Components and Development in Big Data System: A Survey

Jing-Huan Yu*  |  Zi-Meng Zhou

Abstract—With the growth of distributed computing systems, the modern Big Data analysis platform products often have diversified characteristics. It is hard for users to make decisions when they are in early contact with Big Data platforms. In this paper, we discussed the design principles and research directions of modern Big Data platforms by presenting research in modern Big Data products. We provided a detailed review and comparison of several state-of-the-art frameworks and concluded into a typical structure with five horizontal and one vertical. According to this structure, this paper presents the components and modern optimization technologies developed for Big Data, which helps to choose the most suitable components and architecture from various Big Data technologies based on requirements.

Index Terms—Big Data, cloud computing, data analysis, optimization, system architecture.

1. Introduction

Big Data means a collection of data that can not be crawled, managed, and processed by traditional software tools over a specified time. Big Data technologies refer to the ability to quickly obtain valuable information from various types of data[1]. To keep with the desire to store and analyze the daily growth of complex data, a significant number of analytical platforms are available for analysis of complex structured and unstructured data, each of them is designed to handle specific types of data/workloads[2][3].

This article describes the common features in Big Data platforms’ development according to some modern research. It also lists out some widely-used components and applications of Big Data while exploring those products in use[4][5].

1.1. Applications in Big Data

Big Data technologies are widely used in our daily life. Data analytics is the primary purpose and core feature of...
Big Data. With amounts of data emanating from various digital sources, the importance of data analytics has tremendously grown all these years. There are many industries that have been propelled by Big Data, the most representative ones are as follows.

1) Big Data contributions to public healthcare

At all time, healthcare is always a social issue to be considered and benefited from the raising of Big Data. By keeping track of all the patients’ histories, it can significantly help physicians doing long-term experimental projects. With the massive data storage technology, it makes sure that once a patient receives treatment, her/his data will be safely stored in the data center.

2) Big Data contributions to public sectors

In public sectors, Big Data techniques can provide wide areas searching from corner to corner in various public sector units and allied unions. By providing an extensive range of facilities and pipes for data collecting, Big Data can build a wide area monitory system including the power grid maintenance, deceit recognition, financial promotion investigation, ecological fortification, etc.

3) Big Data contributions to education

Teaching by their aptitudes is a critical concept in education, but the lack of educational resources and population bases makes this concept rarely used in real life. Data-driven classrooms provide a solution much further than just coursework reformation and grading development. By collecting various online resources and customized intelligent recommendation services for both students and teachers, these applications can provide adaptive learning solutions and problem tracking services for educators.

4) Big Data contributions to banking zones and fraud detection

From a certain level, Big Data techniques are designed for the area at first. With the raising of anomaly detection, the usage of Big Data has been widely expanded. Detecting the misuse of credit cards or debit cards, archival inspection tracks, and venture credit hazard treatment are the basic usages of Big Data technologies.

1.2. Challenges and Complex Architecture in Big Data

With the widespread use of these techniques, several challenges in Big Data have exposed. Privacy and security, accessing and sharing of information, analytical challenges, and so on make the optimization of Big Data technologies desperately needed. The common motivation of expanding these techniques is to find an alternative way to achieve the highest performance in a cost-effective manner and under given limitation by mostly utilizing the desired factors.

Big Data applications, due to their complex components and frequent updates, form a unique application ecosystem based on the aggregation of computing engines. In this paper, we focus on the products and their improvements in the integration of disparate data sources, combination with high-performance hardware, and execution process optimization. According to the projects studied during the research period, we summarized these components into a five-horizontal-and-one-vertical structure to help to learn and organize the Big Data products from the inside.

2. Vision of Conclusion

Fig. 1 is a partitioning method proposed in this paper, consisting of four main layers and six different modules. Moreover, it should be noticed that this picture does not mean the dependency relationship between each layer. The criteria for this classification are based on the primary use of each layer. Moreover, components playing several roles in different systems may only be mentioned about its most representative function. The exact layer of these components divided into is based on its typical use case.
2.1. Application Layer

The user application layer is the level that users can directly contact, access, and control. According to the characteristics of the enterprise, we divide applications into different types such as Web services, integrated data access, and decision support. Most of the products in this level aim at the specific needs of customers and customized development of the company’s main business. The specific functions and implementations rely on various development frameworks and message components, which are the primary embodiment of cloud computing products.

2.2. Data Processing Layer

Data processing layer is the most complex part of Big Data applications. System architects always face various choices on where to process their data, each choice with possible orders of magnitude differences concerning the performance. Though there are many related studies like cross-platform framework, full stack support, and other solutions (which will be mentioned more in Section 4) to simplify this layer in recent years, the challenges mentioned above are evident in this layer. In order to have more universal and flexible architecture to meet different system requirements, we divide this layer into three smaller modules.

1) Data access module

Components in this module are mainly to achieve read-write separation and the application-oriented query, providing direct data access interfaces for application services like on-line transaction processing (OLTP) and on-line analytical processing (OLAP). In addition to traditional relation database management system (RDBMS) and data warehouses like Oracle and DB2, the open source database is an important type of service that constitutes this level. Compared with solutions that are maintained by the company and constantly tapping the requirements, open source solutions tend to be built for a specific need.

In modern Big Data platforms, there are three kinds of components in data accessing: Real-time query component, structured query language (SQL)-based calculating component, and multi-dimensional query component with both characteristics.
2) Data analysis module

In the early stage of exploration in the module, statistical product and service solutions in full (SPSS), was the first commercialization product in the field of data analysis. This module mostly provides the upper-level components in the whole system, which is usually built on top of the calculation engine and is used to implement various data analysis algorithms.

Unlike the user application layer, this module does not directly provide an interface to system users in common cases. The generated data usually gets stored or iterated as an intermediate result or as data input for other modules in the system. There are many famous frameworks in this layer like TensorFlow[21], Apache SINGA[22], and spark machine learning library (MLlib)[23]. Represent that components in this module are mostly designed for the certain requirement types.

3) Data calculating module

The calculating module provides massive data computing foundations in the whole architecture. This paper divides different components according to different processing models, basically covering batch-only, stream-only, and hybrid frameworks. The products developed for this layer have obvious mash-up characteristics, which are embodied in the mutual penetration, reference, and invocation between these components.

4) Conclusion

The processing layer performs processing in the system through data reading data from storage systems or other third-party systems. According to the implementation details of each specific application, different frameworks will put some of the same components into several layouts. However, we still can map this structure to each of them.

2.3. Data Collecting Layer

The main task of this level is to collect data in batches. Big Data has 4V features: Volume, velocity, variety, and value[1]. As the core module is responsible for the data collection, the specific requirements this layer should handle with can be summarized as follows.

1) Diversified data collecting capabilities: Big Data systems always need to collect data from various sources, so the components set needs to fit the requirements of collecting real-time, incremental, and various forms of data such as tables, files, messages, and third-party systems[24,25]. It also means that platforms should also provide support for all types of distributed data storage systems like RDBMS, Hadoop distributed file system (HDFS), and network attached storage (NAS)[26,27].

In addition to the traditional extract-transform-load (ETL) technology, the growing requirements need the platforms providing more flexible functions like real-time collection, Internet crawler analysis, and more kinds of data sources.

2) Visual rapid configuration capability: There is an important concept in software engineers that the labor cost in maintenance of a system is much higher than the cost in developing, so it is necessary to provide an easy maintaining interface set. Basically, developers would prefer graphical development and access interface, drag-and-drop developing and management graphical user interface (GUI) suites in order to free code writing and reduce acquisition difficulty. These tool suites will finally reach the goal of cutting the cost wasted on simple and repeat work[28].

3) Unified scheduling management and control capabilities: Big Data systems always need multiple data collecting sources, which means that it is important for the system to schedule collection tasks and interfaces.

 Mostly, software manufacturers or communities providing toolkits in this layout may focus on the first feature, about how to collect massive data efficiently. We will list out the most famous ones in this area in subsection 3.2.

2.4. Platform Management Layer

The management layer of Big Data platforms mainly contains application management tools and system
management tools. From the perspective of application management, it is responsible for accessing and monitoring various component instances and scheduling and configuring external users and systems. The system management part includes deploying the developed applications, internal resource and authority assignments, and visual operation and maintenance. The most representative product may be Amazon Web services (AWS), which provides the standard for cloud computing and virtual cluster management product.

3. Representative Components

In this section, we give a detailed review of the components in the data processing layer and data collecting layer. We introduce several typical products in use for each layer and then compare them in the fitness of the layer’s features and core algorithms or techniques.

3.1. Data Processing Layer

The data processing layer consists of three parts: Data accessing module, data analysis module, and data calculating module. According to the use cases and their features, we deliver several different component sets to show the main usage of these modules.

3.1.1. Data Accessing Module

Data accessing products usually provide direct access to data sources, mainly building beyond data storage services and focusing on real-time querying, conditional filtering, and data transportation. There are three typical kinds which can be divided by interfaces and calculating models: Real-time querying component, SQL-based computing component, and multi-dimensional query component.

A. Real-Time Querying Component

As for real-time querying products, Facebook Presto, Apache Drill/Dremel, and Impala are the most representative ones.

1) Presto: Facebook opened the source code of Presto in November 2013. This distributed SQL query engine is designed to perform the high-speed, real-time data analysis. It provides American National Standards Institute (ANSI) structured query language (SQL) as a standard operating interface, including complex queries, aggregations, joins, and window functions. Presto designed a simple abstraction layer for data storage to satisfy queries which store data in different systems (including HBase, HDFS, and Scribe). Fig. 2 shows the Presto’s system architecture.

Presto uses a custom query execution engine and response operators to support SQL syntax. In addition to the improved scheduling algorithm, different clusters end to form a pipeline through the network. This mechanism avoids additional delays caused by unnecessary disk reads and writes. This pipelined execution model runs multiple data processing segments at the same time; once the data is available, it will pass the data from one processing segment to the next. This approach will greatly reduce the end-to-end response time of various queries. Meanwhile, it can avoid a lot of intermediate results in the underlying operation, improving the running effect and cutting wasted resources.

2) Apache Drill and Google Dremel: Dremel is a Google’s interactive data analysis system. Currently used in open source projects is the implementation project, Apache Drill. The features of the Dremel system are as follows.
a) Large-scale: It is possible to form a cluster which contains thousands of scales and process data at the petabyte (PB) level.

b) Nested data models: The Internet’s data is often non-relational. Dremel supports a nested data model like the JavaScript object notation (JSON) format. In the implementation of Drill, it also has the characteristics of schema-free, which can run in HDFS\cite{37}, Not Only SQL (NoSQL)\cite{40}, and cloud storage systems\cite{41}.

c) Concept of Internet search and parallel database management system (DBMS) based: Dremel and Drill’s architecture draws on the concept of service tree used in distributed search engines\cite{39}. Unlike Pig\cite{42} and Hive\cite{43}, it performs queries directly instead of translating them into MapReduce (MR) tasks.

Because of these characteristics, Dremel reduces the minute-level overhead required for MapReduce processing data to the second level as Figs. 3 and 4 show\cite{33,44}.

Dremel uses a multi-level service tree to execute queries (see Fig. 5). A root node server receives the incoming query, reads the metadata from a table, and routes the query to the next layer. The leaf server is responsible for communicating with the storage layer and accessing the data directly on the local disk. When the root node server receives the query, it will ensure the horizontal partition of all these tables and rewrite the query into a union statement. This computation model is ideal for aggregating queries that return fewer results, taking the most advantages of combining a large amount of low-cost hardware.

3) Impala: Impala is a real-time interactive SQL Big Data query tool developed by Cloudera inspired by Google’s Dremel. It is a combination of Google Dremel architecture and massively parallel processing (MPP) architecture\cite{45,46}. Impala uses a similar distributed query engine in a commercial parallel relational database, consisting of query planner, query coordinator, and query exec engine (as shown in Fig. 6).

B. SQL-Based Computing Component

This kind of component is built to facilitate the work habits of data scientists. These products realize the abstraction of data computing and processing services by providing SQL-like languages. The most common components are Hive\cite{44}, Pig\cite{42}, and Spark SQL\cite{47}. These products are not real SQL query tools, but rather provide a SQL-like statement interface to implement operations on each one’s computing model (like the MapReduce\cite{39} tasks in Hive and Spark resilient distributed dataset (RDD)\cite{3} in Spark SQL).

1) Hive: Hive is a data warehouse tool based on Hadoop. It maps structured data files into a database table and provides a simple SQL query function. Hive is designed to convert SQL statements into MapReduce tasks. Due to the low learning cost, users can quickly implement complex MapReduce statistics through SQL-like statements.
Fig. 7 shows the main structure of Hive system, mainly divided into user interfaces, metadata storage, database drivers, and MapReduce engine.

a) User interfaces: There are a variety of user interfaces, but the most used in practice is the CLI and the protocol interface. CLI is the command line interface, which starts a copy of Hive at the same time it starts. Hive provides Hive query language (HQL) operation mode through Java database connectivity (JDBC) and thrift protocols.

b) Metadata storage: Hive stores metadata in third-party databases, such as MySQL and Derby. The metadata mainly contains the name, columns, and partitions of tables in use and their attributes, like the directory where these tables’ data located.

c) Database driver: Though there has been through many versions’ evolution since Hive had been designed out, the main structure has never changed. Driver models mainly include an interpreter, a compiler, an optimizer, and a physical plan and its executor. The HQL query statement sequentially performs lexical and syntax analysis, compilation, optimization, and query plan generation in the drivers. The generated query plan is stored in HDFS and subsequently executed by MapReduce calls.

d) MapReduce and storage: Hive stores data in HDFS and uses the MapReduce engine to do the query executions.

2) Pig: Pig is a large-scale data analysis platform based on Hadoop. It provides a SQL-like language called Pig Latin. The compiler of this language converts SQL-like data analysis requests into a series of optimized MapReduce operations. Fig. 8. shows the architecture of its main framework. Pig provides a simple operation and programming interface for complex, massive data-parallel computing. Pig’s main structure and running process are just like Hive. Both two products convert SQL/SQL-like statements into logical plans and physical plans and can be executed by the MapReduce engine. However, unlike Hive, Pig has its unique Syntax called Pig Latin which has a great difference with ANSI SQL in programming ideas and data structures.
a) Programming ideas: Pig Latin is a data flow programming language, and SQL is a declarative programming language. Pig declares the process of processing and converting data, while SQL only defines what data is needed. To some extent, Pig scripts are similar to the SQL’s query plan which translates declarative results into system steps.

b) Data structure: In SQL, data is stored in a table and bound to a specific schema. Pig’s data structure is loose, and it defines schemes during processing. Also, Pig supports complex nested data structures, which is quite different from SQL’s flatter table structure. Pig’s user design function (UDF) and streaming operations (for adapting UDFs written in different languages) make Pig more flexible and powerful.

3) Spark SQL: Spark SQL’s predecessor is Shark. Fig. 9 shows the evolution between these two components. Just like Hive, Spark SQL provides a quick start-up tool for technicians who are familiar with RDBMS but do not understand Spark RDD. Since Hive runs on Hadoop, a large number of intermediate disk landing processes in MapReduce computing process consume a lot of I/O. Spark group developers modified the optimizer and execution plan in Hive to access Spark[R], an in-memory computing engine. Based on Shark’s foundation, the Spark SQL project has got rid of Hive’s specialized components such as syntax parser and query optimizer, while expanding data compatibility and optimizing operational performance, thus greatly improving components’ expansion capabilities. Benefits from its loose structure are shown in Fig. 10. Spark can adopt several different optimization techniques, such as in-memory columnar storage and byte-code generation. This architecture also makes Spark SQL much more convenient for developers to redesign the SQL parsers, planners, and optimizers according to their requirements.
C. Multi-Dimensional Query Component

One of the most representatives of these components is Apache Kylin\cite{45,46}, the multi-dimensional query component. Although data warehouses have gradually replaced relational databases as the underlying data storage services for Big Data platforms, relational databases still occupy the core position of the storage layer. Rational on-line analytical processing (ROLAP) is an improvement in on-line analytical processing (OLAP) technique, which performs dynamic multi-dimensional analysis on data stored in a relational database. These query components can directly perform multi-dimensional operations on the contents from the relational database system.

Kylin is designed to reduce the latency of queries ten-billion size of data on Hadoop/Spark, using ANSI SQL to provide an interface that supports most of the query functionality. As shown in Fig. 11, Kylin can combine with a wide range of business intelligence tools such as Tableau, Power BI/Excel, microstrategy incorporated (MSTR), Qlik Sense, Hue, and SuperSet directly. With Kylin, users can perform sub-second interactions with Hadoop data, providing better performance than Hive. The biggest feature of Kylin is that users can define a data cube and pre-build it in Kylin with massive data records in Kylin.

3.1.2. Data Analysis Module

Components in the data analysis module are mostly frameworks developed for different programming languages.

R and Python are similar, symbolizing a script-based technology stack, both use statistics techniques for data analysis. The main difference between two technique stacks is that Python is more inclined to engineering development and has direct support for developed algorithms such as word segmentation and clustering required for many projects; while R focuses more on statistical drawing, which provides better visualization support from the native level. R language rarely mentions its technical framework, but rather serves as an interface to other frameworks or Big Data engines. Both languages are based on sample statistics. And the scripting language relies on the characteristics of their execution environments, the support for large-scale data is limited. In most cases, they need to cooperate with C++ or other more efficient and lower level languages.

The Java virtual machine (JVM) based domain is much more systematic due to its mature cross-platform technology, complete development suites support, and the strong engineering management structure. A variety of data analysis tools based on different computing concepts have emerged, most of them are dependent on Apache
Hadoop and Apache Spark. Both build a whole environment for developing suites and provide a powerful and full functioning interface set.

Unlike the other modules in this layer, products are more demand-oriented. It is hard to list out the frameworks developed for all purposes, so we just mention two most popular frameworks in machine learning (ML).

1) TensorFlow: For those who have heard of deep learning but have not been too thorough, TensorFlow may be their favorite deep learning framework. More than the usage of a deep learning framework, its official definition is an open source software library for machine intelligence. So it is better to consider that TensorFlow is an open source library for numerical calculation using data flow graphs\(^{[21]}\). TensorFlow\(^{[48]}\) is not just a deep learning framework but a graph compiler. TensorFlow is a relatively low-level programming framework that supports Python and C++, allows computational distribution on CPU and GPU, even supporting horizontal scaling with a remote procedure call produced by google (gRPC) (see Fig. 12). Most of its kernel codes rather use a combination of highly optimized C++ and CUDA (NVIDIA’s language for programming GPUs\(^{[49]}\)) to write and optimize eigen (high-performance C++ and CUDA libraries) and NVIDIA’s cuDNN (a dedicated deep neural network (DNN)\(^{[50]}\) library for NVIDIA GPUs for convolution and other functions)\(^{[51]}\).\(^{[52]}\).

2) Spark MLlib: MLlib\(^{[51]}\) is the Spark’s ML library. It provides a set of algorithms and frameworks to build a practical, scalable, and easy ML develop environment. There are several tools provided such as\(^{[53]}\):

   a) ML algorithms: The common learning algorithms such as classification, regression, clustering, and collaborative filtering.
   b) Featurization: Feature extraction, transformation, dimensionality reduction, and selection.
   c) Pipelines: Tools for constructing, evaluating, and tuning ML pipelines.
   d) Persistence: Saving and loading algorithms, models, and pipelines.
   e) Utilities: Linear algebra, statistics, and data handling.

Fig. 12. Architecture of TensorFlow system.

Spark MLlib can directly access into Spark’s application programming interface (APIs) and interoperate with Numy in Python and R libraries. Developers can use any Hadoop data source (e.g., HDFS, HBase, or local files), making it easy to plug into Hadoop workflows (example steps in using MLlib are shown in Fig. 13).

Spark uses a new in-memory computation model called RDD, making MLlib run much faster than other
products based on MapReduce (see Fig. 14). The development group also cares about algorithmic optimization: MLlib’s algorithms set uses leverage iteration high-quality algorithms that can yield better results than the one-pass approximations.

3) Conclusion: We display two main strain products in the data analysis module, TensorFlow is a typical Python-interfaced and C++-implemented mixture framework. It depends on the cooperation of front end and execution system, focusing on the solution of providing a high-performance programming library. Spark MLlib is built on the Spark calculating system, transferring common ML algorithms into Spark internal operation chains and taking the advantages of this mature system. These are two main thoughts in developing data analysis module components.

3.1.3. Data Calculating Module

Big Data is a general term for the technologies designed to collect, organize, and process large datasets which cannot be solved by traditional solutions. With the growth of demand of popularity, scale and value in kinds of computing have expanded considerably in recent years.

1) Apache Hadoop

Batch processing has a long history in computing methodology research[^52]. Fig. 15 shows the batch processing engine called MapReduce, which provides sequential operations on a large-scale, static, structured, or semi-structured dataset and returns the result until the computation is complete, which makes it well-suited for calculations access to a complete record set. Due to the benefits of handling large volumes of persistent data, this processing model is frequently used in historical data analysis.

Apache Hadoop[^53] is a processing framework designed for distributed batch processing. It was the first Big Data framework and has been greatly improved by the open-source community. Hadoop reimplemented the algorithms and components according to the papers and reports from Google, containing the following parts that work together to process batch data.

---

[^52]: Batch processing has a long history in computing methodology research.
[^53]: Apache Hadoop is a processing framework designed for distributed batch processing.
a) HDFS: HDFS is a distributed file system module which provides coordinates storage services and data replication through cluster nodes. HDFS is widely used in Big Data frameworks as the storage system due to its efficiency.

b) YARN: YARN is the cluster coordinating component of Hadoop stack, responsible for coordinating and managing the underlying resources and scheduling jobs to be run.

c) MapReduce: MapReduce is the Hadoop’s native batch processing engine. Its processing technique follows map and shuffles and reduces algorithm using key-value pairs.

Fig. 15. MapReduce processing engine.

This methodology tends to be slow for its heavily leverages permanent storage and reading and writing multiple times per task. However, as the disk space is one of the most abundant server resources, MapReduce can handle enormous datasets at a relatively cheap cost. Because it does not encroach on memory, Hadoop cluster achieves running on less expensive hardware than some alternatives. With the support of cluster manager YARN, Hadoop has incredible scalability potential and has already been used in the production cluster which consists of thousands of nodes.

2) Apache Storm

Stream processing systems do computation work over datasets as they enter the system. The main difference between stream processing and batch processing is defining operations. Stream process systems define operations to each item of data, while batch process systems define operations on an entire dataset. This makes stream processors can handle the items as they come into the system. The most significant feature of stream processing is the event-based calculating processes which will not end until the input explicitly stops. Computation results are immediately available and continually updated when new data arrives.

Apache Storm<sup>[54]</sup> is a stream processing framework that focuses on low latency. Apache Storm meets the demands of near real-time processing. It can handle large quantities of data and deliver results with less latency.
Fig. 16 provides a typical structure base on Apache Storm. Apache Storm provides stream processing by topologies, a directed acyclic graph (DAG) based scheduler. These topologies mean that the transformations and steps will be applied to each data item. Topologies are composed of streams, spouts, and bolts. The stream is unlimited data that continuously arrives at the system. Spouts are sources of data streams at the edge of the topology.

Moreover, bolts represent a processing step that consumes streams, applies an operation to them, and outputs the result as a stream. Apache Storm is probably the best solution currently available for near real-time processing. It handles data with extremely low latency for workloads which need to be processed with the minimal delay. Apache Storm is often a good choice when processing that time directly affects user experience\cite{YU et al.}. Storm can integrate with Hadoop’s YARN cluster manager and HDFS storage system, making it easy to hook up to an existing Hadoop deployment (as shown in Fig. 17).

Apache Storm gives developers options to use mini size batches instead of pure stream processing over the datasets, providing users more flexibility to shape the tool to the intended use. However, it tends to negate some of the most significant advantages over other solutions. Moreover, Apache Storm core modules do not offer to order the guarantees of messages, it means that processing upon each message can be guaranteed, but will cause massively a duplicate burden.

3) Apache Spark

In order to fit different use cases in application scenes, there are frameworks using plugins and integrated interfaces to handle both batch and stream workloads.

Apache Spark is a batch processing framework combining SQL, streaming, and sophisticated analytics (see Fig. 18). Spark streaming is the component handling stream processing. As for batch processing, Apache Spark focuses on speeding up batch processing workloads by offering full in-memory computation and processing optimization.

Unlike MapReduce, Spark processes all data in-memory and only accesses the storage layer and the
beginning and end of the entire process, to initially load the data into memory and to persist the final results. All intermediate results are managed in memory. There are several exceptions like cache or OOM cases may force the system taking write operations. Spark uses a model called RDD to implement its in-memory computation. These immutable structures exist in memory and represent collections of data that can be backtracked based on lineage, avoiding access to disk storage each time. It can perform better on disk-related tasks by creating DAGs to represent all the operations the data needs to be operated on and the relationships between them. The scheduler will assign the workloads according to the sequence of the nodes in DAGs, making the coordinate work between the processors more intelligent. Spark streaming, the plugin buffering the stream in sub-second increments, is the stream processing solution for Apache Spark. Fig. 19 shows the process of splitting the input stream into fixed batches. Spark streaming cannot work as well as the native stream processing frameworks, but it does work well in practice.

Apache Spark is a universal option for diverse processing workloads. Trading off high memory usage, it provides excellent speed advantages on batch processing workloads and fits well in handling workloads that value throughput over latency.
4) Conclusion

For batch-only workloads, application demands are not time-sensitive, and on a vast scale, Hadoop can be a good choice for using less expensive equipment than other solutions.

For stream-only workloads, Storm has extensive language support and can deliver lower latency processing.

For mixed workloads, Spark provides high-speed batch processing and micro-batch processing for streaming. It has broad support, integrated libraries and tooling, and flexible integrations.

Options for processing within a Big Data system is plentiful, the best fit for the users’ situation should depend on the state of the data to process, how time-bound the requirements are, and what kind of results developers are interested in.

3.2. Data Collecting Layer

Data collecting plays a significant role in the Big Data cycle. The Internet provides almost infinite data on a variety of topics. Behaviors from system users also bring large valuable data helping to improve services and find demands. We will introduce some most popular products according to different features of these systems.

1) HDFS

Hadoop file system (HDFS)\(^{(36)}\) stores large files in a streaming data access mode and runs on a commodity hardware cluster. It is a file system that stores data across multiple computers in a management network.

HDFS is a most popular file system built for offline collecting purpose, which means that it may not meet the demands such as low latency for data access, storage of large numbers of small files, multi-user writes, and arbitrarily modify files.

HDFS uses typical master/slave architecture. As shown in Fig. 20, an HDFS cluster consists of a single NameNode and multiple DataNodes. NameNode is the master server which stores file system namespace, executes namespace operations, and determines the mapping of file blocks to DataNodes. As for DataNodes, they are processes managing storage services attached to the nodes that they run on, responsible for serving read and write requests. The high availability and disaster back up are implemented by block duplicating mechanism, the file inside the storage system will be split into blocks and store them in a set of DataNodes.

2) HBase

HBase\(^{(35)}\) is a distributed, column-oriented open source database, which implements the BigTable\(^{(56)}\) based on HDFS. HBase is a subproject of Apache’s Hadoop project, and Fig. 21 demonstrates a system built on these components. Different from a general relational database, HBase is a database suitable for unstructured data storage, and HBase is column-based rather than a row-based model.

Fig. 20. HDFS system architecture.

Fig. 21. HBase system architecture.
A typical HBase cluster consists of an HMaster and several slave nodes led by HRegionServer. Each HRegionServer node contains its log module and multiple HRegions, a segment of the whole recorded table divided by keys. HRegion is also the minimum unit of distributed storage and load balance.

The cooperation mode inside HBase is also a typical master-slave model which consists of many layers processing different workloads. HMaster cooperates with ZooKeeper, the cluster managing system, responsible for allocating HRegion to each HRegionServer and their loads balancing, discovering invalid HRegionServer and reallocating them, garbage collecting of HDFS, and handling requests for schema update operations.

3) NoSQL storage system

NoSQL refers to a non-relational database. With the rise of the Internet web2.0 technology, the traditional relational database cannot fit the ultra-large-scale and high-concurrency requirements occurred in massive data exchange scales. The NoSQL database was created to solve the challenges brought by multiple data types of large-scale data collection, especially Big Data application problem. It is hard to classify NoSQL, since each kind of them has its benefits and focus point. Recent years, in-memory database gains great development. This kind of database stores most of its contents in memory instead of storing in disk or other extern storage. LevelDB and RocksDB are the most representative ones in this area. Version controlling is the main issue in blockchain technology. OrpheusDB provides dataset version control services as its highlight design while ForkBase implements structurally-invariant reusable indexes (SIRI) indexes as an efficient storage engine for blockchain and fork-able applications.

4) Message queue products

In addition to the above-mentioned offline requirements, real-time acquisition is now an essential component of Big Data platform. The mainstream architecture uses Flume as data entry and uses Kafka’s customized message queue to process data in real time. Combination with stream processing and in-memory database to form data aggregation could achieve real-time collection and analysis. Such architecture schemes are often based on open source systems and have a high degree of reliability. However, since open source systems are often based on a broader range of application scenarios, they may need additional development to meet the specific demands, and it is difficult to adjust and improve in a short time.
5) Other data collecting channels

In addition to the two data collecting’s basic demands described above, Internet information collection in the form of enterprise crawler systems and third-party data sources fulfill the content in this layer. Most of the data involved in these components needs pretreatments like indexing and slicing. Projects servers these features such as Solr\[62\], Lucent, Nutch, and ES, their primary purpose is to convert the natural language or document structure into a structure more suitable for storing queries.

4. Platform Optimization Directions

With the explosive growth of data volumes, even Big Data technologies face many challenges. Focusing on the topics of high efficiency and scalability, we summarize some of the recent research achievements.

4.1. Combination with High-Performance Materials

In recent years, material technology and heterogeneous computing have gained much development. Using hybrid architecture as the solution for Big Data products enhancement is getting more and more attention from industry.

Non-volatile memory materials like Open-Channel solid-state drive (SSD)>[63] and 3D X-point\[64] with the characteristics of non-volatile and high performance than traditional memory material shines in in-memory storage system optimizations\[65],[66].

Despite the new techniques in storage layer, high-performance computing chips are equipped into the system. Solving the performance bottleneck of traditional databases with dedicated hardware has become a trend\[66],[67]. However, the current acceleration solution has the following two shortcomings.

1) The current acceleration scheme is designed for the certain layer, mostly the SQL execution layer, and the co-processor is usually placed between the storage and the host as a filter. Although there have been many attempts to improve OLAP systems by field programmable gate arrays (FPGAs), the design of accelerating solutions is still a challenge for OLTP systems.

2) As chips become smaller and smaller, internal errors are becoming an increasing threat to system reliability. For a single FPGA chip, the probability of an internal error grows greatly in 5 years, the design of fault tolerance mechanisms is particularly important for large-scale availability systems.

In general, these systems enhance the performance by taking advantages of the characteristics of new materials. However, the defects, instability, and high price of new materials make these solutions hard to become commercial products.

4.2. Internal Process and Algorithm Optimization

Benefit from the development of network transmission technology, Internet of Things also makes great progress\[68],[69]. Optimizations in the process and algorithm most focus on complex-structured data\[70], business logical evaluation-based refactoring\[15],[17],[71], or new data structure and algorithms in traditional cases\[65],[72]. These optimization solutions provide approaches at a more fundamental level such as cutting down memory cost, delaying data transportation, and calculating, which are much more stable and commercially feasible than simply taking advantages of new materials.

4.3. Scalability Improvements

As this article has always wanted to explain, the ecosystem of Big Data components is very complex. To provide one single processing through different environments, Musketeer\[14] and Rheem\[12],[13] provide solutions to
build cross-platform frameworks, helping to increase system scalability. Although these frameworks provide good cross-platform characteristics over the years, we still have not got rid of the existing framework (as shown in Fig. 22) system from 2014. In addition to proving that the structure we have summarized is correct, we still hope to see a new and expanded framework which can achieve improvements by changing the overall structure.

5. Conclusion

The modern Big Data platform architecture has following difficulties in getting a unified structure: In order to meet different scenarios, more components will be adopted, which is the first difficulty; and in order to support more application scenarios as much as possible, each product is often designed to be very broad and consistently requires additional calculations models to adapt to the needs, this is the second difficulty; on the basis of the first two, each computing component will learn from each other, infiltrate each other, and call cyclically from each other, making it almost impossible to directly stratify between components.

Due to its complexity, it is hard to give out a fixed standard for the division of Big Data platform architecture. Though the architecture of them is much similar to each other, different application environments and scenarios of the authors’ requirements, frequent updates of the Big Data system itself, and staggered classification of applications make them never be a framework suitable for even common cases. Reasons such as staggering did not lead to a unified conclusion. In this paper, we try to inspire a common protocol which can aggregate all these processing flows, just like the REST APIs in microservice, by providing a detailed review on the most commonly used components now.
References


Jing-Huan Yu was born in Jiangxi Province, China in 1996. He received the B.S. degree from University of Electronic Science and Technology of China, Chengdu, China in 2018. He is currently pursuing the Ph.D. degree with City University of Hong Kong, Hong Kong, China. His research interests include data mining, database engine, and non-volatile memory.

Zi-Meng Zhou received the M.E. degree from the Department of Computer Science and Technology, Shandong University, Jinan, China in 2015. He is currently pursuing the Ph.D. degree with the Department of Computer Science, City University of Hong Kong. His research interests include non-volatile memories, embedded systems, and data-intensive computing.