Low
Resource Usage	High	Low

IV. IMPLEMENTATION

Now we need to prove the concept of both architectures. Implementation of Lambda architecture and Kappa architecture will be demonstrated in section A and B, respectively. We use Ubuntu 16.04 as an operating system with the machine of 8 core CPU, RAM 30 GB, Hard disk 250 GB for a cluster of one master node, and two worker nodes.

A. *Lambda Architectural Design*

Lambda architecture separates the layer of processing, as described in Section 2. The process of analytics begins with Apache NiFi ingest streaming data to Apache Kafka. Then Kafka distributes the data to both batch layer (1) and speed layer (2). The flume in the batch layer sends that data to persist in HDFS and analyze by using MapReduce. The batch views are the results of this layer. Spark streaming in the speed layer receives the input directly from the Kafka and processes that data before representing in the speed views. The serving layer (3) merges the batch and speed views and put them to Elasticsearch for comparing and indexing via Kibana, as shown in Fig. 4.

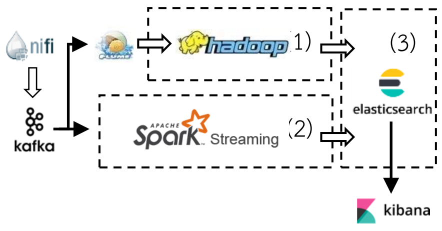


Figure 4. Proposed Lambda Architecture

B. *Kappa Architectural Design*

Kappa architecture focuses only on the streaming data, so it gets rid of the batch layer as described in Section 2. After Apache NiFi ingests streaming data to Apache Kafka, Kafka distributes that data to Spark streaming to analyze in the streaming layer (1). Then the real-time views are presented and sent to persist in Elasticsearch indexed through Kibana in the serving layer (2), as shown in Fig. 5.

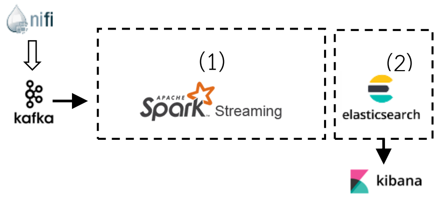
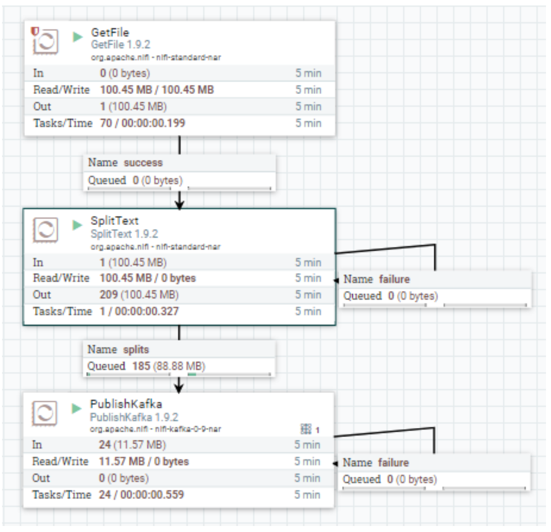


Figure 5. Proposed Kappa Architecture

C. *Design of Transmission Control*

A processor named GetFile is used to control the amount of transmission data from Apache NiFi. Then, the data are split using the SplitText processor and send data through the PublishKafka processor to Apache Kafka, as shown in Fig. 6.


```
GetFile
GetFile 1.9.2
org.apache.nifi - nifi-standard-nar
In           0 (0 bytes)                 5 min
Read/Write   100.45 MB / 100.45 MB       5 min
Out          1 (100.45 MB)               5 min
Tasks/Time   70 / 00:00:00.199           5 min

Name  success
Queued  0 (0 bytes)

SplitText
SplitText 1.9.2
org.apache.nifi - nifi-standard-nar
In           1 (100.45 MB)               5 min
Read/Write   100.45 MB / 0 bytes         5 min
Out          209 (100.45 MB)             5 min
Tasks/Time   1 / 00:00:00.327            5 min

Name  failure
Queued  0 (0 bytes)

Name  splits
Queued  185 (88.88 MB)

PublishKafka
PublishKafka 1.9.2
org.apache.nifi - nifi-kafka-0-9-nar     1
In           24 (11.57 MB)               5 min
Read/Write   11.57 MB / 0 bytes          5 min
Out          0 (0 bytes)                 5 min
Tasks/Time   24 / 00:00:00.559           5 min

Name  failure
Queued  0 (0 bytes)
```

Figure 6. Design of Transmission Control

When the data is distributed from Kafka, Apache Flume helps to spread data to the batch layer. The data is stored in HDFS and processed Word Count using Hadoop MapReduce. While in the speed layer, that data is processed to do Word Count using Spark streaming, as shown in Fig. 7 and 8.

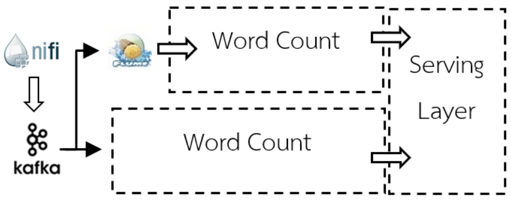


Figure 7. Processing Design in Lambda Architecture

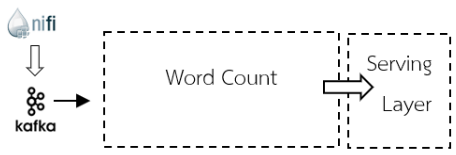


Figure 8. Processing Design in Kappa Architecture

V. Results

This section is part of the experimental results obtained from data transmission testing and performance measurement, as shown below:

A. Accuracy Test

A data (one file) with a size of 3 MB containing 566,849 words was input into both architectures. Then, ten times and one hundred times of the message, which are 30 MB and 300 MB size of data containing 5,668,490 words and 56,684,900 words were also input to investigate more results. The data were sent into the architectures by compressing with one byte per one second. The results of the word countability and the percentage of accuracy of both architectures are shown in Table 1 and 2, respectively.

TABLE I. Word Count and Accuracy, Sorted by Data Size

Data Size	Layer	Lambda Architecture		Kappa Architecture	
		Word Count	***Accuracy (%)***	***Word Count***	***Accuracy (%)***
3 MB	Batch	566,503	99.9389	-	
	Speed / Streaming	514,916	90.8383	514,916	90.8383
30 MB	Batch	5,664,840	99.9356	-	-
	Speed / Streaming	5,149,070	90.8367	5,149,070	90.8367
300 MB	Batch	54,382,464	95.9382	-	-
	Speed / Streaming	49,431,072	87.2032	49,431,072	87.2032

B. Data Processing Time

From Table 3, it is found that Lambda architecture uses the data processing time approximately 2.2 times more than that of Kappa architecture when the data size is 30 MB and 300 MB.

TABLE II. The Data Processing Time, Sorted by Data Size

Data Size	Layer	Data Processing Time (minute)	
		Lambda Architecture	***Kappa Architecture***
3 MB	Batch	0.31	-
	Speed / Streaming	0.08	0.06
30 MB	Batch	1.53	
	Speed/ Streaming	1.24	1.22
300 MB	Batch	27.55	-
	Speed / Streaming	23.59	23.05

C. CPU Usage in Data Transmission

When testing CPU usage for sending 3 MB of data into the architectures, the results showed that the Lambda architecture (average 50.85%) had a higher CPU utilization rate than that of the Kappa architecture (average 39.95%) approximately 10.90% as shown all details in Figure 9.

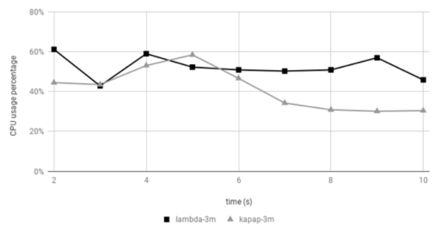


Figure 9. CPU Usage of Sending 3 MB of Data

When testing CPU usage for sending 30 MB of data into the architectures, the results showed that the Lambda architecture (average 68.50%) had a higher CPU utilization rate than that of the Kappa architecture (average 53.62%) approximately 14.88% as shown all details in Figure 10.

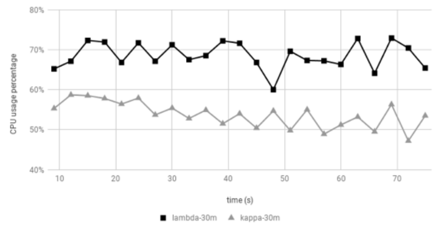


Figure 10. CPU Usage of Sending 30 MB of Data

When testing CPU usage for sending 300 MB of data which is one hundred times of the original size into the architectures, the results showed that the Lambda architecture (average 71.55%) had a higher CPU utilization rate than that of the Kappa architecture (average 54.61%) approximately 16.94% as shown all details in Figure 11.

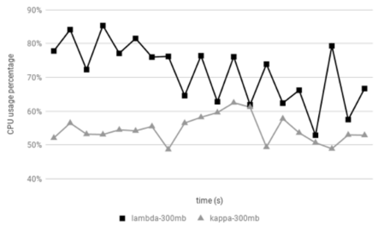


Figure 11. CPU Usage of Sending 300 MB of Data

D. RAM Usage in Data Transmission

When measuring the RAM usage of sending data size 3 MB, the results showed that the Lambda architecture has RAM usage at the average 15.72 GB, which is 52.40% of the available RAM. Whereas, the Kappa architecture has RAM usage at the average 15.23 GB, which is 50.78% of the available RAM as shown all details in Figure 12.

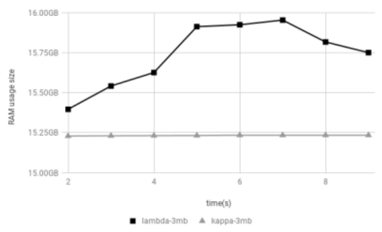


Figure 12. RAM Usage of Sending Data Size 3 MB

When measuring the RAM usage of sending data size 30 MB, the results showed that the Lambda architecture has RAM usage at the average 15.48 GB which is 51.59% of the available RAM. Whereas, the Kappa architecture has RAM usage at the average 15.22 GB which is 50.74% of the available RAM as shown all details in Figure 13.

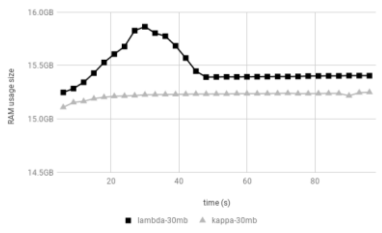


Figure 13. RAM Usage of Sending Data Size 30 MB

When measuring the RAM usage of sending data size 300 MB, the results showed that the Lambda architecture had RAM usage at the average 16.13 GB which is 53.75% of the available RAM. Whereas, the Kappa architecture has an average at 15.38 GB which is 51.25% of the available RAM as shown all details in Figure 14.

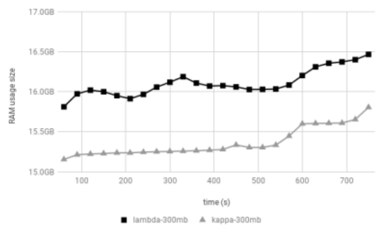


Figure 14. RAM Usage of Sending Data Size 300 MB

VI. DISCUSSION AND CONCLUSION

In this research, we investigated two state-of-the-art architectures: Lambda and Kappa. Experiments had been carried out with the data size 3 MB, 30 MB, and 300 MB. Based on the results of the experiments, it can be seen that the Lambda architecture outperforms Kappa architecture around 9% for the accuracy test when using processing time approximately 2.2 times more than Kappa architecture. Lambda architecture also used more 10-20% of CPU usage and 0.5 GB of RAM usage than Kappa architecture. Since Kappa architecture does not store any data, the processes may have some errors during data transmission or file subdivision. Therefore, it is suitable for tasks which do not require much precision but need quick results. Whereas, Lambda architecture, which requires complicated installation and high budget of maintenance, is suitable for organizations needing high accuracy jobs.

The next step toward more challenging is to subdivide data size before sending into both architectures. Moreover, the experiments should be more focus on the impacts that may occur at the time of data transmission or data processing to provide for the next real-time Big Data analytics architecture.

REFERENCES

[1] M. Hilbert, "Big Data for Development: A Review of Promises and Challenges," vol. 34, pp. 135-174, 2016.

[2] Lambda Architecture. Available: http://lambda-architecture.net

[3] kappa-architecture.com. Available: http://milinda.pathirage.org/kappa-architecture.com

[4] K. Pawar and V. Attar, "A survey on Data Analytic Platforms for Internet of Things," in 2016 International Conference on Computing, Analytics and Security Trends (CAST), 2016, pp. 605-610.

[5] Apache Hadoop. Available: https://hadoop.apache.org

[6] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," presented at the Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, Boston, MA, 2010.

[7] K. Hong. SPARK PROGRAMMING MODEL : RESILIENT DISTRIBUTED DATASET (RDD). Available: https://www.bogotobogo.com/Hadoop/BigData_hadoop_Apache_Spark_Programming_Model_RDD.php

[8] Apache Kafka ® is a distributed streaming platform. What exactly does that mean. Available: https://kafka.apache.org/intro

[9] T. Van-Dai, L. Chuan-Ming, and G. W. Nkabinde, "Big data stream computing in healthcare real-time analytics," in 2016 IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), 2016, pp. 37-42.

[10] B. Lawlor, R. Lynch, M. Mac Aogáin, and P. Walsh, "Field of genes: using Apache Kafka as a bioinformatic data repository," GigaScience, vol. 7, p. giy036, 2018.

[11] Apache NiFi. Available: https://nifi.apache.org

[12] Elasticsearch. Available: https://www.elastic.co/products/elasticsearch

[13] Kibana. Available: https://www.elastic.co/products/kibana

[14] M. Feick, N. Kleer, and M. Kohn, "Fundamentals of Real-Time Data Processing Architectures Lambda and Kappa."

[15] V. Persico, A. Pescapé, A. Picariello, and G. Sperlí, "Benchmarking big data architectures for social networks data processing using public cloud platforms," Future Generation Computer Systems, vol. 89, pp. 98-109, December 2018.

[16] J. Kreps, (2014). Questioning the Lambda Architecture. Available: https://www.oreilly.com/ideas/questioning-the-lambda-architect