Big Data and Computing Visions

www.bidacv.com

Big. Data. Comp. Vis. Vol. 1, No. 4 (2021) 192–199.

Paper Type: Original Article

Review of Prime Issues in Big Data Storage

Zhi Zhou[1], Javad Pourqasem[2,*], Shadi Sayadmanesh[3]

[1]Government Information Headquarters Inspur Software Group Company Ltd, Jinan, China.
[2]Department of Computer Engineering, University of Guilan, Rasht, Iran; jpourmail@gmail.com.
[3]Department of Industrial Management, Allameh Tabatabai University, Tehran, Iran.

Citation:

Zhou, Z., Pourqasem, J., Sayadmanesh, S. (2021). Review of prime issues in big data storage. *Big data and computing visions, 1* (4), 192-199.

Received: 18/06/2021 Reviewed: 22/07/2021 Revised: 21/08/2021 Accept: 02/09/2021

Abstract

Developing the scale and increasing the data set, make the reliability and availability principal affairs in access process and data achievement. In addition, we face the challenges of handling big data in terms of storage and managent. This paper provides the important issues related to the massive storage systems, distributed storage systems, and big data storage mechanisms. Then, we present some analysis models utilizing in big data and describe structure of them in details.

Keywords: Big data, Storage system, Analysis model.

1 | Introduction

In the late 1970s, the concept of "database machine" emerged, a technology particularly used for storing and analyzing data. By increasing the volume of data, the storage and processing capacity of a single mainframe computer system became inadequate. In the 1980s, people suggested "sharing nothing," a parallel database system, to meet demand for increased data volume [1]. The share of nothing is system architecture based on cluster usage and each machine has its own processor, storage, and disk. The Teradata system was the first successful parallel trading database system. Such a database recently became very popular. The latest advances in information technology (IT) make data production easier. So we face the main challenge of collecting and integrating massive data from widely distributed data sources.

Traditional data management and analysis systems are based on the Relational Database Management System (RDBMS). However, such RDBMSs only apply to structured data, other than semi-structured or unstructured data. In addition, RDBMSs are increasingly using more and more expensive hardware. Apparently, traditional RDBMSs could not handle the huge volume and incognity of big data. The research community has proposed solutions from different perspectives. For permanent

Corresponding Author: jpourmail@gmail.com
10.22105/bdcv.2022.325253.1040

storage solutions and large-scale disordered dataset management, distributed file systems [2] and NoSQL databases [3] are good choices.

Such programming frameworks have achieved great success in processing cluster tasks, especially for web page rankings. Various big data programs can be developed based on these innovative technologies or platforms. The acquisition of big data involves data collection, data transfer, and data processing. When we collect raw data, we must use an efficient transmission mechanism to send it to a suitable storage management system to support various analytical applications.

Yet there were many challenges with big data. As internet services developed, queryed indicators and contents were growing rapidly. So search engine companies had to face the challenges of handling such big data. Google created the GFS [4] and MapReduce [5] programming models to tackle the challenges brought by data management and internet-scale analytics. In addition, content produced by users, sensors, and other ubiquitous data sources also displayed onerous data streams that required a fundamental change to the computing architecture and large-scale data processing mechanism.

The rest of this paper has been organized as follows: Section 2 involves the Big Data storage and we review the principal issues including the massive data storage, distributed storage system, and the storage mechanism in details. In Section 3 we consider the analysis model implemented in Big Data and describe some analysis models structure. Section 4 concludes the paper.

2| Big Data Storage

As the data are growth, some requirements in storage and management are needed. Large data storage relates to the storage and management of large-scale data sets to achieve the reliability and availability of data access. Important issues include massive storage systems, distributed storage systems, and big data storage mechanisms. The storage infrastructure needs to provide data storage services with reliable storage space, additionally, it should provide a powerful access interface for querying and analyzing large amounts of data.

2.1| Massive Data Storage

Massive storage technologies can be classified as Direct Connected Storage (DAS) and network storage. It is noteworthy that network storage can be tabulated to Network Attached Storage (NAS) and Storage Area Network (SAN) [6] and [7]. In DAS, different hard drives are directly associated with servers, and data management is server-centric, so that storage devices are accessories, each taking a certain amount of I/O source and managed by an individual application software. DAS is only suitable for connecting servers on a small scale. However, due to its low scalability, DAS will show unfavorable productivity when storage capacity is increased. In this way DAS is mainly used on PCs and servers of small size.

2.2| Distributed Storage System

Distribution storage system is the first challenge brought about by big data is how to develop a large-scale distribution storage system for efficient data processing and analysis [8] and [9]. To use a distributed system to store massive data, the following factors should be taken into consideration:

Consistency. A distributed storage system requires multiple servers to co-store data cooperatively. Since there are more servers, the chances of server failures will be larger. Usually the data is divided into several pieces to be stored on different servers to ensure availability in case of server failure. However, server failure and parallel storage may cause dissonance among different copies of similar data. Consistency refers to ensuring that multiple copies of the same data are identical.

Availability. A distributed storage system operates on multiple sets of servers. As more servers are used, server failures are inevitable. If the whole system was not seriously affected to satisfy customer requests in terms of reading and writing, it would be desirable. This property is called availability.

Partition tolerance. Multiple servers are connected to a storage system distributed by a network. The network could have a link/node failure or temporary congestion. The distributed system must have a certain level of tolerance to problems caused by network failures. It would be desirable that distributed storage still works well when the network is partitioned. In 2010, Eric Breuer proposed a CAP [80, 81] theory that showed that a distributed system could not simultaneously meet the requirements of consistency, availability, and partition tolerance; MIT's Seth Gilbert and Nancy Lynch proved the CAP theory correct in 2002.

2.3| The Storage Mechanism

The storage mechanism for big data is significant research on big data promoting the development of storage mechanisms for big data. Existing storage mechanisms of big data may be classified into three levels: (i) file systems, (ii) databases, and (iii) programming models.

2.3.1| File systems

File systems are the foundation of high-level applications. Google's GFS is an expandable distributed file system to support large-scale, distributed, data-intensive applications [10]. GFS uses inexpensive commodity servers to achieve fault-tolerance and provides high performance services to customers. GFS supports large-scale file applications with more frequent reading of writing. However GFS also has limitations, such as a break point and poor runs for small files. Such restrictions have been overcome by Klossus, the successor to the GFS. In addition, other companies and researchers also have their own solutions to meet various demands for big data storage. For example HDFS and Kosmosfs are derivatives of open source GFS codes. Microsoft developed Cosmos [11] to support its search and advertising business. Facebook uses Haystack [12] to store plenty of small-sized photos. Taobao also developed TFS and FastDFS.

2.3.2| Database technology

Database technology has been evolving for over 30 years. Different database systems have been developed to handle data sets at different scales and support different applications [13]. Traditional relational databases cannot meet the challenges of categories and scales brought on by big data. NoSQL databases (as one, nontraditional relational databases) are becoming more popular for big data storage. NoSQL database features flexible modes, simple and easy copying support, simple API, ultimate consistency, and large volume data support. NoSQL databases are becoming nuclear technology for big data. We will review the following three main NoSQL databases in this section: key-value databases, column-oriented databases, and document-driven databases, each based on specific data models.

Key Value Databases. Key value databases are formed by a simple data model and the data is stored related to key-values. Each key is unique and customers may have queried values according to the input key. Such databases feature a simple structure and modern databases are value key with high expandability and shorter query response times than specified relational databases. Over the past few years, many key value databases have emerged as motivating Amazon's dynamo system [14]. We will introduce Dynamo and several other key-value representative databases.

Dynamo is a very accessible and expandable key distribution value data storage system. It is used to store and manage the status of certain nuclear services that can be realized with key access on Amazon's e-commerce platform. The general mode of relational databases may generate invalid data and limit data scale and availability, while Dynamo can solve these problems with a simple key-object interface that is

formed by simple reading and writing operations. Dynamo achieves traction and availability through data partitions, data copiers, and object version mechanisms. Dynamo's partition scheme relies on compatible hashing [15], which has a main advantage that passing nodes only directly affects adjacent nodes and does not affect other nodes, to split the load for multiple main storage machines. Dynamo copies data to N sets servers, where N is an adjustable parameter in order to achieve high availability and durability. The dynamo system also provides the ultimate consistency, so as to perform asynchronous updates on all copies.

Voldemort is also a key storage system, originally developed for it and still used by LinkedIn [16]. Keywords and values in Voldemort are composite objects composed by tables and images. Voldemort's interface consists of three simple operations: reading, writing, and deletion, all of which are verified by keywords. Voldemort provides simultaneous control of asynchronous updates of multiple versions but does not guarantee the consistency of the data. However, Voldemort supports an optimistic lock for consistent multi-record updates. When conflicts occur between updates and any other operations, the update operation will leave. Voldemort's data copying mechanism is the dynamo mechanism. Voldemort not only stores data in RAM but also allows data to be placed in a storage engine. Voldemort in particular supports two storage engines, including Berkeley D.B. and random access files.

Column-oriented database. Column-oriented databases store and process data according to columns other than rows. Both columns and rows are segmented in multiple nodes to realize the expandability. The columned databases are mainly inspired by Google's BigTable. In this section, we first discuss BigTable and then introduce several derivative tools.

BigTable is a distributed and organized data storage system, designed to process large-scale data (PB class) among thousands of trading servers [17]. Bigtable's basic data structure is a multidimensional sequence mapping with pale, distributed, and continuous storage. Mapping indicators are row keys, column keys, and time stamps, and each value is unfamiliar in the mapping of a byte array. Each row key in BigTable is a 64KB character string. In lexical order, rows are stored and continuously segmented into tablets (as one of the distribution units) to balance the load. Therefore, reading a short row of data can be very effective, as it only involves communicating with a small part of the machines. Columns are grouped by key prelimations, thus forming column families. These pillar families are basic units for access control. Time stamps are 64-bit to detect different versions of cellular values. Customers may be flexible to determine the number of cell copies stored. These versions are sequenced in descending order of time stamps, so the latest version will always be read.

The BigTable API features the creation and reduction of tablets and column families as well as modifying metadata clusters, tables, and column families. Client applications may insert or delete BigTable values, query values from columns, or browse the data subset in a table. Bigtable also supports some other features such as transaction processing in a single row. Users may use such features to perform more complex data processing. Each procedure implemented by BigTable consists of three main components: the main server, the tablet server, and the client library. Bigtable only allows a set of master servers to be distributed responsible for distributing tablets for tablet servers, detecting added servers or deleting tablets, and doing load balances. In addition, it can also change BigTable's outline, for example, creating tables and column families, and collecting waste stored in GFS as well as deleted or disabled files, and using them in specific BigTable examples. Each tablet server manages a tablet suite and is responsible for reading and writing a loaded tablet. When the pills are too large, they will be segmented by the server. The app's client library is used to communicate with BigTable examples.

BigTable is based on many of Google's basic components, including GFS [4], cluster management system, SSTable file format, and chubby [18]. GFS is used to store data and log files. The cluster management system is responsible for task scheduling, resource sharing, processing machine failures, and monitoring machine situations. The SSTable file format is used to store BigTable data internally, it is a 64KB character string. In lexical order, rows are stored and continuously segmented into tablets (as one of the distribution units) to balance the load. Therefore, reading a short row of data can be very effective, as it only involves

communicating with a small part of the machines. Columns are grouped by key prelimations, thus form column families. These pillar families are basic units for access control. Time stamps are 64-bit to detect different versions of cellular values. Customers may be flexible to determine the number of cell copies stored. These versions are sequenced in descending order of time stamps, so the latest version will always be read.

HyperTable similar to BigTable was developed to obtain a set of high-performance, expandable, distributed storage and processing systems for structured and unstructured data [8]. HyperTable relies on distributed file systems, such as HDFS and Distributed Lock Manager. Data representation, processing, and partition mechanisms are similar to those in BigTable. HyperTable has its own query language, called HyperTable Query Language (HQL), allowing users to create, modify and query contextual tables. Because column-oriented storage databases mainly mimic BigTable, their designs are all the same, except for the co-ordinate mechanism and several other features.

Document database. Compared to key value storage, Document database can support more complex forms of data. Since documents do not follow strict modes, there is no need to do mode migration. In addition, key-value pairs can still be saved. We will review three important representatives of document storage systems, as one, MongoDB, SimpleDB, and CouchDB.

MongoDB is an open source and document-oriented database [19]. MongoDB stores documents as Binary Pound Objects (BSON) [20], which is similar to the object. Each document has an ID field as the primary key. The query is expressed in MongoDB with the same syntax. A database driver sends the query to MongoDB as a BSON object. The system allows querying on all documents, including embedded objects and arrays. To enable quick queries, indicators can be created in queryable fields of documents. Copy operations in MongoDB can be run with log files in the main nodes that support all high-level operations performed in the database. When copying, slave owners query all writing operations from the last sync to the master and execute the operation in log files in local databases.

SimpleDB is a distributed database and a web service from Amazon [21]. Data in SimpleDB is organized into different domains where data may be stored, acquired and queryed. Domains include different properties and sets of name/value pairs of projects. The date is copied to different machines in different data centers in order to ensure data safety and improve performance. This system does not support automatic partitioning and thus cannot be expanded by changing the volume of data. SimpleDB allows users to query with SQL. It is worth noting that SimpleDB can ensure ultimate consistency but does not support Muti-Version Synchronous Control (MVCC). Therefore, its conflicts cannot be detected on behalf of the client.

CouchDB is a documentary database written in Erlang [22]. The data on the DOB couch is organized into documents consisting of fields named by keys/names and values, which are organized as objects and access. Each document is presented with a unique identifier. CouchDB allows access to database documents via restful HTTP API. If a document needs to be changed, the client will need to download the whole document to correct it, and then send it back to the database. After a document has been rewritten once, the ID will be updated. The DB couch uses optimal copying to gain scalability without a sharing mechanism. Since different couches may run simultaneously along with other transactions, any type of tapology iteration can be made. The consistency of the DB couch relies on the copying mechanism. Couch DB supports MVCC with its historical records.

3| Massive Data Analysis Model

Big data is generally stored on hundreds and even thousands of trading servers. Therefore, traditional parallel models such as Message Pass Interface (MPI) and open processing (OpenMP) may not be enough to support such large-scale parallel applications. Recently, some proposed parallel programming

models effectively improve NoSQL performance and reduce the performance gap to relational databases. So these models have become the cornerstone of massive data analysis.

MapReduce [5] is a simple but powerful programming model for large-scale computing using a large number of clusters of commercial PCs to achieve automated parallel processing and distribution. In MapReduce, the computing model only has two functions called Map and Reduce, both of which are programmed by users. The Map function processes key-value input pairs and generates key-value pairs. MapReduce then combines all the middle values corresponding to a key and moves them to the Reduce function, which compresses the set value further into a smaller set.

MapReduce has the advantage of avoiding complex steps to develop parallel applications, as such, data scheduling, error tolerance, and intertidal communication. The user only needs to schedule two functions to develop a parallel application. MapReduce's basic framework does not support multiple data sets in one task, which has been mitigated by some recent improvements [23] and [24]. Over the past decades, programmers have been familiar with the advanced SQL declarative language often used in a relational database to describe the task and analyze the dataset. However, MapReduce's summary framework provides only two nontransparent functions that cannot cover all shared operations. Therefore, programmers should spend their time programming basic functions that are normally hard to maintain and reuse. In order to improve programming efficiency, some advanced language systems have been proposed, as such, Sawzall [25] from Google, Latin Pig [26] from Yahoo, Hive [27] from Facebook, and Scope [28] from Microsoft.

Dryad [29] is a distributed execution engine with general purpose for processing parallel applications of coarse grain data. Dryad's operating structure is a guided cyclic graph, in which the vertices represent programs and edges represent data channels. Dryad runs operations on vertices in clusters and transmits data through data channels including documents, TCP connections, and shared memory FIFO. During operations, resources in a logic operation graph are automatically map to physical sources.

The structure of The Lyad operation is coordinated by a central program called The Work Manager, which can be implemented in clusters or workstations through the network. A job manager consists of two parts: 1) application codes used to build a job communication graph, and 2) program library codes used to sort available resources. Data types are transmitted directly between the vertices. So the manager is solely responsible for making decisions that do not prevent any transfer of data. At Dryad, app developers can choose the flexibility of any guided cycle graph to describe the program's communication modes and express the mechanisms of data transfer. In addition, Dryad allows vertices to use any amount of input and output data, while MapReduce only supports one input and output set. DryadLINQ [30] is an advanced Dryad language and is used to integrate the above SQL language-like execution environment.

All-Pairs [31] is a system specially designed for biometrics, bioinformatics, and data mining applications. Focuses on comparing element pairs in two data sets by a given function. All pairs can be expressed as three-tonels (set A, set B, and function F), in which the F function is used to compare all elements in Set A and Set B. The result is a comparison of an output matrix M, also called a cartz product, or cross-joining of Set A and Set B.

All pairs are implemented in four phases of system modeling, input data distribution, batch work management, and result collection. In the first phase, an almost model of system performance will be built to assess how much CPU source is needed and how to do the job partition. In the second phase, a span tree is built to transmit data, which makes it possible to efficiently retrieve the workload of each input data partition. In the third phase, after delivering the data flow to the appropriate nodes, the All-Pairs engine will build a batch processing submission for jobs on partitions, while sequencing them in the batch processing system, and formulating a node running the steering wheel to obtain data. Finally, after the batch processing system is finished, the extraction engine collects the results and combines them into a suitable structure, which is generally a single file list in which all the results are placed in order.

The Pregel [32] system of Google facilitates the processing of large-sized graphs, e.g., analysis of network graphs and social networking services. A computational task is expressed by a guided graph formed by the headed vertices and edges. Each vertex corresponds to a modifiable and user-defined value, and each redirected edge corresponding to a source vertex is formed by the user-defined value and the identifier of a target vertex. When the graph is built, it performs an ittractoral computing program called superstep, among which global sync points are determined until the algorithm is completed and the output is completed. In each superstep, the vertices are parallel, and each vertex executes the same user-defined function to express a given algorithm logic. Each vertex may change the state of its output edges, receive the message sent from the previous superstep, send the message to other vertices, and even correct the topological structure of the entire graph. Edges are not provided with corresponding calculations. The functions of each vertex may be removed by suspension. When all vertices are in inactive status with no messages to transfer, all run the program is completed. The Pregel program output is a set of output values of all vertices. In general, the input and output of the program are isomorph guided graphs.

4| Conclusion

In this paper, we considered the main issues regarding to the storing and managing big date that are produced by applications, sensors, and users that are searched. We reviewed some important issues including the massive storage systems, distributed storage systems, and storage mechanisms and also the analysis models implemented in big data.

References

[1] DeWitt, D., & Gray, J. (1992). Parallel database systems: The future of high performance database systems. *Communications of the ACM*, *35*(6), 85-98.

[2] Howard, J. H., Kazar, M. L., Menees, S. G., Nichols, D. A., Satyanarayanan, M., Sidebotham, R. N., & West, M. J. (1988). Scale and performance in a distributed file system. *ACM transactions on computer systems (TOCS)*, *6*(1), 51-81.

[3] Cattell, R. (2011). Scalable SQL and NoSQL data stores. *Acm sigmod record*, *39*(4), 12-27.

[4] Ghemawat, S., Gobioff, H., & Leung, S. T. (2003, October). The Google file system. In *Proceedings of the nineteenth ACM symposium on operating systems principles* (pp. 29-43). https://doi.org/10.1145/945445.945450

[5] Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, *51*(1), 107-113.

[6] Deng, Y. (2009). Deconstructing network attached storage systems. *Journal of network and computer applications*, *32*(5), 1064-1072.

[7] Kim, S. K. (2005). Enhanced management method of storage area network (SAN) server with random remote backups. *Mathematical and computer modelling*, *42*(9-10), 947-958.

[8] Chen, M., Mao, S., & Liu, Y. (2014). Big data: A survey. *Mobile networks and applications*, *19*(2), 171-209.

[9] Behrmann, G., Fuhrmann, P., Grønager, M., & Kleist, J. (2008, July). A distributed storage system with dCache. *Journal of physics: conference series* (Vol. 119, No. 6, p. 062014). IOP Publishing.

[10] Cattell, R. (2011). Scalable SQL and NoSQL data stores. *Acm sigmod record*, *39*(4), 12-27.

[11] Chaiken, R., Jenkins, B., Larson, P. Å., Ramsey, B., Shakib, D., Weaver, S., & Zhou, J. (2008). Scope: easy and efficient parallel processing of massive data sets. *Proceedings of the VLDB endowment*, *1*(2), 1265-1276.

[12] Beaver, D., Kumar, S., Li, H. C., Sobel, J., & Vajgel, P. (2010, October). Finding a Needle in Haystack: Facebook's Photo Storage. *OSDI* (Vol. 10, No. 2010, pp. 1-8). https://www.usenix.org/conference/osdi10/finding-needle-haystack-facebooks-photo-storage

[13] Thalheim, B. (2013). *Entity-relationship modeling: foundations of database technology*. Springer Science & Business Media.

[14] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., ... & Vogels, W. (2007). Dynamo: Amazon's highly available key-value store. *ACM SIGOPS operating systems review*, *41*(6), 205-220.

[15] Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., & Lewin, D. (1997, May). Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. *Proceedings of the twenty-ninth annual ACM symposium on theory of computing* (pp. 654-663). https://doi.org/10.1145/258533.258660

[16] Sumbaly, R., Kreps, J., Gao, L., Feinberg, A., Soman, C., & Shah, S. (2012, February). Serving large-scale batch computed data with project Voldemort. In *FAST* (Vol. 12, pp. 18-18). https://www.usenix.org/conference/fast12/serving-large-scale-batch-computed-data-project-voldemort

[17] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., ... & Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. *ACM transactions on computer systems (TOCS)*, *26*(2), 1-26.

[18] Burrows, M. (2006, November). The Chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th symposium on Operating systems design and implementation* (pp. 335-350). https://www.usenix.org/legacy/event/osdi06/tech/full_papers/burrows/burrows_html/

[19] Bradshaw, S., Brazil, E., & Chodorow, K. (2019). *Mongodb: the definitive guide: powerful and scalable data storage*. O'Reilly Media.

[20] Bryan, P., & Nottingham, M. (2013). *Javascript object notation (json) patch*. Retrieved from https://www.hjp.at/doc/rfc/rfc6902.html

[21] Murty, J. (2008). *Programming amazon web services: S3, EC2, SQS, FPS, and SimpleDB*. " O'Reilly Media, Inc."

[22] Anderson, J. C., Lehnardt, J., & Slater, N. (2010). *CouchDB: the definitive guide: time to relax*. " O'Reilly Media, Inc.".

[23] Blanas, S., Patel, J. M., Ercegovac, V., Rao, J., Shekita, E. J., & Tian, Y. (2010, June). A comparison of join algorithms for log processing in mapreduce. *Proceedings of the 2010 ACM SIGMOD international conference on management of data* (pp. 975-986). https://doi.org/10.1145/1807167.1807273

[24] Yang, H. C., & Parker, D. S. (2009, April). Traverse: simplified indexing on large map-reduce-merge clusters. *International conference on database systems for advanced applications* (pp. 308-322). Springer, Berlin, Heidelberg.

[25] Pike, R., Dorward, S., Griesemer, R., & Quinlan, S. (2005). Interpreting the data: Parallel analysis with Sawzall. *Scientific programming*, *13*(4), 277-298.

[26] Gates, A. F., Natkovich, O., Chopra, S., Kamath, P., Narayanamurthy, S. M., Olston, C., ... & Srivastava, U. (2009). Building a high-level dataflow system on top of Map-Reduce: the Pig experience. *Proceedings of the VLDB endowment*, *2*(2), 1414-1425.

[27] Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., ... & Murthy, R. (2009). Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB endowment*, *2*(2), 1626-1629.

[28] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., ... & Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. *ACM transactions on computer systems (TOCS)*, *26*(2), 1-26.

[29] Isard, M., Budiu, M., Yu, Y., Birrell, A., & Fetterly, D. (2007, March). Dryad: distributed data-parallel programs from sequential building blocks. *Proceedings of the 2nd ACM SIGOPS/EuroSys European conference on computer systems 2007* (pp. 59-72). https://doi.org/10.1145/1272996.1273005

[30] Ekanayake, J., Gunarathne, T., Fox, G., Balkir, A. S., Poulain, C., Araujo, N., & Barga, R. (2009, December). Dryadlinq for scientific analyses. In *2009 Fifth IEEE international conference on e-science* (pp. 329-336). IEEE.

[31] Moretti, C., Bulosan, J., Thain, D., & Flynn, P. J. (2008, April). All-pairs: An abstraction for data-intensive cloud computing. In *2008 IEEE international symposium on parallel and distributed processing* (pp. 1-11). IEEE.

[32] Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., & Czajkowski, G. (2010, June). Pregel: a system for large-scale graph processing. *Proceedings of the 2010 ACM sigmod international conference on management of data* (pp. 135-146). https://doi.org/10.1145/1807167.1807184