Volume 7, Issue 4, April 2017 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Job Attentive Scheduling Algorithm in Hadoop

Prachi Srivastva[1], Hemant Kr Singh[2], Shafeeque Ahmad[3]
[1]Department of Computer Science and Engineering, Azad IET, Lucknow, Uttar Pradesh, India
[2] Head, Department of Computer Applications, Azad IET, Lucknow, Uttar Pradesh, India
[3] Director, Azad IET, Lucknow, Uttar Pradesh, India

Abstract— In recent years cloud services have gained much attention as a result of their availability, scalability, and low cost. One use of these services has been for the execution of scientific workflows as part of Big Data Analytics, which are employed in a diverse range of fields including astronomy, physics, seismology, and bioinformatics. There has been much research on heuristic scheduling algorithms for these workflows due to the problem's inherent complexity, however existing work has mainly considered execution in a utility grid environment using a generic distributed framework. For our research, we consider the ever-increasingly popular Apache Hadoop framework for scheduling workflows onto resources rented from cloud service providers. Contrary to other distributed frameworks, the Hadoop MapReduce model imposes a functional style onto application definition, and as such presents an interesting and unapproached challenge for workflow scheduling. Investigated in our work heterogeneous approach scheduling on the Hadoop MapReduce platform, to minimize workflow makespan while satisfying a given budget constraint.

Keywords— Scheduling Algorithms FIFO, Fair Scheduling, Capacity Scheduling, Hybrid Scheduler Map Reduce Model, Modified Algorithm, Result analysis.

I. INTRODUCTION

Implementation of the proposed algorithm was carried out in Hadoop version 1.2.1. It includes the addition of several packages, with around 8500 lines of code added in total. This included creation of the org.apache.hadoop. mapred.workflow, org.apache.hadoop.mapred. workflow. schedulers,and org. apache.hadoop. mapred. workflow. Scheduling packages. Several existing classes were also modified to allow workflow execution. In this case, the main modifications were made to theorg.apache.hadoop.mapred . core.util.RunJar , org.apache.hadoop.mapred .JobTracker , and org.apache.hadoop. mapred .JobClient classes. For testing of the implementation, classes were also added to the new packages org.apache.hadoop. workflow.examples and org.apache.hadoop. workflow .examples.jobs . The changes made along with the method of execution are reviewed in detail in this section.

The implementation of workflow scheduling attempts to take advantage of the current job scheduling features as much as possible. As a result, it follows the same basic pattern as job scheduling. Modifications include a customized listener which listens for both job events and workflow events (addition, modification, removal), and the scheduler itself which launches executable workflow jobs as well as assigning tasks to querying TaskTrackers. Due to the complexity of workflow scheduling, our assumptions of known data, and the constraint requirements, much of the scheduling decisions are delegated to an additional SchedulingPlan class. The class is instantiated on the client machine during workflow submission, where it generates the plan based on cluster properties. The plan is then passed to the scheduler (via the JobTracker) during workflow submission, and used to decide how to run both workflow jobs and their composite tasks

II. SCHEDULING ALGORITHMS

There are many algorithms to address issues with different techniques and approaches. Some of them get focus to improvement data locality and some of them implements to provide Synchronization processing. Also, many of them have been designed to minimizing the total completion time.

FIFO Scheduling Algorithm

FIFO is the default Hadoop scheduler. The main objective of FIFO scheduler to schedule jobs based on their priorities in first-come first-out of first serve order. FIFO stands for first in first out which in it Job Tracker pulls oldest job first from job queue and it doesn't consider about priority or size of the job [1].

When the scheduler receives a heartbeat indicating that a map or reduce slot is free:

- It scans through list of jobs to and a job with highest priority and then oldest submit time. If this job has a pending task of the slot type (map/reduce) then that job is selected. Else next job in this order is picked. This goes on till a matching task (type as per slot) is found.
- Next if it's a map slot, then to achieve data locality the scheduler consults NameNode and picks a map task in the job which has data closest to this free slave (on the same node, otherwise on the same rack, or on a remote rack).

- If it's a reduce slot, any of the reduce task is scheduled on the node. Hadoop MapReduce doesn't wait for all map tasks to finish for scheduling a reduce task, so the map task execution and shuffling of intermediate data can run in parallel to have better turn-around time. This is known as early-shuffle.

FIFO scheduler have many limitations such as: poor response times for short jobs compared to large jobs, Low performance when run multiple types of jobs and it give good result only for single type of job. To address these problems scheduling algorithms such as Fair and Capacity was introduced.

Fair Scheduling Algorithm

The FAIR scheduler by Zaharia et. al. [8] has been optimized for multi-user environments, where a single cluster is shared across a number of users. This kind of Hadoop usage is popular in companies that do a lot of data mining operations based on user logs. The FAIR scheduler has been designed to reduce the makespan of short jobs, which are found to be frequent in such large environments [5].

The scheduler chooses jobs from a set of pools, and tries to assign jobs fairly across pools. Task preemption is also supported in order to achieve fairness. The authors use a version of max-min fairness with a minimum slot allocation guarantee. If jobs from a pool have been over assigned, tasks of those jobs are killed in order to free slots for jobs with less than guaranteed allocations.

The FAIR scheduler uses delayed allocation to improve data locality on large clusters.

While allocating a task, unlike the native Hadoop scheduler which uses a best effort philosophy, the FAIR scheduler delays the allocation of a task in the hope that a node with more data locality might ask for that task. If the job at the head of the job queue does not have a data local task, then subsequent jobs with data local tasks are chosen for assignment. If the head of the queue has not been getting a data local task for a specific period of time, then it is forcefully allocated. The authors calculate expected gain in job's response time by using delay scheduling to be:

$$E(\text{gain}) = (1 - e^{t})(D - t)$$

Where t is the rate of arrival of allocation requests from nodes (heartbeats), which is the rate parameter of a Poisson process, D is the expected extension in runtime of a non-data local task compared to a data local task, and w is the wait time for launching tasks locally.

Two locality problems can occur with fair scheduling –

- The head-of-line scheduling - The head-of-line problem is that the job at head-of-line has low data locality then most of its tasks cannot be scheduled on local nodes. This problem mostly occurs for small jobs, which have small input les and hence has less number of data blocks to read. The probability of nding their data on a given node is low whenever they come to head-of-line, still some of their tasks get scheduled.
- The sticky slots - This problem can happen for both small as well as large jobs. The problem is that there is a tendency for a job to be assigned the same slot repeatedly. Suppose multiple jobs each having 10 tasks are submitted together on 10 node cluster. So each job is allocated 1 node, in order of their arrival. When job J finishes a task on node N, Node N requests a new task. At this point, J has 9 running tasks while all the other jobs have 10. The nave Fair scheduling algorithm assigns that slot on node N to same job J again. Consequently, in steady state, jobs never get to leave a slot/node. This leads to poor data locality as the input les are striped across the cluster while all tasks of job are executed on same machine.

Capacity Scheduling Algorithm

Capacity Scheduler [5] is another scheduler designed for sharing of large clusters. The scheduler defines a concept of queues, which are created by system administrator for submitting jobs. Each queue is guaranteed a fraction of the capacity (number of task slots) of the entire cluster. All jobs submitted to a given queue have access to the resources guaranteed for that queue. In a single queue jobs are selected based on priorities. Higher priority jobs are selected before low priority tasks, however, jobs with lower priority are not preempted for higher priority tasks, which could result in priority inversion within a queue.

If there are multiple queues and only a subset of these queues are having running jobs, then there is facility to allocate more than the guaranteed capacity to the subset of active queues. If and when the inactive queues also have job submissions, then their lost capacity is reclaimed. For this tasks of jobs in queues with excess capacity are killed.

The scheduler also supports resource aware scheduling for memory intensive jobs. A job could optionally indicates if it needs nodes with more memory. Tasks of such a job are only allocated to nodes that have more than requested amount of free memory.

Whenever a request for a task comes from a TaskTracker, the scheduler chooses a queue that has the large free capacity, i.e. the queue whose ratio of number of running tasks to the number of guaranteed tasks is the lowest. Task quota for users is also supported. While selecting a job, they are usually chosen in FIFO order, only if the quota of the user of the job is not reached, in which case the next job in the queue is chosen.

III. HYBRID SCHEDULER BASED ON DYNAMIC PRIORITY

Nguyen et al. [22] propose a Hybrid Scheduler algorithm based on dynamic priority in order to reduce the delay for variable length concurrent jobs, and relax the order of jobs to maintain data locality. This algorithm is designed for data intensive workloads and tries to maintain data locality during job execution [11]. They believes, average response time for the workloads approximately 2.1x faster over the Hadoop Fairs with a standard deviation of 1.4x.

It achieves this improved response time by means of relaxing the strict proportional fairness with a simple exponential policy model. This algorithm is a fast and flexible scheduler that improves response time for multi-user Hadoop environments.

IV. MAP REDUCE MODEL

Hadoop: Hadoop is an open source framework that provides reliable, scalable, distributed processing and storage on large clusters of inexpensive servers. Hadoop is written in Java and users can customize the code to parallelize the data process in clusters which contains thousands of commodity servers. The response time depends on the complexity of process and volumes of data. The advantage of Hadoop is its massive scalability.

The Apache Hadoop framework consists of Hadoop kernel, MapReduce, Hadoop Distributed File System (HDFS) and some related projects like HBase, Hive and Zookeeper. At present, Hadoop plays a major role in Email spam detection, search engines, genome manipulation in life sciences, advertising, prediction in financial service and analysis of log files. Linux and Windows are a preferred operating system for Hadoop.

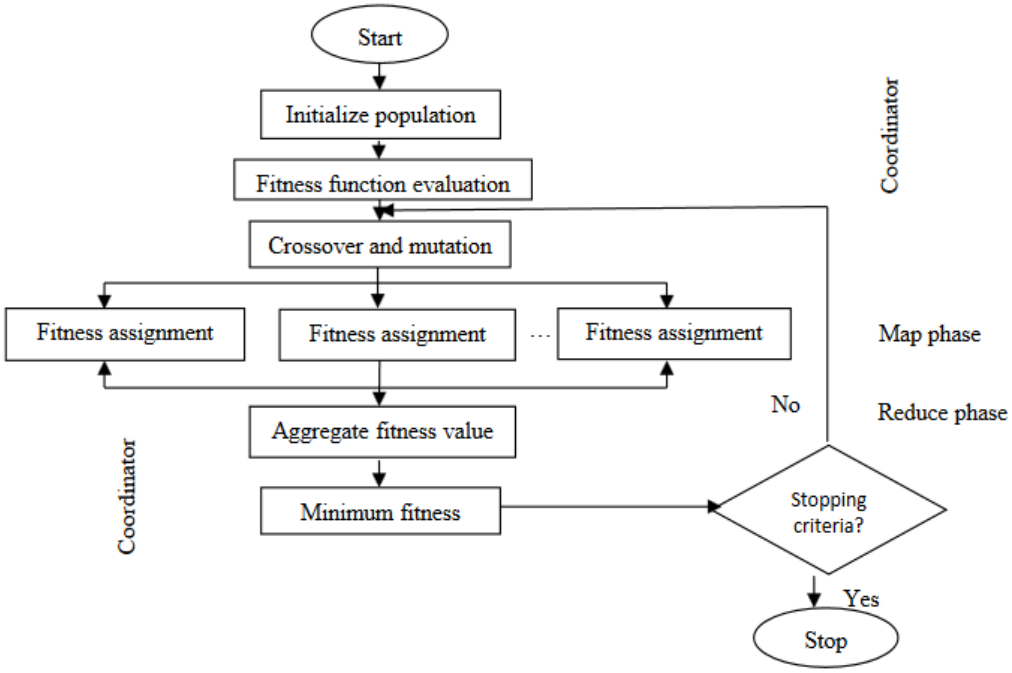

Figure1: Flowchart for MapReduce

V. MODIFIED HETEROGENEOUS

ALGORITHM

1: Procedure schedule (G,M,TP,B)
2:cost= 0
3:for I ∈G.nτ do
4: G.τi.m=Mnm−1
5:G.τi.p=TP.p(G.τi,G.τi.m)
6:G.τi.t=TP.t(G.τi,G.τi.m)
7:cost+=G.τi.t
8:end for
9:B−=cost
10:ifB <0then
11:returnNone
12:end if
13:whileB≥0do▷ O(nτ×nm)
14:update stage times(G)▷ O(|V|+|E|+nτ)
15:distances=calculatecritical(G)▷ O(|V|+|E|)
16:S critical =get critical stages(G,G.τexit,distances)▷ O(|V|+|E|)
17:v=∅
18:for S s ∈ S critical do▷ O(|V|)
19: (Γ,γ) = (S s .slowest,S s .secondslowest)
20:v[Γ]=min{(TP.t(Γ,G.τΓ.m)−TP.t(Γ,G.τΓ.m−1)),(TP.t(Γ,G.τΓ.m)−TP.(γ,G.τγ.m))}TP.p(Γ,G.τΓ.m−1)−TP.p (Γ,G.τΓ.m)
21:end for
22: while v =∅ do▷ O(|V|log|V|)
23: Γ = max τ ∈ v.keys () {v[τ]}

24:new= TP.p(Γ,G.τΓ.m−1)
25:old = TP.p(Γ,G.τΓ.m)
26:ifB <(new−old)then
27:v.remove(Γ)
28:else
29:G.τΓ.m−= 1
30:G.τΓ.p=TP.p(Γ,G.τΓ.m)
31:G.τΓ.t=TP.t(Γ,G.τΓ.m)
32:B−= (new−old)
33:break
34:end if
35:end while
36:ifv=∅then
37:returnG
38:end if
39:end while
40:end procedure

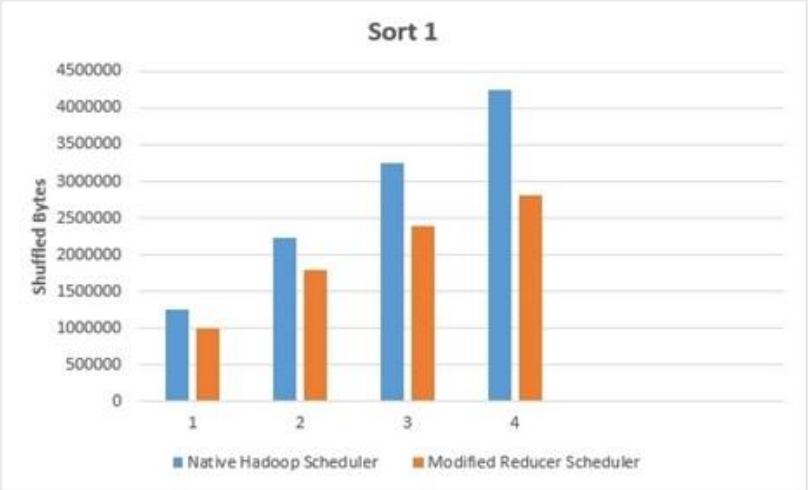

Figure 2: Byte shuffling differences for Sort1

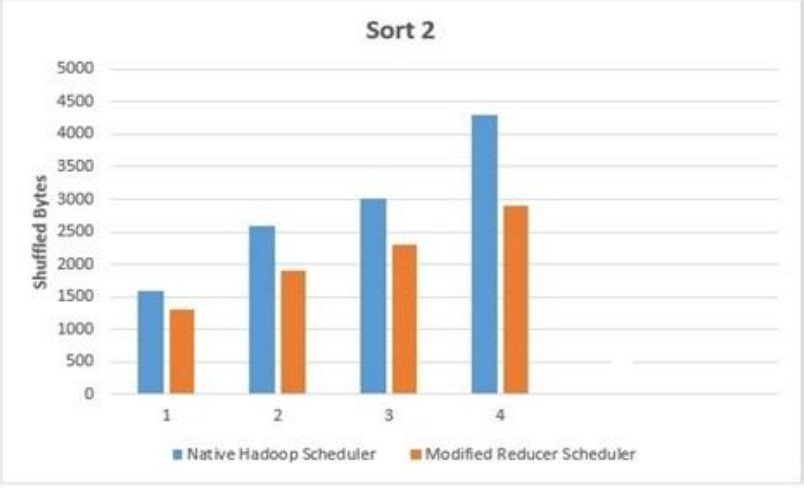

Figure 3: Byte shuffling differences for Sort2

We executed sort job in two (sort1, sort2) different scenarios, that is data sets size range, to get more accurate result. We repeated all the experiments 4-times

As we can see in the Figure 2, Figure 3, Figure 4, Figure 5, Figure 6.Proposed algorithm reduce task scheduler outperform the native Hadoop by 20 to 80%. Modified scheduler can make more accurate decision than native Hadoop. It leads to minimum shuffling and so does minimum network congestion. Execution time have reduced for MapReduce jobs in the Hadoop distributed environment.

The variations in the rate of byte shuffling is due to the varying size of intermediate records, the number of mappers and the number of reducers. We took data sets of various size between 100MB to 500MB. In our experiments the mapper and reducers number varies in between 1 to 8.

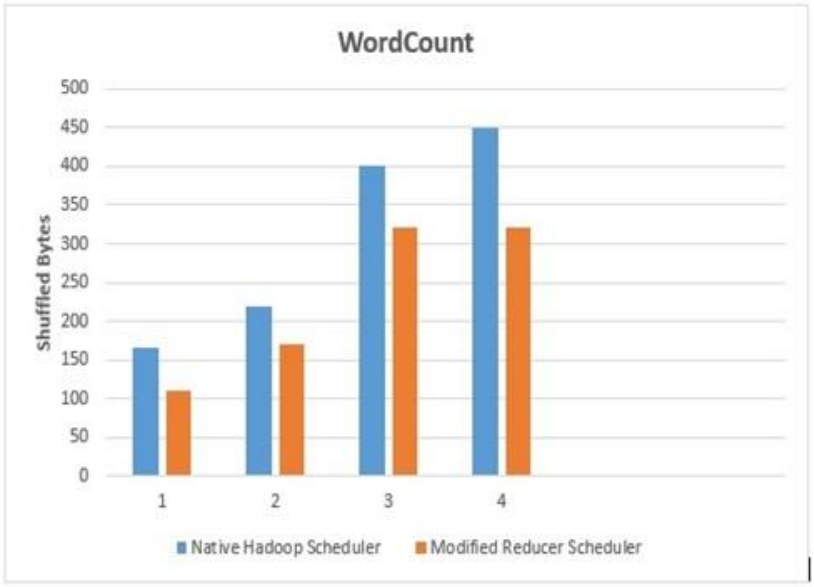

Figure 4: Byte shuffling differences for WordCount

Figure 5: Byte shuffling differences for Multi-File WordCount

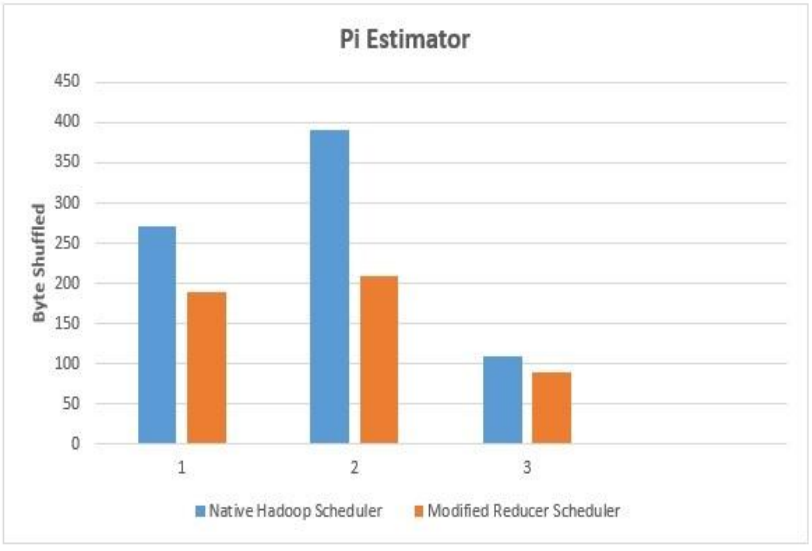

Figure 6: Byte shuffling differences for Pi Estimator

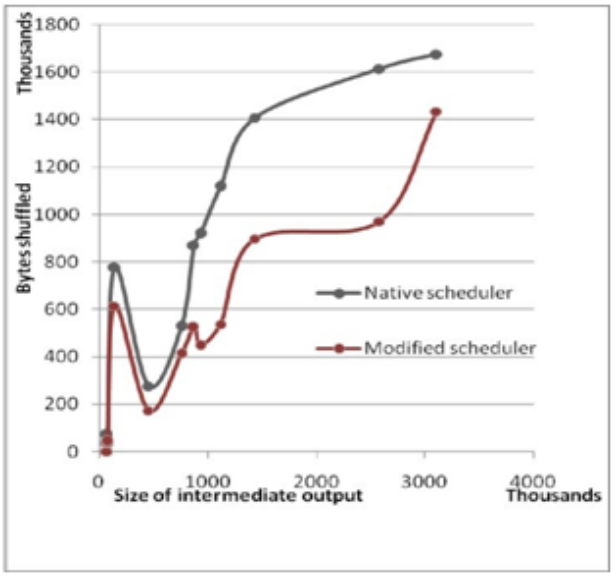

Figure 7: Reduction in Byte Shuffled

Further, as shown in Fig.7, the reduction in bytes shuffled grows with an increase in intermediate output and as a result with increase in the number of bytes input to the MapReduce program .The fluctuation is due to varying number of mappers and reducers. The modified Hadoop scheduler minimizes data-local traffic by 11-80 %.

VI. CONCLUSION

The introduction of the modified heterogeneous algorithm was yet another evolution in cluster computing with Hadoop. It permits the use (and development) of schedulers optimized for the particular workload and application. The new schedulers have also made it possible to create multi-user data warehouses with Hadoop, given the ability to share the overall Hadoop infrastructure with multiple users and organizations.

Hadoop is evolving as its use models evolve and now supports new types of workloads and usage scenarios (such as multi-user or multi-organization big data warehouses). The new flexibility that Hadoop provides is a great step toward more optimized use of cluster resources in big data analytics.

REFERENCES

[1] Braun,T.D., Siegel, H.J, Beck,N., Boloni,L.L., Maheswaran,M., Reuther, A.I., Robertson, J.P., Theys, M.D. and Yao,B. (2001) Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. Journal of Parallel and Distributed Computing, 61,810-837. http://dx.doi.org/10.1006/jpdc.2000.1714.

[2] Subashini,G. and Bhuvaneswari, M.C. (2011) Non Dominated Particle Swarm Optimization for Scheduling Independent Tasks on Heterogeneous Distributed Environments. International Journal of Advances in Soft Computing and Its Applications, 3, 1.

[3] Kaiwartya,O., Prakash,S., Abdullah, A.H. and Hassan,A.N. (2015) Minimizing Energy Consumption in Scheduling of Dependent Tasks Using Genetic algorithm in Computational Grid. KSII Transactions on Internet and Information Systems, 9.

[4] Abraham,A., Liu, H., Zhang, W. and Chang,T.G. (2006) Scheduling Jobs on Computational Grids Using Fuzzy Particle Swarm Algorithm. Springer-erlag , Berlin, Heidelberg, 500-507. http://dx.doi.org/10.1007/11893004_65

[5] Deb,K. (2002) Multi-Objective Optimization Using Evolutionary Algorithms. Wiley Publications.

[6] Osycaka,A. and Kundu,S. (1995) A New Method to Solve Generalized Multicriteria Optimization Problems Using the Simple Genetic Algorithm. Structural Optimization, 10, 94-99. http://dx.doi.org/10.1007/BF01743536

[7] Wang,F., Qiu,J., Yang,J., Dong,B.,Li, X. and Li,Y. (2009) Hadoop High Availability through Metadata Replication. In: Cloud, D.B., Ed., Proceedings of the First International Workshop on Cloud Data Management. http://dx.doi.org/10.1145/1651263.1651271

[8] Munir,E.U., Li, J.-Z ., Shi, S.-F . and Rasool,Q. (2007) Performance Analysis of Task Scheduling Heuristics in Grid.

[9] Wenbin Fang, Bingsheng He, Qiong Luo, and Naga K. Govindaraju. Mars. 2011. Accelerating mapreduce with graphics processors. *IEEE Trans. Parallel Distrib. Syst.*, 22:608–620.

[10] Apache Software Foundation. Dynamic priority scheduler for hadoop. http://issues.apache.org/jira/browse/HADOOP-4768.

[11] Apache Software Foundation. A fair sharing job scheduler. http://issues.apache.org/jira/browse/HADOOP-3746.

[12] Apache Software Foundation. Hadoop. http://hadoop.apache.org/hadoop.

[13] Apache Software Foundation. The hbase project. http://hadoop.apache.org/hbase.
[14] Apache Software Foundation. The hive project. http://hadoop.apache.org/hive.
[15] Apache Software Foundation. The pig project. http://hadoop.apache.org/pig.
[16] Eric Friedman, Peter Pawlowski, and John Cieslewicz. 2009. Sql/mapreduce: a practical approach to self-describing, polymorphic, and parallelizable user-defined functions. *Proc. VLDB Endow.*, 2:1402–1413.

AUTHOR'S PROFILE

Prachi Shrivastava pursuing M.tech Computer Science from Azad IET Lucknow . Completed B.tech from Computer science and engineering in 2013 from MIT Bulandshahr. My area of interest includeds Hadoop, MapReduce, heterogeneous cluster and Scheduling Algorithem.

