Performance Evaluation of In-Memory Computing on Scale-Up and Scale-Out Cluster

Taekyung Yoo, Minsub Yim, Ilgyun Jeong, Yunsu Lee, Seung-Tae Chun
{tkyoo, msyim, igjeong, ysulee, stchun }@datastreams.co.kr
Datastreams Corp.

***Abstract*—Apache Spark framework, which is the implementation of Resilient Distributed Datasets(RDD), is used instead of MapReduce on recent data processing models of Hadoop ecosystem. In this paper, we evaluated the performance and resource usage of real world workloads on scale-up and scale-out clusters using the in-memory caching feature of Spark framework. In our experiments, scale-up processed data more efficiently than scale-out in write intensive workloads such as Sort and Scan, whereas scale-out had strength in those utilizing iterative algorithms such as Join, Pagerank and KMeans. Considering the efficiency in physical factors including performance per watt and the physical space each occupies, we show that it is more advantages to use scale up cluster than scale out.**

I. Introduction

Apache Hadoop [1] is widely used as a distributed computing framework which chooses the scale out method; it uses many cheap nodes instead of a few powerful machines to perform calculations on a large dataset. One common programming model used in Hadoop framework is MapReduce [10], which is developed by Google for processing large datasets. One of the characteristics of MapReduce is that it is a simple model composed of only two steps: a map phase and a reduce phase. It provides users an easy access to high-level cluster computing on a lot of nodes. However, researchers have found a lot of weaknesses of this system including inefficient use of disk space [16], lack of consideration on data locality [17], etc. and many attempts to solve these problems have been in progress.

Resilient Distributed Datasets(RDD) [4] is a new data model to offer a fault tolerant, in-memory data access method to users. It is designed to speed up iterative algorithm and increase performance of most common cases processed by Hadoop MapReduce at the same time. As a result, the trend of data processing model in Hadoop framework has shifted from MapReduce to the implementation of RDD, Apache Spark framework [2], [3].

Because Hadoop framework builds scale out clusters to process its tasks, users have come to believe that scale out clustering has better performance than scale up in Hadoop framework. However, according to a recent experiment on performance comparison of scale up and scale out methods [5], scale out did not always show better results compared to scale up. Especially, scale up had a higher performance per watt ratio than scale out in overall applications. Also, the physical space occupied by scale up had a lower density than that of scale out, as expected. However, this research [5] used MapReduce for a data processing model; If RDD is to be used instead, the following points should be considered:

1) RDD has higher CPU and memory usage than MapReduce because it caches input data in memory, optimizes the execution phase of each task, etc. Thus, CPU performance is more critical in a model using RDD than that using MapReduce.
2) Since a lot of input data reside in memory, network bandwidth may become a new bottleneck instead of disk bandwidth. However, the result showed otherwise when hard disk was used in cluster [12].

To confirm the foregoing assumptions, we compared scale-up and scale-out cluster using in-memory computing through Apache Spark framework.

TABLE I
Workload Input / Output information used in our experiments

Name	Input	Output
Micro Benchmarks		
Wordcount	52GB	740MB
Sort	52GB	52GB
Web Search		
Pagerank	2.8GB	136MB
Machine Learning		
KMeans	37.4GB	-
Analytical Query[a]		
Scan	78.7 GB	137.4 GB
Aggregation	78.7 GB	20GB
Join	82.1 GB	720MB

[a]Analytical query workloads changed the input file format from sequence file to JSON.

II. Workloads

To measure the performance of our cluster setup, we used Intel HiBench benchmark suite [8]. This benchmark suite categorizes workloads into Micro benchmark, Web Search, Machine Learning or Analytical Query. Table I represents input and output file size of workloads used in our experiments. Since HiBench version 5.0 did not have a Spark implementation for join, scan and aggregation, we implemented it simply by changing HiveQL to SparkSQL [6].

TABLE II
THE HARDWARE RESOURCES USED IN OUR EXPERIMENTS

HW	Scale-up	Scale-out
CPU	3.4GHz, 6 Cores * 2	3.4GHz, 4 Cores
RAM	DDR4 256GB	DDR3 32GB
Disk	SSD * 6 (RAID0)	HDD
Network	-	1Gbps
Total cost[a]	$7,733	$1,034

[a]Prices from Amazon, http://www.newegg.com, and Intel recommended customer price

TABLE III
SPARK CONFIGURATION CHANGED IN OUR EXPERIMENTS

Configuration Name	Scale-Up	Scale-Out
spark.executor.memory	16GB	6GB
spark.driver.memory	16GB	6GB
spark.default.parallelism	12[a]	20
spark.shuffle.compress	true	true
spark.shuffle.file.buffer	32KB	4MB
spark.shuffle.split.compress	true	true
spark.io.compression.codec	snappy	snappy
spark.rdd.compress	true	true

[a]In scale-up, the value was set to 11 for this configuration when executing Pagerank workload for eliminating straggler tasks.

A. Micro Benchmark

Micro benchmark is a set of workloads used frequently in real-world jobs. We used Wordcount and Sort in workloads provided by HiBench, and the input file for this experiment was generated by RandomTextWriter included in Hadoop MapReduce example. Wordcount is a well-known example of MapReduce which, obviously, counts the number of words. Similarly, Sort is another famous workload example used in various applications to rearrange the input data.

B. Web Search

Web search is one of typical fields which actively apply MapReduce for processing large data sets. We used Pagerank algorithm in web search benchmark. The input file for Pagerank was created by a job offered by HiBench. Pagerank algorithm is a famous algorithm Google uses for searching. This algorithm computes ranks by calculating the count and the value of edges between vertices.

C. Machine Learning

Machine learning is one of the most popular areas in MapReduce along with web search. We used KMeans clustering which is a well-known clustering algorithm in data mining. The input file for KMeans was created by a job offered by HiBench. KMeans is an iterative algorithm composed of multiple stages. It randomly chooses k points from the input dataset to be used as the center of each cluster. Using the k points, it creates k-clusters by checking the nearest points from each center and calculates the new centroid of each cluster. This process is repeated for a fixed number of iterations or until convergence has been reached.

D. Analytical Query

Query processing based on SQL is a useful tool to hide complex computation or optimization and it offers an easy instruction to users. Analytical queries of HiBench were published in the paper [9] which compared large scale data access methods. Aggregation computes total revenue generated from a specific user IP, and Join calculates the average rank and sum of revenues for a user IP on a certain date. Scan selects all data from the user table.

III. EXPERIMENTAL ENVIRONMENT

Table II shows that the hardware resources used in our experiments. For our tests, we set up two types of nodes: scale-up is a NUMA-architecture machine which includes two of Intel(R) Xeon(R) CPU E5-2643 V3 @ 3.4GHz 6 cores, DDR4 256GB ram and six solid state drive(SSD)s. We organized disks to RAID-0 for accessing data at a high disk throughput. This RAID-0 disks can read or write data at about 1GB/s. The price of this machine was approximately $7,733. Scale-out cluster has 10 nodes, which are set to have the identical hardware specification. each machine contains Intel(R) Xeon(R) CPU E3-1245 V2 @ 3.4 Ghz 4 cores and DDR3 32GB ram, and a single hard disk drive(HDD). Thus, it has about 100MB/s disk bandwidth. The interconnection speed between two nodes is at 1Gbps, and all of nodes are installed in one rack. The price of each node is approximately $1,034.

A. Disk Performance Issue

We already know that SSDs have a better price - performance ratio than HDDs [5]. However, we believe that our experiment environment can inspire researchers or engineers in following topics: 1) the effect of network traffic when scale-up and scale-out clusters have similar disk bandwidths. 2) disk performance issues caused by hardware limitation. (Especially, HDD) 3) how the configuration of Spark framework affects different disk types.

B. Hadoop & Spark Configuration

Our experiments were run on Hadoop YARN framework [7]. To reduce additional overhead of Hadoop Distributed File System(HDFS), we set the replication factor to 1. Since Spark has a cpu-intensive hardware resource pattern, the number of YARN container was set to match the number of physical cpu cores. Thus, scale-up has 12 containers and scale-out has 40 containers, respectively. Like Hadoop MapReduce, Spark configuration has a lot of effects on applications; thus, we suitably changed a few Spark configurations. Table III represents the list of adjustments made in our environment. The values *spark.driver.memory* and *spark.executor.memory* are the maximum usable memory values that do not use the swap memory space. The default value for *spark.default.parallelism*

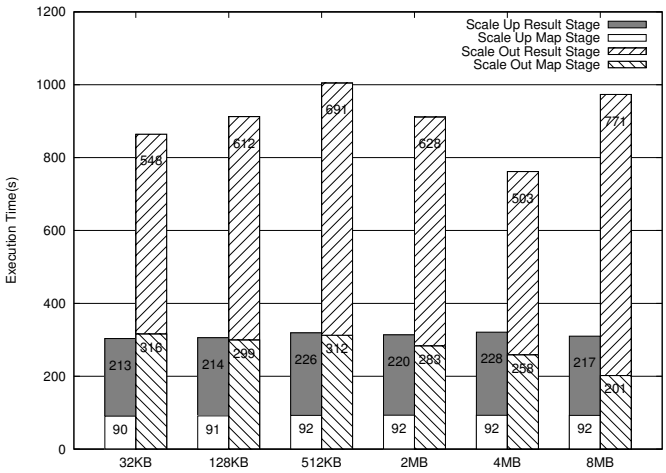


Fig. 1. The Execution Time of Sort for Each Buffer Size

is the total number of cores on nodes where Spark executors are handled, but scale-out cluster had better performance in our environment when half the cores were used. Particularly, changes in *spark.shuffle.file.buffer* caused a huge gap in the resulting values in scale-out when executing write-intensive applications, such as Sort.

Figure 1 represents the execution time of sort with varying *spark.shuffle.file.buffer* sizes. The values are taken from the average time calculated over 5 executions. While the execution time of scale-up was relatively stable with respect to various buffer sizes, the result of scale-out showed diversity in execution time. The execution time of map stage in scale-out was inversely proportional to the buffer size, but the execution time in the result stage increased when the buffer size was set to 8MB. Therefore, we set the buffer size to 4MB in our experiments which yielded the minimum execution time.

C. Power Consumption

Due to our hardware restrictions, we collected each node's CPU watt information using a software called Intel(R) Running Average Power Limit(RAPL) [19]. This feature is included in Intel SandyBridge chips and can estimate current energy based on hardware counters, temperature, and leakage models [18]. Linux kernel provides a power capping framework by Intel RAPL [20] and users can measure energy usage with it.

IV. Experiment Results

Table IV represents the average resource usages in our experiments. Since our scale-out cluster has 10 nodes, we can compare the disk usages of two clusters by multiplying the result from the scale-up cluster by 10. The entry values for the cpu usages are calculated as the average number of cores used per cluster (henceforth, cores per cluster). For example, *cores per cluster* of Wordcount on scale-up and scale-out are 9.4, 9.8, respectively.

Figure 2 shows that the execution time of each workload and the performance per watt ratio. Our experiment results showed that scale-up had better performance than scale-out in workloads that required heavy disk I/O such as Sort or Scan. For workloads that had an iterative algorithm, scale-out processed data more quickly than scale-up. Overall, scale-up had a much higher performance per watt ratio.

A. Micro Benchmarks

1) Wordcount: This workload is composed of 2 stages: a map stage and a result stage. Scale-out finished the result stage faster than scale-up, but the overall execution time was shorter for the latter due to the time save in the map stage. While the number of cores per cluster in the two were similar, with 9.4 and 9.8 respectively, scale-up had higher disk read throughput than scale-out (302.35MB/s, 246.95 MB/s). Therefore, we can conclude that the disk read throughput is a critical factor in performance.

2) Sort: Similar to Wordcount, Sort is composed of a map stage and a result stage. Since it is a write-intensive workload, there was a big performance gap between scale-up and scale-out. Particularly, this workload showed a significant difference of cores per cluster, with 7.8 and 4.8, respectively. Also, the disk usage of scale-out was measured to be almost twice that of scale-up. From these resource usage patterns, the performance ratio scale-up to scale-out turned out to be approximately 2.1.

B. Analytical Query

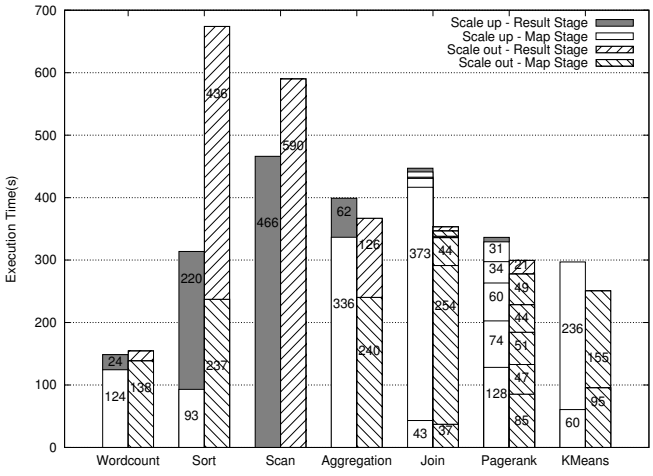
1) Scan: Scan is a single result stage which simply reads all data from the input file and writes it back to HDFS. This workload had the largest disk usage in our experiments. As a result, scale-up had better performance than scale-out as in Sort, but there were a few differences in the resource usage pattern. Since the main task of this workload is directly writing the output to HDFS on the same node, it had little network usage and thus no shuffle between tasks occurred. Therefore, the difference in results between the two clusters heavily depends on the disk throughput and Scan had the highest IO wait value among our workloads.

2) Aggregation: Aggregation is composed of a map stage and a result stages which processes a single table and saves the result to HDFS. Though it uses a similar stage structure to that of Wordcount, it showed different aspects in that scale-out had better performance than scale-up due to the time save in the map stage. One noteworthy feature of this workload is that it shows the advantage of having SSD over HDD very well. Figure 3 shows the disk throughput of each cluster. As seen in 3a, The result stage of Aggregation required disk bandwidth of more than 1GB/s in a moment of time and scale up could offer the speed while scale-out could not. Despite these differences, scale-out had better performance than scale-up because the map stage took a larger portion of the workload than the result stage.

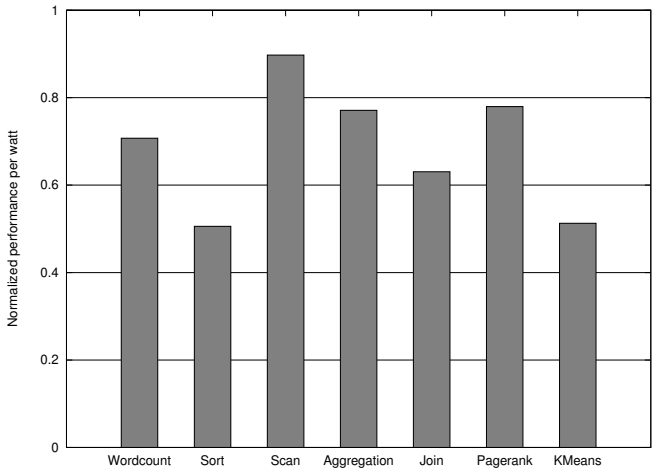
3) Join: Join workload executes a join operation on two tables. Two map stages transform the table information from sequence files into RDD and the rest process proper calculations and save them to HDFS. First two map stages had better performance in scale-out as Aggregation and since most of the

TABLE IV
AVG. RESOURCE USAGES IN OUR EXPERIMENTS

	Usr+Sys	Idle	Wait	Disk Read(MB/s)	Disk Write(MB/s)	Network(MB/s)
Wordcount						
Scale-up	77.98	21.42	0.34	302.35	16.97	-
Scale-out	26.13	51.59	21.97	26.028	3.867	2.47
Sort						
Scale-up	64.85	32.24	2.54	155.25	276.532	-
Scale-out	11.88	64.91	22.49	6.752	14.448	3.65
Scan						
Scale-up	85.52	10.56	3.27	162.88	273.03	-
Scale-out	22.96	39.17	35.91	12.375	21.793	0.32
Aggregation						
Scale-up	87.39	11.91	0.39	188.49	54.72	-
Scale-out	31.87	44.27	22.59	19.438	7.942	2.93
Join						
Scale-up	88.36	11.12	0.24	195.84	13.06	-
Scale-out	41.50	38.71	19.03	22.495	2.337	1.91
Pagerank						
Scale-up	85.02	14.88	0.08	8.57	22.66	-
Scale-out	31.57	63.77	2.89	0.923	4.027	3.40
KMeans						
Scale-up	81.87	17.85	0.16	124.61	4.85	-
Scale-out	31.82	54.97	11.34	13.03	1.95	1.20



(a) The Execution Time of Each Workload



(b) Normalized Performance per Watt Based on Scale-up

Fig. 2. Results of Our Experiments

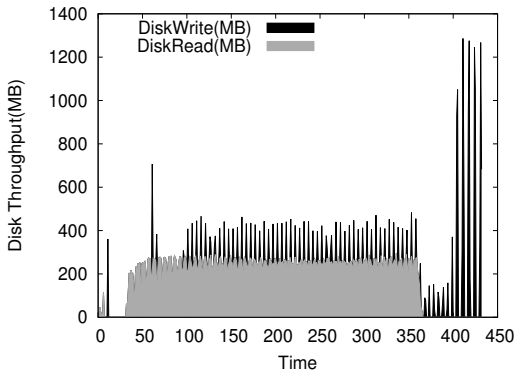
execution time for both clusters was spent in this step, scale-out had better performance than scale-up. The rest four stages cached the intermediate files in memory while processing, which led this workload to have higher cpu usage than others.

C. Web Search

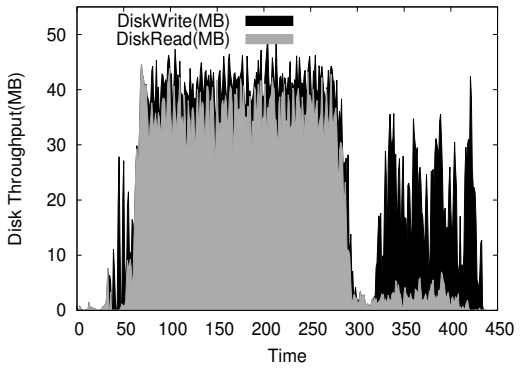
1) Pagerank: Pagerank is composed of 6 stages including two prepare stages, three iterative stages and one result stage because we set the iteration value to 3. The result of this workload was similar to that of Join; scale-out finished the workload more quickly than scale-up because scale-out processed two prepare stages faster, and these two stages took the majority of the execution time in this workload.

D. Machine Learning

1) KMeans: KMeans is an iterative algorithm and scale-out performed better than scale-up just like Pagerank. However, performance factors of this workload differ from those of Pagerank in the following two points: 1) Except for the first stage, both scale-up and scale-out could cache intermediate files created by each stage to memory. 2) due to 1), each stage (except for the first) did not perform any disk I/O but efficiently used cpu by directly accessing data from memory.



(a) Disk Usage of Aggregation in Scale-Up



(b) Disk Usage of Aggregation in Scale-Out

Fig. 3. Disk Usage of Each Cluster

As a result, scale-out could finish the rest 24 stages and hence, the entire task faster than scale-up.

E. Power Usage

Figure 2b showed the normalized performance per watt based on scale-up. Overall, scale-up had better performance per watt than scale-out in all workloads. In addition, rack unit of each node occupied in our experiments was 1U. Considering the physical space and energy consumption of each cluster, scale-up is more powerful in terms of physical aspects.

F. Summary

- Sort and Scan workload results show that scale-up can process faster than scale-out when handling write-intensive workloads.
- Aggregation workload result shows that cpus are not used efficiently since HDD cannot endure peak disk usage.
- Join, Pagerank and KMeans results show that scale-out has better performance than scale-up in workloads which use iterative algorithms.

V. Related Work

The comparison of scale-up and scale-out clustering has been studied for years and there have been numerous researches on choosing the optimal environment given specific conditions. Because Spark framework has replaced MapReduce as a main tool, many researchers studied consequences of the change.

Raja Appuswamy et al. [5] compared the characteristics of scale-up and scale-out cluster in Hadoop framework. This paper revealed that each cluster had advantages as well as disadvantages. The difference in our experiments that we selected RDD as data processing model instead of MapReduce. Our experiments showed that there were differences in results between MapReduce and RDD due to the characteristics of real world workloads and features of RDD.

Juwei Shi et al. [11] compared the performance of Spark and MapReduce frameworks and evaluated resource usage patterns of them. Researches on resource usage patterns can be a great help for users to understand primary differences of the two.

Michael Sevila et al. [13] suggested points worth to consider when comparing scale-up and scale-out clustering on MapReduce. However, this paper does not reflect real-world applications since the experiments only used micro benchmark.

Kamal Ke et al. [14] evaluated the performance of MapReduce using large-scale clusters composed of hundreds of Hadoop nodes. This paper showed that configurations appropriate for massive scale-out clusters depend on each application and that network bandwidth causes IO-intensive cases to have a decrease in performance.

Ahsan Javed Awan et al. [15] compared each real-world application in Spark framework on scale-up machines and measured resource usage patterns in a similar way to ours. Though it offered more detailed in-depth resource patterns of Spark application, our experiments differ from theirs in the following two points: 1) Hardware specifications of scale-out cluster are also provided, which enable a comparison on performance differences due to hardware changes. 2) Our experiments are conducted with a bigger disk bandwidth.

VI. Conclusion

In this paper, we evaluated in-memory computing by RDD in scale-up and scale-out clusters whose costs are comparable. In our experiments, scale-up executed write-intensive workloads like Sort and Scan more efficiently than scale-out, but workloads using iterative algorithms like Join, Pagerank and KMeans had better performance in scale-out because they effectively used cpu and memory by caching intermediate files to memory in each iteration. Furthermore, scale-up is more powerful than scale-out in terms of performance per watt and performance per physical space. In conclusion, when choosing between scale-up and scale-out clusters, users should consider the type of workload to be executed and the fact that scale-up has more physical advantages including lower power usage and less space occupancy.

Acknowledgment

This research was supported by the IITP (Institute for Information & Communications Technology Promotion) research assignment(R7117-16-0144).

REFERENCES

[1] Apache hadoop, http://hadoop.apache.org

[2] Apache spark, http://spark.apache.org

[3] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker and Ion Stoica, "Spark: Cluster Computing with Working Sets", in *2nd USENIX Workshop on Hot Topics in Cloud Computing* (HotCloud'10)

[4] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker and Ion Stoica, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing", in *9th USENIX Symposium on Networked Systems Design and Implementation* (NSDI'12)

[5] Raja Appuswamy, Christos Gkantsidis, Dushyanth Narayanan, Orion Hodson and Antony Rowstron, "Scale-up vs Scale-out for Hadoop: Time to rethink?", in *2013 ACM Symposium on Cloud Computing* (SoCC'13)

[6] Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi and Matei Zaharia, "Spark SQL: Relational Data Processing in Spark", in *SIGMOD'15*

[7] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed and Eric Baldeschwieler, "Apache Hadoop YARN: Yet Another Resource Negotiator", in *2013 ACM Symposim on Cloud Computing* (SoCC'13)

[8] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie and Bo Huang, "The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis", in *2010 IEEE 26th International Conference on Data Engineering Workshops* (ICDEW)

[9] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden and Michael Stonebraker, "A Comparison of Approaches to Large-Scale Data Analysis", in *SIGMOD'09*

[10] Jeffrey Dean, Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", in *4th Symposium on Operating Systems Design & Implementation* (OSDI'04)

[11] Juwei Shi, Yunjie Qiu, Umar Farooq Minhas, Limei Jiao, Chen Wang, Berthold Reinwald and Fatma Ozcan, "Clash of the Titans: MapReduce vs. Spark for Large Scale Data Analytics", in *Proceedings of the VLDB Endowment, Vol.8, No.13*, 2015

[12] Kay Ousterhout, Ryan Rasti, Sylvia Ratnasamy, Scott Shenker and Byung-Gon Chun, "Making Sense of Performance in Data Analytics Frameworks", in *12th USENIX Symposium on Networked Systems Design and Implementation* (NSDI'15)

[13] Michael Sevilla, Ike Nassi, Kleoni Ioannidou, Scott Brandt and Carlos Maltzahn, "A Framework for an In-depth Comparison of Scale-up and Scale-out", in *Proceedings of the 2013 International Workshop on Data-Intensive Scalable Computing* (DISCS'2013)

[14] Kamal Kc, Chin-Jung Hsu and Vincent W. Freeh, "Evaluation of MapReduce in a large cluster", in *2015 IEEE 8th International Conference on Cloud Computing* (CLOUD)

[15] Ahsan Javed Awan, Mats Brorsson, Vladimir Vlassov and Eduard Ayguade, "Performance Characterization of In-Memory Data Analytics on a Modern Cloud Server", in *The 5th IEEE International Conference on Big Data and Cloud Computing* (BDCloud 2015)

[16] Avraham Shinnar, David Cunningham, Vijay Saraswat and Benjamin Herta, "M3R: increased performance for in-memory Hadoop jobs", in *Proceedings of the VLDB Endowment Vol.5, No. 12*, 2012

[17] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker and Ion Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling", in *Proceedings of the 5th European conference on Computer systems* (EuroSys'10), 2010

[18] Vincent M. Weaver, Matt Johnson, Kiran Kasichayanula, James Ralph, Piotr Luszczek, Dan Terpstra and Shirley Moore, "Measuring Energy and Power with PAPI", in *2012 41st International Conference on Parallel Processing Workshops* (ICPPW)

[19] Running Average Power Limit - RAPL, https://01.org/blogs/tlcounts/2014/running-average-power-limit-%E2%80%93-rapl

[20] Power Capping Framework, https://www.kernel.org/doc/Documentation/power/powercap/powercap.txt