MapReduce: State-of-the-Art and Research Directions

Abdelrahman Elsayed, Osama Ismail, and Mohamed E. El-Sharkawi

***Abstract*—Digital data that come from different applications such as, wireless sensor, bioinformatics next generation sequencing, and high throughput instruments are growing in high rate. Dealing with demands of analysis of ever-growing data requires new techniques in software, hardware, and algorithms. MapReduce is a programming model initiated by Google's Team for processing huge datasets in distributed systems; it helps programmers to write programs that process big data. The aim of this paper is to investigate MapReduce research trends, and current research efforts for enhancing MapReduce performance and capabilities. This Study concluded that the research directions of MapReduce concerned with either enhancing MapReduce programming model or adopting MapReduce for deploying existing algorithm to run with MapReduce programming model.**

***Index Terms*—Big data, distributed computing, programming model, scalability.**

I. Introduction and Motivation

Applications in different domains such as bioinformatics, weather forecasting, environmental studies, and fraud detection deal with intensive data. Computing technology must scale to handle the huge volume of data. MapReduce is a programming model aims to deal with data intensive applications. MapReduce enables programmers who have no experience with distributed systems to write applications that process huge datasets in large cluster of commodity machines; it manages data partitioning, task scheduling, and nodes failure [1].

The motivation of this study is:

1) Introduce an overview of MapReduce.
2) Determine strengths, weakness of MapReduce, and research trends of MapReduce.
3) Provide overview of enhancements efforts for MapReduce.
4) Finally, determine the open issues as future research directions.

The rest of paper is organized as follow: Section II introduces a background of MapReduce. Section III lists MapReduce capabilities, limitations, and Research trends. Section IV lists MapReduce enhancements. Section V lists MapReduce Future Research Directions. Finally, section VI provides a conclusion.

II. MapReduce Background

Processing huge volume of data requires distributing data into hundreds or thousands of nodes in order to finalize processing task in short reasonable time. Google' team developed MapReduce to automatically parallelize computation; MapReduce manages data partitioning and distribute it among computation nodes. In addition, it handles node failures. MapReduce helps programmers to concentrate in writing programs that process large data; programmers concentrate on the problem details and MapReduce manage distributed computation issues. Apache Hadoop is an open source implementation of MapReduce [2]. Most of programmers and researcher outside Google use Hadoop in their experiments.

Programmers are restricted to formulate the computation in two functions: Map and Reduce. The input of Map function must be represented as key/value pair; for example, the map function that processes a set of document to compute word counts takes a document as a key and the content of the document as a value. Map function produces a set of intermediate key/values. In case of word count the intermediate keys will be individual words and the value will be the number of occurrences of these words. The input of Reduce function is the output of Map function (intermediate key/value pairs); it processes these values and output it in files; these files could be used for other MapReduce iterations.

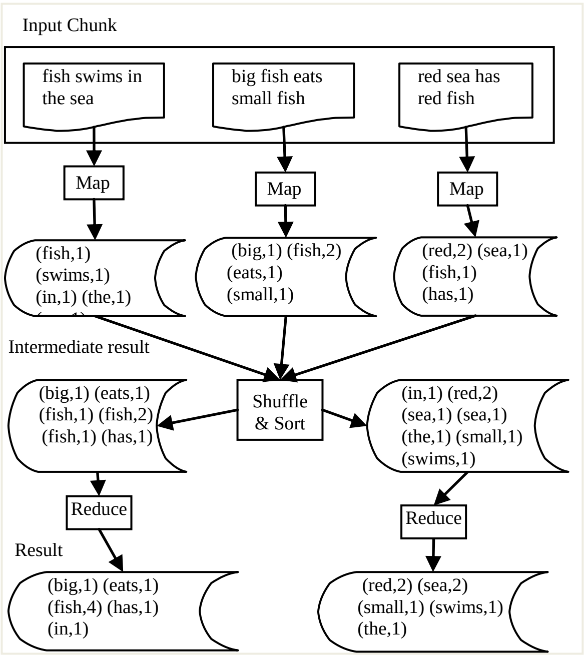
Input Chunk
fish swims in the sea
big fish eats small fish
red sea has red fish
Map
Map
Map
(fish,1) (swims,1) (in,1) (the,1)
(big,1) (fish,2) (eats,1) (small,1)
(red,2) (sea,1) (fish,1) (has,1)
Intermediate result
(big,1) (eats,1) (fish,1) (fish,2) (fish,1) (has,1)
Shuffle & Sort
(in,1) (red,2) (sea,1) (sea,1) (the,1) (small,1) (swims,1)
Reduce
Reduce
Result
(big,1) (eats,1) (fish,4) (has,1) (in,1)
(red,2) (sea,2) (small,1) (swims,1) (the,1)

Fig. 1. MapReduce processes for counting the number of occurrences for each word in a collection of documents

MapReduce efficiently uses network bandwidth by moving computation to data. The input data is managed by Google File System (GFS) [3]; GFS divides input data into a

Manuscript received May 16, 2013; revised July 18, 2013.

A. Elsayed is with Central Laboratory for Agriculture Expert Systems, Agriculture Research Center, Cairo, Egypt (e-mail: master_abdo@yahoo.com)

O. Ismail and Mohamed E. El-Sharkawi are with the Faculty of Computers and Information, Cairo University, Cairo, Egypt (e-mail: Osama@fci-cu.edu.eg, m.elsharkawi@fci-cu.edu.eg).

set of blocks; the block size can be specified by users. GFS replicate each block; the default replication number is three. GFS puts one replica in the same rack and puts the other replica in other rack. The strategy of replica distribution helps in restoring data in case of node or rack failures. MapReduce take replication information into account, and utilizes it on achieving data locality.

MapReduce runs in cluster of nodes; one node acts as a master node and other nodes act as workers. Workers nodes are responsible for running map and reduce tasks; the master is responsible for assigning tasks to idle workers. Each map worker reads the content of its associated split and extracts key/value pairs and passes it to the user defined Map function. The output of Map function is buffered in memory and partitioned into a set of partitions equals to number of reducers.

Master notifies the reduce workers to read the data from local disks of map workers. The result or output of reduce function is appended to output files. Users may use these files as input to another MapReduce call, or use them for another distributed application.

A. MapReduce Example

If we want to count the number of occurrences for each word in a collection of documents, the map function input will be a repository of documents, each document represents a record. The input key for map function will be a document Id, and the value will be string represents document content. The Map function will split each document into a sequence of words w1, w2, w3,…, wn. It then emits a list of key-value pairs. Each word will represent key and the value is always 1. In this example, the addition is associative and commutative operation; so combiner function can be used to aggregate all repeated words that occur in the same document [4].

After finishing the Map phase, Shuffle & Sort phase is start. It takes the intermediate data generated by each Map task, sorts this data with intermediate data from other nodes, divides this data into regions to be processed by the reduce tasks. Finally the Reduce function takes keys and a list of values. For example, given word count example that it is displayed in Fig.1; it takes a fish as key and a list of values {1, 2, 1}. The final result can be collected into one file that contains each word associated with its frequency [5].

B. Dealing with Failure

MapReduce is designed to deal with hundreds or thousands of commodity machines. Therefore, it must tolerate machine failure. The failure may be occur in master node or worker nodes. In case of master failure all MapReduce task will be aborted, and it have to be redone after assigning new master node.

On the other hand, to track worker failure, the master monitors all workers by periodically checking worker status. If a worker doesn't respond to master ping in a certain amount of time, the master marks the worker as failed. In case of failure of map task worker; any map tasks either in progress or completed by the worker are reset back to their initial idle state, and will be assigned to other worker. While in case of failure in reduce task worker, any task in progress on a failed worker is assigned to idle worker. The output of completed reduce tasks is stored in global file system, so completed reduce tasks do not need to be re-executed. In the other hand, the output of map tasks is stored in local disks, so completed map tasks must be re-executed in case of failure.

C. Dealing with Straggler

Straggler is a machine that takes an unusually long time to complete its map or reduce task. Stragglers occur due to several reasons such as, a machine with bad disk. MapReduce have a general mechanism to alleviate straggler problem. The master schedules backup executions for the MapReduce operation that is close to complete. Whenever the primary or backup task execution is completed, the task is marked as completed.

D. MapReduce Applications

Several MapReduce applications have been implemented and executed on Google’s clusters for example, processing crawled documents, web request logs in order to compute various kinds of derived data, such as inverted indices [1]. In addition, MapReduce has been applied for machine learning task, scientific simulation and large scale image processing tasks [6].

Cho, et al proposed a framework for opinion mining in MapReduce and word map. Opinion Mining is a technique for extracting estimation from the internet. It is known as sentiment classification. It reads a text, analyzes it and produces a result like <word | value>, and <word | value>, which is similar to MapReduce data structure. Therefore, it is possible to match it well within MapReduce [7].

Cordeiro, et al used MapReduce for clustering very large moderate-to-high dimensionality dataset [8]. Also Niemenmaa, et al developed a library called Hadoop-BAM for manipulation of aligned next-generation sequencing data in scalable way in Hadoop framework [9].

Further, Chandar used MapReduce for performing join task [10]. Chandar classifies joins algorithms into two categories with three algorithms for each category as follow:

- Two-way joins where joins involving only 2 datasets. The two way join is classified into Map-Side join, Reduce-Side join, and broadcast join.
- Multi-way joins where joins involving more than 2 datasets, and it is classified into : Map-Side Join (for multi-way joins), Reduce-Side One-Shot Join, Reduce-Side Cascade Join

Chandar found that for two-way join Map-Side joins performs well when the key distribution in the dataset is uniformly distributed. In case of one of the datasets is much smaller and it can be easily replicated across all machines the broadcast join is the best option.

E. Classification of MapReduce Algorithms

Srirama, et al studied adapting scientific computing to the MapReduce programming model [11]. They classify algorithm into four classes.

- The first class represents algorithms that can be adapted as a single execution of MapReduce models; factoring integers is an example of this type of algorithms.
- The second class represents algorithms that can be adapted as a sequential execution of a constant

number of MapReduce models; Clustering Large Application (CLARA) [12] is an example of this class of algorithms.

- The third class represents iterative algorithms; where each iteration is represented as an execution of a single MapReduce model; clustering algorithms such as Partitioning Around Medoids (PAM) [12] represents an example of this type of algorithms.
- The fourth class of algorithms represents complex iterative algorithms where the content of one iteration is represented as an execution of multiple MapReduce models. Conjugate Gradient (CG) [13] is an example of this type of algorithms.

III. MapReduce Capabilities, Limitations, and Research Trends

MapReduce has many advantages. First it supports data locality by collocating the data with the compute node; so it reduces network communication cost [2]. Also it support scalability; the runtime scheduling strategy enables MapReduce to offer elastic scalability which means the ability of dynamically adjusting resources during job execution. In addition, it has a fault tolerance strategy that transparently handles failure; it detects map and reduce tasks of failed nodes and reassigns it to other nodes in the cluster. Also it has the ability to handle data for heterogeneous system, since MapReduce is storage independent, and it can analyze data stored in different storage system [14].

In contrast to these positive features MapReduce has some limitations. Srirama, et al in their study clarify that, Hadoop MapReduce framework has some problems with iterative algorithms because each iteration spent some time in background task regardless of the input size. In addition, the input of Hadoop has to be read from the HDFS for every time [11]. Researchers investigate how to handle limitation of MapReduce in implementing iterative algorithms. Haloop [15], iHadoop [16], iMapReduce [17], CloudClustering [18], and Twister [19] are examples of enhancing Hadoop to support efficient iterative algorithms.

Also MapReduce doesn't have expressiveness of popular query languages like SQL; this leads users to spend more time in writing programs for simple analysis. So there is a need for enhancing MapReduce query capabilities [20]. Other limitation of MapReduce, it doesn't have the capabilities of collocation related data on the same nodes. Collocation of data is useful for improving the efficiency of many operations such as joins, indexing, grouping, and columnar storage [21]. In addition to the above limitations it doesn’t support interactive data analysis since it doesn't emit any output until the completion of the job [22].

Research in MapReduce has two directions: the first direction focuses on enhancing MapReduce performance by handling its limitations such as collocation of related data, implementing efficient iterative algorithms, and managing skew of data. The second direction focuses on adapting current algorithms and application to run on MapReduce environments; as we will show in the next sections.

IV. MapReduce Enhancements

Different techniques and efforts initiated in order to get the maximum benefits of MapReduce programming model. Researchers from database communities compared the performance of MapReduce and the parallel database systems, and they try to integrate between them by enabling the features of database and parallel database to MapReduce and vice versa [23]. In this section, we will discuss different MapReduce enhancement techniques in brief.

A. Integrating Database with MapReduce

Abouzeid, et al developed HadoopDB which is a hybrid of MapReduce and DBMS technologies [24]. It has the performance and efficiency of parallel databases such as; sql interface, query optimization. In addition, it inherits scalability and fault tolerance of MapReduce-based systems. HadoopDB uses MapReduce as task coordinator and network communication layer to connect multiple single-node database systems. HadoopDB uses scheduling and job tracking implementation of Hadoop to operate in heterogonous system, and achieve fault tolerance. Performance of parallel database is achieved by doing much of the query processing inside of the database engine. The main drawbacks of HadoopDB are enforcing users to install database management systems, and changing the interface to SQL which contradict with simplicity of MapReduce programming model [25].

Dryad is a Microsoft project. It has similar aims of MapReduce which is facilitate and help developers to write efficient parallel and distributed applications [26]. DryadLINQ [27] integrates between Dryad, and .NET Language Integrated Query LINQ. It supports declarative programming. DryadLINQ translates the data-parallel portions of the program automatically and transparently into a distributed execution plan which is passed to the Dryad execution platform.

B. Integrating Indexing Capabilities to MapReduce

Hadoop++ aims to reach the performance of DBMSs without changing the underlying Hadoop framework [25]. Hadoop++ enhances the performance of Hadoop by injecting indexing and joining capabilities into Hadoop. Hadoop++ proposed Trojan Index to integrate indexing capabilities into Hadoop. The core idea of Trojan Index: is a covering index will be built for each split of data (SData T), and a header (H) is added that contains indexed data size, index size, first key, last key and number of records. Finally, to identify the split boundary, a split footer (F) is used as shown in Fig. 2.

Hadoop++ also proposed Trojan Join to support more effective join processing in Hadoop. Trojan Join applies the same partitioning function on the join attributes for related relations at the load time. Hence, it places the co-group pairs that have the same join key of related relations on the same split. This leads to processing join locally at query time.

Hadoop++ affects Hadoop from inside. So, future changes of Hadoop could be directly used by Hadoop++. In addition, it does not change the interface of the Hadoop.

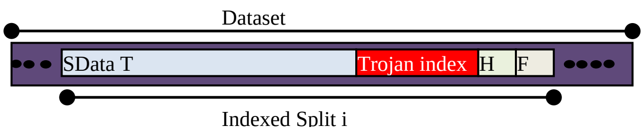

Fig. 2. Indexed data layout in Hadoop++ [Dittrich, et al, 2010].

C. Empower Hadoop by Careful Data Organization

Many research efforts tried to enhance the performance of MapReduce by optimizing structure of data storage. Column-oriented storage techniques [28] and extension of Hadoop such as, Manimal [29], and CoHadoop [21] try to optimize data storage of related data in the same nodes.

Floratou, et al described how to improve the performance of Hadoop by adopting column oriented storage techniques. In addition, they showed that using binary storage formats in Hadoop can provide 3X performance over the native use of text files [28]. The cost of using column storage technique occurred only during loading time of data. Using column oriented storage technique does not change the core of Hadoop. In the other hand column oriented storage technique integrated with lazy record construction technique helps in loading only those columns of records that are actually accessed in a Map phase.

Jahani *et al*. developed a system called Manimal; it extends MapReduce in order to be able to analyze MapReduce programs automatically and applies data aware optimization in appropriate way. It tries to enhance the performance of the MapReduce without losing the appealing of MapReduce programming model. It focuses on programs that have relational style. It uses B+ tree to scan only portion of data that is relevant to program inputs. It analyzes program code to find which fields is used and leverage column-store approach to load files that contain only used fields [29].

CoHadoop [21] is an extension of Hadoop to enable applications to control where data are stored. CoHadoop empowered Hadoop by enabling the capabilities of colocating related data on the same set of nodes. It retains the flexibility of Hadoop since it does not require users to convert their data to a certain format such as a relational database or a specific file format. Instead applications will give hints to CoHadoop that some set of files are related and may be processed jointly.

Achieving collocation obtained by extending Hadoop Distributed File System (HDFS) with a new file level property called locator. In addition, CoHadoop modifies Hadoop's data placement policy in order to make use of this locator property. Locator always represented by integer value, and other data types can be used. In HDFS, each file will be assigned to at most one locator, but many files can be assigned to the same locator. Files that use the same locator placed on the same set of data nodes. Colocation is applied to all blocks including replicas. Fig. 3 displays an example of colocating two files A and B via a common locator. All of A's two HDFS blocks and B's three blocks are stored on the same set of data nodes.

CoHadoop introduced locator table data structure into the named node of HDFS, to manage the locator information and keep track of collocated files. The mappings between locators to the list of files that share these locators are stored in the locator-table.

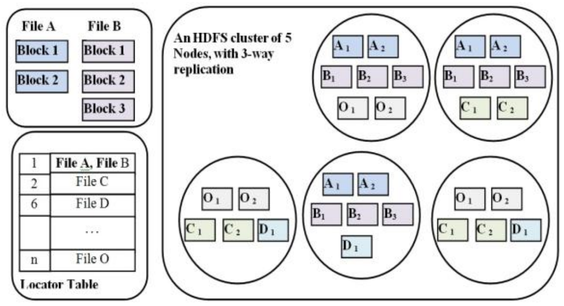

Fig. 3. Example of file colocation in CoHadoop[Eltabakh, et al, 2011]

D. Enrichment MapReduce with Data Warehouse Capabilities

Hive is data warehousing solution built on top of Hadoop. Hive is similar to database in storing data in tables and using primitive data types; it supports queries expressed in a SQL-like declarative language. Hive uses a query language called HiveQL. Queries are compiled into map-reduce jobs to be executed on Hadoop. HiveQL supports custom map-reduce scripts to be plugged into queries. Hive provides a system catalog to persist metadata about tables within the system; this enables Hive to function as a traditional warehouse which can interface with any standard reporting tools. Facebook used Hive in order to build its warehouse. In addition, Facebook warehouse used Hive extensively for both reporting and ad-hoc analyses by more than 100 users [20].

Cheetah is another data warehouse system built on top of Hadoop; it differs from Hive because it is designed specifically for online advertising application. It adds custom features and optimizations to query execution and schema design for advertising applications. Cheetah allows MapReduce programs to take advantages of MapReduce and data warehouse [30].

E. Skew Management in MapReduce

MapReduce is more suitable to deal with data intensive application in distributed computing. But dealing with distributed data requires efficient load balancing to partition data between all computation nodes in uniform way. In some cases skew may be occur in MapReduce. Skew in general is a result of unfairness tasks distribution in distributed environment which leads to delaying all missions or jobs.

In MapReduce Skew happens when there is one node has assigned data to be processed more than others either in Map or Reduce Stage. Also skew may happen if some records require more execution time than others. Skew leads to the situation where some nodes take longer time than other, the default MapReduce deals with this situation by speculative execution strategy. This strategy is not efficient in this case because the speculative nodes will take the same time to execute the same task [31]. In the following we will brief the efforts that try to decrease the skew of data in MapReduce.

Kwon *et al*. [32] developed a system called SkewReduce on top of Hadoop. SkewReduce has a static optimizer that depends on user-supplied cost functions to determine the best way to partition the input data to reduce computational skew.

Gufler *et al*. [33] exploit distributed database partitioning approaches to mitigate skew in MapReduce. They propose two load balancing approaches, The first approach is fine partitioning, which splits the input data into a fixed number of partitions (p), which will be greater than number of reducers (r), in contrast to MapReduce systems where p = r. This retains some degree of freedom for balancing the load on the reducers. Load balance is achieved by distributing expensive partitions to different reducers. The second approach is dynamic fragmentation; in this approach each mapper split large partitions which exceed average partition size by predefined factor into fragments. This approach leads to more uniform and easier load balancing for highly skewed distributions.

SkewTune is a system addresses skew in both the map and reduce phase [34]. SkewTune mitigates skew at runtime by executing the following three steps repeatedly.

1) The first step is Detect; during the execution of MapReduce phase's coordinator observes all tasks and collects their time remaining estimates; in case it found slot is idle, the coordinator identifies the most stragglers that have longest remaining time estimates.
2) The second step is scan; the coordinator stops the straggler task, after that the unprocessed data is scanned to collect information for repartitioning.
3) The third step is Plan; according to repatriating information, and the remaining estimated time of all running tasks the coordinator plan how to reparation the remaining data. After that it repeats again the detect phase and go on.

The approach of SkewTune needs to be explained more for task that has remaining data for the same key group task and it needs more investigations.

F. Enhancing Power Consumption for MapReduce

There are many approaches for cluster energy optimization; these approaches try to save electricity by efficient utilization of power resources within cluster. In the following, we discuss three different approaches for enhancing power consumption.

1) Covering set (CS)

It is a method for cluster energy management. It exploits the replication feature of distributed file systems (DFS). In CS strategy some nodes in the system is specified as special nodes, called CS nodes. At least one copy of each unique data block is kept in these nodes. To save energy during periods of low utilization; CS strategy can power down some or all of the non-CS nodes.

The drawback of CS is the workloads take longer to run when the cluster has been partially powered down, since fewer nodes are available to run workload. Other issues it requires code modifications in the underlying DFS software [35], [36].

2) All-In Strategy (AIS)

It is an alternative cluster energy management to CS strategy. It aims to decrease response time of workload by running the workload on all the available nodes in the cluster. During low utilization period, AIS transitions the cluster to a low power state. According to [36], the experiment results show that in many cases, AIS has energy consumption lower than CS strategy. CS strategy is more effective than AIS only when the computational complexity of the workload is low, and the time it takes to transition a node to and from a low power state is relatively large fraction of the overall workload time.

3) Berkeley energy efficient MapReduce (BEEMR)

BEEMR Motivated by empirical analysis of real-life MapReduce interactive analysis at Facebook. The idea of BEEMR developed in observation that the interactive jobs operate on a small fraction of data; so small pool of machines can be dedicated to these jobs; on the other hand the less time-sensitive jobs can run on the rest of the cluster in a batch way. BEEMR is differ from Covering Set strategy by keeping all dedicated machines for interactive analysis run full power at all time, and it handles distributed file system fault tolerance using error correction codes rather than replication; error correcting codes cannot be applied in case of Covering set. The all in strategy is difficult to be applied in case of MapReduce interactive data analysis, since the cluster is never to be completely inactive [37].

V. MapReduce Future Research Directions

MapReduce is initiated by Google to deal with unstructured data such as web documents. There are needs for researches that focus on rethinking of MapReduce to deal with structured data and stream data [22].

According to this study the research direction of MapReduce can be divided into two directions, first direction concerns enhancing MapReduce programming model, for example current MapReduce methods for handling skew of data still need more investigation in order to tackle key group load imbalance, where majority of data are assigned to few keys. Another enhancement to MapReduce is tracing nodes statistics to discover specific nodes that delay execution time of jobs. This delay may be caused by load imbalance or hardware faults.

The second research direction concerns by converting existing algorithms of different domains to run in MapReduce. This can be done by programmers that are restricted to express the algorithm in map and reduce functions so the future research aims to modify the current algorithms to be applicable for MapReduce. On the other hand another research trend is developing a layer on MapReduce that convert current algorithms automatically or semi automatically to be suitable to MapReduce programming model. Also researchers can investigate new algorithms that take into consideration MapReduce restrictions.

VI. Conclusion

MapReduce has been invented by Google to deal with huge volume of data. In this paper we introduced overview about MapReduce programming model. We presented MapReduce capabilities, limitations, and current research efforts that aim to enrich MapReduce to manipulate with its limitations. Further, we illustrated how performance of MapReduce can be enhanced by managing skew of data. In addition, we introduced current power enhancement methodologies for clusters that run MapReduce jobs. Finally we discussed future research directions of MapReduce.

REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *Proc. the 6th conference on Symposium on Operating Systems Design & Implementation*, 2004, pp. 137-150.

[2] T. White, *Hadoop: The Definitive guide*, 1st ed.: O'Reilly Media, 2010.

[3] S. Ghemawat, H. Gobioff, and S. T. Leung, "The Google file system," in *Proc. the nineteenth ACM symposium on Operating systems principles*, 2003, pp. 29-43.

[4] A. Rajaraman and J. Ullman, *Mining of Massive Datasets*, Cambridge University Press, 2011, ch. 2.

[5] A. M. Middleton, *Data-Intensive Technologies for Cloud Computing.* B. Furht and A. Escalante, *Handbook of Cloud Computing*, New York: Springer, 2010, ch. 5.

[6] S. Chen and S. W. Schlosser, "Map-Reduce Meets Wider Varieties of Applications," *Intel Research Pittsburgh*, IRP-TR-08-05, 2008.

[7] K. S. Cho *et al*., "Opinion Mining in MapReduce Framework," *Secure and Trust Computing, Data Management, and Applications*, 2011, pp. 50-55.

[8] R. Cordeiro *et al*., "Clustering Very Large Multi-dimensional Datasets with MapReduce," in *Proc. the International Conference on Knowledge Discovery and Data Mining*, 2011, pp. 690-698.

[9] M. Niemenmaa, A. Kallio, A. Schumacher, E. Klemela, and K. Heljanko, "Hadoop-BAM: directly manipulating next generation sequencing data in the cloud," *Oxford Journals, Bioinformatics*, vol. 28, no. 6, pp. 876-877, 2012.

[10] J. Chandar, "Join Algorithms using Map/Reduce," M.S. thesis, School of Informatics, University of Edinburgh, United Kingdom 2010.

[11] N. S. Srirama, P. Jakovits, and E. Vainikko, "Adapting scientific computing problems to clouds using MapReduce," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 184-192, 2012.

[12] L. Kaufman and P. Rousseeuw, *Finding Groups in Data An Introduction to Cluster Analysis*. New York: Wiley Interscience, 1990.

[13] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," Technical Report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, 1994.

[14] D. Jiang, B. Ooi, L. Shi, and S. Wu, "The Performance of MapReduce: An In-depth Study," *PVLDB*, vol. 3, no. 1, pp. 472-483, 2010.

[15] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "HaLoop: Efficient Iterative Data Processing on Large Clusters," *PVLDB*, vol. 3, no. 1, pp. 285--296, 2010.

[16] E. Elnikety, T. Elsayed, and H. E. Ramadan," iHadoop: Asynchronous Iterations for MapReduce," in *Proc. the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, 2011, pp. 81- 90.

[17] Y. Zhang, Q. Gao, L. Gao, and C. Wang, "iMapReduce: A Distributed Computing Framework for Iterative Computation," in *Proc. the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, 2011, pp. 1112-1121.

[18] A. Dave, W. Lu, J. Jackson, and R. Barga, "CloudClustering: Toward an iterative data processing pattern on the cloud," in *Proc. the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, 2011, pp. 1132-1137.

[19] J. Ekanayake *et al*., "Twister: A Runtime for Iterative MapReduce," in *Proc. the 19th ACM International Symposium on High Performance Distributed Computing*, 2010, pp. 810-818.

[20] A. Thusoo *et al*., "Hive – A Petabyte Scale Data Warehouse Using Hadoop," in *Proc. 26th IEEE International Conference on Data Engineering*, Long Beach, California, 2010, pp. 996-1005.

[21] M. Eltabakh, Y. Tian, F. Gemulla, A. Krettek, and J. McPherson, "CoHadoop: Flexible Data Placement and Its Exploitation in Hadoop," *PVLDB*, vol. 4, no. 9, pp. 575-585, 2011.

[22] T. Condie *et al*., "MapReduce Online," in *Proc. the 7th USENIX Conference on Networked Systems Design and Implementation*, SWan Jose, California, 2010, pp. 313-327.

[23] M. Stonebraker *et al*., "MapReduce and Parallel DBMSs: Friends or Foes ?" *Communications of the ACM*, vol. 53, no. 1, pp. 64-71, 2010.

[24] A. Abouzeid, K. Bajda Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin, "HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads," *PVLDB*, vol. 2, no. 1, pp. 922-933, 2009.

[25] J. Dittrich et al., "Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing)," *PVLDB*, vol. 3, no. 1, pp. 518-529, 2010.

[26] M. Isard, M. Budiu, and Y. Yu, "Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*, 2007, pp. 59-72.

[27] Y. Yu *et al*., "DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language," in *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, San Diego, 2008, pp. 1-14.

[28] A. Floratou, J. Patel, E. Shekita, and S. Tata, "Column Oriented Storage Techniques for MapReduce," *PVLDB*, vol. 4, no. 7, pp. 419-429, 2011.

[29] E. Jahani, M. Cafarella, and C. R é, "Automatic Optimization for MapReduce Programs," *PVLDB*, vol. 4, no. 6, pp. 385-396, 2010.

[30] S. Chen, "Cheetah: a high performance, custom data warehouse on top of MapReduce," *PVLDB*, vol. 3, no. 2, pp. 1459-1468, 2010.

[31] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "A study of skew in mapreduce applications," presented in the 5th Open Cirrus Summit, 2011.

[32] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "Skew-resistant parallel processing of feature-extracting scientific user-defined functions," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, 2010, pp. 75-86.

[33] B. Gufler, N. Augsten, A. Reiser, and A. Kemper, "Handing Data Skew in MapReduce," in *Proceedings of The First International Conference on Cloud Computing and Services Science*, 2011, pp. 574-583.

[34] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "SkewTune in action: mitigating skew in MapReduce applications," *PVLDB*, vol. 5, no. 12, pp. 1934-1937, 2012.

[35] J. Leverich and C. Kozyrakis, "On the Energy (In) efficiency of Hadoop Clusters," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 1, pp. 61-65, 2010.

[36] W. Lang and J. M. Patel, "Energy Management for MapReduce Clusters," *PVLDB*, vol. 3, no. 1, pp. 129-139, 2010.

[37] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz, "Energy efficiency for large-scale mapreduce workloads with significant interactive analysis," in *Proc. the 7th ACM european Conference on Computer Systems*, Bern, Switzerland, 2012, pp. 43-56.

Abdelrahman Elsayed was born in Egypt 1977; he received his master and BSc in 2008 and 2000 resp. from Information System Dept., Faculty of Computers and Information, Cairo University, Egypt. His research interests are data mining and data intensive algorithms. Now he is working as Research assistant at Central Laboratory for agriculture expert systems. Mr. Elsayed is member of the IACSIT.

Osama Ismail was born in Egypt 1969; He received a MSc. degree in Computer Science and Information from Cairo University in 1997 and a PhD degree in Computer Science from Cairo University in 2008. He is a Lecturer in Faculty of Computers and Information, Cairo University. His fields of interest include, Cloud Computing, Multi-agent Systems, Service Oriented Architectures, and Data Mining.

Mohamed E. El-Sharkawi was born in Egypt; he is currently a professor in Faculty of Computers and Information, Cairo University. He received his Doctor of Engineering and Master of Engineering in 1991 and 1988 resp. in Computer Science and Communication Engineering from Kyushu University, Fukuoka, Japan. He received his BSc. in Systems and Computer Engineering from the Faculty of Engineering, Al-Azhar University, Cairo, Egypt in 1981. His research interests are database systems, query processing and optimization, social networks, and data mining.