Improving Map Reduce Performance using LATE scheduling in big data

Suyash Mishra
Computer sciences
Noida International University
Greater Noida, India

Dr Anuranjan Misra
Computer sciences
Noida International University
Greater Noida, India

Dr Suryakant yadav
Computer sciences
Noida International University
Greater Noida, India.

***Abstract*—The volume of data produced and used in today's world has been growing very rapidly which has forced industry to explore data for better business decision making and incase profit by focusing more on customer needs. MapReduce is the core-processing engine of Hadoop, which accommodate rapidly increasing demands on computing resources required by massive data sets. Highly scalability of the MapReduce processing, allows parallel and distributed processing on multiple computing. Normally Hadoop implementation considers that underline cluster nodes are same in computing ability, configurations and storage. However, this homogeneity of cluster is not necessary every case and performance of MapReduce degrades and suffers due to various limitations in heterogeneous environments where underline nodes have different capability. This paper talks about how in heterogeneous environment, LATE (Longest Approximate Time to End) scheduling performs better and efficiently in comparison to other scheduling. LATE can improve Hadoop response times by almost two times in a clusters.**

***Keywords*—Big data, HDFS, Hadoop, Map-Reduce, Scheduling Algorithm, LATE.**

I. Introduction

Traditional data storage and processing capabilities were limited and was dependent on underline hardware configurations, storage and processing requirements, which deemed to be very different from today. Due to increase in data, volume and unstructured data traditional methods and databases are facing huge threat to ingest and process Big Data processing and storing demands.
Now a days Industry is focusing and making huge investment to conclude how to make better use of Big Data and identify beneficial business insights to lead a better business decisions .Which help business or industry to increase profit. MapReduce is a highly scalable programming model able to process huge volume of data in parallel execution fashion on a huge number of commodity computing nodes. Google [3] has developed MapReduce paradigm was later used and implemented in many open source projects, but the Apache Hadoop harnessed processing most efficiently.

Now a days Industry is focusing and curious to know about efficiently utilize Big Data and analyze to identify beneficial business insights for making them quipped with viable and better business decisions .Which in turn add value to their business ,increase profit ,retain customers and make customers happy with services they are offered.
Advancement in the technology has also brought flood of unstructured data. Analysis of this data is important to extract value from the abundance of data available. The rate at which the data is growing is very high and unpredictable. Businesses can use this data for multiple purpose. This data is further utilized for conducting customer trend analysis, customer's sentiment analysis or to know customer feedback on service or product Company is offering. Which can be further converted into structured data-by-data analyst as per their convenience.

Big data is comprised of huge volume of structured, semi-structured and unstructured data that used for data analytics using various methods.

II. HADOOP

Hadoop is an open source-processing engine designed to compute extremely large unstructured or semi structured datasets efficiently. Hadoop follows distributed processing mechanism, which offers resilience and scalability while processing big data. Primarily Hadoop has two major constituents HDFS and MapReduce. HDFS (Hadoop distributed file system) manages data inspired from UNIX file system responsible for storage management in structured relational form or unstructured form and in any form in between similarly. HDFS is a highly distributed file system ensures highly computation of big data along with scalable facility. The MapReduce programing model manages applications on multiple distributed servers or user related processing task in an efficient manner. This provides an environment to execute a highly efficient data processing by harnessing distributed processing feature, which is based on divide and aggregate paradigm. HDFS gives liberty to user or application programmer to capitalize unlimited storage need. Figure1 below depicts both prime components of Hadoop framework.

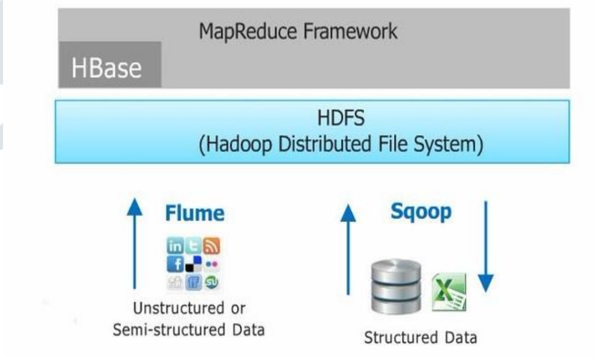

Fig. 1. Hadoop compnents

III. PROCESSING OF BIGDATA USING MAPREDUCE

In the MapReduce paradigm, MapReduce [2] has become the prominent batch-processing model. MapReduce is a very flexible programming method able to parallel process large volume efficiently on large number of computing nodes.
MapReduce methodology consisting of map and reduce functions, which allows application developers to customize them as per their requirement. Along with providing faster processing Map Reduce manages node failure and abstracting design complexity from the programmer.
MapReduce has been developed for effective parallel processing of big data by breaking the load into independent

sub units. Prominent benefit of MapReduce is that it conceals system level implementations and complexities from developer by allowing developer just to focus on its objective. MapReduce is a two-phase approach: Map and Reduce. Every Map phase takes input files present into various nodes of distributed file system. If Map function does, co- located with data partition system will attempt to move data portion to node where Map function exists with data portion to reduce data movement. Hence MapReduce backs concept of as “Moving data closer to compute” to optimize execution time. Post Map function finishes its processing, reduce function is then further run on all values having same interim key value and output key/value pairs as the result. The MapReduce framework based on master/slave architecture having a one node acting as master has Job Tracker and several slave nodes, which runs assigned task, and Tasktrackers, one per node in the cluster, maintain their status. The Job Tracker acts as communicator between users who submits the job and the underline pool of hardware. Job Tracker accepts users assigned map/reduce jobs to the and further jobs are executed on sequence of submission first come/first-served basis. The Job Tracker does resource management required for map, reduce steps, allocated, and monitor slave nodes executing allocated task via tasktrackers. The Tasktrackers accepts and run tasks allocated by JobTracker, manages data movement among the map, and reduce step.

Whole processing can be elaborated in detail as per below processing sequence it has been explained in figure2.

1. Map – Input data is first accepted by master node, which divides data file into smaller datasets, and move them to slave nodes. A slave node may reiterate the process leading to a multilevel tree structure. Map accept similar type of data and gives a list of output.
2. Map logic execution – Map logic will run once for every key value and generates output ordered by key values.
3. Reduce phase execution – the MapReduce system identify a node to run Reduce function, assigns the key value to each processor, and provides that node with all the Map-generated data allocated by same key value.
4. Execute Reduce code – Reduce is executed only once for each key value produced by the Map stage.
5. Produce the result – Final output is generated by consolidating all the Reduce output, and sorts them by key.

Below diagram Figure2, elaborates word count example of MapReduce processing.

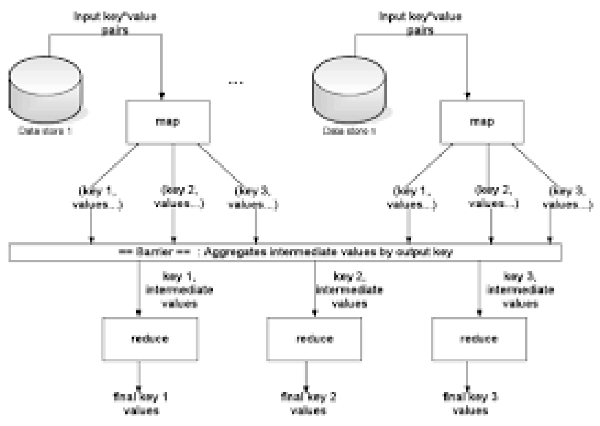

Fig. 2. Steps of Mapredcue execution

In a homogeneous environment, all nodes have equal computing speed and disk capacity. In case of node, failure happens MapReduce attempts to reruns failed tasks on a different node, which is idle/less-occupied at that moment. In some instances a node is executing a task very poorly or slowly known as straggler task, in that case Map Reduce kick-off a speculative copy of slow running task (“backup task”) on another node, which can finish task quickly job will be as slow as the misbehaving task ,if no speculative task is there. Straggling tasks can also be triggered due to impaired hardware and slowness This can improve job processing by 44%.In Heterogeneous environment lack of proper mechanism to address speculative task may increase processing time due to difference in nodes storage and processing capability.

IV. SPEECULATIVE EXECUTION IN HADOOP

When a node is idle, HADOOP selects that idle node for processing basis on below three events. Highest priority is assigned first to failed tasks .Then next priority is given to a task currently in non-execution status. For maps, nodes having local data is first picked-up. In last, Hadoop identify a task to re-run slow running task. Hadoop monitors task passage using a score between zero and one to pick a speculative task. Input data read determine map task progress score .However reduce task who, processing is divided into three steps, each step earn 1/3 of the score:

a) Accept input step, Task receives map outputs.
b) Order step, Sort Map output by key.
c) Reduce step, User-developed logic will be applied on map results list with each key.

In every stage processing percentage is calculated which depends upon the processed data . If a task is half completed then copy step progress score is 1/ 2 * 1/3=1/6 ,While a task till order step is 1/3 +[1/2*1/3]=1/2 , task till reduce step scores 1/ 3 + 1/ 3 + [1/ 2 * 1/ 3]) = 5 /6 . Hadoop job tacker monitors average progress score of each tasks (maps and reduces) to conclude a maximum time given for speculative execution. Task beyond the cut-off time is considered as “Slow” tasks. The speculative scheduling Works efficiently better is homogenous environments as due to same underline capacity and processing capability tasks begin and finishes approximately in same amount times and then speculation begins for task taking more than cut-off time. FIFO is used to determine when multiple jobs are running.

Hadoop scheduler adheres below implicit assumptions while working on speculative scheduling.

a) All nodes are of same processing capacity
b) Due to same processing ability of nodes, each task will take same amount of time to finish
c) To launch a speculative task on a new node no additional cost is required, though using this nodes will be optimally utilized.
d) A task’s progress score dependents on the task it has completed among copy, sort and reduce phases and every step spent 1/3 of the total CPU time.
e) Task secure minimum score is likely to be a straggler task.
f) Tasks using similar jobs among map or reduce assumed to take same amount of time.

Now above assumption of Hadoop processing works quite well in Homogeneous environment however, assumptions 1 and 2 suffers processing in heterogeneous cluster. However, Assumptions 3, 4 and 5 can violateds in a homogeneous environment as well and can contribute performance degradation. This is why Yahoo disabled speculative tasks on some jobs because it performance, and started other approaches to monitors faulty nodes. Similarly Facebook disables speculation for reduce tasks [16].Tasks in MapReduce should not be large task else complete job processing and derail data parallelism concept. For an ideal MapReduce job, the breaking input into equal parts and the distribution of keys between reduce does approximately same work. In case of heterogeneous environment, its complex to utilize the capability of every node efficiently this may cause performance degradation.

V. MAPREDCUE ASSUMPTIONS VIOLETION IN HETEROGENEOUS ENVIROUNMENT

A. *HETEROGENITY*

Generally, Hadoop considers that all underline machines of a data cluster are same and possess same execution power and capacity. However in actual there may be multiple versions and verity of hardware in data cluster. However in case of a virtualized data on individual physical network center many virtual nodes operates e.g., Amazon EC2, colocation of VMs may cause heterogeneity [8].
There are multiple processes advocated for improving MapReduce execution in heterogeneous environments. Each approach aims to improve one of underperforming areas of Map reduce features in a heterogeneous cluster.

Below are two prominent algorithms identified to boost processing are categorized as below:

- Data Locality Algorithms.
- Fault Tolerance Algorithms.

B. *DATA LOCALITY ALGORITHMS*

In case of homogeneous environment, all nodes are believed to have equal processing capability and storage. However In heterogeneous cluster having various nodes with varying processing a good CPU node can finish task quicker than lesser efficient CPU node. Data placement strategy play vital role to determine how data placement can be optimized in Heterogeneous Clusters to improve MapReduce performances. This ensures moving logic closer to data as this saves time and increase processing remarkably. This reduces network congestion and enhances the system processing.

C. *DATA PLACEMENT IN HETEROGENEOUS HADOOP CLUSTERS*

In heterogeneous Hadoop data cluster, a highly efficient machine can complete execution of local data quickly in comparison to slow processing node. Once faster node complete its processing of data present on local disk, the fast node helps, start manage and process incomplete data in distant slow machine. Cross exchange of data across nodes adds lot of network overhead and decreased performance a lot also data transfer cost is high if the volume of transferred data is big. Various algorithms designed to minimize data movement between slow and fast nodes in a heterogeneous clustering.

D. *INITIAL DATA PLACEMENT*

Other approach, which divides a big input dataset into a number of same size small datasets. One of node holds responsibility of data distribution of file segments across the nodes of the cluster. In heterogeneous nodes, it follows round-robin algorithm to selects nodes base on their computing capacity. Based on computing power data segments are assigned e.g. high computing node is expected to process bulk segments. In addition, less computing power node must process a small number of file segments for better and efficient processing.

E. *DATA RETRANSFER*

Input file portions scattered by the initial data placement algorithm above might be unfruitful due to the below following reasons:

a) Fresh data inserted to already present input file.
b) Reorganization is required if data is deleted from already present input File.
c) New node with additional data is included into cluster. There are various dynamic data load-balancing algorithms has been implemented to identify file fragments based on nodes processing capability.

VI. DATA LOCALITY AWARE TASK SCHEDULING FOR HETEROGENEOUS ENVIRONMENTS

Zhuoyao zhang [10] proposed a method in a homogeneous cluster to optimize data locality of Map reduce Approach consider an assumption that each node have same processing capacity and finishes assigned task in same time. However this assumption cannot be correct in heterogeneous environment due to numerous factors due to having varying processing speed of the connected various dissimilar nodes .This requires a dynamic work load kind of mechanism where a node act as resource manager and assigns the task based on node processing capability and maintain status
In a heterogeneous Hadoop cluster, X. Zhang et al. [10] suggested a data locality aware scheduling method. This method considers two performance indicators, which affects efficiency of map tasks one wait time to begin processing while second is transmission time. Execution-waiting time is the lowest wait time task for a task before it can be rescheduled to node having data for processing. Transmission time is a time taken to transfer input data to node task is scheduled.

The objective is to attain an equilibrium between wait time and transmission time during execution in order to achieve optimal execution time by scheduling a task to a node. In a lifecycle of task processing, highest priority assigned to node where input data locally present. If no such tasks, then pick up the job having data present in the closest to requesting node. Further waiting time calculation is done and assessment of data transfer time of the assigned task is performed. If waiting time, is less than data transfer time then algorithm schedules task on node keeps corresponding input data, else task to be scheduled on requesting node.

VII. FAULT TOLERANCE ALGORITHMS (LATE)

Primary benefit of Map reduce is its ability to identify, manage node failures and conceals the logical complexity from the users to manage node failures. Effective management of tasks by a task scheduler increases Hadoop's performance considering cluster is homogenous having all nodes similar processing and storage allowing tasks make progress linearly.

Hadoop's scheduler decides based on progress score check to decide when to re-run a long running task speculatively which looks to slowing overall job progress. Speculative tasks triggered based on an algorithm, which compares each task's progress to the average execution of other task. In homogeneous environment, this approach performs better due clear identification of straggler tasks. Due to data locality and other violation of other assumptions in heterogeneous environment Hadoop's scheduler performance downgrades.
Here we will talk about the algorithms used to enhance fault tolerance support in the heterogeneous cluster of Hadoop.

Longest Approximate Time to End Algorithm (LATE) utilize idle nodes efficiently by allocating tasks .Hadoop allocates a task using one of three approaches. Any failed task gets highest priority next priority is given to task is waiting status, then map tasks who have local data on this machine. Last priority is assigned to speculative tasks.
Hadoop computes a turnaround time for speculative processing by averaging each type of tasks (maps and reduces) progress percentage. A task having less progress percentage in comparison to its category average will be marked as a straggler. LATE identify task which will finish is longest approximate time and will run that speculatively. LATE computes the task's completion time as per progress score provided by Hadoop. Which further computes the task progress rate of each task along with calculation of task's competition time.

VIII. LATE SCHEDULAR

LATE is abbreviation of Longest Approximate Time to End. This approach performs efficiently in homogeneous environment where all cluster nodes assumed to have same processing speeds and no cost overhead while launching a speculative task on another idle /less utilized node. Different methods for calculating time left can be added into LATE. First method known as simple heuristic method which computes the progress rate of each task as Progress Score/T, T is the time since task is in execution, and then computes the time to finish as (1 − Progress Score)/Progress Rate. This has assumption that tasks execution rate is approximately same. However, in some cases, heuristic approach may fail to give desired result, but it is successful in typical Hadoop jobs. Speculative task should be launched in faster computing nodes in order to finish them quickly using a simple method to not launch speculative tasks especially on slow machines.

Suggested heuristic approach gives improved results in comparison to a speculative task to the first available machine on data cluster. Alternate solution is to create speculative task more than one replica, but this consumes many unnecessary resources. Finally, two heuristics are followed to take care of cost overhead as below:

1. Numbers of speculative tasks allowed executing at a time, which is known as Speculative Cap.
2. Correct computation of slow running task to decide node is "enough slow" to be speculated to avid redundant launching of speculative tasks.

LATE algorithm behaves as follows:

When a node is requested to execute as fresh task having currently running task less than computed speculative cap. Speculative tasks running then following are the conditions, which are considered to execute fresh task on that node:

1. If nodes progress is less than Slow Node, Threshold ignore the request, as this will degrade the performance.
2. Correct ranking of task currently in execution and based on time left before speculation cut-off time.
3. Kick-off a copy of the highest-ranked task, which has progress rate less than Slow Task cut-off execution timed.

LATE algorithm works similar to Hadoop's scheduler, wait for until threshold time to a task to complete before deciding for speculation. In order to achieve optimum result, three numbers for LATE is to keep Speculative Cap to 10% of current task slots and allocate the Slow Node Threshold and Slow Task running cut-off to the 25% of node processing limit and task progress rates.

A. Benefits of LATE algorithm into MapReduce processing

LATE has improved MapReduce performance significantly. It has improved MapReduce performance in heterogeneous environment where each nodes has varying processing capability by re-launching slowest tasks only less in number. Considering impact on response time, LATE selects slowest tasks for next execution.
To avoid resource contention LATE caps on number of speculative. However, Hadoop's scheduler comes with a fixed number, which launch all slow tasks and have an equal probability of being launched. This causes exuberant number of speculated tasks.

Computing estimated remaining time instead of computing progress rate, LATE speculatively selects and executes tasks having possibility to optimize job response time, rather than picking up any slow tasks.

B. Estimating Finish Times

Estimate the time remaining to finish task equal to (1 − Progress marks)/Progress Rate. This heuristic works well in most of the scenarios. There are some situations where this heuristic approach will not correctly estimate finish time .As in typical MapReduce, such situations do not occur frequently as elaborated below, we have used the simple heuristic as described above as per below result conducted by Matei Zaharia [18] .

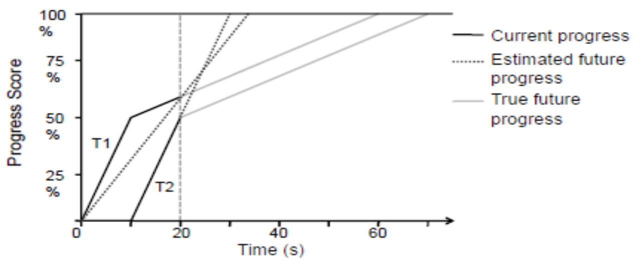

Fig. 3. Processing time using LATE algorithms

How the progress rate examining might underperform, take a situation have two steps, which runs at different rates. Case one when task's progress score increases by 5% per second in the first step, maximum to 50%, and then reduces by 1% in the second stage. Task takes ten seconds in the first stage and fifty-seconds in the next stage, or sixty seconds total to complete. If two copies T1 and T2 of a tasks are launched, one at time zero and other at ten second, and check their progress rates after twenty seconds. Figure3 mimic this scenario. At time twenty, T1 completes first stage and processed one fifth of second stage, makes progress score as 60%, with progress rate equal to 60/20s = 3%/s. Meanwhile, T2 just completes first stage making progress rate equal to 50/10s = 5%/s. The estimated time left for T1 equal to [100 – 60]/ [3/s] = 13.3s. The estimated time left for T2 is equal to [100−50]/[5/s] = 10s. Therefore, it is computed that T2 will finish earlier than T1, normally in reality T2 finishes after T1. This happens due task's progress rate decreases throughout its lifetime and is not related to actual progress. There could be task finishes second step faster there would be no problem. Hence, a correct estimation of longer time left in first step will better estimate of finish times more accurately.

There can be scenarios when a reduce task takes more time and running slowly than the Map tasks, a complex heuristic design can help accurate calculations of finish time. This will consider each phase independently to estimate completion time. To consider per-phase progress rate gives better completed or in-progress steps for that task and for steps yet to be started by using average progress rate of steps of other reduce tasks.

This heuristic assumes a slow performing task in some step will run faster in other step. Challenge with advised step - aware heuristic is dependency on historical averages of progress rates per step. In future work there is plan to investigate the finish time efficiently.

complete. If two copies T1 and T2 of a tasks are launched, one at time zero and other at ten second, and check their progress

rates after twenty seconds. Figure3 mimic this scenario. At time twenty, T1 completes first stage and processed one fifth of second stage, makes progress score as 60%, with progress rate equal to 60/20s = 3%/s. Meanwhile, T2 just completes first stage making progress rate equal to 50/10s = 5%/s. The estimated time left for T1 equal to [100 – 60]/ [3/s] = 13.3s. The estimated time left for T2 is equal to [100−50]/[5/s] = 10s. Therefore, it is computed that T2 will finish earlier than T1, normally in reality T2 finishes after T1. This happens due task's progress rate decreases throughout its lifetime and is not related to actual progress. There could be task finishes second step faster there would be no problem. Hence, a correct estimation of longer time left in first step will better estimate of finish times more accurately.

There can be scenarios when a reduce task takes more time and running slowly than the Map tasks, a complex heuristic design can help accurate calculations of finish time. This will consider each phase independently to estimate completion time. To consider per-phase progress rate gives better completed or in-progress steps for that task and for steps yet to be started by using average progress rate of steps of other reduce tasks.

This heuristic assumes a slow performing task in some step will run faster in other step. Challenge with advised step - aware heuristic is dependency on historical averages of progress rates per step. In future work there is plan to investigate the finish time efficiently.

IX. CONCLUSION AND FUTURE WORK

MapReduce has been regarded as prominent programming paradigm to cope with Big data processing .Though MapReduce offers numerous advantages but there are few trade-offs faced in meeting, the rapidly growing computing demands of Big Data in heterogeneous environment. There are many scheduling methodologies proposed .Our aim is to identify and categorize related scheduling algorithms, their capability to address MapReduce challenge to work efficiently in Heterogeneous environment. .This enables better planning of Big data projects. Future work on this will be develop a scheduling change in Hadoop MapReduce which will work efficiently using LATE scheduling approach as this is described as most suited approach among all proposed scheduling methodologies. In addition, investigation will be done to compute finish time estimation in more detail.

REFERENCES

[1] AdaptiveScheduler,https://issues.apache.org/jira/browse/MAPREDUCE-1380

[2] Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large clusters. (December 2004)

[3] Hadoop MapReduce, http://hadoop.apache.org/mapreduce/

[4] Thusoo, A., Shao, Z., Anthony, S., Borthakur, D., Jain, N., Sen Sarma, J., Murthy, R., Liu, H.: Data warehousing and analytics infrastructure at facebook. In: Proceedings of the 2010 International Conference on Management of Data, SIGMOD 2010, ACM 2010

[5] Ananthanarayanan, G., Kandula, S., Greenberg, A., Stoica, I., Lu, Y., Saha, B., Harris, E.: Reining in the outliers in map-reduce clusters using mantri. In: OSDI 2010, pp. 1–16. USENIX Asoc., Berkeley (2010)

[6] Polo, J., Carrera, D., Becerra, Y., Steinder, M., Whalley, I.: Performance-driven task co-scheduling for MapReduce environments. In: Network Operations and Management Symposium, NOMS, pp. 373–380. IEEE, Osaka (2010)

[7] Wolf, J., Rajan, D., Hildrum, K., Khandekar, R., Kumar, V., Parekh, S., Wu, K.-L., Balmin, A.: Flex: A Slot Allocation Scheduling Optimizer for Mapreduce Workloads. In: Gupta, I., Mascolo, C. (eds.) Middleware 2010. LNCS, vol. 6452, pp. 1–20. Springer, Heidelberg (2010)

[8] Dynamic Proportional share scheduling in Hadoop Thomas sandholm and Kevin Springer Berlin Heidelberg Volume 6253, 2010, pp 110-131

[9] Improving Map Reduce Performance through Data Placement in Heterogeneous Hadoop Clusters- Jiong Xie, Shu Yin, Xiaojun Ruan, Zhiyang Ding,

[10] An Empirical Analysis of Scheduling techniques for Real-time cloud based data processing-linh T.X. Phan Zhuoyao zhang, Qi Zheng Boon Thau Loo University of Pennsylvania

[11] Herodotou, H., and Babu, S. Profiling, what-if analysis, and cost-based optimization of MapReduce programs. In Proc. Int' Conf. on Very Large Data Bases (VLDB) (2011).

[12] MapR. The executive's guide to big data. http://www.mapr.com/resources/white-papers. [21] Pettijohn, E., Guo, Y., Lama, P., and Zhou, X. User-centric heterogeneity-aware mapreduce job provisioning in the public cloud. In Proc. Int'l Conference on Autonomic Computing (ICAC) (2014).

[13] Herodotou, H., Lim, H., Luo, G., Borisov, N., Dong, L., Cetin, F. B., and Babu, S. Starfish: A self-tuning system for big data analytics. In Proc. Conference on Innovative Data Systems Research (CIDR) (2011).

[14] Jinda, A., Quian-Ruiz, J., and Dittrich, J. Trojan data layouts: Right shoes for a running elephant. In Proc. of ACM Symposium on Cloud Computing (SoCC) (2011).

[15] Lama, P., and Zhou, X. Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud. In Proc. Int'l Conf. on Autonomic computing (ICAC) (2012).

[16] Li, X., Wang, Y., Jiao, Y., Xu, C., and Yu, W. Coomr: Cross-task coordination for efficient data management in mapreduce programs. In Proc. Int'l Conference for High Performance Computing, Networking, Storage and Analysis (SC) (2013).

[17] Kambatla, K., Pathak, A., and Pucha, H. Towards optimizing hadoop provisioning in the cloud. In Proc. USENIX HotCloud Workshop (2009).

[18] Li, Z., Cheng, Y., Liu, C., and Zhao, C. Minimum standard deviation difference-based thresholding. In Proc. Int'l Conference on Measuring Technology and Mechatronics Automation (ICMTMA) (2010).

[19] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, Ion Stoica https://cs.stanford.edu/~matei/papers/2008/osdi_late.pdf