Hadoop Performance Modeling For Job Estimation And Resource Provisioning

[1]Prof. Sumeet Pate, [2]Ganesh Phad, [3]Pratik Dhage, [4]Saurabh Wagh

[1]Asst. Professor, [2,3,4]UG Student, [1,2,3,4]Computer Engg. Dept. Shivajirao S. Jondhle College of Engineering & Technology, Asangaon, Maharashtra, India

[1]*sumeetpate09@gmail.com*, [2]*phadganesh37@gmail.com* [3]*dhagepratik94@gmail.com*,

[4]*saurabhsuhaswagh@gmail.com*

Abstract— For information escalated applications MapReduce is vital processing model in exhibit period. Open source execution of MapReduce is Hadoop. It has been embraced by an inexorably developing client group. Distributed computing specialist organizations like Amazon, EC2 Cloud are putting forth the open doors for Hadoop clients to rent a specific measure of assets. Problem for cloud service providers is that they do not have technique to comply user jobs within time. Nowadays user has to estimate the required amount of resources for running his job in the cloud.This paper is displaying Hadoop work execution demonstrate which precisely gauges work finish time. It provides number of resources needed to complete the job within specific time. This model depends on past job execution records. It utilizes Locally Weighted Linear Regression (LWLR) method to compute the execution time of a job. It also uses Lagrange Multipliers technique for estimating resources amount needed to satisfy jobs within given timeline.

Keywords: Cloud computing, Hadoop MapReduce, performance modeling, job estimation, resource provisioning.

I. INTRODUCTION

Many organizations have systems to collect huge amount of data information from World Wide Web, sensor networks and social networks etc. The essential undertaking for them is to perform versatile and opportune examination of these unstructured informational collections. It is extremely intense occupation for conventional system and database frameworks to process these persistently developing datasets.

MapReduce[1] was initially created by Google and now it has turned into a noteworthy figuring model in help of information serious applications. This system is an exceedingly adaptable, fault tolerant and information parallel model that consequently disperses the information and parallelizes the calculation over a group of PCs[2]. Because of open source nature of Hadoop it has turned out to be well known in group than it's countrymen like Mars[3], Phoenix[4], Dryad[5].

This framework displays an enhanced Hadoop Performance (HP) model for work execution estimation and asset provisioning. Key highlights of this framework are as per the following:-

The enhanced HP work mathematically models all the three center periods of a Hadoop work though the HP work does not numerically model the non-overlapping shuffle phase in the primary wave.

The enhanced HP model utilizes locally weighted linear regression (LWLR) strategy to estimate the execution time of a Hadoop work with a fluctuated number of lessened tasks despite what might be expected the HP model utilizes a basic straight relapse procedure for work execution estimation which confines to a consistent number of decreased tasks.

Based on job execution estimation, the improved HP model employs Lagrange Multiplier technique to provision the amount of resources for a hadoop job to complete within a given deadline.

II. LITERATURE SURVEY

A. Compute VM Load from data nodes

Input: ith Node input

Output: Idle or Normal Or Overloaded in percent Compute Load (VM id):

Weight Degree Inputs: The static parameter comprise the number of CPUs, the CPU dispensation speeds, the reminiscence size, etc. active parameters are the memory consumption ratio, the CPU exploitation ratio, the network bandwidth.

Procedure:

Step 1: Characterize a load limit set: F= {F1, F2. .Fm} with each Fire present the total number of the consideration.

Step 2: Calculate the load capacity as weight Load Degree (N) = (∑αi Fi), Where, i= (1,...,m).

Step 3: Ordinary cloud partition degree from the node consignment degree statics as: Load amount avg =∑ (i=1...n) LoadDegree (Ni)

Step 4: Three height node position are defined Load Degree (N) =0 for Inactive.

0 <Load Degree (N) <Load Degree (N) for overfull. Load Degree (N) high ≤ Load Degree (N) for overloaded.

B. Equally Spread Current Execution Throttled Load balancing Algorithm

Input: File form user as Fi.

Output: Equally distributed chunks on data servers
Step 1: Read Fi from data owner with size
Step 2: count total number of data nodes Ni
Step 3: for each (score=read each vm node and call to compute node (k)) Read when k==null End for
Step 4: create data chunks base on server loads score.
Step 5: save all data on data nodes.

C. MapReduce: Simplified data processing on large clusters

MapReduce is a programming model[2] and a related usage for handling and creating expansive datasets that is amiable to a wide assortment of real-world tasks. Clients indicate the calculation in terms of a map and a reduce function, and the fundamental runtime framework naturally parallelizes the calculation crosswise over expansive scale clusters of machines, handles machine disappointments, and schedules inter-machine communication to make productive utilization of the network and disks. Software engineers discover the framework simple to utilize. In excess of ten thousand distinct MapReduce programs have been actualized inside at Google in the course of past four years. A normal of one hundred thousand MapReduce jobs are executed on Google's clusters each day. It processes a sum of more than twenty petabytes of information for every day.

MapReduce keeps running on large clusters of product machines and it is largely scalable.

D. Mars: A MapReduce framework on graphics processors

This paper proposes outline and usage of Mars[3], a MapReduce system, on graphics processors (GPUs). MapReduce is a circulated programming structure initially proposed by Google for the simplicity of development of web seek applications on an expansive number of product CPUs. Contrasted and CPUs, GPUs have a request of greatness higher calculation power and memory bandwidth, yet are harder to program since their models are planned as a unique reason co-processor and their programming interfaces are normally for graphics applications.

ALGORITHM: Graph-Based Video Sequence Matching

Step 1) Segment the video frames and extract features of the key frames. Auto dual-threshold method used to segment the video sequences, and then extract SURF and SIFT features of the key frames.

Step 2) Match the query video and target video.

Assume

Qc = {C1Q, C2Q, C3Q, …..CmQ} and Tc = {C1T, C2T, C3T , ….. CmT}

are the segment sets of the query video and target video from Step1, respectively. For each CiQ in the query video, compute the similarity sim (CiQ,CjT) and return k largest matching results. K=8n, where n is the number of segments in the target set, and 8 is set to 0.05 based on our empirical study.

Step 3) Generate the matching result graph according to the matching results. In the matching result graph, the vertex Mij represents a match between CiQ, andCjT .To determine whether there exists an edge between two vertexes, two measures are evaluated. Time direction consistency: For Mij and Mlm, if there exists (i-l)*(j-m) > 0, then Mij and Mlm satisfy the time direction consistency. Time jump degree: For Mij and Mlm, the time jump degree between them is defined as ?tijlm=max (| ti-tl |, | tj-tm |)

Step 4) Search the longest path in the matching result graph. The problem of searching copy video sequences is now converted into a problem of searching some longest paths in the matching result graph. The dynamic programming method is used in this paper. The method can search the longest path between two arbitrary vertexes in the matching result graph.

These longest paths can determine not only the location of the video copies but also the time length of the video copies.

Step 5) Output the result of detection. We can get some discrete paths from the matching result graph; it is thus easy to detect more than one copy segments by using this method.

E. Phoenix: A parallel programming model for accommodating dynamically joining/ leaving resources

This paper proposes Phoenix[4], a programming model for composing parallel and circulated applications that suit progressively joining/leaving computer resources. In the proposed model, nodes engaged with an application see an expansive and settled virtual node name space.

They convey by means of messages, whose goals are indicated by virtual node names, as opposed to names bound to a physical resource. The paper portrays Phoenix API and show how it permits a straightforward movement of application states, and also dynamically joining/leaving nodes as its by-product. It additionally show through a several application studies that Phoenix model is sufficiently close to normal message passing, along these lines it is a general programming model that encourages porting numerous parallel applications/algorithms to more dynamic environments.

F. Dryad: Distributed data-parallel programs from sequential building blocks

Dryad[5] is a general purpose circulated execution engine for coarse-grain information parallel applications. A Dryad application joins computational "vertices" with communication "channels" to frame a dataflow graph. Dryad runs the application by executing the vertices of this graph on an arrangement of accessible PCs, communicating as suitable through flies, TCP pipes, and shared-memory FIFOs. The vertices gave by the application designer are very basic and are normally composed as consecutive projects with no thread creation or locking. Concurrency

arises from Dryad planning vertices to run at the same time on various PCs, or on different CPU centers inside a PC. The application can find the size and position of information at run time, and alter the graph as the calculation advances to make enough utilization of the accessible resources.

G. The performance of Map-Reduce: An in-depth study

MapReduce[7] has been generally utilized for vast scale information investigation in the Cloud. The framework is all around perceived for its flexible versatility and fine-grained adaptation to internal failure in spite of the fact that its execution has been noted to be suboptimal in the database context. As per a current report, Hadoop, an open source usage of MapReduce, is slower than two best in class parallel database frameworks in playing out an assortment of analytical tasks by a factor of 3.1 to 6.5. MapReduce can accomplish better execution with the allocation of more figure nodes from the cloud to accelerate calculation.

III. EXISTING SYSTEM

To effectively manage cloud resources, a few Hadoop execution models have been proposed. In any case, these models don't consider the overlapping and non-overlapping phases of the shuffle phase which prompts an inaccurate estimation of job execution. As of late, various complex Hadoop execution models are proposed.

Starfish gathers a running Hadoop work profile at a fine granularity with point by point data for work estimation and improvement. On the highest point of Starfish, Elasticiser is proposed for asset provisioning as far as virtual machines.
The HP demonstrate considers both the covering and non-covering stages and uses straightforward direct relapse for work estimation. CRESP estimates work execution and support resource provisioning as far as map and reduce slots.

Limitations:

Collecting the detailed execution profile of a Hadoop job incurs a high overhead which leads to an overestimated job execution time.

The HP model also estimates the amount of resources for jobs with deadline requirements.

Both the HP model and CRESP ignore the impact of the number of reduce tasks on job performance. The HP model is restricted to a constant number of reduce tasks, whereas CRESP only considers a single wave of the reduce phase.

IV. PROBLEM STATEMENT

The capacity to perform scalable and timely investigation on these unstructured datasets is a high need undertaking for some enterprises. It has become hard for traditional network storage and database systems to process these consistently growing datasets. Map Reduce, initially created by Google, has turned into a major processing model in help of information intensive applications. It is an exceedingly adaptable, blame tolerant and information parallel model that consequently circulates the information and parallelizes the calculation over a group of PCs.

V. PROPOSED SYSTEM

Developing the HP model, this paper shows an upgraded HP model for Hadoop work execution estimation and resource provisioning. The genuine responsibilities of this paper are according to the accompanying:

The improved HP work numerically models all the three center phases of a Hadoop job. The upgraded HP model show utilizes LWLR strategy to evaluate the execution time of a Hadoop work with a changed number of lessen errands. Strikingly, the HP model uses an essential direct relapse method for work execution estimation which breaking points to an unfaltering number of diminishing errands. In light of work execution estimation, the upgraded HP indicate uses Lagrange Multiplier system to course of action the measure of advantages for a Hadoop work to complete inside a given due date. The proposed framework called Hadoop execution Displaying for Employment Improvement. In Proposed Framework it exhibit enhanced HP show for Hadoop work execution estimation and asset provisioning. The enhanced HP work scientifically models all the three center phasess of a Hadoop job.

Benfits of proposed system:

The enhanced HP model scientifically models all the three center periods of a Hadoop work. Conversely, the HP work does not scientifically demonstrate the non overlapping mix stage in the primary wave.

The enhanced HP model utilizes Locally Weighted Linear Regression (LWLR) method to evaluate the execution time of a Hadoop work with a differed number of reduce tasks. Conversely, the HP model utilizes a simple linear regression procedure for work execution estimation which confines to a steady number of reduce tasks.

In view of job execution estimation, the enhanced HP model utilizes Langrage Multiplier procedure to provision the measure of resources for a Hadoop job to finish inside a given deadline.

VII. SYSTEM ARCHITECTURE

Modeling Map Phase: In this phase, a Hadoop job reads an input dataset from Hadoop distributed file system (HDFS), parts the input dataset into information pieces in light of a predefined size and after that passes the information chunks to a user-define map function. The guide work forms the information pieces and delivers a guide yield. The map output is called intermediate information.
Modeling Reduce Phase: In this phase, an occupation peruses the arranged middle information as info and goes to a client characterized lessen work. The lessen work forms the middle information and produces a last yield. All in all, the diminish yield is composed over into the HDFS.

Modeling Shuffle Phase: In this phase, a Hadoop job brings the halfway information, sorts it and copies it to at least one reducers. The rearrange assignments and sort undertakings are performed at the same time, along these lines, for the most part consider them as a rearrange stage.

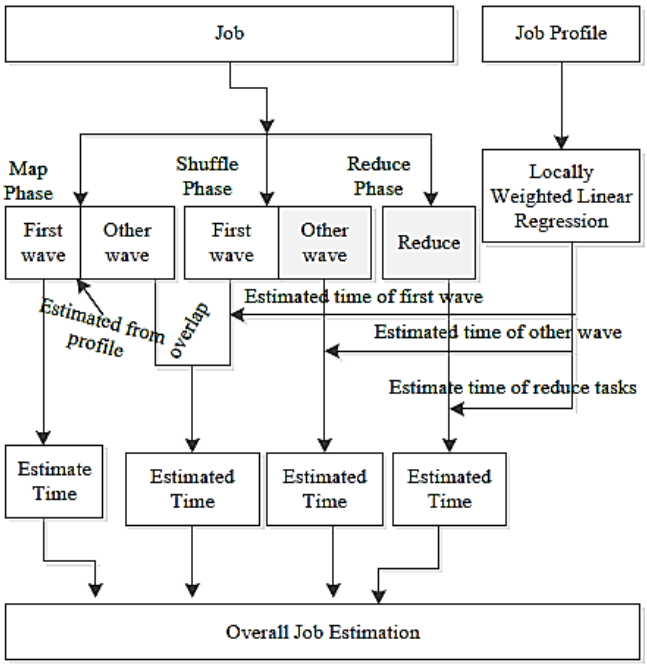

Fig. 7.1 System Architecture

VIII. MATHEMATICAL MODEL

This paper shows the mathematical expressions of the improved HP work in modelling a Hadoop job which finishes in multiple waves. The Table defines the variables used in the improved model.

Table 8.1 Mathematical Model

Variables	Expressions
T_{m-w1}^{low}	The lower bound duration of the map phase in the first wave (non-overlapping).
T_{m-w1}^{up}	The upper bound duration of the map phase in the first wave (non-overlapping).
N_m^{w1}	The number of map tasks that complete in the first wave of the map phase.
N_m^{w2}	The number of map tasks that complete in other waves of the map phase.
T_m^{max}	The maximum execution time of a map task.
T_{sh-w1}^{low}	The lower bound duration of the shuffle phase in the first wave (overlapping with the map phase).
T_{sh-w1}^{up}	The upper bound duration of the shuffle phase in the first wave (overlapping with the map phase).
T_{sh-w1}^{avg}	The average execution time of a shuffle task that completes in the first wave of the shuffle phase.
T_{sh-w1}^{max}	The maximum execution time of a shuffle task that completes in the first wave of the shuffle phase.
T_{sh-w2}^{low}	The lower bound duration of the shuffle phase in other waves (non-overlapping).
T_{sh-w2}^{up}	The upper bound duration of the shuffle phase in other waves (non-overlapping).
T_{sh-w2}^{avg}	The average execution time of a shuffle task that completes in other waves of the shuffle phase.
T_{sh-w2}^{max}	The maximum execution time of a shuffle task that completes in other waves of the shuffle phase.
T_r^{low}	The lower bound duration of the reduce phase.
T_r^{up}	The upper bound duration of the reduce phase.
T_r^{max}	The maximum execution time of a reduce task.
T_{job}^{low}	The lower bound execution time of a Hadoop job.
T_{job}^{up}	The upper bound execution time of a Hadoop job.
T_{job}^{avg}	The average execution time of a Hadoop job.

Algorithm: Compute VM Load from data nodes

Input: ith Node input

Output: Idle or Normal Or Overloaded in percent Compute Load (VM id):

Weight Degree Inputs: The static parameter comprise the number of CPUs, the CPU dispensation speeds, the reminiscence size, etc. active parameters are the memory consumption ratio, the CPU exploitation ratio, the network bandwidth.

Procedure:

Step 1: Characterize a load limit set: F= {F1, F2. .Fm} with each Fire present the total number of the consideration.

Step 2: Calculate the load capacity as weight Load Degree (N) = (∑αi Fi), Where, i= (1,...,m).

Step 3: Ordinary cloud partition degree from the node consignment degree statics as: Load amount avg =∑ (i=1...n) LoadDegree (Ni)

Step 4: Three height node position are defined Load Degree (N) =0 for Inactive.

0 <Load Degree (N) <Load Degree (N) for overfull. Load Degree (N) high ≤ Load Degree (N) for overloaded.

Step 1: Hold the count current active allocation on each vm.

```
publicActiveVmLoadBalancer(DatacenterController dcb)
{
dcb.addCloudSimEventListener(this);
this.vmStatesList = dcb.getVmStatesList();
this.currentAllocationCounts=
Collections.synchronizedMap(new HashMap<Integer,
Integer>());

}
```

Step 2: Find the vm with least number of allocations and if all allocations vm are not allocated then allocated the new ones.

```
if (currentAllocationCounts.size() < vmStatesList.size()){
for (int availableVmId : vmStatesList.keySet()){
if (!currentAllocationCounts.containsKey(availableVmId)){
vmId = availableVmId;
break;
}}}
else {
int currCount;
 int minCount =Integer.MAX_VALUE;
for (int thisVmId : currentAllocationCounts.keySet()){
currCount=currentAllocationCounts.get(thisVmId);
if (currCount < minCount){
```



```
minCount = currCount;
vmId = thisVmId;
 }}}
allocatedVm(vmId);
```

Step 3: Count the number of data nodes of vm

Step 4 :

For each((score = read each vm node and call to computer node(k))
Read each node when k == null

Step 5: Create data chunks base on servers load score

Step 6: Save all data on data nodes.

In practice, job tasks in different waves may not complete exactly at the same time due to varied overhead in disk I/O operations and network communication. Therefore, the improved HP model estimates the lower bound and the upper bound of the execution time for each phase to cover the best-case and the worst-case scenarios respectively.

The author consider a job that runs in both non-overlapping and overlapping stages. The lower bound and the upper bound of the map phase in the first wave which is a non-overlapping stage can be computed.

$$T_{m-w1}^{low} = \frac{T_m^{avg} \times N_m^{w1}}{N_m^{slot}}$$

$$T_{m-w1}^{up} = \frac{T_m^{max} \times N_m^{w1}}{N_m^{slot}}$$

In the overlapping stage of a running job, the map phase overlaps with the shuffle phase. Specifically, the tasks running in other waves of the map phase run in parallel with the tasks running in the first wave of the shuffle phase. As the shuffle phase always completes after the map phase which means that the shuffle phase takes longer than the map phase, therefore the author uses the duration of the shuffle phase in the first wave to compute the lower bound and the upper bound of the overlapping stage of the job.

$$T_{sh-w1}^{low} = \frac{T_{sh-w1}^{avg} \times N_{sh}^{w1}}{N_r^{slot}}$$

$$T_{sh-w1}^{up} = \frac{T_{sh-w1}^{max} \times N_{sh}^{w1}}{N_r^{slot}}$$

As a result, the lower bound and the upper bound of the execution time of a Hadoop job can be computed by combining the execution durations of all the three phases.

$$T_{job}^{low} = T_{m-w1}^{low} + T_{sh-w1}^{low} + T_{sh-w2}^{low} + T_r^{low}$$

$$T_{job}^{up} = T_{m-w1}^{up} + T_{sh-w1}^{up} + T_{sh-w2}^{up} + T_r^{up}$$

Finally, the average of equations to estimate the execution time of a Hadoop job is,

$$T_{job}^{avg} = \frac{T_{job}^{low} + T_{job}^{up}}{2}$$

VI. COMPARATIVE STUDY

Table 6.1 Comparative Study

Sr. No.	Paper Title	Author Name	Problem	Solution
1.	MapReduce:Simpified Data processing on large Clusters	S. Ghemawat& J. Dean	large-scale machine learning problems, clustering problems for the Google News and Froogle products,	Ordering makes it easy to generate a sorted output per partition, which is useful when the output format needs to support efficient random access for data to be sorted.
2.	Mars:AMapReduce Framework on Graphics Processors	T. Wang, Q. Luo, N. K. Govindaraju and B. He, W. Fang	Searching longest path in matching result graph	Dynamic programming method
3.	Phoenix:A Parallel Programming Model for accommodating Dynamically joining/leaving Resources	A. Yonezawa, T. Endo, K. Kaneda, and K. Taura	model is weak in supporting joining leaving nodeslargely unsolved, andcomplications that arise are simply exposed to the programmer	It allows parallel applications thatchange the number of participating nodes at runtime
4.	Dryad:Distributed Data Parallel Program for sequential building blocks	D. Fetterly, M. Isard, M. Budiu, Y. Yu and A. Birrell	long-standing problem to write efficientparallel and distributed applications	It handles by scheduling the use ofcomputers and their CPUs, recovering from communication or computer failures, and transporting data between vertices
5.	The Performance of MapReduce:An In-depth study	D. Jiang, B. C. Ooi, L. Shi, and S. Wu	Due to large database searching time complexity is more	Locality secure hashing

IX. CONCLUSION

This paper has introduced an enhanced HP model to accomplish this objective considering numerous multiple waves of the shuffle phase of a Hadoop job. The enhanced HP demonstrate was at first assessed on an in-house Hadoop clusters and accordingly assessed on the EC2 cloud.

The experimental results showed that the improved HP model outperforms both Starfish and the HP model in job execution estimation. Similar to the HP model, the improved HP model provisions resources for Hadoop jobs with deadline requirements.

X. RESULTS & DISCUSSIONS

Dataset is given input to the Map Reduce phase.

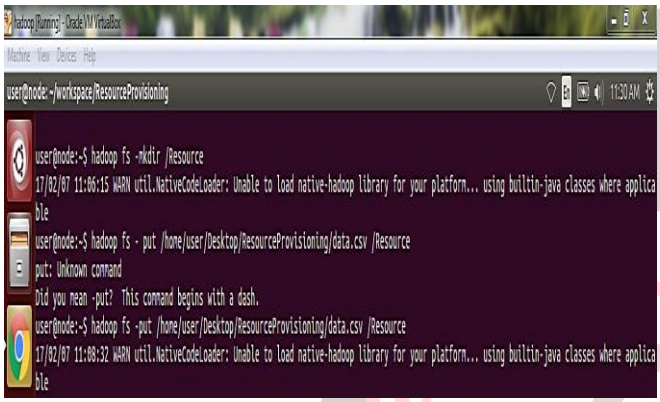

Fig. 2 Resource Provisioning

Job scheduling is done using map 100% and reduce 100%

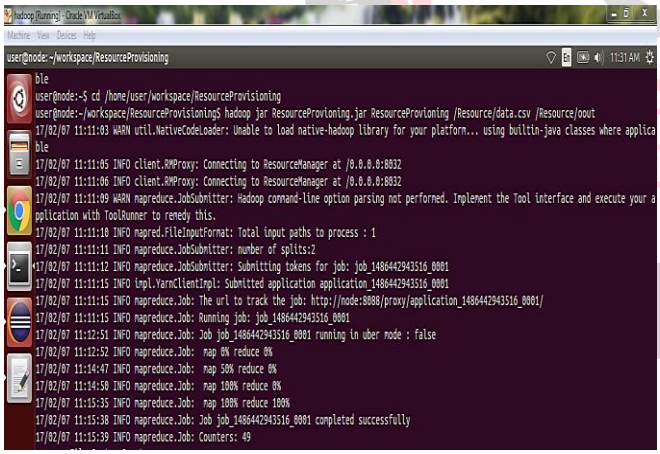

Fig. 3 Job Estimation

Fig. 4Resource Provisioning Graph

Above graph displays the scheduling of jobs given to the job estimation phase.

REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] R. Lämmel, "Google's MapReduce programming model — Revisited," *Sci. Comput. Program.* vol. 70, no. 1, pp. 1–30, 2008.

[3] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, "Mars: a MapReduce framework on graphics processors," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques - PACT '08*, 2008, p. 260.

[4] K. Taura, T. Endo, K. Kaneda, and A. Yonezawa, "Phoenix: a parallel programming model for accommodating dynamically joining/leaving resources," in *SIGPLAN Not.*, 2003, vol. 38, no. 10, pp. 216–229.

[5] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," *ACM SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 59–72, Mar. 2007.

[6] "Apache Hadoop." [Online]. Available: http://hadoop.apache.org/. [Accessed: 21-Oct-2013].

[7] D. Jiang, B. C. Ooi, L. Shi, and S. Wu, "The Performance of MapReduce: An In-depth Study," *Proc. VLDB Endow.*, vol. 3, no. 1–2, pp. 472–483, Sep. 2010.

[8] U. Kang, C. E. Tsourakakis, and C. Faloutsos, "PEGASUS: Mining Peta-scale Graphs," *Knowl. Inf. Syst.*, vol. 27, no. 2, pp. 303–325, May 2011.

[9] B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo, "PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce," *Proc. VLDB Endow.* vol. 2, no. 2, pp. 1426–1437, Aug. 2009.

[10] A. Pavlo, E. Paulson, and A. Rasin, "A comparison of approaches to large-scale data analysis," in *SIGMOD '09 Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, 2009, pp. 165–178.

