# ENHANCED BIG DATA PLATFORMS FOR FAST QUERY RESPONSE WITH HIVE, IMPALA AND SPARKSQL

[1]Revati Khushal Rane, [2]Prof. Samadhan Sonavane
[1]Student, [2]Assistant Professor
[1]Computer Science and Engineering,
[1]School of Engineering and Technology, Sandip University, Nashik, India

***Abstract :*** With the development of cloud computing technologies and mobile internet data is getting generated tremendously hence creating big data. There is a great challenge to analyze and extract important data from it and also getting benefit out of it. Therefore big data processing is facing incredible challenges. In order to provide solution to this issue, this paper focuses on storage of big data and system performance optimization. This paper integrates three big data tools i.e Hive, Impala and SparkSQL which supports SQL-like queries in big data environment. These platforms are used to fast respond to user's query when the optimized system will automatically select the particular platform to best perform a query. In addition to this approach this paper provides in-memory cache and in-disk cache for the fast data retrieval for the repeated SQL commands. The proposed approach improves performance and efficiency of the data retrieval significantly.

***IndexTerms* - Big data tools, in-memory cache, in-disk cache, big data processing, optimized platform selection.**

## I. INTRODUCTION

As indicated by the most recent research results of American journal of CIO, as 70% of activity is finished by batch processing in the business process managed by IT, it makes "Unfit to control activity load processing resources[1]" become one of the greatest difficulties for enormous big data application[2]. The objective of this study is to realize a multiple big data processing platform with high performance, high availability and high scalability that will be compatible with any existing business intelligence and analysis tools. Enterprises do not need to import this platform, but it will eliminate a lot of traditional software. This platform will support SQL-like query statements in order to process big data, so existing tools relying on relational databases as a data source can be made compatible with a minimum of modification, or even no modifications to the new platform at all, and provide companies with relatively easy access to the advantages of the new platform, such as high performance and high availability. The speed at which big data is read is significantly increased by amortizing the I/O delay time through a reliable distributed file system. This study will build a Multiple Big Data Processing Platform using (1) Apache Hive, (2) Cloudera Impala and (3) BDAS Spark SQL, as shown in Fig. 1; these three systems support SQL-like command-based data warehouse and underlying systems (1) Apache Hadoop, (2) the MapReduce platform and (3) BDAS Spark.

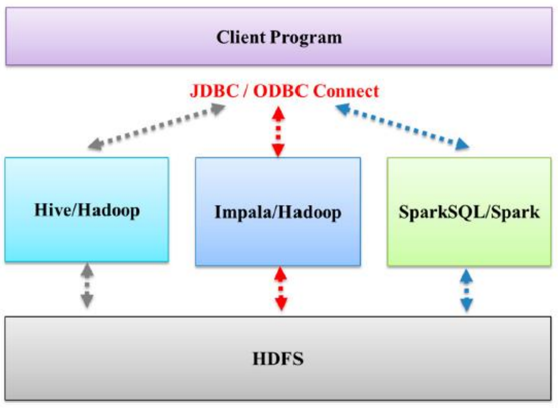

Figure 1. multiple big data processing Platforms

## II. RELATED WORK

So as to give open source solution for the issue of big data, this paper focuses around the solution of data and performance optimization. In the first, it plans to understand MapReduce/HDFS as indicated to the background problems, at that point, this research will build following tools(1) Hive[3],(2) Impala[4], (3) Spark SQL [5], (4) Hue [6] and (5) Memcached [7] because of interest for highlights of a wide range of big data and analysis. This section introduces about key computing technologies.

### A. Introduction to Mapreduce/HDFS

The most straightforward Hadoop structure [8] can be separated into the upper MapReduce [9]and the lower HDFS[10], as is appeared in Fig. 2. The server can be separated into Master node and Worker node dependent on the utilization, the Master node is responsible to give task to appropriate worker node and worker node is responsible to perform given task.The server in Master hub performs two arrangements of programs, one is JobTracker in charge of giving task of MapReduce calculation layer and the other is NameNode program which is in charge of the administration of HDFS data layer. The server in Worker node has likewise two arrangements of projects, one is TaskTracker program which acknowledges the direction of JobTracker and plays out the undertakings of calculation layer and the other is DataNode program which is relating to NameNode and which is in charge of performing data reading and writing and performing duplicate technique of NameNode.

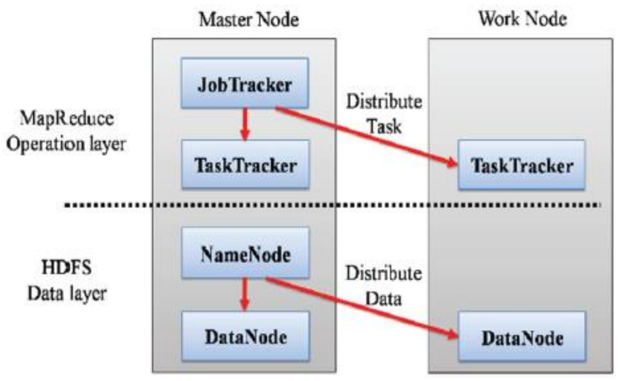

Figure 2. hadoop structure diagram

B. Distributed Data Storage platform-Hive

Hive is a tool of the Apache Hadoop. At the point when the construction of Hive framework is completed, it will begin a HiveServer to get and process the customer's Hive-QL commands. The user can associate Hive at the Hadoop stage through a direction line interface (CLI) or Java database availability (JDBC) driver and afterward it send Hive-QL directions to Hive. After Hive gets a Hive-QL direction, the Hive-QL order will be incorporated to a Java program, and Hadoop MapReduce executes the activity asked for from a query. When a Hive-QL query conjures the table creation, the framework will assemble another Hive's own table (an exchange table) for Hive at the Hadoop stage and makes its metadata in the meantime. Metadata will be put away in the relational database MySQL called Metastore, and from that point Hive-QL inquiry can get the data in Metastore to search for where information are found and what number of information to retrive. Notice that Hive's metastore data is normally utilized for Impala and SparkSQL with the goal that Impala/SparkSQL sharing a similar Hive's metastore data is separately ready to play out the quick information retrival from HDFS where the plain huge amount of information has been put away in HDFS.

C. SQL Query Engine- Impala

Impala, developed by Cloudera is used to speed up for this kind of SQL-Like query statement it uses its own process called MPP (massively parallel processing) query engine. Impala uses LLVM (low level virtual machine, written in C++) [22] to compile these statements. Using an LLVM can significantly reduce the compiling cost. Impala is capable of handling Hive-like SQL commands, and fortunately is able to access Hive’s metastore information as well. The main functioning of Spark is the similar as Hadoop MapReduce, but it uses in-memory cluster computing. The storage system is also compatible with HDFS.

D. SparkSQL

SparkSQL is a Hive-like system which is like based on Spark MapReduce. It is fully compatible with Hive as well as being capable of accessing Hive’s metastore data like metadata stored in the MySQL database.

*E. Hue*

Hue is a web-based GUI for Hadoop and Impala. Hue in Hadoop or Impala is just like phpMyAdmin in MySQL. This study employed Hue as a user-machine interface because users can easily perform operations and observations in addressing the problem of the Hadoop ecosystem’s difficulty of operation.

F. Memcached

Memcached [11] is a key-value distributed memory caching system. The key length is limited to 250 characters and a single datum cannot exceed 1MB. Currently, it is frequently used in websites and database search cache. In Figure 6, the technique has provided Spymemcached metastore information to locate the data in the memory cache or in the disk cache so that Spymemcached is able to upgrade data between the memory cache and the disk cache asynchronously. This paper intends to implement rapid data retrieval the in-memory cache and the in-disk cache if there is no necessary to start Hive, Impala, or SparkQSL for dealing with an SQL query.

## III. Related Method

The combination of three big data processing tools—Hive, Impala, and SparkSQL—can be boosted through the following two ways: (1) the rapid data retrieval from in-memory caching or in-disk caching using Memcached if data have been cached earlier; otherwise (2) platform selection for choosing the appropriate tool to speed up the query/response operation at HDFS. It should provide a metric to indicate the system efficiency and thus a performance index has been found in this paper to show the performance evaluation among different approaches.

A. Multi-System Compatibility Solution

Hive requires very few memory resources, but it will write temporary files to HDFS frequently, resulting in long computation times and poor performance. In contrast, parkSQL needs more memory because most of the operations are done in memory and uses less write files to hard disk. SparkSQL, however, may risk a routine crash as the memory is not sufficient. On the other hand Impala supports most standard SQL statements superior for more advanced

## B. Platform Selection

The integration of big data tools aims to accomplish the best performance and efficiency of data retrieval. In order to fulfil this goal, the so-called platform selection executes the appropriate tool to finish a job as soon as possible. In this manner, the approaching query direction will be dispatched to the suitable tool to accelerate data retrieval. Hive is a tool which is supported for SQL command that is executed by MapReduce at Hadoop device. It must write to disk too often, bringing about low execution of the query task. To address this issue, Impala utilizes C++ language to re-compose its high-performance Message Passing Interface (MPI) search engine, significantly reducing hard disk writing and fundamentally enhancing execution. Spark utilizes in-memory MapReduce technology, guaranteeing that SparkSQL has a quick processing speed. In spite of the fact that these three tools are practically comparative, their condition prerequisites and performances differ. Whenever the remaining memory limit is 2 GB or less on every server, Impala and SparkSQL had a great deal of page swapping, causing to a great degree low execution or crash. At the point when the information scale was bigger, it caused the JVM I/O exception and made the program crash. When the rest of the memory limit was adequate, SparkSQL was quicker than Hive and Impala. Impala's utilization of memory resources was between those of SparkSQL and Hive. This measure of remaining memory was adequate for Impala's maximum performance. In the experiment as found in Section 2.4, every server was dispensed 20 GB of memory allocated to each computing node and set the rest of the amount of memory to 3 GB as critical point 1 (signified as L1) and 15 GB as critical point 2 (meant as L2). The program will automatically choose Hive when the rest of the measure of memory was under L1, Impala between L1 to L2, SparkSQL when bigger than L3 as appeared in fig 3.

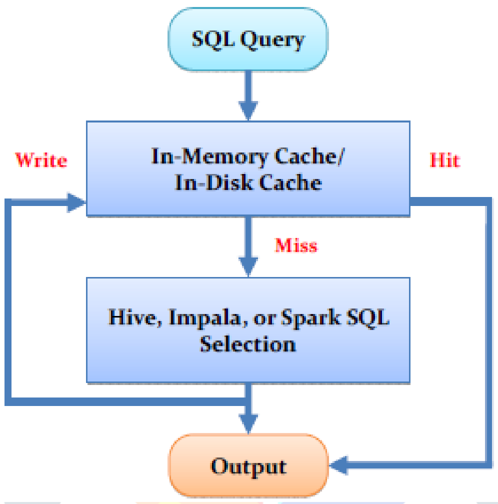
SQL Query
In-Memory Cache/ In-Disk Cache
Write
Hit
Miss
Hive, Impala, or Spark SQL Selection
Output

Figure 3. platform selection and caching mechanism

## C. Caching Mechanism

A caching mechanism was more essential in some of the environment that has many duplicate SQL commands. Each search query will require time and resources. These can be diminished while the cache hits. In this manner, this study planned a rapid in-memory cache and a substantial limit in-disk cache and the flowchart is appeared in Figure 12. This examination utilized Memcached to build up the proposed in-memory cache to yield profoundly effective data retrieval however it experienced the capacity limitations. Conversely, the thought was to utilize a HDFS distributed file system for the in-disk cache since it saved the search result as a content document and transferred it to the specified folder in HDFS. According to the least recently used (LRU) cache replacement policy, outdated dara in the memory cache was erased to maintain the limit size of memory for a cache. The advantage was that you can diminish the amount of memory utilized, keep just certain "hot stuff" stored in a cache, put different less-prevalent, rarely accessed information in the database, and did not compose a duplicate to store until the point when the following same demand to the database happened. Some new data could be included the database, however but data did not instantly become stored in a cache. The component of the in-memory cache together with the in-disk cache was made out of six algorithms to manage (1) deleting outdated information in the memory cache,(2) deleting outdated document in the disk cache (3)compute and add up-to-date data to memory cache(4)compute and add up to date data to disk cache(5) visit current information (hit or miss) in the memory cache (6) visit current information (hit or miss) in the disk cache.

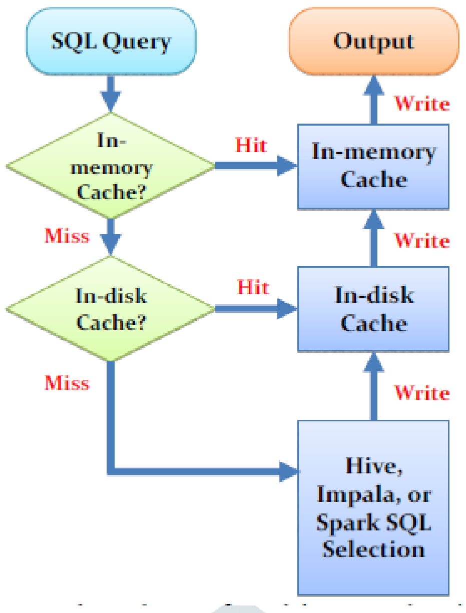

Figure 4. flowchart of caching mechanism

Since each block of in-memory cache has timeliness, Memcached checking the data will automatically replace the cache by a LRU cache replacement approach when the framework is full or the cache has been put away for more than one month. With the usage of the LRU strategy for the in-disk cache, the history list has executed this approach in this program. This list records the location and the last access time to this cache. At the point when a cache hits, the program refreshes the list once. Checking this this, the program can discover some stored information that had not been gotten to for quite a while furthermore, will erase that quickly. The client, through the interface of this program, can enter the purge x command to erase cached data which had not been accessed in specific x days.
The search query results are at the same time put away in-memory and in-disk, however the in-memory reserve has the most high priority access need. At the point when the in-memory cache hits, the outcome won't be recovered by the in-disk cache and big data tools which have been used in this study. At the point when the in-memory cache misses, the outcome will be recovered through the in-disk cache and will be consequently composed again into the in-memory cache. Whenever both in-memory and in-disk cache have missed, the outcome will be recovered by big data tools and written to the in-memory and in-disk caches.

## IV. EXPERIMENTAL RESULTS AND DISCUSSION

Different versions of experimental environment are shown in Table I. Three test environment are used each environment is allocated different memory size as shown in Table II. Each platform is analyzed by executing different sizes of test data. The size of test information is recorded in TABLE III. Different Test SQL query are recorded in TABLE IV. The test method used are shown in TABLE V. In test environment I the of execution time among different methods are appeared in Figs. 5, 6 and 7. The trial results demonstrate that Hive can still work even if there is lack of memory however Impala and Spark SQL have crashed in response to certain sizes of information estimate. The representations of execution time between strategies are appeared in Figs. 8,9 and 10 in test condition II. The exploratory outcomes demonstrate that Impala's execution is increasingly conspicuous with medium sums of memory. In test condition III, the representations of execution time between techniques are appeared in Figs. 11, 12 and 16. The test results demonstrate that Spark SQL's execution is great with increasingly satisfactory measures of memory.

TABLE I THE STABLE VERSION OF PLATFORMS

| Platform | Version |
|---|---|
| Apache Hadoop | 2.3.0 |
| Apache Hive | 0.12.0 |
| Apache Spark | 1.6.0 |
| Cloudera Impala | 1.3 |
| Oracle Java (JDK) | 7u65 |
| Scala | 2.10.3 |

TABLE II THE STABLE VERSION OF PLATFORMS

| Group | Memory size |
|---|---|
| Test environment I | Reserve 2GB remaining memory space |
| Test environment II | Reserve 10GB remaining memory space |
| Test environment III | Allocate all memory space |

TABLE III TEST DATA

| Group | Data size |
|---|---|
| Test data I | 10GB- About 40 million records |
| Test data II | 100GB-About 400 million records |
| Test data III | 500GB- About 2 billion records |
| Test data IV | 1TB- About 4 billion records |

TABLE IV TEST SQL COMMAND

| Category | Function |
|---|---|
| SQL command I | Only search specific fields |
| SQL command II | Add the comparison condition |
| SQL command III | Execute the commands contained Table JOIN |

TABLE V TEST METHOD

| Method | Command |
|---|---|
| Hive | Use "enforced hive" command to run the platform |
| Impala | Use "enforced impala" command to run the platform |
| Spark SQL | Use "enforced shark" command to run the platform |
| In-memory Cache | Reaction time when cache hit |
| In-disk Cache | Reaction time when cache hit |

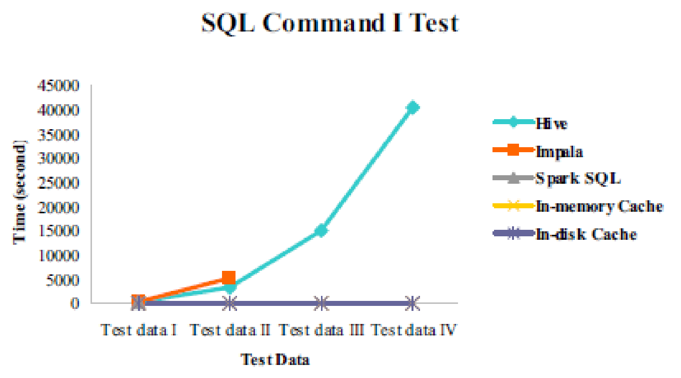

Figure 5. Execution time of SQL query 1 at test environment 1

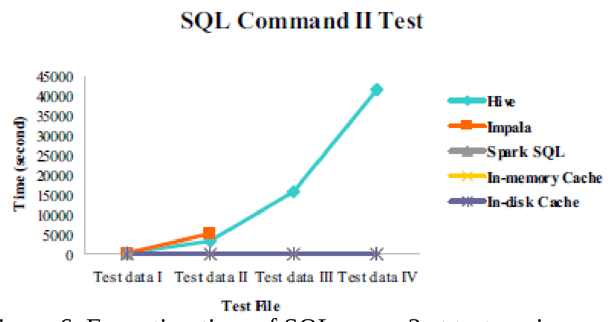

Figure 6. Execution time of SQL query 2 at test environment 1

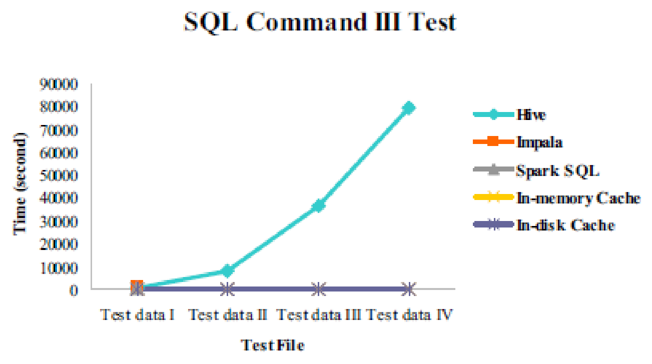

Figure 7. Execution time of SQL query 3 at test environment 1

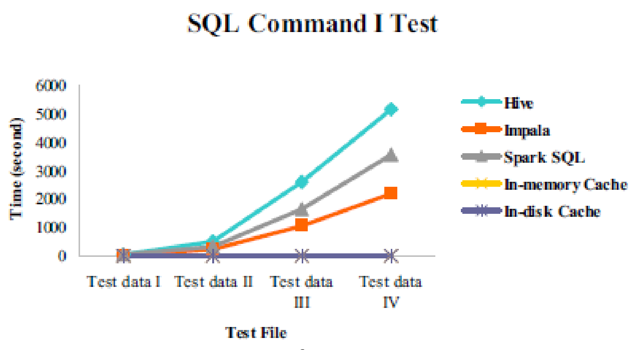

Figure 8. Execution time of SQL query 1 at test environment 2

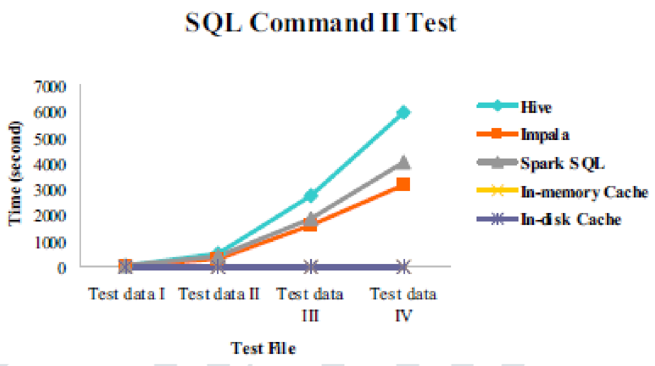

Figure 9. Execution time of SQL query 2 at test environment 2

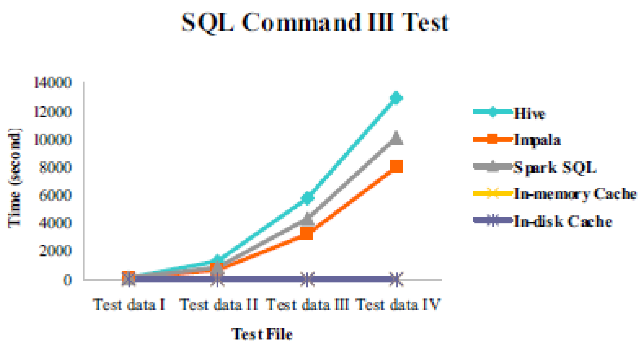

Figure 10. Execution time of SQL query 3 at test environment 2

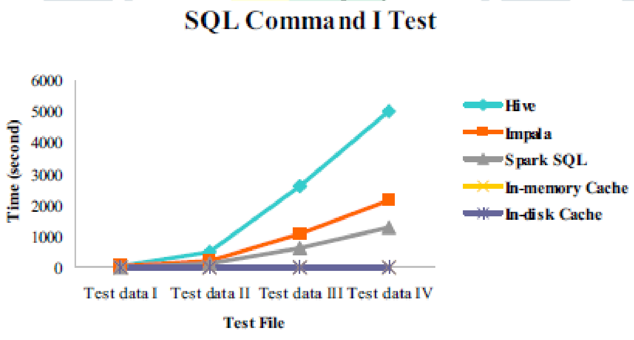

Figure 11. Execution time of SQL query 1 at test environment 3

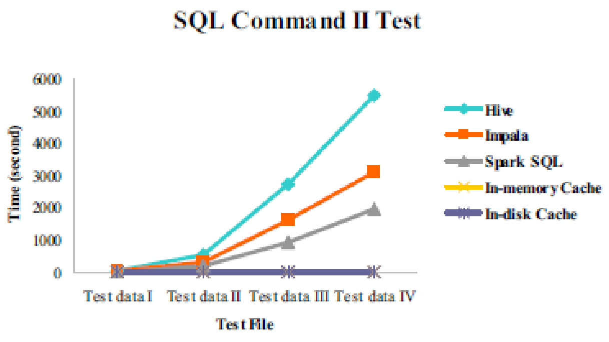

Figure 12. Execution time of SQL query 2 at test environment 3

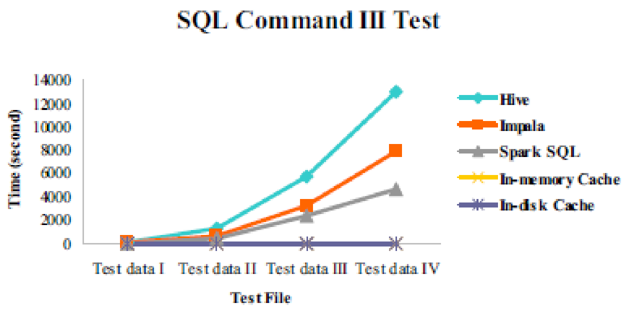

Figure 13. Execution time of SQL query 3 at test environment

## V. CONCLUSION

This paper has accomplished by automatically detecting the status of a cluster through checking the remaining memory size at computing nodes and then choosing the the appropriate and suitable tool to manage or execute the SQL command for a quick query response. Additionally to more enhance the system as well as to improve the performance it incorporated cache system like in-memory cache and in-disk cache for fast retrieval of query and to decrease the response time Additionally this type of fast query response gives contribution to implement applications for example, OLAP, data mining, and real time statistics, by utilizing complex SQL commands for real world use cases. In future work, we will be reformulating this study by targeting updation queries intsead of just seach queries used in this study and more ever to enhance the system will note down the frequency and give rank to the queries which will help us to predict which query will be placed in in-memory cache. Additionally we will reformulate with newer versions of the components i.e Hive, Impala, and SparkSQL.

## REFERENCES

[1] C. D. Wickens, “Processing Resources in Attention Dual Task Performance and Workload Assessment,” Office of Naval Research Engineering Psychology Program, No. N-000-14-79- C-0658, July, 1981.
[2] H.-C. Chen, R. H. L. Chiang, and V. C. Storey, “Business Intelligence and Analytics: From Big Data to Big.
[3]Impact,” MIS Quarterly, Vol. 36, No. 4, pp. 1165-1188, December, 2012. A. Thuso, “Hive - a petabyte scale data warehouse using Hadoop,” 2010 IEEE 26th International Conference on Data Engineering, pp. 996-1005, March 1-6, 2010.
[4] A. Kamburov,R. Cavill,T. M.D.Ebbels,R. Herwig1, and H. C. Keun, “Integrated pathway-level analysis of transcriptomics and metabolomics data with IMPaLA,” Bioinformatics, Vol. 27, Iss. 20, pp. 2917-2918, September, 2011.
[5] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia, “Spark SQL: Relational Data Processing in Spark,” SIGMOD '15 Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 1383-1394, May 31- June 04, 2015.
[6] Hue - http://gethue.com/
[7]F. Li, S. Zhan, and L. Li, “Research on using memcached in call center,” 2011 International Conference on Computer Science and Network Technology (ICCSNT), Vol. 3, pp. 1721-1723, Dec. 24-26, 2011.
[8] M. Adnan, M. Afzal, M. Aslam, R. Jan, and A. M. Martinez- Enriquez, “Minimizing big data problems using cloud computing based on Hadoop architecture,” 2014 11th Annual High Capacity Optical Networks and Emerging/Enabling Technologies (Photonics for Energy), pp. 99-103, Dec. 15-17,
2014.
[9] M. Maurya, and S. Mahajan, “Performance analysis of MapReduce programs on Hadoop cluster,” Proceeding of World Congress on Information and Communication Technologies, pp. 505-510, 2012.
[10] A. K. Karun, and K. Chitharanjan, “A review on hadoop — HDFS infrastructure extensions,” Information & Communication Technologies (ICT) 2013 IEEE Conference on, pp. 132-137, 2013.
[11] Fitzpatrick, B. Distributed Caching with Memcached. Linux J. **2004**, 2004, 5.
[12] Chang, B.R.; Tsai, H.-F.; Chen, C.-Y.; Guo, C.-L. Empirical Analysis of High Efficient Remote Cloud Data Center Backup Using HBase and Cassandra. Sci. Program. **2015**, 294614, 1–10.
[13] Li, P. Centralized and Decentralized Lab Approaches Based on Different Virtualization Models. J. Comput.Sci. Coll. **2010**, 26, 263–269.
[14] Almgren, K.; Kim, M.; Lee, J. Extracting Knowledge from the Geometric Shape of Social Network Data Using Topological Data Analysis. Entropy **2017**, 19, 360.
[15] Fan, S.; Lau, R.Y.; Zhao, J.L. Demystifying Big Data Analytics for Business Intelligence through the Lens of Marketing Mix. Big Data Res. **2015**, 2, 28–32.
[16] Borthakur, D. The Hadoop Distributed File System: Architecture and Design. Hadoop Proj. Website **2007**, 11, 21.
[17] Dean, J.; Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. Commun. ACM **2008**, 51,107–113.
[18] Ghemawat, S.; Gobioff, H.; Leung, S.T. The Google File System. In Proceedings of the ACM SIGOPS Operating Systems Review (SOSP ’03), Bolton Landing, NY, USA, 19–22 October 2003; Volume 37, pp. 29–43.
[19] DeCandia, G.; Hastorun, D.; Jampani, M.; Kakulapati, G.; Lakshman, A.; Pilchin, A.; Vogels,W. Dynamo: Amazon’s Highly Available Key-Value Store. ACM SIGOPS Oper. Syst. Rev. **2007**, 41, 205–220.
[20] Casado, R.; Younas, M. Emerging Trends and Technologies in Big Data Processing. Concurr. Comput. Pract. Exp. **2015**, 27, 2078–2091.
[21] Abadi, D.; Babu, S.; Özcan, F.; Pandis, I. SQL-on-Hadoop Systems: Tutorial. Proc. VLDB Endow. **2015**, 8, 2050–2051.
[22] Bajaber, F.; Elshawi, R.; Batarfi, O.; Altalhi, A.; Barnawi, A.; Sakr, S. Big Data 2.0 Processing Systems: Taxonomy and Open Challenges. J. Grid Comput. **2016**, 14, 379–405.

[23] Zlobin, D.A. In-Memory Data Grid. 2017. Available online:http://er.nau.edu.ua/bitstream/NAU/27936/1/Zlobin%20D.A.pdf Chang, B.R.; Tsai, H.-F.; Tsai, Y.-C. High-Performed Virtualization Services for In-Cloud Enterprise Resource Planning System. J. Inf. Hiding Multimed. Signal Process. **2014**, 5, 614–624.
[24] Proxmox Virtual Environment. Available online: https://p.ve.proxmox.com/ Chang, B.R.; Tsai, H.-F.; Chen, C.-M.; Huang, C.-F. Analysis of Virtualized Cloud Server Together with Shared Storage and Estimation of Consolidation Ratio and TCO/ROI. Eng. Comput. **2014**, 31, 1746–1760
[25] Thusoo, A.; Sarma, J.S.; Jain, N.; Shao, Z.; Chakka, P.; Zhang, N.; Antony, H.S.; Liu, R.; Murthy, R. Hive—A Petabyte Scale Data Warehouse using Hadoop. In Proceedings of the IEEE 26th International Conference on Data Engineering, Long Beach, CA, USA, 1–6 March 2010; pp. 996–1005.
[26]LLVM3.0ReleaseNotes.Availableonline:http://releases.llvm.org/3.0/docs/ReleaseNotes.html.
[27] Gibilisco, G.P.; Krstic, S. InstaCluster: Building a big data cluster in minutes. arXiv **2015**, arXiv:1508.04973.
[28] Fitzpatrick, B. Distributed Caching with Memcached. Linux J. **2004**, 2004, 5.
[29] Chang, B.R.; Tsai, H.-F.; Chen, C.-Y.; Guo, C.-L. Empirical Analysis of High Efficient Remote Cloud Data Center Backup Using HBase and Cassandra. Sci. Program. **2015**, 294614, 1–10.