International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-8 Issue-6, August, 2019

DAG-CPM Scheduler for Parallel Execution of Critical Jobs

D C VINUTHA, G T RAJU

Abstract*: *MapReduce applications having multiple jobs may be dependent on each other such as iterative Page View application [2] performs the required operation in several iterations before generating the result. Each iteration is considered as single job. Conventional Hadoop MapReduce schedules the jobs sequentially, but not customized to handle multi job application. Also, it will not perform the parallel execution of the dependent jobs. This prolongs the execution time to complete all the jobs. Therefore a new scheduler DAG–CPM Scheduler uses the critical path job scheduling model, to identify the jobs present in the critical path. Critical path job scheduling is optimized to offer support for multi job applications, critical path job is a series of jobs, if execution of a job is delayed, then time required to execute all jobs will be prolonged. DAG–CPM Scheduler schedules multiple jobs by dynamically constructing the job dependency in DAG for the currently running job based on the input and output of a job. DAG represents the dependency among the jobs, this dependency graph is used to insert a pipeline between the output of one job as input for map tasks of another job and it executes the dependent jobs in parallel which results into a substantial reduction in the execution time of an application. Experimental analysis on the proposed approach has been carried out on Page View application on Academic and research web server log file, such as, NASA and rnsit.ac.in of 10 GB data set. PigMix2 is executed on 8GB data set. Experimental results reveal that the average execution time is decreased by 41% compared to Hadoop in respect Page View application and Execution speed is 37.7% faster compared to Pig and DAG–CPM Scheduler can run 24.3% faster when compared to DAG–CPM Scheduler without pipeline.

Keywords*: *Critical Path, DAG, MapReduce, Pipeline.

I. INTRODUCTION

MapReduce is a programming model to perform the computation in a parallel and distributed environment [1]. It is used in most popular applications like scientific computing, log analysis etc. These applications may have multiple jobs with dependency among each other. Page view application [2] performs the required operations in several iteration, each iteration is considered as a single job, and it emits the output and executes the job sequentially. Conventional MapReduce will not execute the dependent jobs in parallel, as it does not provide the synchronization between the dependent jobs, because it is necessary for the jobs to have the complete input data before being submitted. Conventional MapReduce Framework will not schedule the multiple jobs to execute in parallel [5-8], because it is not preserving the information of dependency between the jobs. Here overlapping of dependent jobs is not taken into consideration. Few software's such as Pig [7]for apache Hadoop[8], Tenzing for MapReduce[11],Scope[9] for Microsoft Dryad[10],Hive[12] and RoPE [13] these software's converts the query in to a DAG by analyzing the dependency among the jobs, to schedule the multiple dependent jobs for parallel execution. These software's' has their own schedulers/ scheduling policies and cannot use the schedulers/scheduling policy of different software. The main objective of this paper is to execute the maximum number of dependent jobs in parallel to reduce the overall completion time.

MapReduce follows master/ slave architecture, master node is the Job tracker, which supervises the resources of all the slave nodes, schedules the map/reduce tasks for the submitted jobs. After the user submits a job, the client fragments the input data into multiple splits of size 64 MB, these input data blocks are distributed and stored in the data nodes of cluster and then the Job tracker launches the map task to process the input split, the number of map task is equal to the total number of input splits. Map task generates the intermediate data as <key, value> pair, these are grouped in to partitions using hash function. The map task outputs are shuffled in reduce phase, sorted and merged by reduce tasks to generate the final output. Conventional MapReduce executes the submitted jobs only after receiving the complete input data from the dependent jobs. This will have an effect on overall run time.

The contributions for this paper are as follows

1. Enhanced the conventional MapReduce, to execute the scheduled dependent jobs in parallel, by dynamically constructing the dependency DAG for the currently running job based on the output and input directories.
2. To overlap the execution of the dependent jobs, Pipeline is introduced /inserted between the output and input of the dependent jobs, this result into significant improvement in the execution time.
3. Execution time of each job is estimated using Locally Weighted Liner Regression (LWLR) by utilizing job log files / execution history.
4. Developed a critical path job scheduling model for job scheduling based on the precise prediction of the critical path using Critical Path Method algorithm.

Revised Manuscript Received on October 30, 2019.
* Correspondence Author
D C V*, Research Scholar, Dept. of CSE, RNS Institute of Technology, Bengaluru, Associate Professor, Dept. of ISE Vidyavardhaka College of Engineering, Mysuru. Visvesvaraya Technological University, Belagavi, Karnataka.
G T R, Professor, Dept. of CSE, RNS Institute of Technology, Bengaluru, Visvesvaraya Technological University, Belagavi, Karnataka.

II. RELATED WORK

Scheduling in MapReduce is classified as coarse grain and fine grain scheduling. Several prior studies for fine grain scheduling provided certain strategies/policies for various functions. Fair Scheduling [8] allocates equal resources among for all the users. Capacity scheduling [8] facilitates the fairness for resource-intensive jobs. Quinchy [5], Delay scheduling [6] concentrates on correlation between data locality and fairness. The author in [4] emphasizing on classifying the users, multiple resource scheduling [25] focuses on heterogeneous requirements of resources and fairness between the users. LATE [18] and MCP [17] emphasizes on straggler tasks. To provide support for intricate queries, various methods offers easy-to-use SQL like languages to streamline MapReduce programming for producing multi job workloads. For example, Pig [7] for Apache Hadoop[8], Scope [9] for Microsoft Dryad[10], and Tenzing [11] for Google MapReduce use conventional query enhancement policies from the database area to translate a query into tree and then uses the bottom-up manner to traverse the tree for converting the tree in to implementation plan. This method offers scheduling to defend the dependency relationship between multiple jobs at a higher level compared to MapReduce. Hence, jobs are not scheduled based on the execution information of job or introduce a pipeline between the dependent jobs. Several works aims on improving multiple-job workload scheduling. Hive-QL [12], YSmart [26], MRShare [27], Starfish [28], RoPE [13] and stubby[29] all these utilized a sequence of hand-crafted procedures to further enhance the job execution strategies. HiveQL executes multiple passes above the logical plan using certain optimization procedures. YSmart interprets queries depending on four job primitive kinds: selection-projection, join, aggregation and sort. Then it combines the MapReduce jobs based on their primitives to curtail the number of jobs. MRShare joins related jobs into set and evaluate every set as a single job depending on the cost model. Starfish offers only a cost-based pattern enhancement for group of jobs. RoPE [13] adjusts execution strategy created on expectations of code and data properties.

MapReduce online [30] and HPMR [31] introduces data pipeline between map and reduce phase, for adaptive load balancing. Parallel execution of dependent jobs is not taken in to account. For Data intensive iterative application, extended MapReduce is an Haloop [32], it provides support for various stages in a job and different caching mechanisms are introduced to support loop-aware task scheduling.

III. PROPOSED SYSTEM

DAG–CPM Scheduler is an enhancement of conventional Hadoop to schedule the multi-job applications. The same workflow ID is used assigned for multiple jobs of an application and these jobs are submitted using the same ID and sent to the Job Tracker simultaneously. Figure 1 shows the proposed system architecture which contains three new components:

– Dependency Parser examines the dependency between the multiple jobs based on the input and output directories.
– Job Time Estimator approximates job execution time based on the jobs log files and configuration of the job.
– To make a robust scheduling decision for the submitted jobs, it utilizes the predicted job execution time and dependency information in DAG.

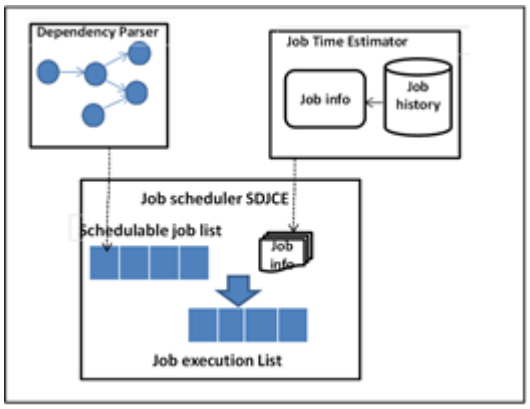


Fig 1. Proposed Architecture

3.1 Dependency Management

The Higher level software like Pig [7], Scope [9], Tenzing [11] analyzes job dependency by converting query into a tree, then translates the tree into executable code by visiting the tree in a bottom-up fashion. In DAG - CPMS, user utilizes the same workflow id to submit MapReduce application having multiple jobs', these jobs are submitted and send to the Job tracker in parallel. After complete submission of the jobs, job dependency DAG is created dynamically depending on the input and output directories, for the jobs whose input data is ready, these jobs be stored into the schedulable job list. Upon on the arrival of new job or completion of a currently running job, the dependency parser will update the dependency DAG.

In some situation, there may be a dependency among the previous iteration and the current iteration job, even though there is no input–output dependency. In such case, we need to insert a dependency edge among these jobs. This is achieved by analyzing every read path of jobs in HDFS with output path of other jobs. This type of dependency is called "Hard Dependency".

3.2 Data pipelining between the input and output of a job

In conventional MapReduce, there is no synchronization between input and output of the dependent job, job dependency information in DAG is not maintained. Therefore it is not possible to identify, to which job the output of an upstream job need to be used as an input for downstream job, which is stored in the scheduler queue. As a result, the dependent job waits till it receives the complete input data from all other jobs. In this proposed work, pipeline is used between the dependent jobs, as map task is required to handle the fixed size of continuous data blocks. As there is no dependency among the map tasks within a job, it immediately launches the appropriate map task, after a portion of the output data is generated from an upstream job and it is forwarded quickly using a pipeline to be used as an input for the downstream job. In this work, dependency parser creates the dependency DAG dynamically for MapReduce application, this can be used to identify to which job the upstream job output data need to be pipelined as an input data for a downstream job.

DAG–CPM Scheduler notices the output generated from an upstream job and then informs all the downstream jobs to launch new map task. This notification has starting and ending offset of a block, and a file name to read. Jobs are added to the scheduler list, only after all the upstream jobs enter the output phase and at least a block of data is generated from the dependent job. For hard dependent jobs, after completing the hard dependent upstream jobs, the downstream jobs are placed in to the schedulable job list and the remaining upstream jobs move to the output phase. After executing the entire dependent jobs, *DAG–CPM Scheduler* informs the number of maps task to the downstream job reduce tasks, the map task outputs are shuffled, sorted and merged in the reduce phase to produce the final result.

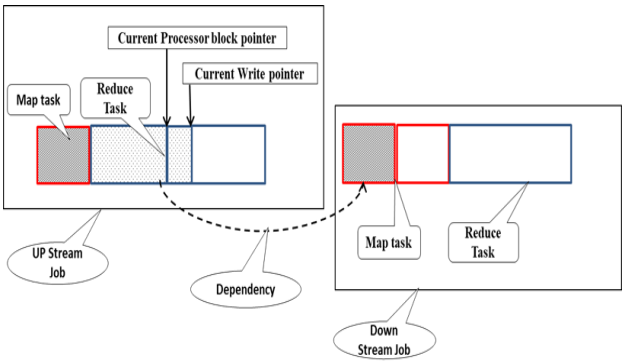


Fig 2. Data Pipeline between Dependent jobs

Figure 2 describes the process of data pipeline used to send the output of a job as an input for other job. In this proposed work, Data intensive application like Page view is used for experimental analysis. To guarantee, the correct execution order of the downstream jobs and not to slip a map inputs or addition of map task should not process the same data block. For every output task in the upstream jobs the current processed block pointer is maintained. Map task is launched for the downstream job, if a pointer is increased. The pointer is always increased, never it will be decreased, this guarantees that always the map tasks processes the different block. Every time the pointer is incremented by a block length, to guarantee that all the data blocks are processed. The reduce tasks are not started till it receives all the map outputs.

Data pipeline is not used between the hard dependent jobs, because all the map tasks existing in a downstream jobs requires to read complete output of "hard dependent" upstream jobs. The downstream jobs are included into the job scheduler queue only after all of its upstream jobs are finished. During pipeline insertion "hard dependency" case may be ignored as this type of data dependency are very rare and also will not result into major impact on the execution time.

3.3 Scheduling in DAG–CPM Scheduler

The Job Scheduler in conventional MapReduce is enhanced by utilizing the execution information of a job and inserting a pipeline between the dependent jobs. Therefore *DAG–CPM Scheduler* can be used in MapReduce [15] Framework directly and also, it can be used in a high level software's, like Pig, Hive, Scope etc. The procedure to schedule the job using *DAG–CPM Scheduler* is given below.

- User utilizes the same id to submit MapReduce applications having multiple jobs' into *DAG–CPM Scheduler*, these jobs can be submitted and send to the Job tracker in parallel.
- After the complete submission of all the jobs, job dependency DAG is created dynamically and approximates the jobs execution time.
- If currently running job queue is empty and free slots are available in the system, then the scheduler will select the job present in the critical path.

3.3.1Problem Statement

Consider the MapReduce application, the number of schedulable jobs is N. Every job i has X_i map tasks and Y_i reduce tasks and average run time of map and reduce task is AT_i^M and AT_i^R. G = (V,E) represents the DAG, the vertex V denotes the job i.e. (V =N), directed edge (U,V) is inserted between the nodes U And V, iff, The output of job U is used as an input for job V. MapReduce system has several Task trackers with s slots. The Scheduling arrangement for dependency graph G of a job is denoted as queue of job vertices: $j_1, j_2, \ldots\ldots j_n$, which fulfills the dependency constraint. The main objective of proposed scheduling is to enhance the system of parallelism and reduce the completion time of an application.

If a new job is dynamically submitted with the same ID, it will change the dependency of jobs in DAG, in the meantime unpredicted situations may occur in real time system, such as, some tasks may get straggled due to competition for the resources among the tasks, in such situation, the speculative copy of the stragglers tasks will be executed, this leads to an impact on the best possible solution. Therefore, the scheduling is streamlined using the procedure given below.

- Makes decision only on the order of the job execution as job and task scheduling of a job are separated.
- To schedule the job's, job scheduling list is not prepared in the beginning. Instead job scheduling list is obtained using iterative method. Therefore the proposed approach satisfies the constraints of dynamic job submission and abides insecurity through real time execution. In every iteration, the jobs to be scheduled in the pending list are sorted and select the suitable job to include for the executable job list.

3.3.2Job Execution Time Estimation

After submitting the MapReduce application, first it estimates every job run time to make scheduling decisions of all the jobs. Job run time is computed dynamically in [16-18].but these will not meet the proposed scheduling requirements. Approximate the run time for every the job's before running. To register the execution time for every job *DAG–CPM Scheduler* uses the job history data base. Job log file is used to approximate the execution time for the frequently occurring jobs. Job names are used to differentiate various types of jobs [25-28]. To calculate the run time of each job and downstream jobs input data size, two parameters have to be predicted are, the speed of processing map and reduce phase for each task and the output data size. Processing speed is the ratio input data size to the time spent in every phase. In this work, Locally Weighted Linear Regression (LWLR) [14] model is used for approximating the execution time.

Fraction of observations are used for estimations, instead of using the complete data set.it performs the estimation using a fraction of observation rather than using the complete data set, LWLR calculates the target *E* at *I* using m observations whose I_i values are close to *I* and weights are assigned for these *m* observations depending on the distance from *I*, higher weight is given in the estimation for the observations where input data size is closer to the present input data size. Recent observations are selected to make the precise estimation, as recent observations denote the current status of the system. Weight for the observation is computed using equation 1.

$$W_i(I) = \begin{cases} \frac{1}{\text{rank}(\text{dist}(I,I_i)) + \text{time}(I,I_i))} & \text{rank}(\text{dist}(I,I_i)) \le m \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Where rank (dist (I, I_i)) denotes the rank of the distance between the observation Ii and I (higher value for large distance), time (I, I_i) denotes the gap between the ongoing id and observation id i.

Weights are assigned only for m observations, where Ii values are very near to I. using weight function given in equation 1, highest weights obtained during estimation are assigned to the observations which are near to I in terms of both distance and time. To estimate the dependent jobs execution time, approximate the input data size for every job as a first step. LWLR model is used to estimate the size of input, output data and to approximate the runtime of every phase for the jobs whose input data is ready. If input data is not ready, the size of the input data is approximated first based on the dependent jobs output data size and the input data size is used to approximate the execution time of every phase and the output data size. If job log file is not available, then mean value of all the current jobs is used to approximate the run time of each phase and size of output data.

MapReduce has many map and reduce tasks. Map and combine are the two phases in map task, shuffle, reduce and sort are the three phases of reduce task. The estimation of run time of a job is given in equation 2.

ET_job = NR_map*(ET(map)+ET(combine)) +
NR_reduce (ET(shuffle)+ET(Sort)+ET(Reduce)) (2)

ET(X) denotes the run time of map, combine, shuffle, sort and reduce phase. NR_{map} and NR_{reduce} denotes the total number of rounds of reduce and map tasks. The shuffle phase and map phase almost runs in parallel, the first round of the shuffle phase usually completes approximately when the map tasks finishes. After finishing all the map tasks, the remaining rounds of the shuffle phase will be executed very quickly. Hence equation 1 is simplified by ignoring the shuffle time, which is given in equation 3.

ET_job = NR_map* (ET(map)+ET(combine)) +
NR_reduce*(ET(sort)+ET(Reduce)) (3)

Typically, the map and reduce tasks of a job executes in several rounds, all the slots are utilized for execution, except in the last round. As a result, the run time of either the map or the reduce tasks is to be consider in the last round to make the scheduling decisions. This method aims to prevent significant decrease in the system of parallelism, in the last round if the number of tasks is small and has longer run time. The jobs present in the critical path with the longest total last round execution time are scheduled first; As a result, wider options will be available to fill up the free slots in the last rounds. The run time of map and reduce task of only the last round is taken into account to compute the run time of a job, which is given in equation 4.

ET_{job}=ET(Map)+ET(combine)+ET(sort)+ET(reduce) (4)

In this work, using data pipeline the downstream job map task and upstream job reduce task works in parallel. Hence, either the run time of map task of downstream map job or reduce task of upstream job is required to be computed in this proposed work, hence equation 4 in divided in to 2 parts to estimate the job execution time, which in given in equation (5)

ET_{jobmap} = ET (Map) + ET (Combine)
$ET_{jobreduce}$ = ET (sort) + ET (Reduce) (5)

3.3.3Algorithm for Job Scheduling

In a system if free slots are available and the currently running job queue is empty, then it is necessary to make job scheduling decision to find out the order of the job for execution, which depends on the execution time of each submitted job. *DAG–CPM Scheduler* uses Critical Path Method (CPM) [23] to schedule the dependent jobs. CPM is a network based diagram, which is used to predict the total execution time with 3 main objectives.

– To calculate the completion time of an application.
– To identify till what extent each job in the scheduler list can be delayed without delaying the completion time of an application, which is called non critical activity.
– To identity the job for the highest risk that cannot be delayed without changing the completion time of an application, which is called critical activity.

CPM is used to identify the critical and non-critical activity, and then it streamlines the critical activities for scheduling. Critical Path is the path having a job with the longest total last round task runtime. Whenever the free slots are available, jobs present in the critical path are scheduled first, to avoid the extension of the completion time of an application. An application having a multiple jobs is the input, dependency among the job is represented in DAG, G=<V, E> and run time for each job is approximated. Each vertex in Graph G represents the job and ET_{job_map} represents the estimated run time of map task and ET_{job_reduce} represents the estimated runtime of reduce task of each job. New vertex s is included and directed edge is inserted from source s to the vertex (where incoming degree of a vertex is zero). ($ET_{job\_map}(s) = ET_{job\_reduce}(s) = 0$) New sink vertex t is added and directed edge is created to t from the vertex whose out degree is zero ($ET_{Job\_map}(t) = ET_{Job\_reduce}(t) = 0$).

Topological sort is used to create a list of jobs, and Critical Path method uses two phases: forward phase and backward phase. In forward phase, it determines the Early Start Time(EST), Early Finish Time (EFT), in backward phase it calculates Latest Start Time (LST) and Latest Finish Time (LFT), and Total Float is calculation is considered for identification of critical activity. Early Start Time(EST) is computed using equation 6, which is defined as the maximum early start time of all of its immediate previous jobs or Early Start Time(EST) is defined as the maximum EFT of all its upstream jobs.

EST_{job} (i) = $max_{<j,i>\epsilon E}$ (EST(j)+max (ET_{Job_reduce} (j), ET_{Job_map}(i))) (6)

Early Finish Time (EFT) is defined as the sum of early start of the predecessor task and the duration to finish the task (Execution time).Here, it overlaps the map task of downstream job and reduce task of upstream job. Hence, only the upstream job execution time is considered as the duration (D_{ij}).

$$EFT_j = \max(EST_i + D_{ij}) \quad (7)$$

Latest Finish Time (LFT) is defined as minimum value of LST of all its downstream jobs, which is calculated using equation 8.

LFT_{job} (i)= $min_{<i,j>\epsilon E}$ (LFT(j) - ET_{Job_reduce} (j) - max(0,ET_{Job_map} (j)-ET_{Job_reduce} (i))) (8)

Latest Start Time (LST) of a job is defined as the LFT minus the duration of job execution (Dij), which is calculated using equation 9.

$$LST_j = LFT_i - D_{ij} \quad (9)$$

Total Float (TF) is how long a job can be delayed /extended without extending the completion time, Calculate the TF using equation 10 for each job. if TF is zero for a job then that job is called the critical job, therefore, the execution of these jobs cannot be delayed.

$$TF=LST-EST \quad (10)$$

The set of critical jobs in a path is called critical path. If a DAG has more than one critical path, then the job with shortest execution time is scheduled first. Example of job dependency is shown in figure 3.Table 1 shows the calculated values to find the critical chain. In figure 3, each vertex denotes the job name and run time of a job. From table 1, it is observed that the path, s-D-C-E-t, is a critical path.

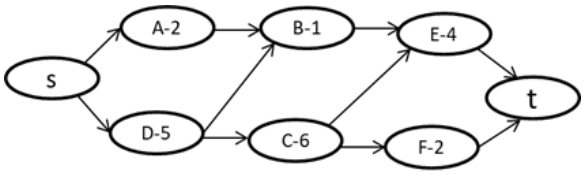


Fig 3. Job Dependency Example

Table 1. Critical Chain Calculation

	EST	EFT	LST	LFT	TF
A	**0**	**2**	**8**	**10**	**8**
B	**5**	**6**	**10**	**11**	**5**
C	**5**	**11**	**5**	**11**	**0**
D	**0**	**5**	**0**	**5**	**0**
E	**11**	**15**	**11**	**15**	**0**
F	**11**	**13**	**13**	**15**	**2**

Starvation is one of the issue during job scheduling, the continuous submission of new jobs to the system, the non-critical jobs are always delayed, to overcome from this issue, submitted jobs are assigned priority and then the job priorities are utilized to schedule the jobs. Priorities of the jobs are incremented as the time increases. For scheduling the job, highest priority jobs are selected first as scheduling candidates, and then the Critical Path Method (CPM) is used to select a job for executing.

IV. EXPERIMENTAL RESULTS AND DISCUSSIONS

To conduct an experiment, Hadoop cluster is created with 10 nodes and the configurations are the systems with quad-core, 8GB memory and 500GB disk space.

a. Data Intensive Iterative Application – Page View

Data intensive iterative application like Page View is executed to evaluate the performance optimization using pipeline in *DAG–CPM Scheduler*. Page View [2] occurs at a specific moment in time, when Web browser displays a Web page. i.e., set of page files that impart to a single presentation in a Web browser. The page View recognition step decides which page file requirements is part of the same page view and what content was assisted. Figure 4 shows the pseudo code for page view identification application. The Page View application is executed on the log files of academic and research website such as rnsit.ac.in, NASA website respectively, with a data size of 10 GB. The execution time in all the iteration and the degree of parallelism is given in figure 5. Experimental results of Page View application on conventional Hadoop and DAG-CPMS is tabulated in Table 2 and Table 3.Experimental result reveals that the time taken to complete all the jobs in conventional Hadoop is 301 seconds and 177 seconds in DAG–CPM Scheduler, so DAG–CPM Scheduler executes the application 41% faster compared to Hadoop. In this proposed work, the downstream job map phase starts as soon as the upstream job emits a block of output data. Therefore, the resources are utilized efficiently during the execution time of job.

```
PageView_Id()
{
    Let O(f1,...,fn) denote an open list of page files.
    Let T(t1,...,tn) denote the times associated with O
    Let Cv(f1,...,fm) denote the current view.
    Let rv denote the view referring file.
    Let fi and ri denote the requested and referring files.
    Set initial page view, k=0
    for each log entry, Le, do
    { if(O ≡ ϕ ) then
      Set rv = re; tk^e=ti
      if( rv ∉ Cv) then
```

```
Let F(f1,..., fn) denote a list of page files.
Let r denote a single page file.
Let L(p,c1,...,cm) denote a global list if
    Links
FindLink(F,r)
{
  for each L where p=r, do
      if( Li(r,c) = F
      return i
```

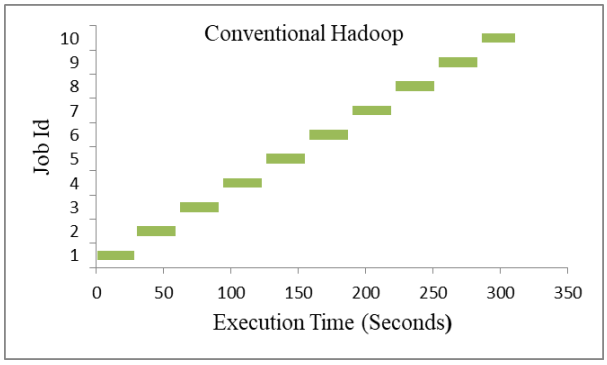
Fig 4: pseudo code for Page View Identification [2]

Table 2: Execution Time in Conventional Hadoop

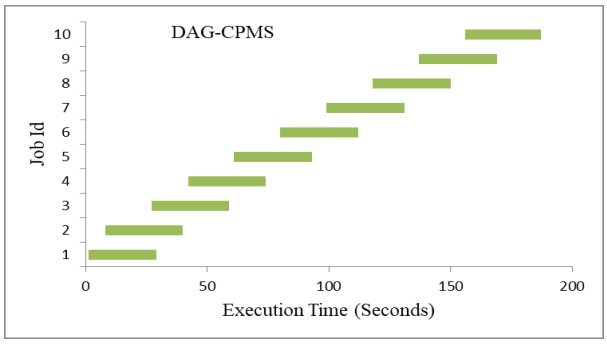
Job Id	Start Time	End time
1	0	27
2	28	57
3	59	88
4	90	119
5	121	150
6	152	181
7	183	212
8	214	243
9	245	274
10	276	301

Table 3: Execution Time in DAG-CPM Scheduler

Job Id	Start Time	End time
1	0	28
2	6	38
3	24	56
4	38	70
5	56	88
6	74	106
7	92	124
8	110	142
9	128	160
10	146	177



(a) Conventional Hadoop



(b) DAG-CPMS

Fig 5: Execution Time and parallelism in Page View Application (a) Conventional Hadoop (b) DAG-CPMS

b. Database Operations

PigMix2 [24] benchmark is a set of queries and it is a combination of simple queries. It is used to evaluate the pig performance as database application.it consists of very frequently performed operations like join, select, count etc and executed this on a dataset of 8 GB for evaluating the performance and also to show that job scheduling in DAG–CPM Scheduler is better compared to PIG. Figure 6 shows the DAG, here the path 3-5-6 is assumed as critical path and also represents the dependency among the jobs. Figure 7 shows the run time of all the jobs. Execution time obtained from the experiment for PigMix2 application on Pig, DAG-CPM Scheduler with and without pipeline is tabulated in Table 4, 5 and 6. Experimental results reveal that there is a substantial improvement in DAG–CPM Scheduler, Execution speed is 37.7% faster compared to Pig. DAG–CPM Scheduler can run 24.3% faster when compared to DAG–CPM Scheduler without pipeline. Figure 7 shows that data pipeline improves the performance and also makes the robust scheduling decisions to schedule the jobs, which is demonstrated in figure 7. DAG–CPM Scheduler without pipelining schedules job 2 after job 4, but DAG–CPM Scheduler with pipeline schedules job 5 after job 4, because job 5 is present in the critical path. Job 4 releases few free slots at around 10 seconds, as few reduce task completes. DAG–CPM Scheduler without pipelining will not start job 5 at this point because job 4 has not finished the execution completely. In DAG–CPM Scheduler with pipeline will launch job 5, as soon as job 4 generates a block of output data.

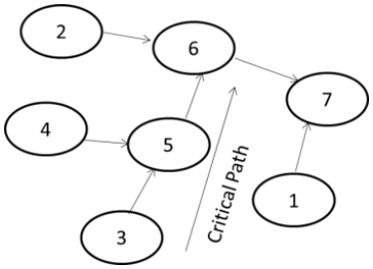


Fig 6: Job dependency DAG

Table 4: Execution Time on Pig

Job Id	Start Time	End Time
1	15	24
2	10	28
3	0	7
4	2	19
5	29	56
6	63	81
7	82	90

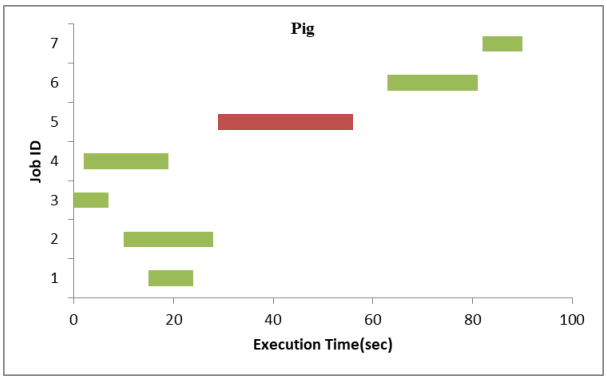
Table 5: Execution Time on DAG-CPM Scheduler without Pipeline

Job Id	Start Time	End Time
1	24	33
2	10	28
3	0	7
4	2	19
5	20	56
6	57	68
7	69	74

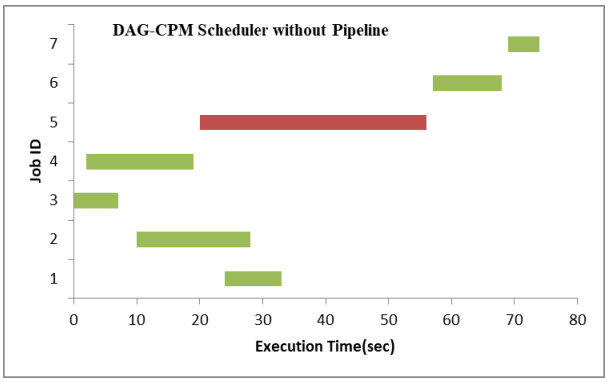
Table 6: Execution Time on DAG-CPM Scheduler with Pipeline

Job Id	Start Time	End Time
1	26	34
2	20	38
3	0	7
4	2	19
5	10	57

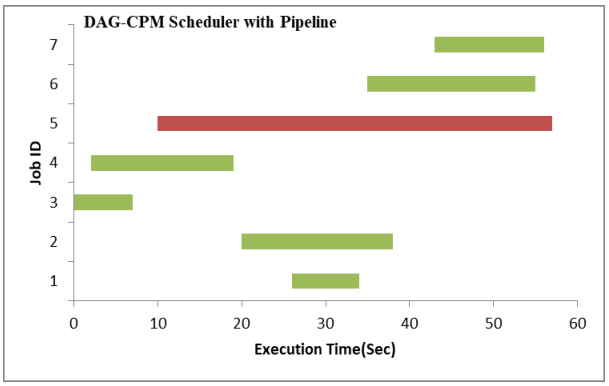
6	35	55
7	43	56



(a) Pig



(b) DAG-CPM without pipeline



(c) DAG-CPM with pipeline

Fig 7: Execution Time and parallelism in Pig Mix2 in (a) Pig (b) DAG-CPM Scheduler with pipeline (c) DAG-CPM Scheduler without pipeline

V. CONCLUSION

In MapReduce application, Job scheduling plays a vital role in scheduling group of jobs. DAG–CPM Scheduler schedules multiple jobs by dynamically constructing the dependency DAG for the currently running jobs based on the output and input directories. Data pipeline is used to overlap the reduce task and map task, by approximating the run time of jobs using historical information and multiple jobs are scheduled using critical job path model. Data intensive iterative application such as Page View application and Pigmix2 has been used to evaluate the performance. Experimental results revealed that for Pigmix2, DAG–CPM Scheduler is 37.7% faster compared to Pig and 24.3% faster compared DAG–CPM Scheduler without pipeline. For Page View application, DAG–CPM Scheduler improves the execution time by 41% compared to Hadoop. DAG–CPM Scheduler *has* improved the execution time by utilizing the resources effectively.

REFERENCES

1. J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, January 2008.
2. G T Raju and P S SathyaNarayana "Knowledge Discovery from Web Usage Data: Complete Preprocessing Methodology". International Journal of Computer Science and Network Security, vol 8, No.1, pp 179-185, Jan 2008.
3. J. Wolf, A. Balmin, D. Rajan, K. Hildrum, R. Khandekar, S. Parekh, K.-L.Wu, and R. Vernica, "On the optimization of Schedulers for MapReduce workloads in the presence of shared scans," *The VLDB Journal*, vol. 21,no. 5, pp. 589–609, Oct. 2012.
4. T. Sandholm and K. Lai, "MapReduce optimization using regulated dynamic prioritization," in *Proc. of SIGMETRICS*, 2009.
5. M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg," Quincy: Fair scheduling for distributed Computing clusters," in *Proc. of the ACM Symposium on Operating Systems Principles (SOSP)*,2009.
6. 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 the ACM EuroSys*, 2010.
7. C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin:A not-so-foreign language for data processing," in *Proc. of the ACM SIGMOD*, 2008.
8. "Apache hadoop, http://hadoop.apache.org/."
9. R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver,and J. Zhou, "SCOPE: Easy and efficient parallel processing of massive data sets," *Proc. VLDB Endow.*, vol. 1, pp. 1265–1276, August 2008.
10. M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proc. of the ACM EuroSys*, 2007.
11. B. Chattopadhyay, L. Lin, W. Liu, S. Mittal, P. Aragonda, V. Lychagina, Y. Kwon, and M. Wong, "Tenzing a SQL Implementation on the MapReduce framework," *Proc. VLDB Endow.*, vol. 4, pp. 1318–1327,September 2011.
12. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu,P. Wyckoff, and R. Murthy, "Hive: A warehousing solution over a mapreduce framework," *Proc. VLDB Endow.*, vol. 2, pp. 1626–1629, August 2009.
13. Agarwal, S. Kandula, N. Bruno, M.-C.Wu, I. Stoica, and J. Zhou, "Reoptimizing data-parallel computing," in *Proc. of the USENIX conference on Networked systems design and implementation (NSDI)*, 2012.
14. W. S. Cleveland and S. J. Devlin, "Locally weighted regression: An approach to regression analysis by local fitting," *Journal of the American Statistical Association*, vol. 83, no. 403, pp. 596–610, 1988.
15. S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proc. of the ACM Symposium on Operating Systems Principles (SOSP)*,2003.
16. J. Polo, D. Carrera, Y. Becerra, J. Torres, E. Ayguade?, M. Steinder, and I. Whalley, "Performance-driven task co- scheduling for MapReduce environments," in *IEEE Network Operations and Management Symposium (NOMS)*, 2010.
17. Q. Chen, C. Liu, and Z. Xiao, "Improving MapReduce performance using smart speculative execution strategy," *IEEE Transactions on Computers (TC)*, vol. 63, no. 4, pp. 954–967, April 2014.
18. M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica,"Improving MapReduce performance in heterogeneous environments," in *Proc. of the USENIX conference on Operating systems design and implementation (OSDI)*, 2008.
19. M. Lakshmi and P. Yu, "Limiting factors of join performance on parallel processors," in *Proc. of fifth International Conference on Data Engineering (ICDE)*, 1989.

20. C. K. Roy and J. R. Cordy, "A survey on software clone detection research," School of Computing Queen's University at Kingston Ontario, Canada, Tech. Rep. No.2007-54, 2007.
21. A. Okcan and M. Riedewald, "Processing theta-joins using MapReduce," in *Proc. of the ACM SIGMOD*, 2011.
22. Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "Skew-resistant parallel processing of feature-extracting scientific user-defined functions," in *Proc. of the ACM symposium on Cloud computing (SoCC)*, 2010.
23. J. E. Kelley, Jr and M. R. Walker, "Critical-path planning and scheduling," in *Proc. of eastern joint IRE-AIEE-ACM computer conference*,1959.
24. "Pigmix2, https://issues.apache.org/jira/browse/pig-200."
25. A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica,"Dominant resource fairness: Fair allocation of multiple resource types," in *Proc. of the USENIX conference on Networked systems design and implementation (NSDI)*, 2011.
26. R. Lee, T. Luo, Y. Huai, F. Wang, Y. He, and X. Zhang, "YSmart: Yet another SQL-to-MapReduce translator," in *Proc. Of the 31st International Conference on Distributed Computing Systems (ICDCS)*, 2011.
27. T. Nykiel, M. Potamias, C. Mishra, G. Kollios, and N. Koudas, "MRShare: Sharing across multiple queries in MapReduce," *Proc. VLDBEndow.*, vol. 3, no. 1-2, pp. 494–505, Sep. 2010.
28. H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, "Starfish: A self-tuning system for big data analytics," in *Proc.of the Fifth Biennial Conference on Innovative Data Systems Research (CIDR)*, January 2011.
29. H. Lim, H. Herodotou, and S. Babu, "Stubby: A transformation-based optimizer for MapReduce workflows," *Proc. VLDB Endow*, vol. 5, no. 11, pp. 1196–1207, Jul. 2012.
30. T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "MapReduce online," in *Proc. of the USENIX conference on Networked systems design and implementation (NSDI)*, 2010.
31. S. Seo, I. Jang, K. Woo, I. Kim, J.-S. Kim, and S. Maeng, "HPMR: Prefetching and pre-shuffling in shared MapReduce computation environment ,"in *Proc. of the IEEE Intl. Conf. on Cluster Computing and Workshops (CLUSTER)*, 2009.
32. Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "HaLoop: Efficient iterative data processing on large clusters," *Proc. VLDB Endow*, vol. 3,no. 1-2, pp. 285–296, September 2010.

AUTHORS PROFILE

D C Vinutha has received M. Tech., Degree from Visvesvaraya Technological University, Belagavi, Karnataka. Currently working as Associate Professor in the Department of Information Science and Engineering, Vidyavardhaka college of Engineering, Mysuru, Research scholar RNS Institute of Technology, Bengaluru. She has 20 years of experience in teaching. Her area of research interests includes Big Data, Data Mining. She is a IETE, IEI, ISTE life member. She has published 6 papers in leading reputed International Conferences/Journals/National conferences.

Dr. G T Raju, has received M.E. Degree from Bangalore University, in 1995 and Ph.D. from Visvesvaraya Technological University, Belagavi, Karnataka in 2008. Currently working as Vice Principal, Professor and Head, in the Department of Computer Science & Engineering, RNS Institute of Technology, Bengaluru. He has 24 years of experience in teaching and research. His area of research interests includes Web Mining, KDD, Image Processing, and Pattern Recognition. He has published more than 90 papers in leading reputed International Journals/ Conference proceedings. He has authored five Technical books. He has completed two funded research projects. Ten Research Scholars have been awarded Ph.D. Degree under his supervision.