biggy: An Implementation of Unified Framework for Big Data Management System

Yao Wu, Hong Chen, Cuiping Li, Henan Guan
Key Laboratory of Data Engineering and Knowledge Engineering of Ministry of Education, Beijing, China
School of Information, Renmin University of China, Beijing, China
Email: {ideamaxwu, hongchen, cuipingli, henanguan}@ruc.edu.cn

***Abstract*—Various tools, softwares and systems are proposed and implemented to tackle the challenges in big data on different emphases, e.g., data analysis, data transaction, data query, data storage, data visualization, data privacy. In this paper, we propose datar, a new prospective and unified framework for Big Data Management System (BDMS) from the point of system architecture by leveraging ideas from mainstream computer structure. We introduce five key components of datar by reviewing the current status of BDMS. Datar features with configuration chain of pluggable engines, automatic dataflow on job pipelines, intelligent self-driving system management and interactive user interfaces. Moreover, we present biggy as an implementation of datar with manipulation details demonstrated by four running examples. Evaluations on efficiency and scalability are carried out to show the performance. Our work argues that the envisioned datar is a feasible solution to the unified framework of BDMS, which can manage big data pluggablly, automatically and intelligently with specific functionalities, where specific functionalities refer to input, storage, computation, control and output of big data.**

***Index Terms*—big data management system, data processing, unified framework, datar, biggy**

I. Introduction

A. From Computer to Datar

As Alan Turing proposed the question "Can machines think?" [1], the imitation game begins. However, before that, Von Neumann had started an engineering research on computer and described a logical design of a computer using the stored-program concept in 1945, which has controversially come to be known as the Von Neumann architecture [2]. Before these, English mathematician and computer pioneer Charles Babbage proposed the Analytical Engine, a designed mechanical general-purpose computer. The goal of these pioneers is to design a computing machine better than human brains, which can liberate human from manual work and tedious computation.

During the last several decades, data management principles such as relational model of data, physical and logical independence, declarative querying and cost-based optimization have led to several fields of researches and a prosperous industry. More importantly, these technical advances have laid the foundation for managing and analyzing big data today. Many novel challenges and opportunities associated with big data necessitate rethinking many aspects of these data management platforms, while retaining other desirable aspects. The practice [3] and theory [4] contributions of Bachman and Codd open up the research on database. And the steps on the road to data management never stop such as, Ingres [5], Postgres [6], Mariposa [7], C-Store [8], VoltDB [9], AsterixDB [10] and P-Store [11] in database field, as well as, Megastore [12], Spanner [13], MillWheel [14], Azure CosmosDB[1] and TiDB[2] in distributed system field.

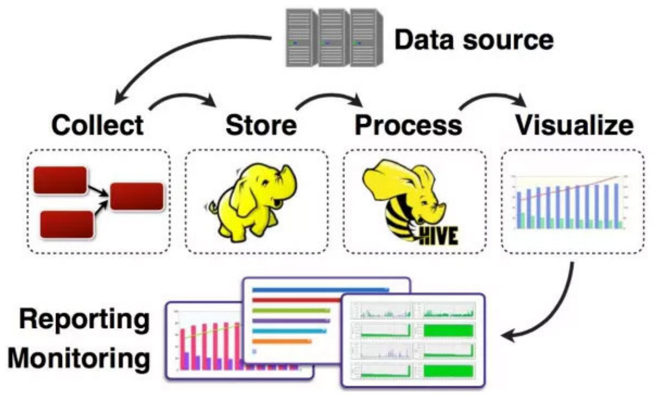


Fig. 1. A typical workflow for big data management.

Michael Stonebraker proposed "On Size Doesn't Fit All", and in this paper, we try to argue that "All Can Fit in One". Since the computing power of machines becomes stronger, we can sniff the shift from computation to data management to explore more in-sight information and knowledge from data. Jim Gray foresighted the transformation from computation-intensive to data-intensive science discovery and brought forward The Fourth Paradigm [15]. He also thought the only way to cope with such paradigm was to develop a new generation of computing tools to manage, visualize, and analyze massive data. As we all know, Big Data Management System (BDMS) is a complex set of functionalities, we think it necessary to propose a unified architecture to guide the design of BDMS. From these observations, we summarize and conclude with five main components in BDMS to provide a better explanation for a full understanding of our proposed datar architecture.

As shown in Fig. 1, BDMS consists of several core components such as, collect, storage, process and visualize. Compared with traditional database systems, BDMS architecture is more flexible and open for varied requirements due to different

[1]https://azure.microsoft.com/zh-cn/services/cosmos-db/
[2]https://www.pingcap.com/

focus-ons. In this paper, we unify the BDMS as **datar**, a general framework to design and build BDMS, corresponding to term **computer**.

As we all know, the mainstream computer architecture is divided into five parts, i.e., input, storage, computation, control and output, in which, computation is the center. If we look closely, we can find that, BDMS is much the same as computer, consisting of (data) input, (data) storage, (data) computation (query/analysis), (data) control (transaction/recovery) and data output (visualization), in which, data storage is the center. In other words, we can name a computer Fast Computation Processing System (FCPS). Likewise, the BDMS is called as datar, focusing on data. To better explain the similarities and differences between a computer and a datar, in terms of architecture, we use Fig. 2 to illustrate. In Fig. 2 (a), five core components of a computer are shown in separate rectangles, while in Fig. 2 (b), five corresponding parts are shown. A computer and a datar share the similar functionalities with different emphases on computation and data storage.

B. What is Datar

Definition (Datar) *A datar is a set of coherent softewares/systems based on a unified architecture that can manage (big) data pluggablly, automatically and intelligently with specific functionalities, where specific functionalities refer to input, storage, computation, control and output of the (big) data.* Datar is featured with Interactive Interface Clients, Pluggable Engines Configuration, Automatic Dataflow on Job Pipelines and Intelligent Self-driving System Management based on the unified framework. In this paper, we implement datar with these features as **biggy**, a data-storage-centered solution to datar implementation.

A datar, i.e., a full-function BDMS, consists of five parts, data input, data storage, data computation, data control and data output. Compared with the computation-centered computer, a datar is data-centered. We take AsterixDB [10] for example, which is a new, full-function BDMS. Data input is how data gets into the system. In AsterixDB, data feed is a built-in mechanism allowing new data to be continuously ingested into system from external sources, incrementally populating the datasets and their associated indexes [16]. Data storage is how the data is stored in the system and how the indexes are built. In AstrixDB, data and index are stored based on LSM structure [17]. Data computation is how to mine valuable information from stored data. A bunch of methods can be applied, such as popular in-memory computation framework Spark on AsterixDB [18]. Besides, the execution of data processing is also part of data analysis, like Hyracks [19] in AsterixDB. Data control is how to control data when it is processed. It is different from the traditional database systems which have strict ACID properties. Another important aspect of datar is data output, e.g., visualization. Cloudberry[3] is a research prototype to support interactive analytics and visualization of large amounts of spatial-temporal data using AsterixDB. Based on these features of AsterixDB, it is ideal for us to explain the five main components of BDMS by one system. The key drawback of taking AsterixDB as BDMS is that it is a strongly coupled system, which is not suitable for varied and dynamic requirement in real scenarios when processing big data. And it is not easy for developers to combo it with new emerging engines. Datar is proposed to achieve a unified framework for building your own BDMS more flexible.

[3]http://cloudberry.ics.uci.edu/

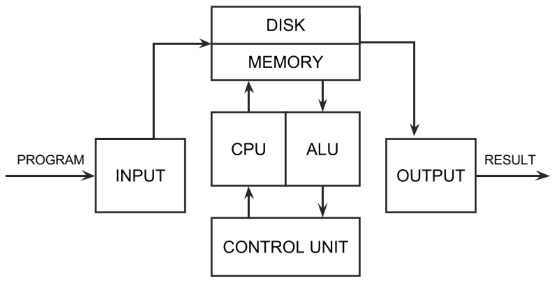
C. How Comes Datar

In the late 1970s, the concept of "database machine" emerged, which is a technology specially used for storing and processing data. With the increase of data volume, the storage and processing capacity of a single computer system became inadequate. In the 1980s, people proposed "share nothing", a parallel database system, to meet the demand of the increasing data. The share-nothing system architecture is based on the use of cluster and every machine has its own processor, storage, and disk. In the late 1990s, the advantages of parallel database was widely recognized in the database field. Later on in 21st century, data system research came into a new era, and broke the traditional concepts from row-based store to column-based store, from disk-based query to in-memory based analysis, and from ACID properties to CAP theorem.

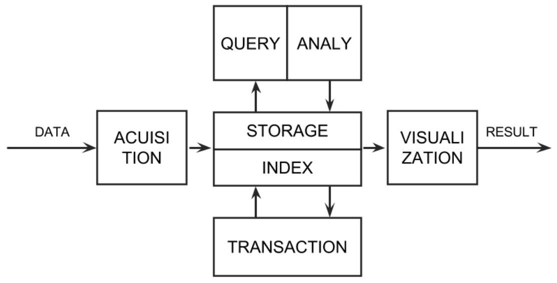
However, many challenges facing big data arise. With the development of Internet services, indexes and queried contents were rapidly growing. Therefore, search engine companies had to face the challenges of handling such big data. Google created storage systems Google File System [20] and programing models MapReduce [21] to cope with the challenges brought about by data management and analysis at the Internet scale. Over the past few years, nearly all major companies, including Oracle, IBM, Microsoft, Google and Facebook, have started their big data projects. Big data shows great value in real application and challenges arise.

Big data is emerging and reaching every corner of industries and every filed of researches. Big data can be interpreted as a term for large or complex datasets that traditional data processing applications are inadequate. Challenges in big data include data analysis, data transaction, data query, data storage, data visualization, data privacy and so on. Various tools and systems are proposed and developed to tackle these challenges on different emphases. In this paper, we describe the BDMS from a new perspective, the view of a computer architecture, to propose a unified framework datar. We focus our attention on the system architecture in BDMS and break it down into five main components to elaborate. The envisioned datar is implemented as biggy with favorable features. The key contributions can be summarized as,

- We review current big data management systems by five core components and state our contributions.
- A unified architecture for big data management, i.e., datar, is proposed and explained to manage big data pluggablly, automatically and intelligently.



(a) Computer Architecture



(b) Datar Architecture

Fig. 2. Computer VS Datar comparison in terms of architecture.

- We propose ConfChain to connect the pluggable engines, a new data structure BigData to manage data flow, and Job Pipeline to execute jobs.
- We implement the envisioned architecture as biggy based on several popular engines to fulfill the functions of input, storage, computation, control and output, and demonstrate biggy in details by running examples.
- We evaluate the performance of our proposed unified framework from different aspects to show the feasibility and potentials.

Rest of the paper is organized as follows. In Section II, we introduce BDMS as datar from five aspects, input, storage, control, computation and output. Section III introduces the framework and key features of datar. The implementation of biggy and demonstrations are given in Section IV. Performance evaluation is carried out in Section V. Finally, we conclude with futures in section VI.

II. Break Down BDMS

A. Data Input

Data can be input into storage since its generation or be “inserted” from other resources.

1) Data Generation: Big data can be generated from various sources, such as unstructured web data, enterprise internal data, government data, and other data from more sources, e.g., scientific applications data and pervasive sensing data. These datasets have their unique data characteristics in scale, time dimension, and data category.

2) Data Feed: Data feed is also known as data acquisition, having continuous data arrive into DBMS from external sources and incrementally populate a persisted dataset and associated indexes. A simple way of having data being put into a big data management system on a continuous basis is to have a single program fetch data from an external data source, parse the data, and then invoke an insert statement per record or batch of records. In AsterixDB, a fault-tolerant data feed facility is built that scales through partitioned parallelism by using a high-level language. Flume[4] and Kafka[5] are two popular tools used as data feeds.

Key-Value	Column	Document	Graph
Dynamo [22]	BigTable [23]	MongoDB [24]	Giraph [6]
Voldemort [25]	Cassandra [26]	SimpleDB[7]	Neo4j [8]
Redis [27]	HBase [28]	CouchDB [29]	OrientDB [9]
MemcacheDB [30]	HyperTable [10]		Pregel [31]
Scalaris [32]	C-store [8]		FlockDB [11]
TiKV [12]			

TABLE I
Popular big data storage system.

B. Data Storage

Data storage focuses on raw data storage and indexes storage in this paper. Besides, schemes storage, configuration files storage and views storage are also data storage but out of our scope.

1) Data Storage: Big data storage refers to the storage and management of large-scale datasets while achieving reliability and availability of data accessing. Various storage systems emerge to meet the demands of massive data. There are four main types of NoSQL databases: key-value DB, column-oriented DB, document-oriented DB and graph-oriented DB. Table I shows examples of popular big data storage systems.

[4]https://flume.apache.org/
[5]https://kafka.apache.org/
[6]http://giraph.apache.org/
[7]https://aws.amazon.com/simpledb/
[8]https://neo4j.com/
[9]http://orientdb.com/
[10]http://www.hypertable.org/
[11]https://github.com/twitter-archive/flockdb
[12]https://www.pingcap.com/

In addition, NewSQL [33] is a class of modern relational database management systems that seek to provide the same scalable performance of NoSQL systems for online transaction processing (OLTP) read-write workloads while still maintaining the ACID guarantees of a traditional database system. Examples are TiDB, VoltDB [9], NuoDB [13] and Spanner [13].

2) Data Index: Index is an effective method to reduce the expense of disk reading and writing, and improve query speeds in both relational databases that manage structured data, and other technologies that manage semistructured and unstructured data. However, index has the additional cost for storing index files which should be maintained dynamically when data is updated. Basic structures include Hash table, Tree-based index, Multidimensional index, and Bitmap index. Big data index [34] has additional requirements, such as parallelism and easily partitioned into pieces for parallel processing, e.g., Latent Semantic Indexing and HMM Indexing.

C. Data Computation

Data computation can be from simple SQL-like query to complex machine learning techniques. There is no clear boundary to distinguish them, but we introduce them by two coarse categories as data query and data analysis.

1) Data Query: MapReduce [21], Dryad[14] [35], All-Pairs, Pregel [31], Spark[15] [36] are the popular programming models and execution engines. More in-memory databases are proposed to accelerate the computation.

2) Data Analysis: The analysis can be from simple statistic to deep data mining technology. Nowadays, deep learnig has become a trend in analysis of big data. MLlib[16] [37] by Java, Scipy, Theano, Caffe[17] [38] by Python, TensorFlow[18] [39] by C++, Torch, etc.

D. Data Control

According to CAP theorem, it is not feasile for BDMS to fulfill ACID properties from traditional databases. However, BASE theorem provides an alternative. CAP theorem says it is impossible for a protocol to guarantee both consistency and availability in a partition-prone distributed system. Most NoSQL database system architectures favor one factor over the other.

1) Data Transaction: In databases, a transaction is a set of separate actions that must all be completely processed, or none processed at all. In partitioned databases, trading some consistency for availability can lead to dramatic improvements in scalability. NewSQL is a kind of next-generation scalable relational database management systems for OLTP that provide scalable performance of NoSQL systems for read-write workloads, as well as maintaining the ACID guarantees of a traditional database system.

[13]http://www.nuodb.com/

[14]https://www.microsoft.com/research/project/dryad/

[15]http://spark.apache.org/

[16]http://spark.apache.org/mllib/

[17]http://caffe.berkeleyvision.org/

[18]https://www.tensorflow.org/

2) Resource Management and Coordination: Big data computation always runs on thousands of machines, which needs the resource management among clusters [40]. Mesos [41] and Hadoop YARN [42] are two popular resource management systems, and Apache ZooKeeper [43] enables highly reliable distributed coordination.

3) Data Recovery: Big data applications as well as operational systems must be supported by a robust and rapid recovery process. The scale-out nature of the architecture can also be difficult for traditional backup applications to handle. As database architecture has fundamentally changed to meet new application requirements, data protection needs to be redefined and re-architected as well.

E. Data Output

Visualization is the best way to present the results of big data management. Besides, in this section, we also mention data sharing as a way of data output.

1) Data Visualization: Visualization helps us take a deep look into the big data, which provides us a interactive and graphic way to embrase the inside of big data. Tools like Tableau[19], Plotly[20], Visual.ly[21], Zeppelin[22] are emerging. Scalability and dynamics are two major challenges in visual analytics [44].

2) Data Sharing: Sharing data [45] can increase the potential benefit to society of the subject's participation by providing greater opportunities for scientific discovery, researchers may have an ethical duty to share their data unless doing so would increase risk to the subjects. For example, some data like personal data and enterprise internal data cannot share since its privacy and confidential. Therefore, some guidelines our regulations should be made to lead us properly share data, rather than share all or share nothing.

III. Build Up Datar

A. Datar Hypothesis

As we have discussed, we propose a unified framework of big data management systems, **datar**, to make one more step. The idea essentially comes from computer. A datar is a set of coherent softewares/systems that can manage big data pluggablly, automatically and intelligently with specific functionalities. Pluggibility means any of the five parts can be replaced by corresponding existing engines easily, automation means the flow from data input to data output can be executed coherently in a pipeline mode and intelligence means datar can mine valuable information in depth as well as self-driving system management.

B. A Unified Framework

Apache Beam [46] is an advanced unified programming model and implements batch and streaming data processing jobs that run on any execution engine. BDAS, the Berkeley

[19]http://www.tableau.com/

[20]https://plot.ly/

[21]http://visual.ly/

[22]https://zeppelin.apache.org/

Data Analytics Stack, is an open source software stack that integrates software components being built by the AMPLab to make sense of big data. The core component Apache Spark is a unified engine for big data processing. Apache AsterixDB [10] is a scalable, open source BDMS. TiDB is a Hybrid Transactional/Analytical Processing (HTAP) database, inspired by the design of Google F1 and Google Spanner. Xiaomi Inc proposes their own open source ecosystems [47] for collecting and processing large-scale data in face of varied business requirements.

The need for high-level abstraction in data centers is necessary [48], [49]. The above systems have their characteristics to process big data as BDMSs, while datar is designed to overcome their drawbacks. Apache Beam aims to process batch and streaming data, and an abstraction of Runners is proposed to adapt different underlying runners. Datar is more specific on the unified framework, and divide the pluggable engines into five types of core components. The engines are chained to easily build your own BDMS to run job pipelines. RDD works as the abstraction data model in Spark and in datar, BigData sits the same position. The difference is that Spark is developed from scratch while datar can make popular engines fit in its unified framework. AsterixDB is a full-function BDMS with strongly coupled components, which is not easy to be compatible with other systems inherently. On the contrary, Xiaomi's platform is too loosely coupled which lacks of unified design. It is hard to reason about the data consistency, scalability and fault-tolerance offered by an assembly 'gluing' together different systems. To be summarized, datar is designed in a high-level abstraction, and provides a unified framework to build your own BDMS with pluggable engines. Job pipelines are executed on the chained engines with data model BigData automatically.

Fig. 3 shows the unified framework of datar, which consists of three main parts, Clients, Framework and Engines. Clients are interactive interfaces for users to interact with the running instance. Framework is the core of datar, which makes the engines pluggable, dataflow automatic and management intelligent. Engines are plugged to the unified framework. Different Wrappers need to be implemented to adapt corresponding engines into biggy Framework. In Framework part, biggy Inception parses the requests from clients and biggy BusKeeper collects the context information of running instance to help intelligent self-driving management.

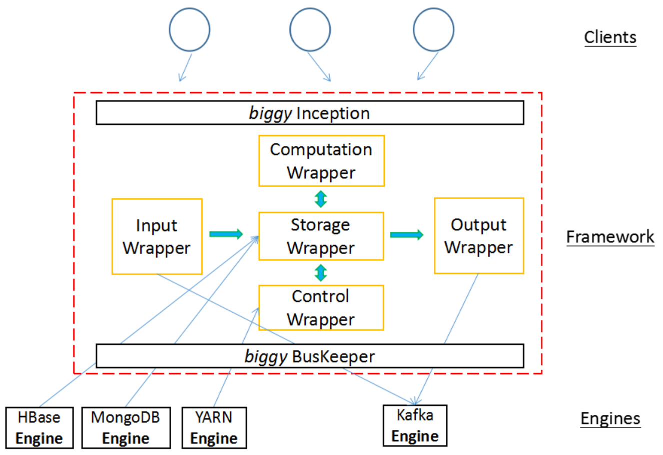


Fig. 3. Framework of datar.

C. Key Features and Techniques

Datar provides a unified framework to build your own BDMS that can manage big data pluggablly, automatically and intelligently. We present the key techniques to achieve these features.

1) Configuration Chain: Users can build their own BDMS based on biggy framework by ConfChain. The configuration chain is connected by pluggable engines of five types. Once users configure the underlying engines by editing configuration file in biggy, the biggy instance generates a particular BDMS based on the unified framework according to users' configurations of engines.

2) Data Flow: We implement a new data structure BigData, which supports fault-tolarence based on linage techniques and suitable for distributed computation. BigData works similar with RDD, which is designed for our biggy data state management. BigData can transform with HDFS data, File data, HBase data by hdfsBD, fileBD, hbaseDB. BigData supports two types of operators, Action and Transformation. An Action operator creates a new BigData and a Transformation operator updates data within previous BigData. Linage techniques keep the changes of BigData during all operations.

3) Job Pipeline: Jobs run on biggy within BigData by Pipeline. There are five raw Pipes including InputPipe, StoragePipe, ControlPipe, ComputationPipe and OutputPipe. A job Pipeline consists of several Pipes, and Pipes are connected in a pipeline mode. Each Pipe includes a set of tasks for a certain class, for example, a StoragePipe may contain WriteToHBaseTask and WriteToFileTask. Tasks are the actual execution units in a job pipeline.

4) Intelligent Management and Interactive Clients: biggy collects context information to achieve self-driving intelligent management. We have not implemented the intelligent features in our released gamma-version, but some related work can be found on this direction [50], [51]. We provide interactive clients for user to access the BDMS to manipulate, including desktop, web and command line. Currently, basic command-line manipulations are supported.

IV. Play with biggy

A. biggy Implementation

To put the envisioned datar into practice, we implement it as **biggy**. The greatest difficulty in implementing biggy is how to make abstractions, high-level abstractions for unified framework and low-level abstractions for flexible functionalities. biggy is based on several different system engines to fulfill the functions of input, storage, computation, control and output. Fig. 4 shows the implementation framework of

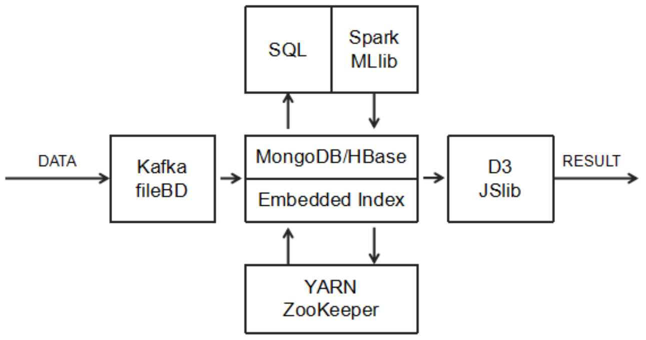


Fig. 4. Implementation of biggy.

biggy. Currently, we implement biggy based on Kafka, MongoDB/HBase, Spark MLlib, YARN/ZooKeeper, and D3. Kafka reads data from external sources into biggy. fildBD is the function we implement to read files on disk to biggy BigData data model. MongoDB and HBase are the two supporting storage engines. Indexes are the internal ones in them. The released gamma-version of biggy is a standalone version, but we still plug YARN as resources manager and ZooKeeper as coordination service to make biggy full-functional. For data computation, MongoDB/HBase SQL is used for basic data query by drivers, and Spark MLlib for complex data analysis. Data output takes D3 JSlib as the pluggable visualization tool to show the results on web pages.

By different levels of abstractions and implementation, we make biggy more pluggable and automatic rather than just gluing them together. Further work of supporting most popular systems (e.g., TensorFlow) as plugins needs done with fulfillment of intelligence. Current biggy is a standalone implementation of datar by different popular engines that can run job pipelines with BigData model.

B. Code Organization

We implement biggy by Java using Maven as the project management tool. Several design patterns are used to make the code work such as Factory Pattern, Singleton Pattern and Chain of Responsibility Pattern. The overview of code organization is shown in Fig. 5. A high-resolution picture can be found here[23]. As we can see, biggy instance bigo sits at the center, the pluggability on upper side is based on factory pattern and chained together, the automation is based on job pipelines and BigData model at lower side, the clients are left-side and intelligent part is right-side. Currently, the clients and intelligent parts are unimplemented, and biggy is set on standalone mode.

Code in black color is the core of biggy, code in red is the main framework of biggy, code in dark orange depends on the pluggable engines, and code in green is application-specific. To see implementation details please go check out the source code with gamma-version.

[23]https://www.processon.com/view/link/5b0e4eeee4b06350d445fcb3

C. Opensource Development

There are three types of users for biggy, i.e., Engine Writers, Framework Writers and App (-lication) Writers. The project is opensourced and can be found at Github[24]. We focus on the implementation of Framework to provide a unified framework for customizing your own BDMS. Engine Writers are responsible to develop new pluggable engines (e.g., HBase) and adapt them into biggy framework by Wrappers. Some Wrapper templates are provided for Engine Writers to use. App Writers are the end users of biggy, who can run their application-specific jobs on a customized BDMS.

D. biggy Deployment

For better explanation, relations among several key terms are shown in Fig. 6. bigo is an instance of biggy and bigdb is the physical files where store the data. Fig. 7 shows the configuration of biggy. Before installation and use of biggy, it is necessary to set the five parts in configuration file. biggy checks if the targeted engines are usable. For each module, there should be an Wrapper to make it fit in biggy, which is implemented by programmers and can be shared for public use. That is why we make it opensourced and any modules/systems/libraries follow the design of biggy can be plugged in to work. Fig. 8 shows how to chain the pluggable engines, in which bigo is the running instance of biggy. Servers of the pluggable engines should get started, such as MongoDB server and Kafka Server. After the deployment of customized BDMS, App Writers can run job pipelines on it with easy coding.

E. biggy Demonstrations

We present four running examples to show how biggy works. Fig. 9 shows the demonstration of biggy, including running examples WordCount, Sort, KMeans and PageRank. Take WordCount for example, given a set of words, the goal is to count the number of each word. We input data from test file *egDBcount.txt* by implementing *egBDIOPipeJobReadFile.java*, store the data into MongoDB by *egHBasePipeJobWriteDB.java*, compute the results using Spark by *egSparkPipeJobWordCount.java*, and output the visualization by *egD3PipeJobVisual.java*. Control part is the default setting on a standalone instance. Two more classes of *egSparkPipeJobWordCountTask001.java* and *egD3PipeJobVisualTask001.java* should be concreted to realize specific data operations. Give the customized BDMS, WordCount pipeline can be executed automatically on BigData model by connecting the pipes in *egWordCount.java*. The other three examples follow the similar implementation. Given the unified framework and simple APIs to use, App Writers can easily deploy their own BDMSs, run job pipelines and care little about the underlying complexity. We give a short summary for each example.

1) Example WordCount: Given a set of words, the goal is to count the number of each word. Fig. 9 (a) shows the results. As we can see, Spark counts four and YANR counts one.

[24]https://github.com/Ideamaxwu/biggy

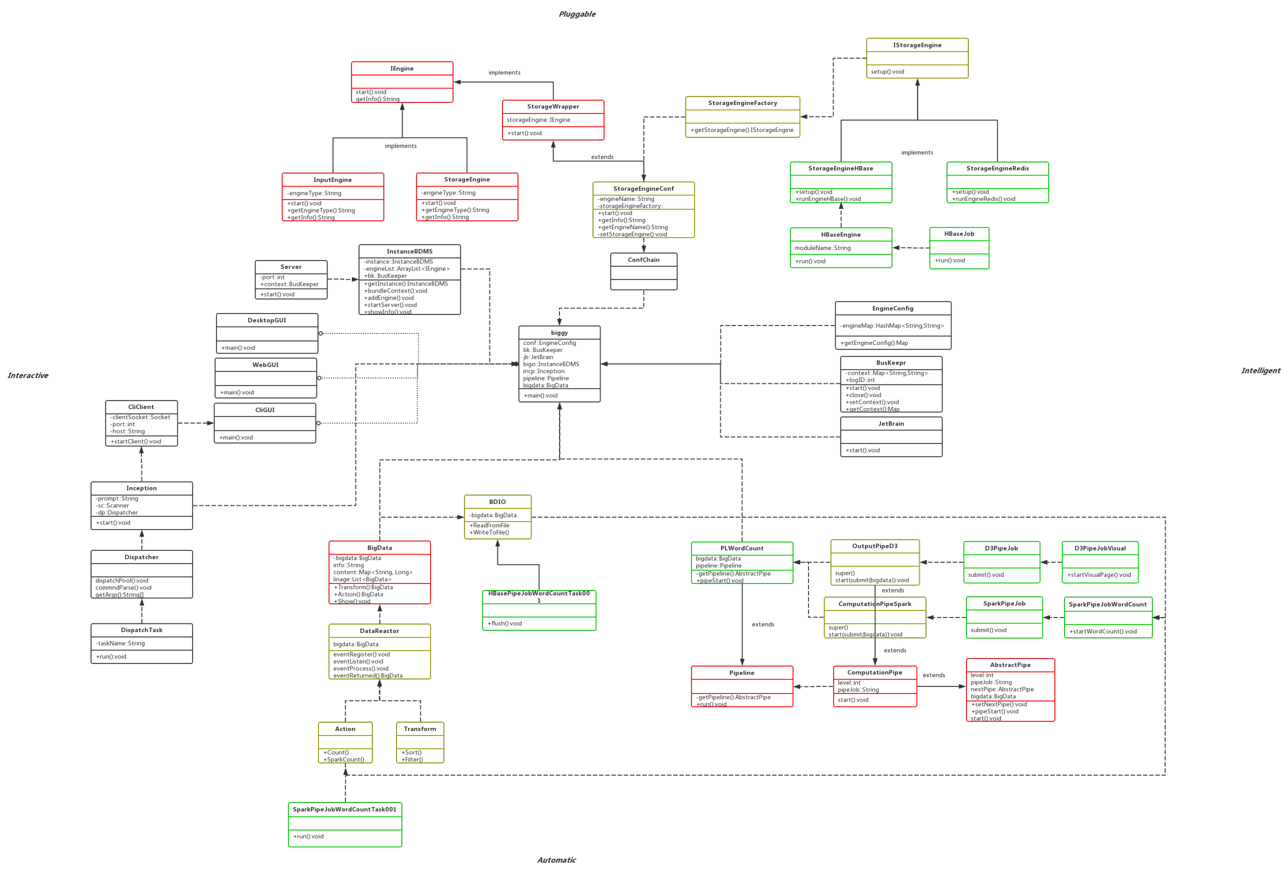


Fig. 5. Code organization og biggy.

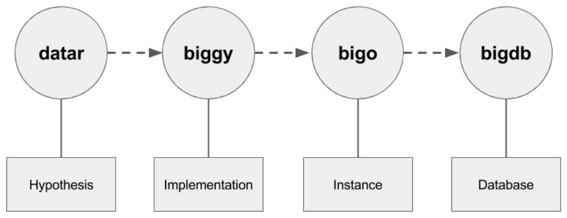
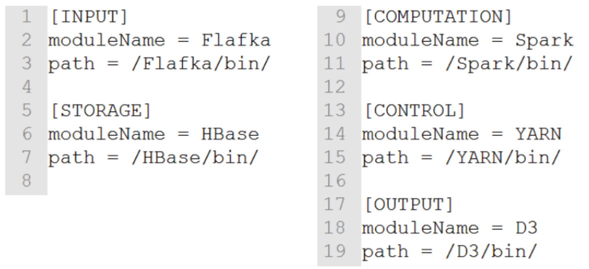


Fig. 6. Relation among several key terms.

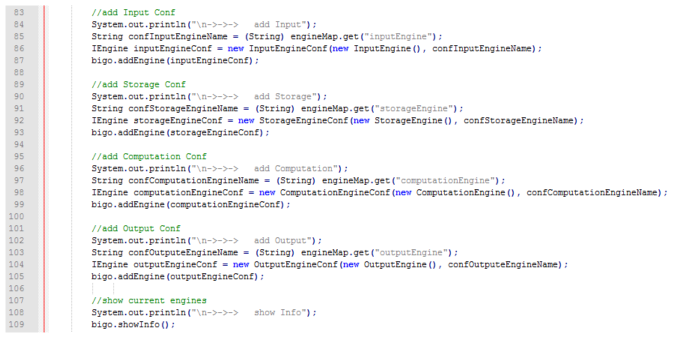
2) Example Sort: Given a random set of strings, the goal is the sort them by alphabet order. In Fig. 9 (b), horizontal label shows the order from A to Z.

3) Example KMeans: KMeans clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean. In our example, each observation is a coordinate with (x, y). Fig. 9 (c) shows the clustering results.



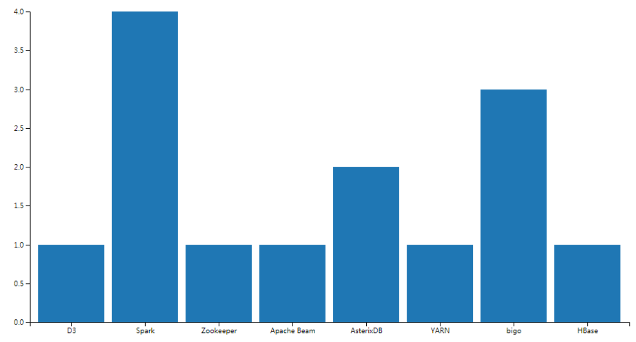
```
[INPUT]
moduleName = Flafka
path = /Flafka/bin/

[STORAGE]
moduleName = HBase
path = /HBase/bin/

[COMPUTATION]
moduleName = Spark
path = /Spark/bin/

[CONTROL]
moduleName = YARN
path = /YARN/bin/

[OUTPUT]
moduleName = D3
path = /D3/bin/
```

Fig. 7. Configuration of biggy.

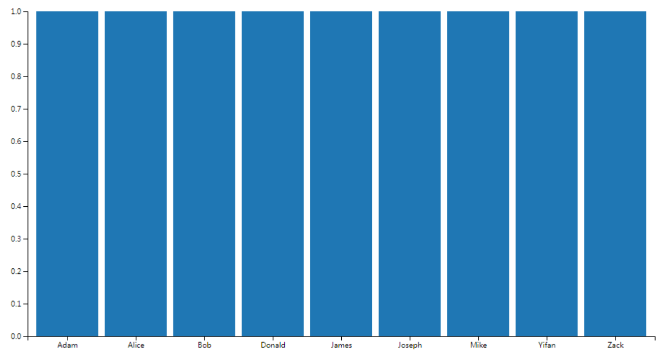


```
//add Input Conf
System.out.println("\n->->->   add Input");
String confInputEngineName = (String) engineMap.get("inputEngine");
IEngine inputEngineConf = new InputEngineConf(new InputEngine(), confInputEngineName);
bigo.addEngine(inputEngineConf);

//add Storage Conf
System.out.println("\n->->->   add Storage");
String confStorageEngineName = (String) engineMap.get("storageEngine");
IEngine storageEngineConf = new StorageEngineConf(new StorageEngine(), confStorageEngineName);
bigo.addEngine(storageEngineConf);

//add Computation Conf
System.out.println("\n->->->   add Computation");
String confComputationEngineName = (String) engineMap.get("computationEngine");
IEngine computationEngineConf = new ComputationEngineConf(new ComputationEngine(), confComputationEngineName);
bigo.addEngine(computationEngineConf);

//add Output Conf
System.out.println("\n->->->   add Output");
String confOutputeEngineName = (String) engineMap.get("outputEngine");
IEngine outputEngineConf = new OutputEngineConf(new OutputEngine(), confOutputeEngineName);
bigo.addEngine(outputEngineConf);

//show current engines
System.out.println("\n->->->   show Info");
bigo.showInfo();
```

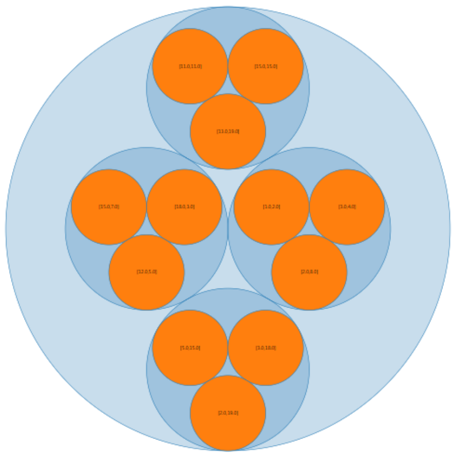
Fig. 8. Example of biggy ConfChain.



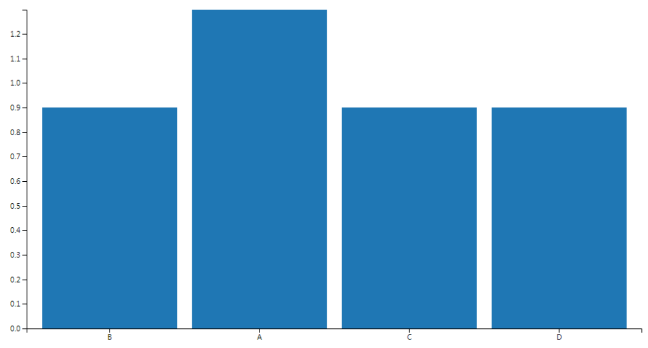
(a) WordCount example



(b) Sort example



(c) KMeans example



(d) PageRank example

Fig. 9. Examples of biggy demonstrations.

4) Example PageRank: PageRank is a way of measuring the importance of website pages. In our test data, each item follows format ($A - > B$), which means A is linked to B. The importance score of each page is showed in Fig. 9 (d).

V. BIGGY EVALUATION

We evaluate biggy on examples WordCount, Sort, KMeans, PageRank by efficiency and scalability in five parts. They are four typical classes of algorithms [52]. To be noted, all the four algorithms are executed on Spark runtime engine.

A. Experiment Setup

The set of experimental evaluation is designed for standalone biggy on a small-scale and personal-computer testbed. The performance mainly depends on the underlying pluggable engines and more experiments in real applications are welcome. From our experiments, we can get a general qualitative and quantitative understanding of biggy performance.

1) Hardware Configuration: The implementation of unified framework biggy is deployed on a laptop with a standalone mode. The laptop has a memory of 8GB, 64-bit Windows 7 system, Intel(R) Core(TM) i7 CPU at 2.5 GHz. We run each experiment three times and report the average results.

2) Software Configuration: For the set of experiments, Java version is 1.8.0, MongoDB is 3.2.3, Spark is 2.3.0, D3 is version 4 and biggy version is gamma.

3) Data Preparation: We use different datasets for four examples, and the dataset samples can be found in the source code. We generate random data from 1K to 1M to evaluate biggy on our laptop. For time and space efficiency, we use default data size 1M, and varied data size is applied for scalability evaluation. In KMeans, number of clusters is 100 and number of iterations is 20. In PageRank, the default number of pages is 1,000. To be noted, 1M means the number of items is 1M, not the disk storage, and the total disk storage is depends on size of each item.

B. Time Efficiency

Time efficiency is shown in Fig. 10. Because Storage and Computation takes too much time in real applications, for better display effects, we count these two parts in seconds and other parts in milliseconds. As we can see, storage is the most time-consuming part, and the reason is we store the data into MongoDB by inserting every tuple, which costs a lot. Computation comes later and Input/Output do not take much time. Control part takes little time because we run experiments in standalone mode. The negligible time of Framework shows the efficiency of our proposed unified framework. KMeans ranks top due to iterations during clustering.

C. Space Efficiency

Space efficiency is shown in Fig. 11. We evaluate used memory of each algorithm, which is calculated by the difference of total amount of memory and amount of free memory in the Java virtual machine. And the used memory is measured in MBs. Fig. 11 shows, the usage of memory in Java VM at

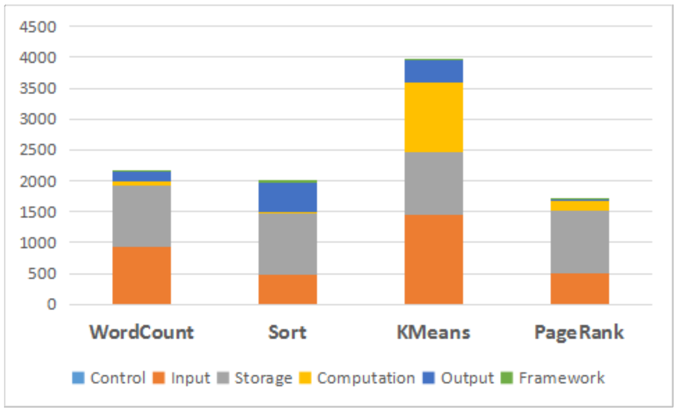


Fig. 10. Time efficiency on different types of algorithms.

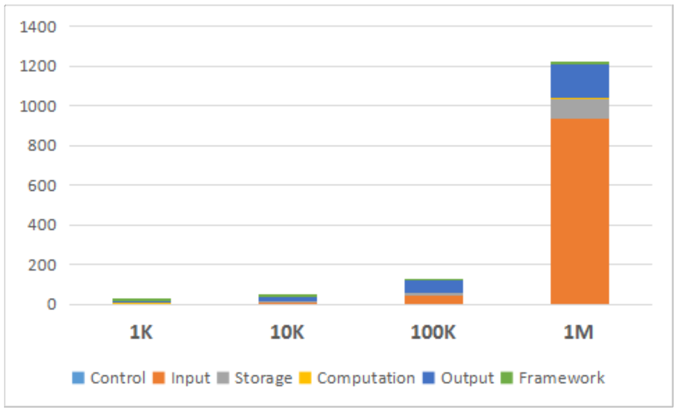


Fig. 12. Scalability on different size of datasets for time efficiency.

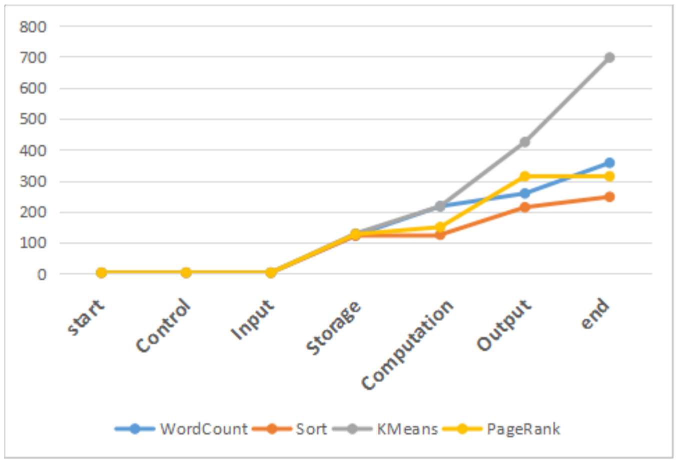


Fig. 11. Space efficiency on different types of algorithms.

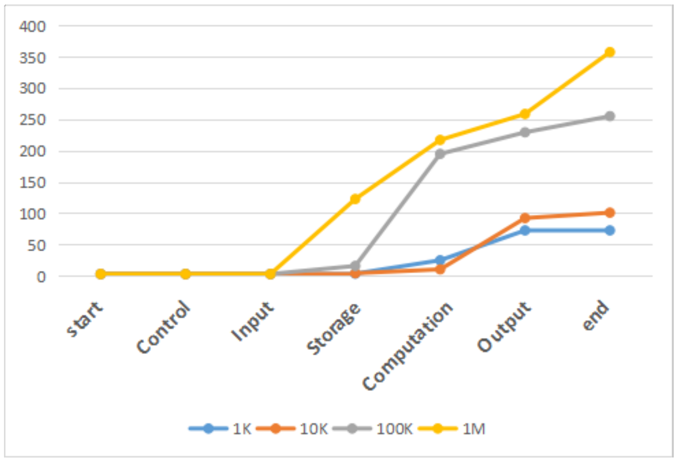


Fig. 13. Scalability on different size of datasets for space efficiency.

the start of each Pipe stage. As we can see, used memory increases as the pipeline goes on and KMeans costs most.

D. Scalability

Scalability evaluation is shown in Fig. 12 and Fig. 13 and is based on WordCount example. Table II shows the details of the results on scalability for time efficiency. The data size varies from 1K to 1M, both time and space increases with the size of data, which is reasonable. Same as previous experiments, Storage and Computation time is measured in seconds, and the rest in milliseconds. We do not optimize the garbage collection of Java in this released version. If so, better results may get. For limitation of pages, detailed results are on Github.

Data Size	1K	10K	100K	1M
Control	0	1	1	1
Input	3	9	42	933
Storage	1.809 (s)	3.337 (s)	12.797 (s)	98.91 (s)
Computation	2.859 (s)	3.077 (s)	3.367 (s)	5.766 (s)
Output	8	22	59	173
Framework	5	6	6	4

TABLE II
DETAILS OF SCALABILITY FOR TIME EFFICIENCY.

VI. CONCLUSION AND FUTURE WORK

We propose datar as a unified framework for BDMS from the perspective of a computer in five components and present biggy as an implementation of datar to manage big data pluggablly, automatically and intelligently with specific functionalities. biggy presents only the first step towards a unified framework of big data management system, and we hope it helps frame important research questions. In particular, some problems we are exploring next include:

- Implementation of intelligent self-driving management .
- From standalone to cluster mode.
- System optimization to improve efficiency.
- More popular engines support and multi-OS support.

Many technical challenges must be addressed before this potential unified framework can be realized fully. We support and encourage fundamental research towards addressing the architecture design and technical challenges to achieve the promised benefits of big data. More resources about this unified framework for B are accessible on Github and so is the contact information.

References

[1] A. M. Turing, "Computing machinery and intelligence," *Mind*, vol. 59, no. 236, pp. 433–460, 1950.

[2] J. Von Neumann, "First draft of a report on the edvac," *IEEE Annals of the History of Computing*, no. 4, pp. 27–75, 1993.

[3] C. W. Bachman, "On a generalized language for file organization and manipulation," *Commun. ACM*, vol. 9, no. 3, pp. 225–226, 1966.

[4] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM*, vol. 13, no. 6, pp. 377–387, 1970.

[5] M. Stonebraker, E. Wong, P. Kreps, and G. Held, "The design and implementation of INGRES," *ACM Trans. Database Syst.*, vol. 1, no. 3, pp. 189–222, 1976.

[6] M. Stonebraker, "The postgres DBMS," in *Proceedings of the ACM SIGMOD*, 1990, p. 394.

[7] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu, "Mariposa: A wide-area distributed database system," *VLDB J.*, vol. 5, no. 1, pp. 48–63, 1996.

[8] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, and et al, "C-store: A column-oriented DBMS," in *Proceedings of VLDB*, 2005, pp. 553–564.

[9] M. Stonebraker and A. Weisberg, "The voltdb main memory DBMS," *IEEE Data Eng. Bull.*, vol. 36, no. 2, pp. 21–27, 2013.

[10] S. Alsubaiee, Y. Altowim, H. Altwaijry, A. Behm *et al.*, "Asterixdb: A scalable, open source bdms," *Proceedings of the VLDB Endowment*, vol. 7, no. 14, pp. 1905–1916, 2014.

[11] R. Taft, N. El-Sayed, M. Serafini, Y. Lu, A. Aboulnaga, M. Stonebraker, R. Mayerhofer, and F. Andrade, "P-store: An elastic database system with predictive provisioning," in *Proceedings of the 2018 SIGMOD*, 2018, pp. 205–219.

[12] J. Baker, "Megastore : Providing scalable, highly available storage for interactive services," in *Biennial Conference on Innovative Data Systems Research*, 2011, pp. 223–234.

[13] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, and et al, "Spanner: Google's globally-distributed database," *Acm Transactions on Computer Systems*, vol. 31, no. 3, p. 8, 2013.

[14] T. Akidau, A. Balikov, K. Bekirolu, S. Chernyak *et al.*, "Millwheel: fault-tolerant stream processing at internet scale," *Proceedings of the VLDB Endowment*, vol. 6, no. 11, pp. 1033–1044, 2013.

[15] T. Hey, "The fourth paradigm c data-intensive scientific discovery," in *International Symposium on Information Management in a Changing World*, 2012, pp. 1–1.

[16] R. Grover and M. J. Carey, "Data ingestion in asterixdb." in *EDBT*, 2015, pp. 605–616.

[17] S. Alsubaiee, A. Behm, V. Borkar, Z. Heilbron, Y.-S. Kim, M. J. Carey, M. Dreseler, and C. Li, "Storage management in asterixdb," *Proceedings of the VLDB Endowment*, vol. 7, no. 10, pp. 841–852, 2014.

[18] Y. Bu, Y. Bu, Y. Bu, Y. Bu, and Y. Bu, "Large-scale complex analytics on semi-structured datasets using asterixdb and spark," *Proceedings of the VLDB Endowment*, vol. 9, no. 13, pp. 1585–1588, 2016.

[19] V. Borkar, M. Carey, R. Grover, N. Onose, and R. Vernica, "Hyracks: A flexible and extensible foundation for data-intensive computing," in *2011 IEEE 27th ICDE*. IEEE, 2011, pp. 1151–1162.

[20] S. Ghemawat, H. Gobioff, and S. T. Leung, "The google file system," in *Nineteenth Acm Symposium on Operating Systems Principles*, 2003, pp. 29–43.

[21] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[22] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, and et al, "Dynamo: amazon's highly available key-value store," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 205–220, 2007.

[23] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, and et al, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems*, vol. 26, no. 2, p. 4, 2008.

[24] K. Banker, *MongoDB in action*. Manning Publications Co., 2011.

[25] R. Sumbaly, J. Kreps, L. Gao, A. Feinberg, C. Soman, and S. Shah, "Serving large-scale batch computed data with project voldemort," in *Proceedings of the 10th USENIX conference on File and Storage Technologies*. USENIX Association, 2012, pp. 18–18.

[26] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[27] J. L. Carlson, *Redis in Action*. Manning Publications Co., 2013.

[28] L. George, *HBase: the definitive guide*. " O'Reilly Media, Inc.", 2011.

[29] J. C. Anderson, J. Lehnardt, and N. Slater, *CouchDB: the definitive guide*. " O'Reilly Media, Inc.", 2010.

[30] B. Fitzpatrick, "Distributed caching with memcached," *Linux journal*, vol. 2004, no. 124, p. 5, 2004.

[31] G. Malewicz, M. H. Austern, A. J. Bik, and et al, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD*. ACM, 2010, pp. 135–146.

[32] T. Schütt, F. Schintke, and A. Reinefeld, "Scalaris: reliable transactional p2p key/value store," in *Proceedings of the 7th ACM SIGPLAN workshop on ERLANG*. ACM, 2008, pp. 41–48.

[33] M. Stonebraker, "Newsql: An alternative to nosql and old sql for new oltp apps," *Communications of the ACM. Retrieved*, pp. 07–06, 2012.

[34] F. B. Adamu, A. Habbal, S. Hassan, R. Les Cottrell, B. White, and I. Abdullahi, "A survey on big data indexing strategies."

[35] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3. ACM, 2007, pp. 59–72.

[36] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets." *HotCloud*, vol. 10, pp. 10–10, 2010.

[37] X. Meng, J. Bradley, B. Yuvaz, E. Sparks, S. Venkataraman *et al.*, "Mllib: Machine learning in apache spark," *JMLR*, vol. 17, no. 34, pp. 1–7, 2016.

[38] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, and et al, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.

[39] M. Abadi, A. Agarwal, P. Barham, E. Brevdo *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[40] M. Zaharia, B. Hindman, A. Konwinski, A. Ghodsi *et al.*, "The data-center needs an operating system," in *Usenix Conference on Hot Topics in Cloud Computing*, 2011, pp. 17–17.

[41] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, and et al, "Mesos: A platform for fine-grained resource sharing in the data center." in *NSDI*, vol. 11, no. 2011, 2011, pp. 22–22.

[42] V. K. Vavilapalli, A. C. Murthy, C. Douglas *et al.*, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 5.

[43] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems." in *USENIX annual technical conference*, vol. 8, 2010, p. 9.

[44] L. Wang, G. Wang, and C. A. Alexander, "Big data and visualization: methods, challenges and technology progress," *Digital Technologies*, vol. 1, no. 1, pp. 33–38, 2015.

[45] R. A. Poldrack and K. J. Gorgolewski, "Making big data open: data sharing in neuroimaging," *Nature neuroscience*, vol. 17, no. 11, pp. 1510–1517, 2014.

[46] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, and S. Whittle, "The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," *Proceedings of the Vldb Endowment*, vol. 8, no. 12, pp. 1792–1803, 2015.

[47] L. Jun, Y. Hangjun, W. Zesheng, Z. Peng, X. Long, and H. Yanxiang, "Big-data platform based on open source ecosystem," *Journal of Computer Research and Development*, vol. 54, no. 1, pp. 80–93, 2017.

[48] M. Schwarzkopf, "Operating system support for warehouse-scale computing," 2015.

[49] I. Gog, "Flexible and efficient computation in large data centres," 2018.

[50] J. M. Hellerstein, V. Sreekanti, J. E. Gonzalez, J. Dalton, A. Dey, S. Nag, and et al, "Ground: A data context service," in *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research*, 2017.

[51] A. Pavlo, G. Angulo, J. Arulraj, H. Lin, J. Lin, L. Ma, and et al, "Self-driving database management systems," in *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research*, 2017.

[52] J. Shi, Y. Qiu, U. F. Minhas, L. Jiao, C. Wang, and B. Reinwald, "Clash of the titans: Mapreduce vs. spark for large scale data analytics," *Proceedings of the VLDB Endowment*, vol. 8, no. 13, pp. 2110–2121, 2015.