Big data processing in the cloud - Challenges and platforms

Cite as: AIP Conference Proceedings **1910**, 060013 (2017); https://doi.org/10.1063/1.5014007
Published Online: 07 December 2017

Svetoslav Zhelev and Anna Rozeva

Big Data Processing in the Cloud - Challenges and Platforms

Svetoslav Zhelev [1, a] and Anna Rozeva [1, b]

[1]*Technical University of Sofia, 8 Kliment Ohridski blv., 1000 Sofia, Bulgaria*

[a)] Corresponding author: sk@goodlightsolutions.com
[b)] arozeva@tu-sofia.bg

Abstract. Choosing the appropriate architecture and technologies for a big data project is a difficult task, which requires extensive knowledge in both the problem domain and in the big data landscape. The paper analyzes the main big data architectures and the most widely implemented technologies used for processing and persisting big data. Clouds provide for dynamic resource scaling, which makes them a natural fit for big data applications. Basic cloud computing service models are presented. Two architectures for processing big data are discussed, Lambda and Kappa architectures. Technologies for big data persistence are presented and analyzed. Stream processing as the most important and difficult to manage is outlined. The paper highlights main advantages of cloud and potential problems.

1. INTRODUCTION

Most big data applications require real time data processing, historic data analysis and data persistence. Depending on the application needs the data architect has to choose:

- The most appropriate database(s) - relational database management system (RDBMS) and/or NoSQL database(s);
- The most appropriate data processing framework and data processing type - streaming, batch processing or both;
- The most appropriate cloud provider or in-house hosting based on the infrastructure requirements - scalability, availability, pricing.

Each architecture and platform has its strengths and weaknesses and it is important that they are known in advance before initiating the application implementation phase.

Cloud computing delivers on-demand computing resources on a pay-for-use basis. It offers elastic resources - scale up or down quickly to meet demand [4]. Clouds can reduce the costs for infrastructure maintenance significantly but they have also some pitfalls, which have to be carefully evaluated. Clouds are not a “silver bullet” and they may be unsuitable for some use cases.

While designing an application architecture the first problem that is usually encountered is storage. Big data has three main characteristics: volume (amount of data), velocity (speed of incoming and outgoing data) and variety (range of data types and sources). Often the volume and velocity are so high that terabytes of storage is required, i.e. Facebook has 600 Tb of data, and ability to process gigabytes per second. Data variety can be even a bigger problem if not addressed properly.

There are two main approaches on how to implement a streaming system - native streaming and micro-batching. In native streaming all incoming data entries are processed one by one as they arrive. In micro-batching small batches are created from incoming data according to predefined time constant - usually every few seconds. All processing frameworks/platforms fall in one of the two categories and each one has its advantages and disadvantages. If the big data application requires machine learning (ML) then it is wiser to consider which ML libraries have integration with the chosen data processing framework and if these libraries have the appropriate algorithms for the task at hand.

The paper is organized as follows: Section 1 discusses cloud computing's s service models and cloud types; architectures for processing big data are introduced in Section 2; Section 3 presents big data persistence; techniques for stream processing and cloud scaling are shown in Section 4; the paper concludes with statements for future work.

2. CLOUD COMPUTING

2.1 Cloud Computing Service Models

Cloud computing has the following 3 service models [1]:

- Saas (Software as a Service) provides applications through a browser and has a user-oriented focus. An example of SaaS is G Suite offered by Google. Users have their data accessible from any computer with Internet access and they cannot lose data in case of computer failure. This service model targets end users and is unsuitable for big data processing.
- PaaS (Platform as a Service) provides server side technologies like databases, web servers, application engines, etc. This helps application developers to be more efficient and focused on the application itself because the provider does resource procurement, capacity planning, software maintenance and patching.
- IaaS (Infrastructure as a Service) provides access to networking, data storage and computers (virtual machines or dedicated hardware). It offers the highest flexibility compared to the other 2 models but in exchange the user himself has to configure, provision and maintain the infrastructure and software.

2.2 Cloud Types

There are 3 types of clouds [2]:

- Public clouds - owned by companies that offer cloud services over public network. Generally they are cheap and have evolved enough to meet the quality-of-service requirements of most customers.
- Private clouds - built specifically for a single organization. They are managed internally or by another company and they may be hosted internally or externally. Private clouds are secure and provide more control over resources. They require up-front investment and may not be an affordable solution to most companies.
- Hybrid clouds - a combination between private and public clouds with the aim to address both their limitations.

2.3 Cloud Advantages and Pitfalls

The greatest advantage of cloud computing is the dynamic resource utilization - the application can scale up and down on demand. The centralization of infrastructure and pay-as-you-go model lowers the costs significantly. Many cloud providers offer free services like monitoring, free storage, etc.

Although cloud computing provides clear benefits over traditional approaches it has some pitfalls as well. Without a doubt Amazon is one of the best cloud providers worldwide. They started their Web Services cloud platform in 2006 and currently have the richest cloud stack. However, even Amazon has annual uptime percentage of 99.95% defined in their Service Level Agreement (SLA). If we are building critical application service 0.05% could be significant downtime. This is approximately 5 min a week, 22 min per month and almost 4 hours and a half per year! Typically public cloud resources are shared between multiple tenants. Data is isolated but there is a potential security risk. Big cloud providers have several data centers across the globe. If we are implementing an application, which has sensitive data falling into special regulations (like medical data) we need to consider the data location as well. Sometimes cloud providers bill the clients not only for the computing resources they use but also for data transfer/requests to the machines. This can increase significantly the price for the cloud services for some applications and must be carefully considered.

3. BIG DATA ARCHITECTURE

Good big data processing architecture is fault-tolerant, scalable and extensible. There are 2 architecture types, which proved themselves in real life applications. Each one has its strengths and weaknesses and choosing the right one depends on the application requirements.

3.1 The Lambda Architecture

The Lambda Architecture shown in Fig.1 has three layers: batch, speed, and serving. The batch layer manages historical data and recomputes results (machine learning). It iterates through the whole data and has high latency. The speed layer processes incoming data and provides results in nearly real time. The results are not so accurate compared to the batch layer. The serving layer gets results from the other two layers and enables queries. Although Lambda architecture offers high flexibility it comes with a price. The complexity and the need to maintain two different code bases for data processing makes it difficult to implement and support [11], [12].

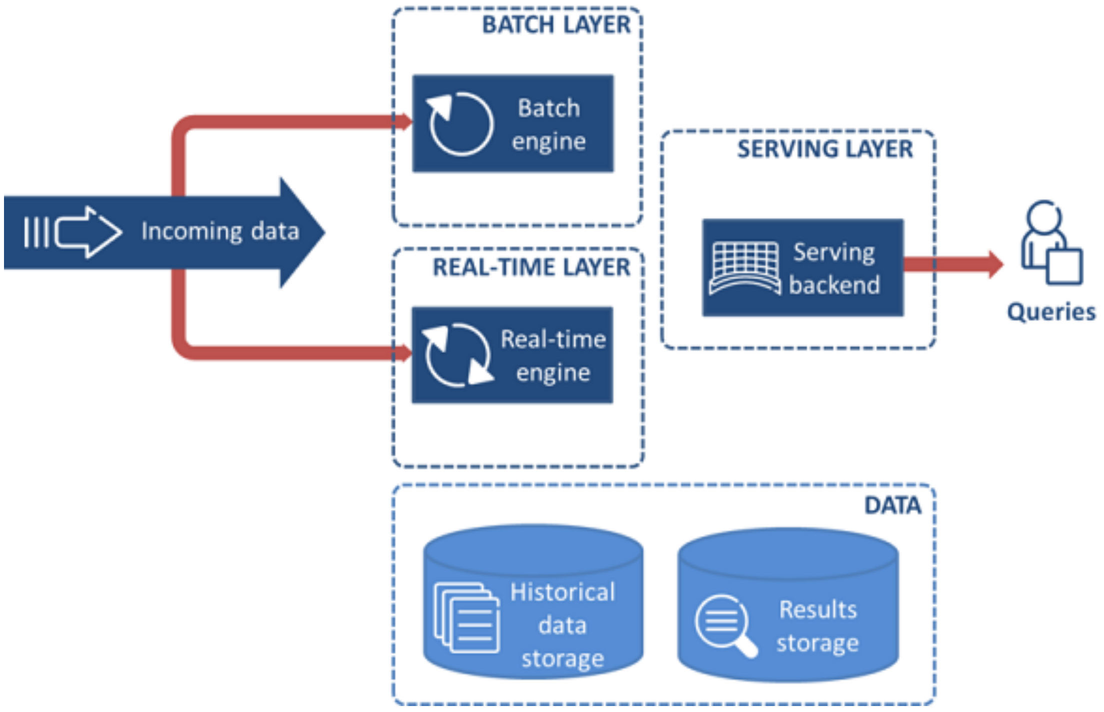

FIGURE 1. Lambda Architecture (taken from https://www.ericsson.com/research-blog/data-knowledge/data-processing-architectures-lambda-and-kappa/)

3.2The Kappa Architecture

The Kappa architecture, shown in Fig.2 tries to simplify Lambda architecture. It combines the batch and speed layers into one and has only two layers: stream processing and serving. The stream-processing layer executes processing jobs. Depending on what data we need to process different jobs are run (real time data, historic data). The serving layer is used to query results like in Lambda architecture. Kappa trades flexibility for simplicity [11, 13].

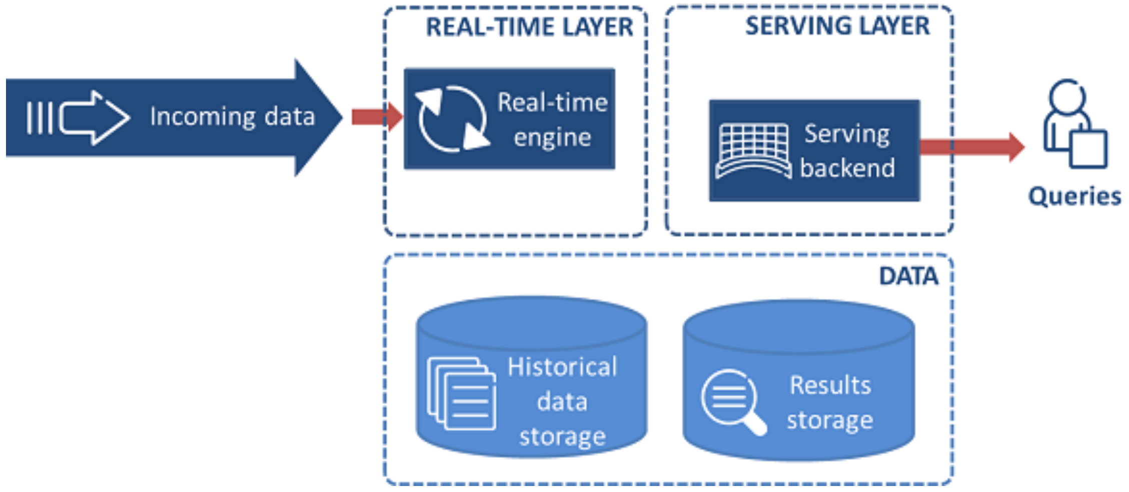

FIGURE 2. Kappa Architecture (taken from https://www.ericsson.com/research-blog/data-knowledge/data-processing-architectures-lambda-and-kappa/)

Choosing the right architecture depends on the application requirements. If algorithms that process real time and historical data are the same and the application does not need heavy and time consuming computations like machine learning then we could go with Kappa architecture. In some cases it might not be possible to simplify data processing and we would be forced to use Lambda architecture.

4. BIG DATA PERSISTENCE

As any technology, relational databases (RDBMS) have strengths and weaknesses. They have been around for decades and they are great for certain tasks - financial software for example. However, they do not scale well horizontally and this is huge disadvantage when working with big data because it is impossible to keep all data on one machine. In order to overcome this problem NoSQL databases emerged. They are good at scaling but they do not solve all persistence problems and as usual the right technology depends on the application requirements. In certain cases more than one database may be used. This is known as polyglot persistence [3].

The database consistency model probably has the biggest impact on application's architecture and business logic in regard of persistence. RDBMS and most graph databases implement the ACID (Atomic, Consistent, Isolated, Durable) consistency model. This means that once a transaction is completed its data is consistent and stable on disk. On the other hand most NoSQL databases are schemaless and implement the BASE (Basic Availability, Soft-state, Eventual consistency) consistency model. They value availability, but do not offer guaranteed consistency of replicated data at write time. This should be taken into account by the architect/developer and application needs to handle eventual data inconsistencies [5].

In NoSQL databases data is distributed across nodes in the cluster and that makes it difficult to query. There are four types of NoSQL databases and due to their different nature there is no common query language as it is by the RDBMS. Some of them have their own query languages, other like key-value stores do not need a query language. However, most of them have support for MapReduce programming model, which allows us to take advantage of the multiple machines in the cluster. MapReduce consists of two separate tasks. The map job takes a set of data and converts it into key/value pairs. The reduce job takes the output from map and combines the values with same keys [3, 6, 7].

NoSQL Database Types

Types of NoSQL implementations are the following:

- Key-value databases are the simplest implementation of NoSQL and as the name suggests they are designed for storing, retrieving, and managing associative arrays (key-value pairs). They are suitable for use cases like storing cache, session data, user profiles/preferences and shopping carts. It is not recommended to use key-value stores if you have relationships among data, multioperation transactions, query by data [3], [8].
- Document databases store documents like XML, JSON, BSON and others. They support query mechanism and have a flexible schema. Document databases are suitable for events logging, CMS/blogging platforms, web analytics, e-commerce platforms. They are not advisable if you need complex transactions or queries against varying aggregate structures [3].
- Column-family databases group the data by columns. A record is a tuple that contains a key mapped to values and values are grouped into multiple column families. It is suitable for event logging, CMS/blogging platforms and counters, i.e. count and categorize web page visitors, expiring usage - create demo users active for period of time. If an application requires ACID transactions then these databases are not a wise choice [3], [9].
- Graph databases store entries (nodes) and relations between them. This allows the data to be interpreted in different ways based on its relations. It makes them convenient for storing connected data like social networks, routing and location based services, recommendation engines. Graph databases may be unable to handle big amounts of data and updates on all or subset of entries are difficult to perform [3], [10], [13].

5. STREAM PROCESSING

Stream processing is probably the hardest and most important aspect of big data. Usually data is incomplete and heterogeneous because it comes from various sources with varying reliability. The processing framework/platform should be able to transform it to appropriate format suitable for analysis. It needs to be able to scale as well in order to handle stream peaks. There are two type of streaming systems - native streaming and micro batching [14], [15].

5.1 Native Streaming

The great advantage of native streaming is its expressiveness. Because it takes stream as it is, it is not limited by any unnatural abstraction over it. Also as the records are processed immediately upon arrival, the latencies of these systems are always better than its micro-batching companions. State full operations are also much easier to implement. Native streaming systems have usually lower throughput and fault-tolerance is much more expensive as it has to take care (persist & replay) of every single record.

5.2 Micro-batching

Splitting a stream into micro-batches inevitably reduces system expressiveness. Some operations especially state management or joins and splits, are much harder to implement, as systems have to deal with the whole batch. The batch interval connects two things, which should never be connected - an infrastructure property and business logic. On the other hand, fault tolerance and load balancing are much simpler to implement. Micro-batching system can be built atop native streaming quite easily.

There are a lot of frameworks available for stream processing and it is impossible to cover all. That is why only Storm and Spark will be mentioned. They are widely adopted and they both are top-level Apache projects.

5.3 Storm

Storm has many use cases: real time analytics, online machine learning, continuous computation, distributed RPC, ETL, and more. It can handle a million of tuples processed per second per node. It is scalable, fault-tolerant, guarantees that data will be processed, and is easy to set up and operate. Storm integrates with a lot of queuing and database technologies. A Storm topology consumes streams of data and processes them in arbitrarily complex ways, repartitioning the streams between each stage of the computation however needed [16], [17].

5.4 Spark

Apache Spark has been rapidly adopted by enterprises across a wide range of industries. Netflix, Yahoo, and eBay have deployed Spark at a massive scale, collectively processing multiple petabytes of data on clusters of over 8,000 nodes. Spark is a general framework for distributed computing that offers high performance for both batch and interactive processing. It has a rich ecosystem of libraries and huge community. Spark is suitable for data integration and ETL, interactive analytics or business intelligence, high performance batch computation, machine learning and advanced analytics, real-time stream processing [18], [19], [20].

5.5 Automatic Cloud Scaling

Elasticity is one of the main cloud advantages. Cloud providers allow users to configure thresholds for scaling up and down depending on machine load (CPU, RAM, HDD, etc). However, these static options are not sufficient for the dynamic and complex applications living in the clouds. A more appropriate approach would be if we use the application metrics and make decisions based on machine learning algorithms. In that way we can predict application peaks and scale up or down cloud's virtual machines or services. This will significantly reduce company expenses and will be far more accurate than the static thresholds currently used. Booting up services and virtual machines takes time. Depending on applications that need to be installed and the necessary configurations this process can take up to several minutes. In order to have enough time to boot the necessary virtual machines to address the increased load on the application, we need to be able to predict high peaks at least 10 minutes in advance. It will ensure reasonable response time and smooth user experience.

CONCLUSION AND FUTURE WORK

We intend to implement automatic cloud scaling platform by using widely adopted open source projects like Prometheus (metrics server), Spark (streaming and machine learning processing), Juju (management and deployment of services to clouds) and Apache Tomcat (web server) [21], [22], [23]. Java is a first choice language for enterprise solutions and it will be the main platform language - Fig. 3.

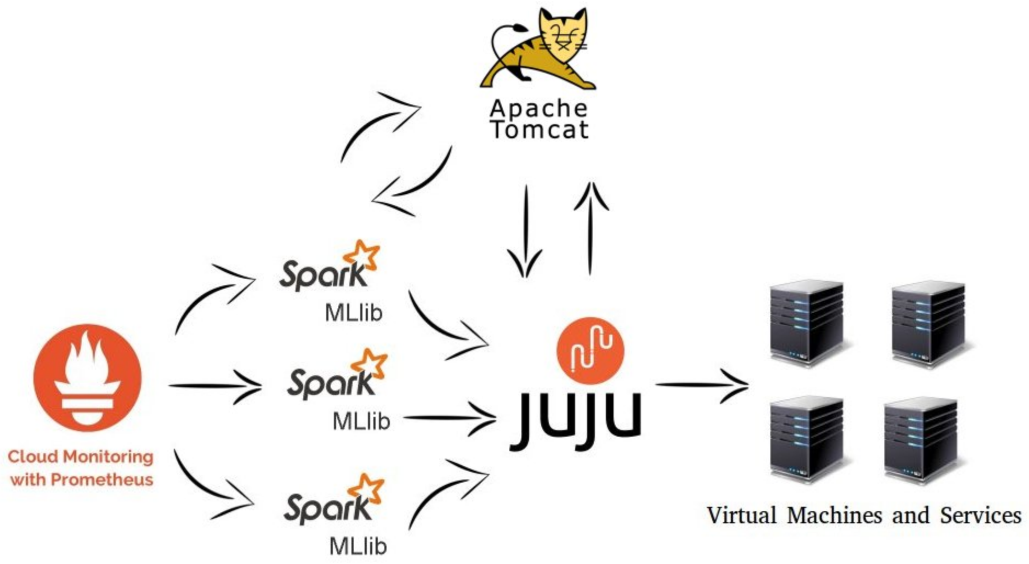
Apache Tomcat
Spark MLlib
Spark MLlib
Spark MLlib
Cloud Monitoring with Prometheus
JUJU
Virtual Machines and Services

FIGURE 3. Architecture of automatic cloud scaling platform.

Prometheus has data exporters for vast number of applications, databases and web servers. We will be able to cover both frontend and backend application bottlenecks. It has been used already by many companies, which is a good foundation for future real world use of the platform.

Juju will allow us to deploy and manage our services/virtual machines to major public clouds (Amazon Web Services, Google Compute Engine, and Microsoft Azure). It is developed by Canonical Ltd., the company behind Ubuntu.

Spark MLlib has a lot of machine learning algorithm implementations and offers high performance.

The platform will offer web user interface for administration and configuration purposes - training neural networks, configuring cloud services, etc. A general overview of the platform is shown in Fig. 3.

REFERENCES

1. M. Kavis, Architecting the Cloud, John Wiley & Sons, ISBN: 9781118617618 (2014)
2. Q. Zhang, L. Cheng and R. Boutaba, Cloud computing: state-of-the-art and research challenges, Springer London, ISSN 1867-4828 (2010)
3. P. J. Sadalage, M. Fowler, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence, Addison-Wesley Professional, ISBN: 9780133036138 (2012)
4. What is cloud computing. Retrieved 10.04.2017 from https://www.ibm.com/cloud-computing/learn-more/what-is-cloud-computing/
5. Graph Databases for Beginners: ACID vs. BASE Explained. Retrieved 10.04.2017 from https://neo4j.com/blog/acid-vs-base-consistency-models-explained/
6. What is MapReduce. Retrieved 10.04.2017 from https://www-01.ibm.com/software/data/infosphere/hadoop/mapreduce/
7. MapReduce. Retrieved 10.04.2017 from https://en.wikipedia.org/wiki/MapReduce
8. Key-value database. Retrieved 10.04.2017 from https://en.wikipedia.org/wiki/Key-value_database
9. Column family. Retrieved 10.04.2017 from https://en.wikipedia.org/wiki/Column_family
10. Graph database. Retrieved 10.04.2017 from https://en.wikipedia.org/wiki/Graph_database
11. Data processing architectures - Lambda and Kappa. Retrieved 10.04.2017 from https://www.ericsson.com/research-blog/data-knowledge/data-processing-architectures-lambda-and-kappa/
12. Lambda architecture. Retrieved 10.04.2017 from https://en.wikipedia.org/wiki/Lambda_architecture
13. Kappa Architecture on Bluemix. Retrieved 10.04.2017 from https://www.ibm.com/blogs/emerging-technology/kappa-architecture-on-bluemix/
14. P. Zapletal, Comparison of Apache Stream Processing Frameworks: Part 1. Retrieved 10.04.2017 from http://www.cakesolutions.net/teamblogs/comparison-of-apache-stream-processing-frameworks-part-1
15. P. Zapletal, Comparison of Apache Stream Processing Frameworks: Part 2. Retrieved 10.04.2017 from http://www.cakesolutions.net/teamblogs/comparison-of-apache-stream-processing-frameworks-part-2
16. Hadoop, Storm, Samza, Spark, and Flink: Big Data Frameworks Compared. Retrieved 10.04.2017 from https://www.digitalocean.com/community/tutorials/hadoop-storm-samza-spark-and-flink-big-data-frameworks-compared
17. Why use Storm. Retrieved 10.04.2017 from http://storm.apache.org/
18. Apache Spark. Retrieved 10.04.2017 from https://databricks.com/spark/about
19. The 5-minute guide to understanding the significance of Apache Spark. Retrieved 10.04.2017 from https://mapr.com/blog/5-minute-guide-understanding-significance-apache-spark/
20. Spark Guide. Retrieved 10.04.2017 from https://www.cloudera.com/documentation/enterprise/5-8-x/topics/spark.html
21. Getting started with Juju. Retrieved 10.04.2017 from https://jujucharms.com/docs/stable/getting-started
22. Prometheus introduction. Retrieved 10.04.2017 from https://prometheus.io/docs/introduction/overview/
23. Apache Tomcat introduction. Retrieved 10.04.2017 from http://tomcat.apache.org/tomcat-9.0-doc/introduction.html