IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 19, Issue 6, Ver. IV (Nov.- Dec. 2017), PP 12-18
www.iosrjournals.org

Performance Modeling of Map Reduce Jobs in Heterogeneous Cloud Environments

[1]H.Shyam Sundhar, [2]Eruguralla Sathish Babu,[3]Bukya Mohanbabu,

Dept of Computer Science & Engineering,
Vivekananda Institute of Technology & Sciences, Karimnagar, Ts, India.

Abstract: *Many companies start using Hadoop for advanced data analytics over large datasets. While a traditional Hadoop cluster deployment assumes a homogeneous cluster, many enterprise clusters are grown incrementally over time, and might have a variety of different servers in the cluster. The nodes' heterogeneity represents an additional challenge for efficient cluster and job management. Due to resource heterogeneity, it is often unclear which resources introduce inefficiency and bottlenecks, and how such a Hadoop cluster should be configured and optimized. In this work1, we explore the efficiency and performance accuracy of the bounds-based performance model for predicting the Map Reduce job completion times in heterogeneous Hadoop clusters. We validate the accuracy of the proposed performance model using a diverse set of 13 realistic applications and two different heterogeneous clusters. Since one of the Hadoop clusters is formed by different capacity VM instances in Amazon EC2 environment, we additionally explore and discuss factors that impact the Map Reduce job performance in the Cloud.*

Keywords: *Map Reduce, heterogeneous clusters, performance modeling,* efficiency

Date of Submission: 13-12-2017 Date of acceptance: 28-12-2017

I. Introduction

In recent years, there has been an emergence of several data-parallel middleware platforms primarily targeted towards large-scale data processing in the cloud environment, for example, Map Reduce [3] and its popular open-source Hadoop implementation. These middleware platforms provide simple yet powerful and extremely versatile programming models, and can be applied to a wide spectrum of domains where large-scale data analytics is performed. To better manage cloud resources, there has been a series of efforts aimed at developing performance models [4], [5], [6], [7], [9], [10] for Map Reduce applications. These performance models are used for predicting the job completion times resources. They are also used as a basis for the resource provisioning problem: and estimating the amount of resources required to complete applications with given deadlines. One major shortcoming of the existing models is that they do not consider the resource heterogeneity which causes inaccurate prediction on heterogeneous clusters. We argue that cluster deployments are grown incrementally over time. It is not unusual for companies to start off with an initial cluster, and then gradually add more compute and I/O resources as the number of users increases. More often than not, it is economical to add newer more powerful servers rather than discard the old hardware. So, the problem is to design and offer the efficient performance models that work well in heterogeneous environments. Performance models that rely on extensive fine-granularity profiling techniques [4], [5] or complex analysis using ma chine learning [9], are unlikely to work well for clusters with high degrees of heterogeneity due to the complexity and execution variability in the heterogenous environment. In this paper, we explore the applicability and performance accuracy of an alternative bounds-based approach introduced in the ARIA project [10] for homogeneous Hadoop clusters. The model utilizes a few essential job characteristics such as the average and maximum phase execution times (e.g., map, shuffle, and reduce phases) during the task processing. We argue that this simple model does work well for heterogeneous environments. Intuitively, in a heterogeneous Hadoop cluster, slower nodes result in longer task executions. These measurements then are reflected in the calculated average and maximum task durations that comprise the job profile. While the bounds-based performance model does not explicitly consider different types of nodes, their performance is implicitly reflected in the job profile and therefore is effectively incorporated in predicting the job completion times. We validate the accuracy of the proposed bounds-based performance model using a diverse set of realistic applications and two different testbeds: i) an Amazon EC2 heterogeneous cluster formed by different capacity VM instances, and ii) the UPenn Hadoop cluster with three different types of nodes. The predicted completion times are within 10% of the measured ones for 12 (out of 13) applications in the workload set. While evaluating the bounds-based performance model, we make a few interesting observations on the behavior of heterogeneous Hadoop cluster where machines vary in capabilities:The number of map and reduce slots per node should be configured dynamically according to the nodes' capacity. In a cluster with a high degrees of heterogeneity (e.g., Amazon EC2's small, medium, and large VM instances), an efficient use of the cluster requires us to vary the number of map and reduce slots per node based on its compute capacity. However, in the UPenn cluster the nodes differ slightly in processor speeds. Therefore, we use a fixed number of map and reduce slots on each node.

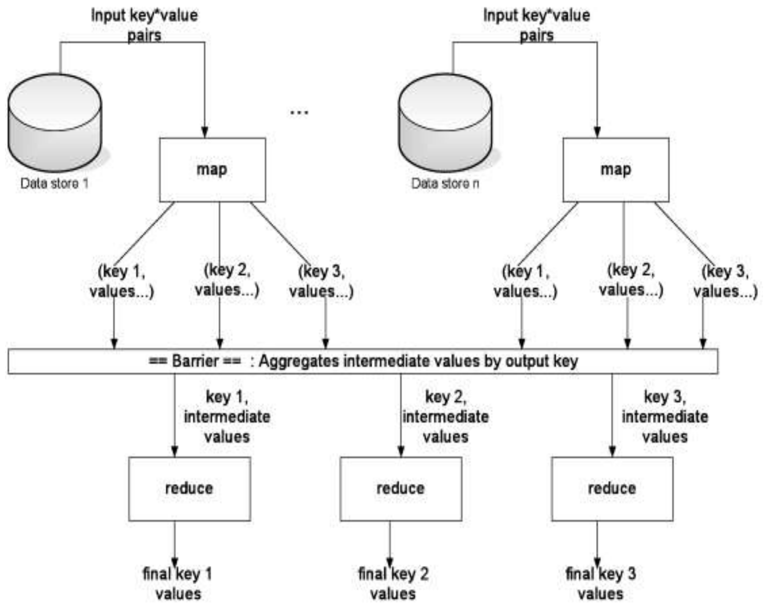

Figure 1: A Map Reduce computation.

A simple "rule-of-thumb" [12] states that the completion time of a MapReduce job is decreased twice when it is processed by two times larger Hadoop cluster. We compare the completion times of 13 applications in our experimental set when they are processed on the EC2- based Hadoop clusters formed by small versus large VM instances respectively. The CPU and RAM capacity of a large VM instance in EC2 is four times larger compared to a capacity of a small VM instance. However, for most applications in our set, the measured speedup is much lower than expected under the traditional "rule-of-thumb" (i.e., 4 times). The reason is that Cloud environments use VM instance scaling with respect to CPU and RAM capacity, while the storage and network bandwidth (that is critical for MapReduce job performance) is only fractionally improved for different capacity VM instances.This further stresses the importance of MapReduce job profiling and performance modeling for making the right business decisions on the choice of the underlying Hadoop clusters. The rest of the paper is organized as follows. Section II provides background information on MapReduce. Section III describes the bounds-based performance model and argues why it works well in heterogeneous environment. Section IV evaluates the accuracy and effectiveness of the proposed approach. Section V outlines related work. Section VI summarizes our contribution and gives directions for future work.

II. Background

This section provides an overview of the MapReduce [3] abstraction, execution, and scheduling. In the MapReduce model, computation is expressed as two functions: map and reduce. The map function takes an input pair (k1, v1) and produces a list of intermediate key/value pairs. The intermediate values associated with the same key k2 are grouped together and then passed to the reduce function. The reduce function takes intermediate key k2 with a list of values and processes them to form a new list of values.

$$\begin{aligned} map(k_1, v_1) &\rightarrow list(k_2, v_2) \\ reduce(k_2, list(v_2)) &\rightarrow list(v_3) \end{aligned}$$

MapReduce jobs are executed across multiple machines: the map stage is partitioned into map tasks and the reduce stage is partitioned into reduce tasks. The map and reduce tasks are executed by map slots and reduce slots. In the map stage, each map task reads a split of the input data, applies the user-defined map function, and generates the intermediate set of key/value pairs. The map task then sorts and partitions these data for different reduce tasks according to a partition function. In the reduce stage, each reduce task fetches its partition of intermediate key/value pairs from all the map tasks and sorts/merges the data with the same key (it is called the shuf- fle/sort phase). After that, it applies the user-defined reduce function to the merged value list to produce the aggregate results (it is called the reduce phase). Then the reduce outputs are written back to a distributed file system. Job scheduling in Hadoop is performed by a master node called the JobTracker, which manages a number of worker nodes in the cluster. Each worker node in the cluster is configured with a fixed number of map and reduce slots. The worker nodes periodically connect to the JobTracker to report their current status and the available slots. The JobTracker decides the next job to execute based on the reported information and according to a scheduling policy. The popular job schedulers include FIFO (First In First Out), Fair [14], and Capacity scheduler [2]. The assignment of tasks to slots is done in a greedy way: assign a task from the selected job immediately whenever a worker reports to have a free slot. If the number of tasks belonging to a MapReduce job is greater than the total number of slots available for processing the job, the task assignment will take multiple rounds, which we call waves. The Hadoop implementation includes counters for recording timing information such as start and finish timestamps of the tasks, or the number of bytes read and written by each task. These counters are written to logs after the job is completed.

III. Bounds-Based Performance Model

In this section, we provide an outline of the bounds-based performance model proposed in ARIA [10] for predicting the Map Reduce job completion time, and then explain why this model does work well in heterogeneous environments. The proposed Map Reduce performance model [10] evaluates lower and upper bounds on the job completion time. It is based the Make span Theorem [10] for computing performance bounds on the completion time of a given set of n tasks that are processed by k servers, (e.g., n map tasks are processed by k map slots in Map Reduce environment). The assignment of tasks to slots is done using an online, greedy algorithm: assign each task to the slot which finished its running task the earliest. Let avg and max be the average and maximum duration of the n tasks respectively. Then the completion time of a greedy task assignment is proven to be at least:

$$T^{low} = avg \cdot \frac{n}{k}$$

and at most $T^{up} = avg \cdot \frac{(n-1)}{k} + max$

The difference between lower and upper bounds represents the range of possible completion times due to task scheduling non-determinism. As motivated by the above model, in order to approximate the overall completion time of a Map Reduce job J, we need to estimate the average and maximum task durations during different execution phases of the job, i.e., map, shuffle/sort, and reduce phases. These measurements can be obtained from the job execution logs. By applying the outlined bounds model, we can estimate the completion times of different processing phases of the job. For example, let job J be partitioned into NJ M map tasks. Then the lower and upper bounds on the duration of the entire map stage in the future execution with S J M map slots (denoted as T low M and T up M respectively) are estimated as follows:

$$T_M^{low} = M_{avg}^J \cdot N_M^J / S_M^J \qquad (1)$$

$$T_M^{up} = M_{avg}^J \cdot (N_M^J - 1)/S_M^J + M_{max}^J \qquad (2)$$

Similarly, we can compute bounds of the execution time of other processing phases of the job.
The reduce stage consists of the shuffle/sort and reduce phases. The shuffle phase begins only after the first map task has completed. The shuffle phase completes when the entire map stage is complete and all the intermediate data generated by the map tasks has been shuffled to the reduce tasks and has been sorted. The shuffle phase of the first reduce wave may be significantly different from the shuffle phase of subsequent reduce waves. This happens because the shuffle phase of the first reduce wave overlaps with the entire map stage, and hence depends on the number of map waves and their durations. Therefore, from the past execution, we extract two sets of measurements: (Sh^1_{avg}, Sh^1_{max}) for the shuffle phase of the first reduce wave (called first shuffle) and (Sh^{typ}_{avg}, Sh^{typ}_{max}) for shuffle phase of the other waves (called, typical shuffle). Moreover, we characterize the first shuffle in a special way and include only the non-overlapping portion (with map stage) in our metrics: Sh1 avg and Sh1 max. This way, we carefully estimate the latency portion that contributes explicitly to the job completion time. The typical shuffle phase is computed as follows:

:

$$T_{Sh}^{low} = \left(N_R^J / S_R^J - 1\right) \cdot Sh_{avg}^{typ} \qquad (3)$$

$$T_{Sh}^{up} = \left((N_R^J - 1)/S_R^J - 1\right) \cdot Sh_{avg}^{typ} + Sh_{max}^{typ} \qquad (4)$$

The reduce phase begins only after the shuffle phase is complete. From the distribution of the reduce task durations of the past run, we compute the average and maximum metrics: R_{avg} and R_{max}. Similarly, the

Makespan Theorem [10] can be directly applied to compute the lower and upper bounds of completion times of the reduce phase (T^{low}_R , T^{up}_R) when a different number of allocated reduce slots S J R is given. Finally, we can put together the formulae for the lower and upper bounds of job completion time:

$$T_J^{low} = T_M^{low} + Sh_{avg}^1 + T_{Sh}^{low} + T_R^{low} \qquad (5)$$

$$T_J^{up} = T_M^{up} + Sh_{max}^1 + T_{Sh}^{up} + T_R^{up} \qquad (6)$$

This simple model should work well in heterogeneous environments. Intuitively, in a heterogeneous Hadoop cluster, slower nodes result in longer task executions. These measurements then are reflected in the calculated average and maximum task durations that comprise the job profile. While the bounds-based performance model does not explicitly consider different types of nodes, their performance is implicitly reflected in the job profile and used in the future prediction.

IV. Related Work

In the past few years, performance modeling in MapReduce environments has received much attention, and different approaches [4], [5], [9], [10], [11] were offered for predicting performance of MapReduce applications. Morton et al. [7] propose an estimator for evaluating the MapReduce job progress. It uses debug runs of the same query on a data sample to estimate the relative processing speeds of map and reduce stages. Their later work [6] propose ParaTimer which extends the work for estimating the progress of parallel queries expressed as Pig scripts that can translate into directed acyclic graphs (DAGs) of MapReduce jobs. Tian and Chen [9] propose an approach to predict MapReduce program performance from a set of test runs on a small input datasets and small number of nodes. By executing 25-60 diverse test runs, the authors create a training set for building a regression-based model of a given application. The derived model is able to predict the application performance on a larger input and a different size Hadoop cluster. However, for each application, a different set of test runs is required to derive the specific model for the application, which limits its applicability in general cases. In Starfish [5] Herodotou et al. apply dynamic Java instrumentation to collect run-time monitoring information about job execution at a fine granularity and extract diverse metrics. Such a detailed job profiling enables the authors to predict job execution under different Hadoop configuration parameters, derive an optimized the cluster configuration, and solve cluster sizing problem [4]. However, collecting a large set of metrics comes at a cost. To avoid significant overhead profiling should be only applied to a small fraction of tasks. ARIA [10] proposes a framework that automatically extracts compact job profiles from past application run(s). These job profiles form a basis of a Map Reduce analytic performance model that computes the lower and upper bounds on the job completion time. ARIA provides a fast and efficient capacity planning model for a MapReduce job with timing requirements. In later work [11], this model is extended with additional framework for deriving the scaling factors that are used for predicting the job completion times of MapReduce applications for processing larger datasets. There is a body of work focusing on performance optimization of MapReduce executions in heterogeneous environments. Zaharia et al. [15], focus on eliminating the negative effect of stragglers on job completion time by improving the scheduling strategy with speculative tasks.

The Tarazu project [1] provides a communication-aware scheduling of map computation which aims at decreasing the communication overload when faster nodes process map tasks with input data stored on slow nodes. It also proposes a load-balancing approach for reduce computation by assigning different amounts of reduce work according to the node capacity. Xie et al. [13] try improving the MapReduce performance through a heterogeneity-aware data placement strategy: a faster nodes store larger amount of input data. In this way, more tasks can be executed by faster nodes without a data transfer for the map execution. Polo et al. [8] show that some MapReduce applications can be accelerated by using special hardware. The authors design an adaptive Hadoop scheduler that assigns such jobs to the nodes with corresponding hardware. One major shortcoming of these existing performance models is that they are valid only for homogeneous Hadoop clsuters and do not take into account resource heterogeneity.

4.1 The Late Scheduler

We have designed a new speculative task scheduler by starting from first principles and adding features needed to behave well in a real environment. The primary insight behind our algorithm is as follows: We always speculatively execute the task that we think will finish farthest into the future, because this task provides the greatest opportunity for a speculative copy to overtake the original and reduce the job's response time. We explain how we estimate a task's finish time based on progress score below. We call our strategy LATE, for Longest Approximate Time to End. Intuitively, this greedy policy would be optimal if nodes ran at consistent speeds and if there was no cost to launching a speculative task on an otherwise idle node. Different methods for estimating time left can be plugged into LATE. We currently use a simple heuristic that we found to work well in practice: We estimate the progress rate of each task as Progress Score/T, where T is the amount of time the task has been running for, and then estimate the time to completion as (1 − P rogressScore)/P rogressRate. This assumes that tasks make progress at a roughly constant rate. There are cases where this heuristic can fail, which we describe later, but it is effective in typical Hadoop jobs. To really get the best chance of beating the original task with the speculative task, we should also only launch speculative tasks on fast nodes – not stragglers. We do this through a simple heuristic – don't launch speculative tasks on nodes that are below some threshold, SlowNodeThreshold, of total work performed (sum of progress scores for all succeeded and in-progress tasks on the node). This heuristic leads to better performance than assigning a speculative task to the first available node. Another option would be to allow more than one speculative copy of each task, but this wastes resources needlessly. Finally, to handle the fact that speculative tasks cost resources, we augment the algorithm with two heuristics:

1. A cap on the number of speculative tasks that can be running at once, which we denote Speculative Cap.
2. A Slow Task Threshold that a task's progress rate is compared with to determine whether it is "slow enough" to be speculated upon. This prevents needless speculation when only fast tasks are running.

In summary, the LATE algorithm works as follows:

1. If a node asks for a new task and there are fewer than Speculative Cap speculative tasks running:
2. Ignore the request if the node's total progress is below Slow Node Threshold.
3. Rank currently running tasks that are not currently being speculated by estimated time left.
4. Launch a copy of the highest-ranked task with progress rate below Slow Task Threshold.

Like Hadoop's scheduler, we also wait until a task has run for 1 minute before evaluating it for speculation. In practice, we have found that a good choice for the three parameters to LATE are to set the Speculative Cap to 10% of available task slots and set the Slow Node Threshold and Slow Task Threshold to the 25th percentile of node progress and task progress rates respectively. We use these values in our evaluation. We have performed a sensitivity analysis in to show that a wide range of thresholds perform well. Finally, we note that unlike Hadoop's scheduler, LATE does not take into account data locality for launching speculative map tasks, although this is a potential extension. We assume that because most maps are data-local, network utilization during the map phase is low, so it is fine to launch a speculative task on a fast node that does not have a local copy of the data. Locality statistics available in Hadoop validate this assumption.

4.2 Estimating Finish Times

At the start of Section 4, we said that we estimate the time left for a task based on the progress score provided by Hadoop, as (1 − P rogressScore)/P rogressRate. Although this heuristic works well in practice, we wish to point out that there are situations in which it can back- fire, and the heuristic might incorrectly estimate that a task which was launched later than an identical task will finish earlier. Because these situations do not occur in typical MapReduce jobs (as explained below), we have

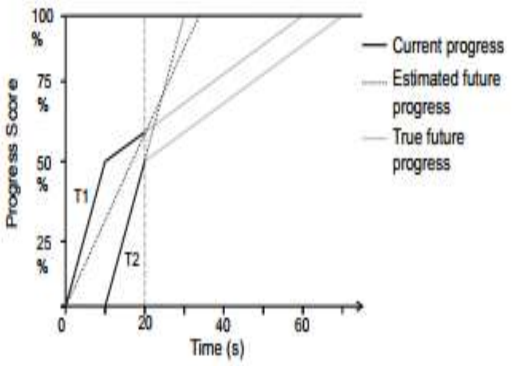

Figure 2: A scenario where LATE estimates task finish orders incorrectly.

Used the simple heuristic presented above in our experiments in this paper. We explain this misestimation here because it is an interesting, subtle problem in scheduling using progress rates. In future work, we plan to evaluate more sophisticated methods of estimating finish times. To see how the progress rate heuristic might backfire, consider a task that has two phases in which it runs at different rates. Suppose the task's progress score grows by 5% per second in the first phase, up to a total score of 50%, and then slows down to 1% per second in the second phase. The task spends 10 seconds in the first phase and 50 seconds in the second phase, or 60s in total. Now suppose that we launch two copies of the task, T1 and T2, one at time 0 and one at time 10, and that we check their progress rates at time 20. Figure 2 illustrates this scenario. At time 20, T1 will have finished its first phase and be one fifth through its second phase, so its progress score will be 60%, and its progress rate will be 60%/20s = 3%/s. Meanwhile, T2 will have just finished its first phase, so its progress rate will be 50%/10s = 5%/s. The estimated time left for T1 will be (100% − 60%)/(3%/s) = 13.3s. The estimated time left for T2 will be (100%−50%)/(5%/s) = 10s. Therefore our heuristic will say that T1 will take longer to run than T2, while in reality T2 finishes second. This situation arises because the task's progress rate slows down throughout its lifetime and is not linearly related to actual progress. In fact, if the task sped up in its second phase instead of slowing down, there would be no problem – we would correctly estimate that tasks in their first phase have a longer amount of time left, so the estimated order of finish times would be correct, but we would be wrong about the exact amount of time left. The problem in this example is that the task slows down in its second phase, so "younger" tasks seem faster. Fortunately, this situation does not frequently arise in typical MapReduce jobs in Hadoop. A map task's progress is based on the number of records it has processed, so its progress is always representative of percent complete. Reduce tasks are typically slowest in their first phase – the copy phase, where they must read all map outputs over the network – so they fall into the "speeding up over time" category above. For the less typical MapReduce jobs where some of the later phases of a reduce task are slower than the first, it would be possible to design a more complex heuristic. Such a heuristic would account for each phase independently when estimating completion time. It would use the the per-phase progress rate thus far observed for any completed or in-progress phases for that task, and for phases that the task has not entered yet, it would use the average progress rate of those phases from other reduce tasks. This more complex heuristic assumes that a task which performs slowly in some phases relative to other tasks will not perform relatively fast in other phases. One issue for this phase-aware heuristic is that it depends on historical averages of per phase task progress rates. However, since speculative tasks are not launched until at least the end of at least one wave of tasks, a sufficient number of tasks will have completed in time for the first speculative task to use the average per phase progress rates. We have not implemented this improved heuristic to keep our algorithm simple. We plan to investigate finish time estimation in more detail in future work.

V. Conclusion And Future Work

Hadoop is increasingly being deployed in enterprise private clouds and offered as a service by public cloud providers. We observe that in many practical deployments today, clusters are grown incrementally over time, and therefore combine heterogeneous resources. In this work, we have focused on the design and evaluation of a performance model for predicting job completion times in heterogeneous MapReduce environments. We have assessed the efficiency and accuracy of the bounds-based performance model using a diverse set of 13 realistic applications and two different testbeds. The predicted completion times are within 10% of the measured ones for 12 (out of 13) applications in the workload set. In our future work, we plan to explore the properties of the job profiles as well as the analysis of the job completion time breakdown to utilize potential performance benefits of the job execution on different types of heterogeneous nodes in the cluster, (e.g., executing the CPU-intensive jobs on the nodes with a higher CPU capacity). We are also going to investigate the strategy that determines the optimal cluster deployment for applications with completion time requirement. Motivated by the real-world problem of node heterogeneity, we have analyzed the problem of speculative execution in MapReduce. We identified flaws with both the particular threshold-based scheduling algorithm in Hadoop and with progress-rate-based algorithms in general. We designed a simple, robust scheduling algorithm, LATE, which uses estimated finish times to speculatively execute the tasks that hurt the response time the most. LATE performs significantly better than Hadoop's default speculative execution algorithm in real workloads on Amazon's Elastic Compute Cloud.

References

[1]. F. Ahmad et al. Tarazu: Optimizing MapReduce on heterogeneous clusters. In Proc. of ASPLOS, 2012.
[2]. Apache. Capacity Scheduler Guide, 2010.
[3]. J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM, 51(1), 2008.
[4]. H. Herodotou, F. Dong, and S. Babu. No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics. In Proc. of ACM Symposium on Cloud Computing, 2011.
[5]. H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. Cetin, and S. Babu. Starfish: A Self-tuning System for Big Data Analytics. In Proc. of 5th Conf. on Innovative Data Systems Research (CIDR), 2011.
[6]. K. Morton, M. Balazinska, and D. Grossman. ParaTimer: a progress indicator for MapReduce DAGs. In Proc. of SIGMOD. ACM, 2010.
[7]. K. Morton, A. Friesen, M. Balazinska, and D. Grossman. Estimating the progress of MapReduce pipelines. In Proc. of ICDE, 2010.
[8]. J. Polo et al. Performance management of accelerated mapreduce workloads in heterogeneous clusters. In Proc. of the 41st Intl. Conf. on Parallel Processing, 2010.
[9]. F. Tian and K. Chen. Towards Optimal Resource Provisioning for Running MapReduce Programs in Public Clouds. In Proc. of IEEE Conference on Cloud Computing (CLOUD 2011).
[10]. A. Verma, L. Cherkasova, and R. H. Campbell. ARIA: Automatic Resource Inference and Allocation for MapReduce Environments. Proc. of the 8th ACM Intl Conf. on Autonomic Computing (ICAC), 2011.

[11]. A. Verma, L. Cherkasova, and R. H. Campbell. Resource Provisioning Framework for MapReduce Jobs with Performance Goals. Proc. of the 12th ACM/IFIP/USENIX Middleware Conference, 2011.

[12]. T. White. Hadoop:The Definitive Guide. Page 6,Yahoo Press.

[13]. J. Xie et al. Improving mapreduce performance through data placement in heterogeneous hadoop clusters. In Proc. of the IPDPS Workshops: Heterogeneity in Computing, 2010.

[14]. M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. In Proc. of EuroSys. ACM, 2010.

[15]. Z. Zhang, L. Cherkasova, A. Verma, and B. T. Loo. Automated Profiling and Resource Management of Pig Programs for Meeting Service Level Objectives. In Proc. of IEEE/ACM Intl. Conference on Autonomic Computing (ICAC), 2012.

Author's Profile

1. H. Shyam Sundhar, Presently working as Assistant Professor in Dept of Computer Science and Engineering, Vivekananda Institute Of Technology & Sciences, Karimnagar

2. Eruguralla Sathish Babau, Presently working as Assistant Professor in Dept of Computer Science and Engineering, Vivekananda Institute Of Technology & Sciences, Karimnagar He is completed B.Tech(CSIT) from BVRIT, M.Tech(VLSI & ES) from KITS-Warangal. He had 10years of teaching experience and Gate Qualified in 2006. His interested areas are Big Data, Data mining, Networking.

3. Bukya Mohanbabu, Presently working as Assistant Professor in Dept of Computer Science and Engineering, Vivekananda Institute Of Technology & Sciences, Karimnagar. He is completed B.Tech(CSE) from PRRM Engineering College, M.Tech (WT) from Aurora's Engineering College. He had 9 years of teaching experience ,Active Member in college activities, web developer, guided to PG students. His interested areas are Data mining, Networking. Big Data.

IOSR Journal of Computer Engineering (IOSR-JCE) is UGC approved Journal with Sl. No. 5019, Journal no. 49102.

H.Shyam Sundhar."Performance Modeling of MAP Reduce Jobs in Heterogeneous Cloud Environments." IOSR Journal of Computer Engineering (IOSR-JCE) 19.6 (2017): 12-18.