Optimized Capacity Scheduler for MapReduce Applications in Cloud Environments

Adepu Sree Lakshmi, N. Subhash Chandra and M. BalRaju

Abstract Most of the current-day applications are data centric and involves lot of data processing. Technologies like hadoop enable data processing with automatic parallelism. Current-day applications which are more data intensive and compute intensive can take advantage of this automatic parallelism and the methodology of moving computation to data. In addition to it the Cloud computing technology enables users to establish the required clusters with required number of nodes instantly. Cloud computing has made easy for the users to execute large data applications without any requirement to establish/maintain the infrastructure. As cloud gives readily installed infrastructures, using hadoop on cloud has become common. The existing schedulers are very effective in static cluster environments but lack performance in virtual environments. The purpose of this work is to design an effective capacity scheduler for MapReduce applications for virtualized environments like public clouds by making scheduling decisions more intelligent using the characteristics of job and virtual machines.

Keywords Big data · Cloud computing · CloudSim · Hadoop MapReduce · Virtual machine

A. Sree Lakshmi (✉)
Geethanjali College of Engineering and Technology, Hyderabad, Telangana, India
e-mail: adepu.sreelakshmi@gmail.com

N. Subhash Chandra
CSE Department, CVR College of Engineering, Hyderabad, Telangana, India
e-mail: subhashchandra.n.cse@gmail.com

M. BalRaju
CSE Department, Swami Vivekanandha Institute of Techology, Hyderabad, Telangana, India
e-mail: drraju.jb@gmail.com

V. E. Balas et al. (eds.), *Data Management, Analytics and Innovation*, Advances in Intelligent Systems and Computing 808,
https://doi.org/10.1007/978-981-13-1402-5_12

1 Introduction

Hadoop [1] uses MapReduce framework where the given application executes in parallel on multiple nodes. The data of application is divided into multiple chunks of equal size and distributed to multiple nodes for processing. Map task perform the map function of the application on a split of data and generates the key value pairs which are further hashed and shuffled to an appropriate reduce task. Reduce task performs reduce logic on the output (Key/Value pairs) received from different mappers.

There are different schedulers provided in hadoop package which are pluggable. The schedulers provided in hadoop package are FIFO, Capacity scheduler, Fair scheduler. The default scheduler used in hadoop 2.0 is capacity scheduler. Our papers [2] can be referred for description of schedulers provided in hadoop package. All the schedulers provided in hadoop are optimal in fixed cluster environment. When executed in cloud environments it does not give optimized performance as the cloud is completely virtualized and each physical machine is shared by multiple virtual machines. Our paper [3] gives the study of performance of virtual machines for MapReduce applications in public cloud environment Amazon EMR. The execution time for hadoop MapReduce jobs can be further optimized by making scheduling decision more intelligent. If scheduling of MapReduce tasks considers the characteristics of virtual machine along with the job characteristics, we can make optimal scheduling.

Capacity Scheduler [4] in hadoop works in the form of hierarchy of queues and jobs are always placed in leaf queues. The available resources can be shared by multiple queues according to the configuration given in capapcityscheduler.xml file. In this paper we have improved the capacity scheduler by embedding the knowledge of virtual machine state and the resource requirements of the job. The proposed work is done in 3 modules: 1. Job resource requirements classification 2. Virtual machine state classification 3. Scheduling based on the above two classifications. Our experiments show that there is an improvement of 18–25% in the execution time of the hadoop jobs on virtualized environments. Less execution time is a major factor in cloud environments as users pay as use for the infrastructures. Reducing the execution time when executed on public clouds would reduce the costs incurred for cloud resources usage.

2 Background and Related Work

Capacity Scheduler is the default scheduler in Hadoop 2.0 [5]. One of the biggest advantages of Hadoop is multi-tenancy feature which enables multiple tenants to share the cluster in such a way that resources are allocated to their applications in a timely manner with respect to capacities allocated. Capacity scheduler enables sharing of large cluster among multiple users giving minimum capacity guarantee. It ensures that a single job/user/queue does not monopoly the resources available in the cluster.

Capacity scheduler mainly uses the concept of job queues and ensures capacity guarantees for queues. It uses hierarchy of queues where jobs are submitted to leaf queue. Certain capacity of resources is allocated to queues which can be used by the jobs submitted in the child leaf queues under it. The capacity guaranteed for the queues is configurable. Free resources can be allocated beyond the capacity of the queue also if not utilized by the other queues in the cluster. When an underutilized queue needs resources, the resources released by the over utilized queue are allocated to them. Hence elasticity is provided so that underutilized capacity of a queue can be provided to queues that need more resources beyond its allocated capacity.

Once the first job is submitted to the queue all the resources are given to it. When the job of other queue is submitted then further released resources are allocated to both the queues in accordance to the capacities guaranteed to the queues. Capacity scheduler does a static way to scheduling based on the resource requirements of the job and resources available in the cluster. Though it gives best run times in cluster environments. It needs some improvements when executed in virtualized environments.

Other works done in the direction of schedulers have done optimizations in different directions to decrease the execution time of the MapReduce applications. The authors of [6] designed a context aware scheduler for Hadoop1.x which improves the overall throughput of the system by leveraging the cluster heterogeneity. It schedules the tasks by context, i.e., job characteristics and resource characteristics of the nodes in the cluster. The authors neglected the IO bound shuffle phase where map outputs are moved to the reducer. Their work mainly classifies the node statically to determine whether a node is fast or slow. SAMR [7] self adaptive MapReduce scheduling algorithm in heterogeneous environment adapts to the continuously varying environment automatically by calculating the progress of tasks dynamically. It splits the given job into many fine grained tasks and while executing these fine-grained maps and reduce tasks stores the historical information on every node. Based on the historical information SAMR will adjust the time weight of each stage of the map and reduce tasks.

The authors of [8] designed a fine-grained and dynamic MapReduce task scheduling scheme for the heterogeneous cloud environment. Their work is also based on collecting historical and real time online information from each node in the cluster and selects the appropriate parameters in order to identify slow running tasks. Nodes are statically classified as high performance and low performance irrespective of the load on peer virtual machines running on the same physical node.

Research Work in [9] has a scheduling framework that takes into account the actual resource requirements of the job. Their scheduler classifies tasks into schedulable and non-schedulable classes based on whether a job will overload a node. They used only CPU utilization but other job features need to be considered relatively. Scheduling related works by different researchers [10–19] in the direction of dynamic resource allocation try to optimize the execution times in different directions like cost, IO time, Network IO, reduce operation time, etc.

3 Proposed Work

In virtual environments each physical machine has multiple virtual machines hosted on it and shares the resources in the physical machine and accesses them through hypervisor. When a Hadoop cluster is created in a virtualized environment the performance is not similar to that of dedicated cluster environment. An optimized scheduler is required for virtualized environments to schedule the Hadoop MapReduce applications. A better performance can be obtained by including the information related to job characteristics and VM characteristics in scheduling decisions. When a job is scheduled to a VM its performance is very much affected by the other virtual machines running on the same physical host. This is more effective in the case of disk performance as every read/write operation done by a VM goes through hypervisor and the disk I/O is shared by multiple VMs. This has more effect in Hadoop jobs as the map/reduce application input/output are done to disk. The amount of CPU and IO required by every job has different performance in different VMs depending on the current CPU and IO usage of the VM and the other VMs existing in the physical machine.

If job characteristics is understood to be whether it is CPU-intensive or IO-intensive then a proper VM can be chosen to give better performance of the MapReduce application execution. Scheduling can be made dynamic which includes the features of job, current VM characteristics. Our proposed work is to design a dynamic capacity scheduler which schedules MapReduce applications considering the job and VM characteristics. As the amount of data processed except the last map task is same and the map function applied is same, understanding the job characteristics is easy after execution of few map tasks, which can be used in scheduling the future map tasks of that job. Similarly if the current state/characteristics of every virtual machine is regularly monitored and a tag is set regularly based on the amount of CPU utilization and IO Utilization of each VM. If a VM is tagged to be using more IO then that VM is more appropriate for a map task which is less IO-intensive.

To simulate our work, we used CloudSimEx [20] which is a MapReduce simulator and an extension of CloudSim simulator [21]. CloudSimEx has simulation for both cloud environment and MapReduce execution. But as our scheduler is an improvement over current capacity scheduler, we built the existing scheduling model of capacity scheduler on CloudSimEx. To model capacity scheduler the following changes are made to the CloudSimEx tool.

1. Limited number of virtual machines, the number of virtual machines required by user is taken as parameter through cloud.yaml file as the cloud is usually rented by specifying the number and type of machines required.
2. In cloudSimEx a scheduling plan is built for all cloudlets considering the virtual machines available in the data centres and all cloudlets are submitted at a time. Cloudlets are mapped to VMs initially itself. As number of VMs is restricted based on user request, initial binding of cloudlets VM's is removed. It is being modelled in an incremental fashion. Cloudlets are submitted initially to VMs based on the capacity guaranteed (**Algorithm 1**).
3. Job submissions are made through queues where each queue is associated with capacity guarantee (queue_capacity). If queue_capacity is [60, 40] then 60% of resources are assigned to queue1 and 40% of resources to queue2. If nVMs = 5 then 3 VMs are allocated to queue1 and 2 VMs for queue2. Cloudlets are modelled to assign to nVMs as per the user provided queue_capacity.

4. Submit cloudlet method is overridden to submit cloudlet to a specified VM so that once the task assigned to slave node running on that VM is completed it can assign a new task to it within the constraints of capacity Scheduler (**Algorithm 2**).
5. Cloudlet return is being modified in order to invoke submitCloudlet(VM vm) to submit a cloudlet to the VM according to queue capacity allotment.
6. Job_submitted_count[] array is used to maintain the number of cloudlets submitted for each queue to enable the capacity allocation constraints. No of resources allocated to job in queue 0 is indicated in job_submitted_count[0].
7. job_index[] array to hold the index of each job submitted to cloudletList. This is being used to enable to move to next job in the queue easily instead of comparing with every task in the cloudletList. As every job is divided into tasks executed in cloudlets, when a task tag does not match with VM tag, the remaining tasks of the same job need not be compared with the VM tag, and we can directly compare with the cloudlet of next job. job_index[0] = 0 which is the index of cloudlet of the first job submitted and job_index [1] is assigned with the index of the first cloudlet of the second job submitted.

submitCloudlets() is invoked only once during the initial scheduling to VMs. First cloudlet is submitted and job_submitted_count is updated to reflect the current allocation done to the queue to which the job request belongs. If that request allocated resource (VM) count is less than the guaranteed queue capacity the next index also belongs to the same job request otherwise the index is made to point to next job request to fulfil its guaranteed queue capacity.

Algorithm1 submitCloudlets()

```
1   q_c = Queue_Capacity;
2   count, index, q= 0;  lastindex = job_index[q] – 1;
3   for each vm in VmsCreatedList
            3.1 cloudlet = get cloudlet at index
            3.2 if (cloudlet instanceof ReduceTask &&
                !isAllMapTaskFinished(cloudlet.getCloudletId())) then continue;
            3.3 r = request to which cloudlet belongs
            3.4 set vmid of cloudlet to vm
            3.5 put(cloudletid,vmid) into scheduling plan
            3.6 send CLOUDLETSUBMIT event;
            3.7 cloudletsSubmitted++;  count++;
            3.8 job_submitted_count[q] = count;
            3.9 for  i = q to job_index.length
                        3.9.1   job_index[i] = job_index[i] – 1;
            3.10        lastindex = job_index[q];
            3.11        if (!((count < q_c[q]) & (index < lastindex))) then
                        3.11.1 index = lastindex
            3.12        count = 0;  q++;
            3.13        lastindex = job_index[q];
            3.14        endif
            3.15        add cloudlet to CloudletSubmittedList
            3.16        remove cloudlet from CloudletList
4   end for
```

SubmitCloudlets(int Vmid) is invoked for submission of cloudlets as each VM is ready to accept a new task on finishing the previously allocated task. Index is made to point to the queue which has less allocated resources than the guaranteed capacity. If the task is a reduce task and its corresponding map tasks are not completed then cloudlet from next queue is considered. If the queue is last queue tried and no more cloudlets next to it in the cloudletlist we simply return as in step 4.4.2, otherwise point to the next queue. But if the selected task is a map task and within the capacity guaranteed for the corresponding queue then that task is submitted to the VM.

Algorithm2 submitCloudlets(int vmid)

```
1  if (getCloudletList().size() > 0) then
2  done = false;
3  index = 0;
4  for each vm in VmsCreatedList
   4.1 if (vm.getId() == vmid) then
        4.1.1  q = 0;
        4.1.2  if ((job_submitted_count[q] < q_c[q]) & (job_index[q] > 0)) then break;
        4.1.3  else q++;
        4.1.4  break;
        4.1.5  endif
   4.2  if (q != 0) then vindex = job_index[q – 1];
   4.3  cloudlet = get cloudlet at index
   4.4  if (cloudlet instanceof ReduceTask &&
                !isAllMapTaskFinished(cloudlet.getCloudletId())) then
        4.4.1  if (q == q_c.length – 1) index = job_index[q – 1];
        4.4.2  if (index = = 0) return;
        4.4.3  endif
        4.4.4  else
                4.4.4.1     index = job_index[q];   q = q + 1;
        4.4.5  endif
        4.4.6  cloudlet = get cloudlet at index
        4.4.7  endif
        4.4.8  r = request to which cloudlet belongs
        4.4.9  set vmid of cloudlet to vm
        4.4.10put(cloudletid,vmid) into scheduling plan
        4.4.11send CLOUDLETSUBMIT event
        4.4.12cloudletsSubmitted++;
        4.4.13job_submitted_count[q]++;
        4.4.14if (q == job_submitted_count.length – 1) then job_index[q] – = 1;
        4.4.15if (q > 0)
        4.4.16if (job_index[q] == job_index[q – 1])
                4.4.16.1    t = job_index[q];
                4.4.16.2    job_index[q] = 0; job_index[q – 1] = 0;
                4.4.16.3    for (int i = q + 1; i < job_index.length; i++)
                 4.4.16.3.1        job_index[i]= job_index[i]–t;
```

```
            4.4.17 endif
            4.4.18 endif
            4.4.19 else
            4.4.20 for (int i = q; i < job_index.length; i++)
                        4.4.20.1    job_index[i]= job_index[i]– 1;
            4.4.21 endif
            4.4.22 getCloudletSubmittedList().add(cloudlet);
            4.4.23 getCloudletList().remove(cloudlet);
        4.5         endif
        4.6         endfor
    5   endif
```

Our work is a contribution to CloudSimEx with the scheduling techniques of Hadoop. We implemented the scheduling algorithms FIFO, Capacity Scheduler of Hadoop MapReduce applications in CloudSimEx. This contribution enables people working in the area of scheduling, load balancing of Hadoop jobs for virtualized environments to simulate their work. Our proposed work of designing a dynamic capacity scheduler is simulated in CloudSimEx by including the job characteristics and VM characteristics in scheduling decisions. When a cloudlet is returned, submit cloudlet decides about which job is to be given to the VM based on the job tag and VM tag within the constraints of queue capacity.

Jobs are categorized to be CPU-intensive, IO-intensive and every job is associated with a 2-digit tag which indicates the intensiveness of the job. MSB indicates the CPU-intensiveness and LSB denotes IO-intensiveness. Tag with value 01 indicates a job which is IO-intensive and value 10 indicates a job which is CPU-intensive. Similarly every VM is associated with 2-bit tag to indicate the current load of CPU and IO of the host on which the virtual machine is deployed. Cloudlets can be submitted with reference to the tag of jobs and virtual machines. Jobs can be assigned a tag by user if the intensiveness of the job is known or can be calculated after few map tasks get executed. When a virtual machine tag indicates as IO heavy then a task which is more CPU-intensive and less IO-intensive would be appropriate choice.

Algorithm submitCloudlets()
// CapacitySchedulerLoadAware

```
    1. q_c = QueueCapacity;
    2. index,q= 0;  lastindex = job_index[q] – 1;
    3. for  each vm in VmsCreatedList
    4. cloudlet = first cloudlet in cloudletList;
          5.1 done = false;
          5.2 for idx=0 to job_index.length
          5.3 q = idx;
          5.4 if (idx == 0) then  index = idx
          5.5 else   index = job_index[idx – 1]
          5.6 if (job_submitted_count[q] < q_c[q]) then
                5.6.1  cloudlet = Cloudlet at index
          5.7 if (cloudlet instanceof ReduceTask &&
               !isAllMapTaskFinished(cloudlet.getCloudletId())) then  continue;
          5.8 r= request to which cloudlet belongs
          5.9  if (((int) r.job.getTag() & (int) vm.getTag()) == 0) then
                    5.9.1   set vmid of cloudlet to vm
                    5.9.2   put(cloudletid,vmid) into scheduling plan
                    5.9.3   send CLOUDLETSUBMIT event
                    5.9.4   cloudletsSubmitted++;
                    5.9.5   done=true;
                    5.9.6   break;
          5.10         end if
          5.11         end if

6   end for
7   if(!done)
          7.1 cloudlet= first cloudlet in cloudletList;
          7.2 r = request to which cloudlet belongs
          7.3 q = 0;
          7.4 set vmid of cloudlet to vm
          7.5 put(cloudletid,vmid) into scheduling plan
          7.6 send CLOUDLETSUBMIT event,
          7.7 cloudletsSubmitted++;
8   end if
9   job_submitted_count[q] = job_submitted_count[q]+1;
10  for (int i = q; i < job_index.length; i++)
          10.1         job_index[i] – = 1;
11  lastindex = job_index[q];
12  if (!((job_submitted_count[q] < q_c[q]) & (index < lastindex)) ) then
          12.1         index = lastindex;  q++;  lastindex = job_index[q];
13  end if
14  add cloudlet to CloudletSubmittedList
15  remove cloudlet from CloudletList
16  end for
```

4 Evaluation and Results

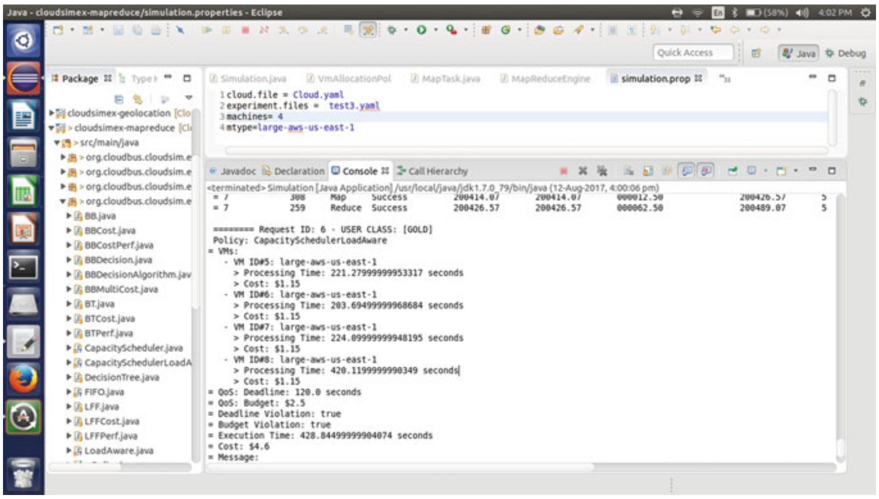
Simulation.properties:
cloud.file = Cloud.yaml
experiment.files = test3.yaml
machines = ≪4,5,6,7,8≫
mtype = large-aws-us-east-1

MapReduce jobs used: MapReduce_9_2.yaml, MapReduce_15_2.yaml, MapReduce_30_2.yaml, MapReduce_50_1.yaml, MapReduce_100_3.yaml

Experiment file is specified as yaml file in which jobs are considered as part of queues in the order of submission. Different combinations of MapReduce applications are being submitted and the execution times of CapacityScheduler and CapacitySchedulerLoadAware by varying number of virtual machines. One of the sample of experiment file is as below (Fig. 1).

```
Expertiment:test3.yaml
!!org.cloudbus.cloudsim.ex.mapreduce.Experiment
workloads:
 - !!org.cloudbus.cloudsim.ex.mapreduce.Workload
    [
     CapacitySchedulerLoadAware, Public,
       {
        GOLD: 100.0,
        SILVER: 60.0,
        BRONZE: 0.0
       },
     [
      #[Submission Time, Deadline, Budget, Job, user class]
      !!org.cloudbus.cloudsim.ex.mapreduce.models.request.Request
      [200000, 120, 2.5, MapReduce_50_1.yaml, GOLD],
  !!org.cloudbus.cloudsim.ex.mapreduce.models.request.Request
      [200000, 120, 2.5, MapReduce_100_3.yaml, GOLD],
 !!org.cloudbus.cloudsim.ex.mapreduce.models.request.Reques
      [200000, 120, 2.5, MapReduce_15_2.yaml, GOLD],
 ]]
```

Test case	nVMs ($N = 4$)		nVMs ($N = 5$)		nVMs ($N = 6$)		nVMs ($N = 7$)	
	CS	CSLA	CS	CSLA	CS	CSLA	CS	CSLA
MR-30-2, MR-9-2	101.88	87.05	81.78	68.47	68.52	58.272	59.094	57.67
MR-50-1, MR-30-2	313.57	240.57	255.82	187.52	224.89	157.9	201.369	137.49
MR-50-1, MR-100-3	504.17	444.42	398.1	353.18	316.19	192.6	212.959	198
MR-100-3, MR-50-1	530.75	488.97	439.21	347.13	371.79	319.91	325.07	305.14



```
cloud.file = Cloud.yaml
experiment.files =  test3.yaml
machines= 4
mtype=large-aws-us-east-1
```

```
<terminated> Simulation [Java Application] /usr/local/java/jdk1.7.0_79/bin/java (12-Aug-2017, 4:00:06 pm)
 = 7          308    Map     Success       200414.07      200414.07      000012.50              200426.57       5
 = 7          259    Reduce  Success       200426.57      200426.57      000062.50              200489.07       5

 ======== Request ID: 6 - USER CLASS: [GOLD]
 Policy: CapacitySchedulerLoadAware
= VMs:
   - VM ID#5: large-aws-us-east-1
     > Processing Time: 221.27999999953317 seconds
     > Cost: $1.15
   - VM ID#6: large-aws-us-east-1
     > Processing Time: 203.69499999968684 seconds
     > Cost: $1.15
   - VM ID#7: large-aws-us-east-1
     > Processing Time: 224.09999999948195 seconds
     > Cost: $1.15
   - VM ID#8: large-aws-us-east-1
     > Processing Time: 420.1199999990349 seconds
     > Cost: $1.15
= QoS: Deadline: 120.0 seconds
= QoS: Budget: $2.5
= Deadline Violation: true
= Budget Violation: true
= Execution Time: 428.84499999904074 seconds
= Cost: $4.6
= Message:
```

Fig. 1 CapacitySchedulerLoadAware execution

Fig. 2 MapReduce 30-2, MapReduce 9-2

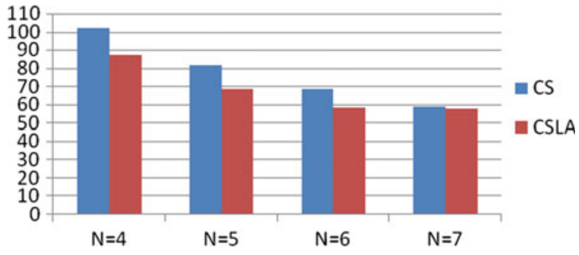
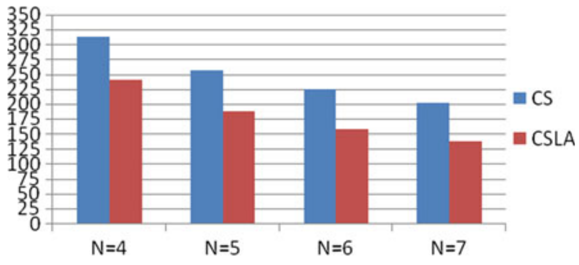


Fig. 3 MapReduce 50-1, MapReduce 30-2



The above results Figs. 2, 3, 4 and 5 indicate an improvement in execution time of the MapReduce jobs on an average of 20%. In few cases there is no major difference in the execution times as the jobs submitted are appropriate to the VM characteristics. There is an improvement brought in the job exection time where MapReduce applications are scheduled in appropriate to VM characteristics (Fig. 6).

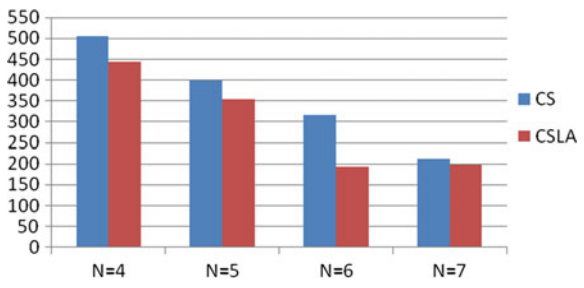


Fig. 4 MapReduce 50-1, MapReduce 100-3

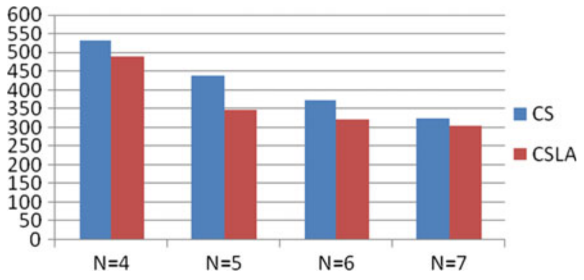


Fig. 5 MapReduce 100-3, MapReduce 50-1

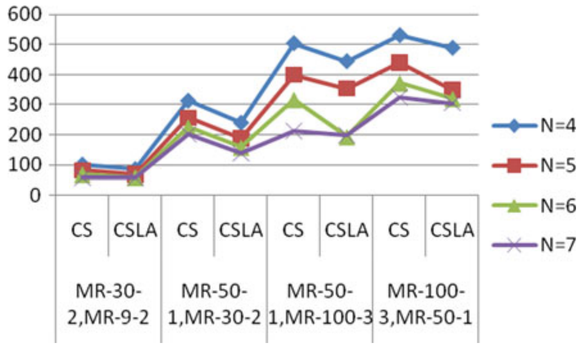


Fig. 6 CapacityScheduler versus CapacitySchedulerLoadAware

5 Conclusion

The MapReduce applications which automatically use the advantage of availability of multiple nodes by executing map and reduce in parallel can further be optimized if the scheduler logic includes a concept of learning of jobs and machine characteristics. If a virtual machine is loaded with CPU-intensive task then scheduler can schedule an IO-intensive task so that there could be an appropriate usage of resources available on the virtual machine. Our work was to design an optimal scheduler for MapReduce applications which can schedule balanced set of workloads to all nodes in the cluster. It not only enables fast execution of the tasks but can also be used to make users understand about the characteristics of the job submitted to the cluster. Job characteristics enable user to use proper configuration of Hadoop cluster for further executions of the same job. Our primary goal is to reduce the execution time of the Hadoop map tasks by including the behaviour of map task and virtual machine in scheduling decisions.

References

1. Hadoop The definitive guide, O'Reilly & Yahoo Press, Tom White.
2. Sree Lakshmi, A., BalRaju, M., Subhash Chandra, N. (2016). Towards optimization of hadoop map reduce jobs on cloud. In *IEEE International Conference on Computing, Analytics and Security Trends (CAST 2016)*, Dec 2016. ISBN: 978-1-5090-1338-8.
3. Sree Lakshmi, A., Bal Raju, M., & Subhash Chandra, N. (2015). Scheduling of parallel applications using map reduce on cloud: A literature survey. *International Journal of Computer Science and Information Technologies, 6,* 112–115.
4. Apache Hadoop Capacity Scheduler YARN: https://hadoop.apache.org/docs/r2.4.1/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html.
5. https://hortonworks.com/blog/understanding-apache-hadoops-capacity-scheduler/.
6. Kumar, A. K., Krishna, V., Voruganti, K., & Prabhakara Rao, G. V. (2012). CASH: Context aware scheduler for Hadoop. In *ICACCI '12 Proceedings of the International Conference on Advances in Computing, Communications and Informatics*.
7. Chen, Q., Zhang, D., Guo, M., Deng, Q., & Guo, S. (2010). SAMR: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment. *Computer and Information Technology, International Conference*, 2736–2743.
8. Mao, Y., Qi, H., Ping, P., & Li, X. (2016). FiGMR: A fine grained mapreduce scheduler in the heterogeneous cloud. In *Proceedings of the IEEE International Conference of Information and Automation*, Ningbo, China, August 2016.
9. Deshmukh, S., Aghav, J. V., & Chakravarthy, R. (2013). Job classification for mapreduce scheduler in heterogeneous environment. In *2013 International Conference on Cloud & Ubiquitous Computing & Emerging Technologies.*
10. Wylie, A., Shi, W., Corriveau, J. P. (2016). A scheduling algorithm for hadoop mapreduce workflows with budget constraints in the heterogeneous cloud. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops.*
11. Kang, H., Chen, Y., Wong, J. L., Sion, R., & Wu, J. (2011). Enhancement of Xen's scheduler for MapReduce workloads. In *Proceedings of the 20th international symposium on High performance distributed computing*, New York, NY, USA, pp. 251–262.
12. Yazdanov, L., Gorbunov, M., & Fetzer, C. (2015). EHadoop: Network I/O aware scheduler for elastic MapReduce cluster. In *2015 IEEE 8th International Conference on Cloud Computing*. https://doi.org/10.1109/cloud.2015.113.
13. Ehsan, M., Chandrasekaran, K., Chen, Y., & Sion, R. (2016). Cost-efficient tasks and data co-scheduling with affordhadoop. *IEEE transactions on cloud computing*. https://doi.org/10.1109/tcc.2017.2702661.

14. Das, R., Singh, R. P., Patgiri, R. (2016). Mapreduce scheduler: A 360-degree view. *International Journal of Current Engineering and Scientific Research (IJCESR), 3*(11), ISSN (print): 2393–8374, (online): 2394–0697.
15. Kim, S., Kang, D., & Choi, J. (2015). I/O characteristics and implications of big data processing on virtualized environments. *Applied Mathematics & Information Sciences An International Journal, 9*(2L), 591–598.
16. Kim, S., Kang, D., Choi, J., & Kim, J. (2014). Burstiness-aware I/O scheduler for MapReduce framework on virtualized environments. In *2014 International Conference on Big Data and Smart Computing (BIGCOMP)* (pp. 305–308). https://doi.org/10.1109/bigcomp.2014.
17. Tian, W., Li, G., Yang, W., & Buyya, R. (2016). HScheduler: An optimal approach to minimize the makespan of multiple MapReduce jobs. *The Journal of Supercomputing, 72*(6), 2376–2393. https://doi.org/10.1007/s11227-016-1737-4.
18. Wang, X., Shen, D., Yu, G., Nie, T., & Kou, Y. (2013). A throughput driven task scheduler for improving mapreduce performance in job-intensive environments. In *2013 IEEE International Congress on Big Data* (pp. 211–218).
19. Yao, Y., Wang, J., Sheng, B., Lin, J., Mi, N. (2014). HaSTE: Hadoop YARN scheduling based on task-dependency and resource-demand. In *2014 IEEE International Conference on Cloud Computing*. 978-1-4799-5063-8/14.
20. http://nikgrozev.com/2014/06/08/cloudsim-and-cloudsimex-part-1/.
21. http://www.cloudbus.org/cloudsim/.