© https://fti-tn.net/publications Future Technologies and Innovations (FTI) Proceedings: 4th international conference on computer applications and information security (iccais'2021) / March 19 / 2021/ Tunisia: https://fti-tn.net/iccais-2021-list-of-papers

Lambda Architecture Based Big Data System for Air and Ground Surveillance

Mustafa Umut DEMİREZEN
AI and Technology Management,
ROKETSAN Missiles Inc.,
Ankara,
TURKEY
Email: umut@demirezen.tech
(Corresponding author)

Tuğba Selcen NAVRUZ
Elecrical and Electronics Engineering,
Gazi University,
Ankara,
TURKEY
Email: selcen@gazi.edu.tr

***Abstract*—Processing both streaming and batch data at thesame time for big data analytics is very rare, even it is not possible to implement in real time applications such as air/groundmonitoring and surveillance. A good solution requires for a combination of batch and speed layers which must be processed together in a strictly coordinated manner. This paper introduces,a new type of system for handling of the problem. It also presents design and implementation processes based on Lambda Architecture which is a new generation big data technology. In order to explain the novel system, a brief research on the Lambda Architecture is summarized. Building and configuring all the necessary Big Data frameworks for a Lambda Architecturerequires three different layers work in a coordinated manner. This is very hard task to achieve for basic big data systems. A new type of hybrid system is proposed with a different concept having different layers on the same platform. The implementationresults have shown that the proposed system provides successful solutions to achieve both processing batch and stream data, to visualize data queries and to reduce code maintenance for areal time air/ground surveillance and monitoring problem of air traffic over Ankara, Turkey. It is expected that the new system proposed and implemented in this article might be used not only this specific application but also in other real-time and eventuallyconsistent applications.**

***Index Terms*—Big Data, Lambda Architecture, Data in Motion, Data at Rest, Air/Ground Surveillance**

I. INTRODUCTION

For Big Data applications, simultaneous processing of data streams, both real-time and offline, is an essential requirement for a long time. The data processing requirements for batch data and real-time processing operations dictate different technologies and, as a result, a particular type of Big Data architectures. One of the concepts that have been developed and widely used today is Lambda Architecture (LA) [1]. Architecture compromises of three distinct layers for processing both data in motion (DiM) and data at rest (DaR) at the same timeand a serving layer for presenting the results. Every Lambda Architecture layer is related to a specific task for processing data with different characteristics, combining the processed results from these layers and serving these merged data sets for visualization, query, and data mining purposes. The speed layer is mainly responsible for processing the streaming/real-time data (DiM) and very vulnerable to delaying and recurring data situations. The batch layer is responsible for processing the offline data (DaR), calculating the pre-defined analytics operations, and correcting the errors that sometimes occur on data arrival at the speed layer. The serving layer is in charge of ingesting data from the batch and speed layer, indexing and combining resulting data sets for the required analytical queries. The serving layer requires the unique capability to ingest and process both streaming data (DiM) in real-time as received and bulk data (DaR) in huge quantities. It has to be emphasized that Lambda Architectures are eventually consistent systems for big data processing applications and can be used for dealing with CAP theorem [2]. The batch layer corrects data ingestion inconsistencies caused by thereal-time layer after finishing the data processing. Eventually, accurate data is served and made available at the serving layer to provide information to the remaining operations.

It is evident that these complex data processing operations and serving the corrected data as accurately as possible require highly coordinated and continuous operation between speed and batch layers together. It is a tremendous advantage of LA that consisting of three separate layers, provide flexible usage of different Big Data technologies for each layer. However, this also brings some drawbacks. Even though LA is an auspicious direction, its success depends on an effective combination of the right and mature technologies. The fundamental problem is developing a Big Data application for each layer separately and integrating them to work together and in an interoperable manner. The usage of different technologies in LA layers requires separate development and maintenance efforts foreach layer. For some reason, if the data model or the data format in the application changes or additional new analytics capability is required, it finally results in updating, testing, anddeploying this big data application at all layers.

This work aims to present an end-to-end big data system and architecture, Lambda Architecture, a more modern generation of big data technology, for real-world applications. In the first section, to design a successful LA, candidate technologies are briefly reviewed for each layer, layer-wise design constraints investigated, and the same software development framework's usage in different layers (same code for different layers -

SC-FDL) approach introduced. In the second section, the suggested LA was applied to a real-world case study for Air/Ground Surveillance, and real-world performance tests under different real-world data ingestion conditions were performed. In the last section, the implementation and testing results were shared and have shown that the proposed system provides robust performance to achieve both processing batch and stream data, visualization of data queries, and reduce code maintenance for real-time air/ground surveillance.

II. Related Work

The main advantage of building a Lambda Architecturebased Big Data system is handling the requirement of de- veloping a fault-tolerant architecture for preventing data loss against hardware problems and unexpected mistakes during processing both DaR and DiM. It works well under the workloads, which require minimum latency read and update operations. This sort of system has to support ad-hoc queries, be linearly scalable and extensible.

One of the early examples of lambda architecture was used for IoT based smart home application [3]. Another application proposed a multi-agent big data processing approach to implementing collaborative filtering to build a recommendation engine by using LA [2]. For prime examples of LA, Apache Spark/Spark Streaming, Hadoop YARN [4], and Cassandra [5] technologies were used for real-time, batch, and serving layers, respectively, for LA architectures. Task-based serving, batch, and speed layers were selected to build real-time and batch views, furthermore querying the aggregated data. Apache Hadoop and Storm frameworks are mature enough technologies to implement and deploy for LA applications with different requirements [6]. An alternative strategy for utilizing the speed and batch layers of LA was implemented [7]. If time restrictions are not applicable and time periods are in degrees of minutes, continuously operating a speed layer is unnecessary. Using stream processing only when batch processing time exceeds the system's response time is a design to utilize the cluster resources efficiently [8]. Using performance models for software systems to prognosticate a big data system's execution metrics and cluster resources and then operating the speed layer is an application-specific approach [9]. Another work suggested a basic set of procedures to examine the difficulties of volatility, heterogeneity and desired low latency performance by decreasing the overall system timing (scheduling, execution, monitoring, and error recovery) and latent faults [10].

III. Lambda Architecture based Big Data SystemA

Lambda Architecture compromises a speed, a batch, and a serving layer to process incoming data and respond to the queries over stored historical and newly received new data. When a query request is received at the serving layer, the response is generated by querying both real time and batch views at the same time and merging the results obtained from these layers. Both real-time and batch databases at the serving layer are queried, and results are merged into one resultant data set so that a near real-time data set can be formed in response to the query. A scalable distributed data transfer system (data bus) provides data transfer operation continuously to simultaneously batch and speed layers. The data processing and analytics operations are executed in a real-time manner on the speed layer and offline manner on the batch layer. A conceptual visualization of the Lambda Architecture is shown in Figure 1. Incoming data from the data ingestion bus is sent both speed and batch layer, and then using these new and old data, several views are generated by these layers, and the results are hosted on the serving layer of LA. Several existing big data technologies can be used in all three layers to build a LA. Each available big data technology framework can be used for its particular data processing capability to deal with that type of data and support analytics operations according to LA's polyglot persistence paradigm.

The batch layer manages, operates, and stores immutable primary data set blocks. The incoming very recent data are only appended to the previously stored historical data on the batch layer. In fact, on the batch layer, update and delete opera-tions are not allowed. As required, continuous data processing and analytics operations are executed to produce batch views by using this data. When coordinated with the speed layeror on a predefined amount of new data arrival, a new batch view computation operation is re-executed one after another and aggregated to form new batch views. This operation is continuous and everlasting. The batch views are always generated from all the immutable data set stored in the batch layer. Full batch data processing from beginning to end and analytical calculations take too much time depending on the size of both incoming and stored historical data. Batch layer data processing and importing progress must be monitored to ensure whether batch view generation is completed before the speed layer is overloaded.

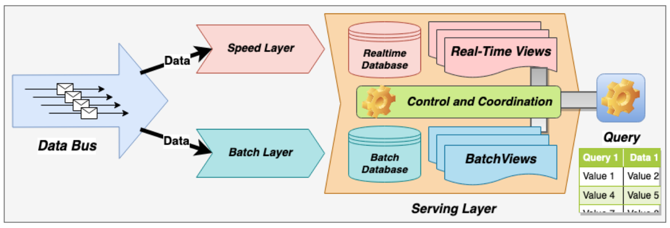

Fig. 1. Conceptual diagram of a lambda architecture

The serving layer is used to respond to the incoming queries using real-time and batch views generated via both speed and batch layers. So, the serving layer requires bigdata storing capabilities like NoSQL databases with different characteristics. Because of the different types of data ingestion patterns, both bulk and real-time data ingestion or ingestion capabilities must be supported at this layer. The serving layer is vulnerable to delaying or missing streaming data situations. Inconsistent data analytics and query responses may occur under these circumstances and are eventually corrected solely by the batch layer.

To meet the low latency analytics and responsive query requirements, the LA's speed layer is used to compensatefor the batch views' staleness and serves the most recently received data, which are yet not processed by the batch layer. The speed layer works on the streaming data in a real-time manner and stores its output in the serving layer as real-time views depending on its limited capacity. Because of the nature of the real-time operation requirements, the speed layernecessitates high read and write operations on the servinglayer. Real-time views are only required to store recent data until the batch layer completes its operation cycle one or two time(s). When the batch layer finishes the data processing and analytics calculation operations, the data stored as real- time views during the batch processing are discarded anddeleted from the serving layer. After batch view generation ends, some real-time views have to be flushed or cleanedfrom the real-time layer depending on the data processed atthe batch layer. This operation is an essential and crucialstep in reducing the stress on the real-time database at the serving layer. Monitoring the speed layer resources and takingaction depends on the resource consumption and capacity requirement, precise coordination between the layers, and specific performance metrics for all layers. Under improper conditions or faulty coordination with the speed layer, thebatch view will be stale for at least the processing time between the start and the end time for the batch operations or even more. This phenomenon requires critical and strict coordination between speed and batch layers. As soon as the coordinated data processing operation between speed and batch layers is completed, bulk data import is required onthe serving layer. After the final batch views are ready, data ingestion on the serving layer is completed.

A. *Technology Selection for Lambda Architecture Layers*

The data ingestion part (data-bus) is formed to receive a high volume of real-time data for the Lambda Architecture. One of the most widely used and common frameworks fordata bus technologies is Apache Kafka [11], and it is very mature, proper, scalable, fault-tolerant, and eligible for this purpose. It supports high throughput data transfer and is a scalable, fault-tolerant framework for data bus operations. For the speed layer, Apache Samza [12], Apache Storm [13], and Apache Spark (Streaming) [14] are excellent choices. Apache Hadoop [4], Apache Spark is very common for the batchlayer operations. Apache Cassandra [5], Red is [15] , Apache H Base [16], MongoDB [17] might be used as a speed layer data base. These databases support high-speed real-time data ingestion and random read and write as well. MongoDB [17], Couch base DB [18], and Splout SQL [6] can be used as a batchlayer database. Apache F link [19] is also a good candidate for data processing operations, and is a framework and distributed processing engine for stateful computations for unbounded andbounded data streams. On the other hand, Apache Spark isan open-source data processing framework built for speed, with ease of use and complex analytics. It gives an extensive, all-in-one framework to handle big data processing requirements with diverse data sets and the source of data. Apache Spark supports a programming model, discretized streams (D-Streams) [20]. Generally, using NoSQL databases for LA is very common instead of relational databases. Scalable and advanced capabilities for data ingestion are the main reasons to be used at the serving layer. Druid [21]] is an open-source, column-oriented, distributed, real-time analytical data store. The distribution and query model of Druid has similarities with the ideas from current generation search infrastructures. Druid real-time nodes provide the capability to ingest, query, and create real time views from incoming data streams.

Generally, using NoSQL databases for LA is very common instead of relational databases. Scalable and advanced capabilities for data ingestion are the main reasons to be used at the serving layer. Druid [21] is an open-source, column-oriented, distributed, real-time analytical data store. The distributionand query model of Druid has similarities with the ideas from current generation search infrastructures. Druid real-timenodes provide the capability to ingest, query, and create real- time views from incoming data streams. Data streams thatare indexed via real-time nodes are instantly available forquerying. Druid also has built-in support for generating batch views by using Hadoop and running Map-Reduce jobs to partition data for batch data ingestion.

B. *Same Coding for Different Layers Approach*

Developing applications with Big Data frameworks maybe complicated, and debugging an algorithm on a Big Data framework or platform maybe even more challenging. Traditionally, application development and testing have to bedone at least twice for Lambda Architecture for batch and speed layers. Besides the configuration and coordination requirements of each layer, the most critical disadvantage of Lambda Architecture is that sometimes it is not practical to write the same algorithm twice with different frameworks for the developers. Developing, debugging, testing, and deploying separate software with different frameworks on large hardware clusters for Big Data applications requires extra effort and extreme work. To overcome this handicap, using the same big data processing technology for different layers is a practical approach.

By using Spark and Spark Streaming together provides usage of the same code for batch and online processing forBig Data applications. As for the Lambda Architecture, Spark Streaming and Spark can be used for developing the speed layer and the batch layer applications. Both layers can besupported by this framework effectively. Nevertheless, one problem remains that the serving layer has to be integrated with both layers for data processing and providing the data ingestion for both layers.

IV. Case Study: Lambda Architecture Performance Tests for Air Traffic Visualization and Monitoring

For the Aviation domain, an Air/Ground Surveillance system characteristically necessitates high-speed distributed streaming data processing. Automatic Dependent Surveillance-Broadcast (ADS–B) is a cooperative surveillance technology in which an aircraft determines its position via GPS signals using satellites, periodically broadcasts it in specific message format, and makes itself to be tracked$. Air traffic control ground stations can receive the information emitted from the aircraft or be received by other aircrafts to provide situational awareness. It is also a collision-avoidance system for aircraft. ADS-B enhances safety by making an aircraft visible to air traffic control (ATC) and other aircrafts with position andvelocity data transmitted every second in specific message formats. There are several types of certified ADS-B data links, but the most common ones operate at 1090 MHz, essentially a modified Mode S transponder or 978 MHz. Signals contain 112 bits of data, encoded using pulse position modulation witha data rate of 1Mbps. The ADS-B data can be recorded and stored for further flight analysis. The data is received from multiple sources and physical locations at the same time with high streaming rates.

For this study, software-defined radios located in geographically different areas to improve signal reception coverage were used for decoding the received ADS-B signals from air and ground vehicles at the edge. This data is then sent as decoded messages into a data ingestion framework to feed both the LA's speed and batch layers. After the required analytics operations are applied for both speed and batch layers, the results are shown on the visualization dashboard developed using theLA's serving layer.

TABLE I
SELECTED BIG DATA TECHNOLOGIES FOR THE APPLICATION

Layer / Component Name	Big Data Technology
Batch Layer	Hadoop / Apache Spark
Speed Layer	Spark Streaming
Serving Layer	Druid
Data Bus	Apache Kafka
Layer Coordination and Control Agent	YARN Application
Monitoring Agent	YARN

Time and safety-critical applications require minimal down-time, scalability, low latency, low maintenance, development effort, and good fault tolerance. Although some of those requirements can be fulfilled by the Big Data technologies, low maintenance and development efforts are generally hardto solve. After careful inspection and evaluation, considering the application requirements, Big Data technologies have been selected and given in Table I. To implement the SC-FDL concept, the Apache Spark framework was selected for both he batch and the speed layers. Druid real-time analytics database is chosen for the serving layer. Apache Kafka was selected for a distributed streaming platform as a data bus.For the speed layer and batch layer coordination and control, a YARN application was developed. YARN was deployed for clutster management and resource management system.

A. Proposed End-To-End Lambda Architecture Based System

An application based on Lambda Architecture called CyFly was developed to visualize real-time aircraft positions at the same time it is used for post-aircraft analytics by querying the ingested historical data. It also provides collision avoidance detection with real-time analytics. The architectural overview of the system is shown inFigure 2.

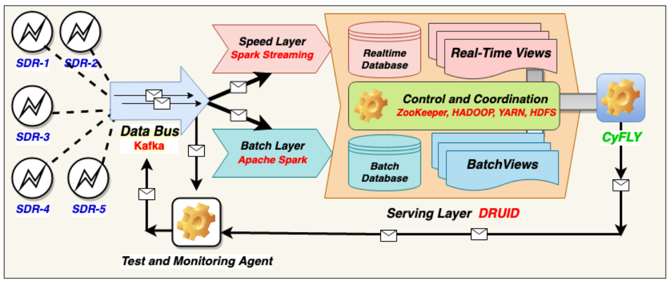

Fig. 2. Lambda architecture for air traffic monitoring application

5 remote SDR are located different locations in Ankaraand used as an ADS-B signal decoder. After decoding the receiving signals and converting the binary data to the text- based messages, software on the single board computer sends this information to the data bus. In this Lambda Architecture, Apache Kafka is deployed as the primary data bus and integrated with SDR via TCP sockets. The data bus is integrated with the speed layer by using Spark Streaming + Kafka Integration receiver-less method. Incoming raw data from different SDRs over Apache Kafka are joined in D Streams, then ETL, and filtering operations are applied at the speed layer. At the same time, this raw data coming from Kafka is stored inthe HDFS. At the batch layer, ETL and filtering operationsare applied with Spark and then results are written in theHDFS again. After this operation, the batch data ingestion operation for Druid is started with Druid's indexing services. As of the batch layer, both Hadoop indexer job and Druid's indexing service are used for ingesting offline data. If the data is less than several Gigabytes, the Druid indexing service is used; otherwise, the Hadoop Indexer job is started by a coordination agent for efficient data ingestion to Druid as a serving layer. The coordination agent is mainly responsiblefor monitoring and triggering batch ingestion jobs accordingto offline data size after ETL operations. If a certain amount ofoffline data reaches a predetermined value and ETL operation is completed, an indexing operation is started concerning offline data size by this agent. New segments are generated continuously and are produced query focused views at the serving layer. After batch ingestion, some real-time views are overridden by the batch views with recent data. Generally,data delay and recurrent data receiving situations may happen.A Data retention policy can be used with Druid to discard delayed data for a certain amount of time, and the missingdata can be corrected with batch layer ingestion. A Red iskey-value store is also connected with the Spark Streaming framework at the speed layer. Publish-subscribe operation is implemented for updating the aircraft's' information. This database is used to buffer more recent data for 5 minutes to newly connected clients for aircraft visualization. A Node.js service is connected to Red is and stream all the information to

clients via using Web Socket technology. Historical data can be queried and visualized by using another web service. For fault tolerance and high availability, Hadoop HDFS is deployed on a cluster. It is also used as deep storage for the Druid. YARN and Zoo Keeper are deployed on a cluster for resource management and coordination services, respectively.

B. Perforamance Tests for Different Data Ingestion Scenarios

The system was tested under several data ingestion conditions with synthetic data by engaging a distributed data generator application. These working conditions were selected to simulate real-world conditions as good as possible, summarized as a data rate increase, data rate decrease, instant data rate increase, instant data rate decrease, recurrent data receiving, and delayed data receiving. To measure the system performance, latency, and the number of correct query results in a time window is calculated. Latency is defined as the time difference between the first ingestion time of data in the system and the visualization time of the same data on the dashboard. According to the performance requirements, this value must be less than 2 seconds. Predefined analytics queries were prepared and executed on control time windows (3 minutes), and the results of the queries were compared with the ground truth values. This test was applied sequentially. For the next time window, all the experiments were executed again. For every test case given above, experiments were executed 100 times to obtain more accurate and statistically significant results.

V. RESULTS

Proposed LA based system is deployed on a cluster. For the cluster hardware, 12 Dell PowerEdge R730 and 13 Dell PowerEdge R320 servers are used. With this cluster, over 6 TB of RAM, more than 500 cores, and more than 0.6 PB disk space are available for data processing and storage. Under normal ingestion conditions, an average of 10000 messages per second was selected to model real air traffic data. Clus- ter's computation capability is far beyond these ratings, butair traffic over Ankara does not generate too many ADS-B messages according to daily observations and statistics. For each data ingestion test, this nominal data rate was increased or decreased according to a particular pattern that can accurately model the real air traffic. Test and monitoring agent sent test data to the data bus and collected the necessary results and statistics for each test case. All the results are given in Table II for the first and second time window.

For all test cases in the first-time window, the latency performance criteria are met. This value is never greater than 589 ms for all test conditions. This result shows thatthe system can effectively process all the incoming data and meets the performance criteria without delay. For this test time window, the number of correct queries was equal to the ground truth. Under delayed data conditions, because of missing data problems, system query results were not accurately correct. Only 9879 of 10000 queries were correctly returned in this time window. These query errors must be corrected the next time window by the batch layer. Under the recurrent data receiving conditions, the system had an excellent capability to handle this case effectively. Results show that the LA system works accurately enough and as expected under all test cases in the first time window.

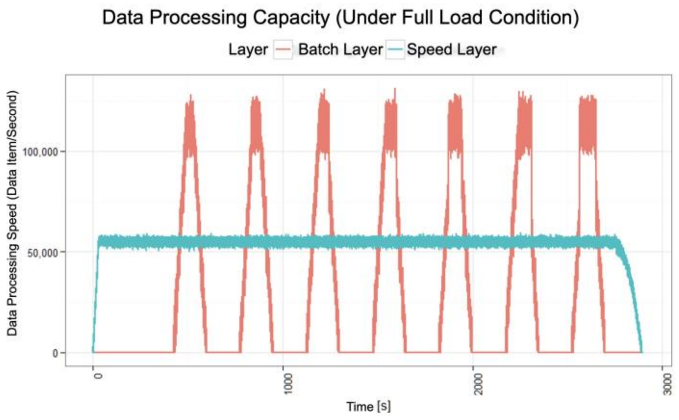

Fig. 3. The speed and batch layer load under heavy data condition

For all test cases, results obtained in the second time window are given in Table II as well. The latency performancecriteria were met during the tests, and the latency value was never greater than 577 ms for all test conditions. This result shows that the system can effectively process all the incomingdata and meet the performance criteria without any delay. For the second test time window, the number of correct queries was equal to the ground truth values. Under delayed dataconditions, missing data were eventually corrected by the batch layer. Under the repetitive data receiving conditions, thesystem had an excellent capability to handle this problematic case effectively. The results are shown that the system works correctly and prone to incoming data problems eventually.

Offline data can be queried by using the same application. Nevertheless, in the case of delaying and repetitive data receiving cases, queries are eventually accurate. After batch layer operation is completed, query-focused views are overridden by resulting batch views, and this layer is vital to obtaining accuracy and precision for query results. Data processing rate under full data load conditions is shown in Figure 3. It isseen that the batch and speed layers work as expected in a complementary manner.

VI. CONCLUSION

This paper presents a Lambda Architecture-based big data system to propose an air/ground surveillance by applying the same code for different layers approach. The proposed system's capabilities related to LA have been tested for real- world cases using ADS-B messages obtained from 5 remote locations with software-defined radio hardware. The design and implementation processes based on Lambda Architecture and performance tests for different data igestion load condi- tions are exhibited.

The implementation and testing results have shown thatthe proposed system provides superior performance to achieve

TABLE II
TEST RESULTS FOR DIFFERENT DATA INGESTION CONDITIONS

Test Case	Avg. Latency (ms) 1st Time Window	Query Results (Result/Ground Truth)	Avg. Latency (ms) 2nd Time Window	Query Results (Result/Ground Truth)
Normal Conditions	422.5 ± 71	10000 / 10000	428.5 ± 51	10000 / 10000
Data Increase	501.8 ± 87	10000 / 10000	499.8 ± 77	10000 / 10000
Data Decrease	423.6 ± 65	10000 / 10000	413.2 ± 55	10000 / 10000
Instant Data Rate Increase	469.8 ± 62	10000 / 10000	452.8 ± 52	10000 / 10000
Instant Data Rate Decrease	448.4 ± 44	10000 / 10000	438.4 ± 54	10000 / 10000
Delayed Data	426.5 ± 52	9879 / 10000	436.5 ± 78	10000 / 10000
Repetitive Data	428.7 ± 88	10000 / 10000	435.7 ± 48	10000 / 10000

both processing batch and stream data, visualization of data queries, and reduce code maintenance for real-time air/ground surveillance. According to the experiment results, the Latency values were always below 500 ms, and the LA system was corrected missing and repeated data ingestion problems almost until in the second time window. It is also shown that developing a Lambda Architecture requires integrating different technologies, even if it sometimes can be tough to implement. The experiments highlighted once more time that Lambda Architectures are eventually accurate systems to develop and implement the issues considered in this article.

ACKNOWLEDGEMENTS

This work presents the re-implementation of the results of the limited part of the thesis given in [22].

REFERENCES

[1] N. Martz and J. Warren, *Big Data Principles and Best Practices Of Scalable Realtime Data Systems*. New York, CA: Manning, 2015.

[2] B. Twardowski and D. Ryzko, “Multi-agent architecture for realtime big data processing,” *ACM International Joint Conferences of Web Intelligence and Intelligent Agent Technologies (IAT)*, pp. 333–337,2014.

[3] A. Villari, M. Celesti, Fazio, and A. Puliafito, “AllJoyn Lambda: An architecture for the management of smart environments in IoT,” *International Conference on Smart Computing Workshops*, pp. 9–14, 2014.

[4] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O’Malley, S. Radia, B. Reed, and E. Baldeschwieler, “Apache Hadoop YARN: Yet Another Resource Negotiator,” in *Proceedings of the 4th Annual Symposium on Cloud Computing*. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: 10.1145/2523616.2523633

[5] A. Lakshman and P. Malik, “Cassandra: A Decentralized Structured Storage System,” *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, April 2010. [Online]. Available: 10.1145/1773912.1773922

[6] Z. Hasani and G. Velinoc, “Lambda Architecture for Realtime Analytic,” in *ICT Innovations Conference, Macedonia*, 2014, pp. 133–143.

[7] J. Kross, A. Brunnert, C. Prehofer, T. A. Runkler, and H. Krcmar, “Stream Processing on Demand for Lambda Architectures,” in *Computer Performance Engineering*, M. Beltrán, W. Knottenbelt, and J. Bradley, Eds. Cham: Springer International Publishing, 2015, pp. 243–257.

[8] S. Nadal, V. Herrero, O. Romero, A. Abelló, X. Franch, S. Vansummeren, and D. Valerio, “A software reference architecture for semantic-aware Big Data systems,” *Information and Software Technology*, vol. 90, pp. 75–92, 2017. [Online]. Available: https://doi.org/10.1016/j.infsof.2017.06.001

[9] M. P. D. Pont, R. S. Ferreira, W. W. Teixeira, D. M. Lima,and J. E. Normey-Rico, “MPC with Machine Learning Appliedto Resource Allocation Problem using Lambda Architecture,” *IFAC-PapersOnLine*, vol. 52, no. 1, pp. 550–555, 2019. [Online]. Available: https://doi.org/10.1016/j.ifacol.2019.06.120

[10] V. A. da Silva, A. J. C. dos, de Freitas Edison Pignaton, L. T. J., and G. C. F., “Strategies for Big Data Analytics through Lambda Architectures in Volatile Environments,” *IFAC- PapersOnLine*, vol. 49, no. 30, pp. 114–119, 2016. [Online]. Available: https://doi.org/10.1016/j.ifacol.2016.11.138

[11] J. A. Shaheen, “Apache Kafka: Real Time Implementation with Kafka Architecture Review,” *International Journal of Advanced Science and Technology*, vol. 109, pp. 35–42, 2017. [Online]. Available: 0.14257/ijast.2017.109.04

[12] S. A. Noghabi, K. Paramasivam, Y. Pan, N. Ramesh, J. Bringhurst, I. Gupta, and R. H. Campbell, “Samza: Stateful Scalable Stream Processing at LinkedIn,” *Proc. VLDB Endow.*, vol. 10, no. 12, pp. 1634–1645, August 2017. [Online]. Available: 10.14778/3137765.3137770

[13] M. H. Iqbal, , and T. R. Soomro, “Big Data Analysis: Apache Storm Perspective,” *International Journal of Computer Trends and Technology*, vol. 19, no. 1, pp. 9–14, 2015. [Online]. Available: 10.14445/22312803/ijctt-v19p103;https://dx.doi.org/10.14445/22312803/ijctt-v19p103

[14] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster Computing with Working Sets,” in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*. USA: USENIX Association, 2010, p. 10.

[15] G. Kaur and J. Kaur, “In-Memory Data processing using Redis Database,” *International Journal of Computer Applications*, vol. 180, pp. 26–31, 2018.

[16] M. N. Vora, “Hadoop-HBase for large-scale data,” in *Proceedings of 2011 International Conference on Computer Science and Network Technology*, vol. 1, 2011, pp. 601–605. [Online]. Available: 10.1109/ICCSNT.2011.6182030

[17] A. Celesti, M. Fazio, and M. Villari, “A Study on Join Operationsin MongoDB Preserving Collections Data Models for Future Internet Applications,” *Future Internet*, vol. 11, no. 4, pp. 83–83, 2019. [Online]. Available: 10.3390/fi11040083;https://dx.doi.org/10.3390/fi11040083

[18] M. A. Hubail, A. Alsuliman, M. Blow, M. Carey, D. Lychagin, I. Maxon, and T. Westmann, “Couchbase Analytics: NoETL for Scalable NoSQL Data Analysis,” *Proc. VLDB Endow.*, vol. 12, no. 12, pp. 2275–2286, August 2019. [Online]. Available: 10.14778/3352063.3352143

[19] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, “Apache Flink^{TM}: Stream and Batch Processing in a Single Engine,” *IEEE Data Eng. Bull*, vol. 38, pp. 28–38, 2015.

[20] T. Zaharia, H. Das, T. Li, S. Hunter, I. Shenker, and Stoica, “Discretized streams: fault-tolerant streaming computation at scale,” *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pp. 423–438, 2013.

[21] F. Yang, E. Tschetter, X. Léauté, N. Ray, G. Merlino, and D. Ganguli, “Druid: a real-time analytical data store,” *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, pp. 157–168, 2014.

[22] M. U. Demirezen, “Büyük veri uygulamaları için bir lamda mimari gelis¸tirilmesi / Developing a lambda architecture for big data processing applications,” Ankara, Aralık 2015. [Online]. Available: https://tez.yok.gov.tr/UlusalTezMerkezi/TezGoster?key=WY5CM7tPNE2z_YM6pBu0t9xbu0Fi98SPu47VrKdWON0NL3fd08lbqE1Y8Dd8Fxe5