A Hadoop Yarn Scheduling Based on Node Computing Capability and Data Locality in Heterogeneous Environments

Ximei Du [a], Yong Liu and Caili Zhao [b]

Henan University of Science and Technology, Luoyang 471000, China.

[a]dxmduximei@163.com, [b]zclsunshine@163.com

Abstract

Due to the computing nodes that make up the Hadoop cluster usually have different performances, the original scheduling algorithm in Hadoop does not take into account the performance difference of different nodes, resulting in the occurrence of busy and unbalanced among nodes, which affects the efficiency of job execution.At present,the implementation of Hadoop mainly focuses on homogeneous clusters. It is assumed that most of task can obtain data locally, and the location of data is not fully considered, leading to unnecessary data transmission, extra network bandwidth and time. In view of the two problems mentioned above, a resource allocation algorithm based on node computing capability and data locality is proposed in this paper, according to the execution ability of different nodes and the location of input data. Based on the historical information of nodes, the current load state and the locality of the nodes, the algorithm assigns the resource to the task request of the input data at the same node. Otherwise, the resource on the node is allocated to the resource request according to the node's computing capacity.Experimental results show that compared with the existing scheduling algorithms, the proposed scheduling algorithm can effectively reduce the completion time and improve resource utilization.

Keywords

Hadoop; Node computing capability; Data locality;Heterogeneous environments.

1. Introduction

MapReduce (MRV1), a popular programming model, proposed by Google, has been well used to process large datasets in Hadoop, an open source cloud platform. Its new version MapReduce 2.0 (MRV2) developed along with the emerging of Yarn has achieved obvious improvement over MRV1. YARN employs a dynamic resource allocation mechanism, which uses global ResourceManager (RM) and application ApplicationMaster(AM) instead of centralized JobTracker for resource management and job scheduling / monitoring. Its basic idea is to split the MapReduce's jobtracker into two separate services: RM and AM. RM is responsible for the management and distribution of the entire system resources, while AM is responsible for the management of individual applications. In addition, it does not use a partition of map or reduce slots, but rather a resource abstraction called a container, which encapsulates the multidimensional resources of nodes that include CPU and memory on a node. In the latest Hadoop YARN, in order to solve the problems, such as main node failure and resource allocation, the resource scheduler is replaced by the resource scheduler. The Hadoop YARN scheduler organizes the resources in a hierarchical queue, and allows the queues share the resources on all nodes.

Hadoop YARN provides three built-in scheduling algorithms and implements three resource schedulers: FIFO Scheduler, Capacity Scheduler, and Fair Scheduler. However, none of these three schedulers take into account the heterogeneity of computing resources, nor do they adequately

consider the locality of the data, resulting in unnecessary data transmission, which consumes additional network bandwidth and time. At present, heterogeneous environments has become commonplace. The heterogeneous environment in this work refers to the computing nodes with different processing capabilities in terms of central processing unit (CPU), memory, and I/O. Moreover, due to the diversity of applications, these schedulers do not meet the needs of users to allocate resources reasonably and reduce the job execution time [2].

In the field of distributed computing, resource allocation is actually a task scheduling problem. Its main task is to make the best match between resources and jobs / tasks based on the current situation of resources (including CPU, memory, network and other resources) on each node in the current cluster and the Quality of Service requirements for each user's job. Due to the diversified requirements of users on job service quality, task scheduling in distributed systems is a multi-objective optimization problem. Further, it is a typical NP problem [3].

In this paper, a resource allocation algorithm (NCLDR) based on node computing capability and data locality is proposed, which is a resource aware scheduler to improve the resource allocation of MapReduce in heterogeneous environments. NCLDR takes into account Hadoop's support for data locality and makes a reasonable allocation of resource requests for tasks by utilizing the current computing capability of participating nodes.

The rest sections of the paper are organized as follows. Related work is given in Section II. Section III gives an introduction to the resource allocation algorithm and model based on node computing capability and data locality, and the implementation of the algorithm is described in detail. In Section IV, testing environment, and corresponding scenarios are designed for the verification and evaluation of the proposed algorithm (NCLDR). Finally, Section V concludes the paper, proposes the challenges that will be faced and points out some future work.

2. Related Work

Job scheduling is an NP-hard optimization problem that requires processing and completing jobs with the available resources in the cluster. In a typical MapReduce scenario, many Chinese and foreign scholars have done a lot of work on the scheduling problem in order to improve the efficiency of resource utilization and reduce the completion time of the job.

The dynamic rack load balancing algorithm (DRWB) proposed in [4], is used to estimate the performance and workload of the node by analyzing the log files of the Hadoop, and dynamically assigns the node tasks with large load to the small load nodes. In literature [5], a LEEN algorithm based on position perception and fair perception is proposed. The LEEN algorithm estimates the data distribution and determines the input of Reduce according to the frequency of occurrence of the Key value after the Shuffle phase. The CloudMR proposed in [6] combines data from the same rack based on locality of data to reduce data transmission between racks. At the same time, the data amount and the number of task slots are dynamically allocated to the node according to the node performance and task characteristics. The above research seldom considers the effect of node performance heterogeneity on Map phase execution. Each node has different computational capabilities, which easily leads to the failure of the nodes to complete the Map task at the same time. Xie et al. proposed a data placement and migration strategy in a heterogeneous environment in [7]. The workload is distributed according to the calculation rate of each node, and the data is migrated according to the cluster load during operation. However, the paper does not consider the impact of data size, and does not delve into the relationship between the reduction of execution time and the locality of data. In addition, data migration strategies may in turn affect data locality.Zaharia et al. [8] proposed a modified version of the speculative execution strategy called the LATE algorithm. A different metric was proposed in LATE to estimate the backup task. Estimate the remaining time of the task directly, regardless of the progress of the task. However, since the parameter used in LATE cannot represent all cases, LATE has unstable execution time at each stage. Therefore MCP [9] was proposed, which uses

average progress to identify slow tasks. According to the results, struggles can be appropriately judged when the data between tasks is skewed. However, MCPs still have many defects, which can only be used for MR, excluding YARN with the new resource management framework.

In order to improve the job completion time, there are many related studies. Liu et al. [10] proposed a new model by collecting real-time data and the model achieved high accuracy. Based on the exponential smoothing model for each stage, a dynamic strategy is proposed. A new adaptive task scheduling strategy was proposed in [11] based on dynamic workload adjustment (ATSDWA) according to the dynamic load changes of each task node in a heterogeneous Hadoop cluster and the performance difference of different tasks. He et al. [12] proposed a scheduling algorithm that automatically adjusts job priority based on historical progress. Based on real-time monitoring of job progress information, it performs exponentially smooth forecasting of job progress rates, calculates the remaining execution time of the job, and dynamically estimates the idle time of the job. The priority order of jobs in the job queue is continuously updated in real time, and jobs with small idle time are preferentially scheduled. However, none of them are suitable for the Hadoop YARN structure.

In view of the current research status, this paper designs and implements a resource allocation algorithm based on node computing power and data locality by considering the data location and the computing ability of nodes to different tasks, which can improve the utilization of resources and shorten the execution time of the job.

3. Resource Scheduling Algorithm Based on Node Computing Capability and Data Locality

3.1 Cluster Environment Model

The model of the Hadoop cluster environment is represented by $E=\{J,R,C\}$, where J represents the job collection, R is the set of node resources in the cluster, and C represents the collection of containers.

The job set $J=\{J_1,\ J_2,\cdots,\mathrm{J_m}\}$ indicates that the current cluster has m jobs. Among them, the job $J\mathrm{i}=\left\{j^{type},I_{\mathrm{s}},\vec{T},\vec{S}\right\}$, j^{type} represent the type of job, ls represent the length of the job, $\vec{T}=\left\{\mathrm{t}^{a},t^{d}\right\}$ contains the detailed time information of the job, t^{a} indicates the arrival time of the job, and t^{d} indicates the deadline of the job. $\vec{S}$ represents the resource information required for the job.

The resource set in the cluster $R=\{R_1,R_2,\cdots,R_n\}$ consists of n nodes and all node resources, where $R_{\mathrm{j}}=\left\{cpuSpeed_j,ioSpeed_j,load_j,M_j,resM_j\right\}$ represent the resource on the j node, $cpuSpeed_j$ and $\mathrm{io}Speed_j$ respectively represent the CPU running rate and the read and write rate on the node j, load_j represents the load of node j, M_j represents the total resource amount of node j, and $\mathrm{res}M_j$ represents the remaining free resource amount of node j. Where $\mathrm{resM_j}\le M_j,0\le j\le n$.

The container set $C=\{C_1,\ C_2,\cdots C_{\mathrm{k}}\}$ consists of k containers, which represents all container collections of resource applications in the Hadoop MRV2 cluster.

The container $C_{\mathrm{t}}=\left\{J_{t,i},R_{t,j},resMC_t\right\}$, $0\le \mathrm{i}<m,0\le t<k$, where, $J_{t,i}$ indicates the job Ji to which Ct belongs, $R_{t,\mathrm{j}}$ indicates the node R_{j} to which the container t belongs, and $\mathrm{res}MC_t$ represents the amount of resources requested by the container Ct.

The resource request set $\mathrm{Req}=\{req_1,req_2,\cdots,req_u\}$ for all jobs $J_i\in J$. When the resource container for the r request is on the node Rj, the node computing capacity is expressed as $C\mathrm{apable}_j$.

3.2 Node Computing Capability

In heterogeneous environments, nodes with different computing power have different processing speed, and thus there is a difference in the execution time for completing the same task. In order to allocate appropriate resources for tasks and improve the completion time of tasks, we need to calculate the computing power of the node resources.

The computing power of a node is closely related to with the remaining resources on the node and the type of job. Different types of jobs also have different resource requirements in the system, so the computing power on the nodes is also different. For CPU-intensive jobs, nodes with sufficient computing resources are better than nodes with sufficient I/O resources. Therefore, we need to distinguish the type of job, which has a crucial role in the computing power of the node. The resource scheduler obtains the node's related information (CPU rate, load information, remaining free resources, job failure records) from NM (Node Manage), and calculates the node's computing capability. It is necessary to dynamically update the node status when a task is completed, because the node's computing power changes at this time. $Capable_j$ denotes the execution capability exhibited by the node Rj where the resource container of the resource request is located, The expressions are as follows:

$$Capable_{\mathrm{j}} = (\alpha * \frac{cpuSpeed_j}{\sum_{j=1}^{n} \frac{cpuSpeed_j}{n}} + \beta * \frac{M_j}{\sum_{j=1}^{n} \frac{M_j}{n}} + \gamma * \frac{\psi}{failNote_{t,i} + 1}) * (1 - load_j) \quad (1)$$

$$\mathrm{load}_{\mathrm{j}} = \frac{M_{\mathrm{j}} - resM_j}{M_j} \quad (2)$$

In the above formula, $failNote_{t,i}$ represents the failure record of the job Ji belonging to Ct on node Rj; α,β,and γ are constants, which are the weight values of the CPU rate, memory and failure record of the container respectively ,and α+β+γ=1. For a single node, the probability of job failure is very small, so the value of γ is set to 0.1. The value of α, β is determined by the job type: For CPU-intensive tasks, CPU has greater impact on task execution ability, so it sets α =0.6, β =0.3, γ=0.1; For data-intensive tasks, the effect of memory on task execution performance is more significant, so it sets α=0.3, β=0.6, and γ=0.1. For mixed tasks, memory and data have different effects on execution capability, where α = 0.45, β = 0.45, and γ = 0.1. For specific tasks, parameters can be further adjusted to optimize the scheduling effect. The task execution failure factor Ψ on the node is constant, indicating that the execution of the failed task on the node affects the subsequent node resources.

Therefore, when the resource request of the job arrives, the resource scheduler can calculate the node computing capability that meets the request in the cluster, and allocate resources to the task request with the data perception technology. Improve resource utilization and job completion time.

3.3 Data Location Perception

Data locality is the location of data to be processed, which is close to the node where the task is located. In the ideal situation, the scheduler allocates the container resource for the map task on the node where the input data for this task is located, so that before the task is executed, the system does not need to transfer the input data over the network to the node on which the container is located. Due to bandwidth resources are limited and data transmission time over the network is prone to a performance bottleneck, Hadoop resources scheduler should allocate as many map tasks to the desired node as possible to reduce data transmission.

When the job is submitted, AM requests the resources from RM to execute the task.through periodic heartbeat in the form of five tuples <Priority, host, capability, containers, relax_locality>. Let the set of resource request for all Ji∈J be $\mathrm{Req} = \{req_1, req_2, \cdots, req_u\}$.

$$\mathrm{req}_r = \{\mathrm{Priority}_r, host_r, capacity_r, containers_r, relax\_localty_r\}, \quad r \in [1,u] \tag{3}$$

Where, $\mathrm{Priority}_r, host_r, capacity_r, containers_r, relax\_localty_r$ respectively represent the priority of the request, the amount of resources (the form of memory and CPU), the expected host location, the number of containers, and whether the locality is relaxed. The system should request resources for the job's AM, map tasks, and reduce tasks. Each resource request for a job represents a series of task requests because the same type of task usually has the same resource request. However, the location of resource requests for map tasks will be different, so we need to consider their data locality.

The process of resource request and scheduler resource allocation is shown in Fig.1. The AM uses periodic heartbeats to request the resources demanded for task execution in the job from RM. The RM stores the resource request in the data structure and puts it into the maintenance queue. When the cluster has idle resources, that is, when the NM reports the heartbeat to the RM, the RM triggers a NODE_UPDATE event to inform the ResourceScheduler to allocate resources for the task according to the scheduling policy.

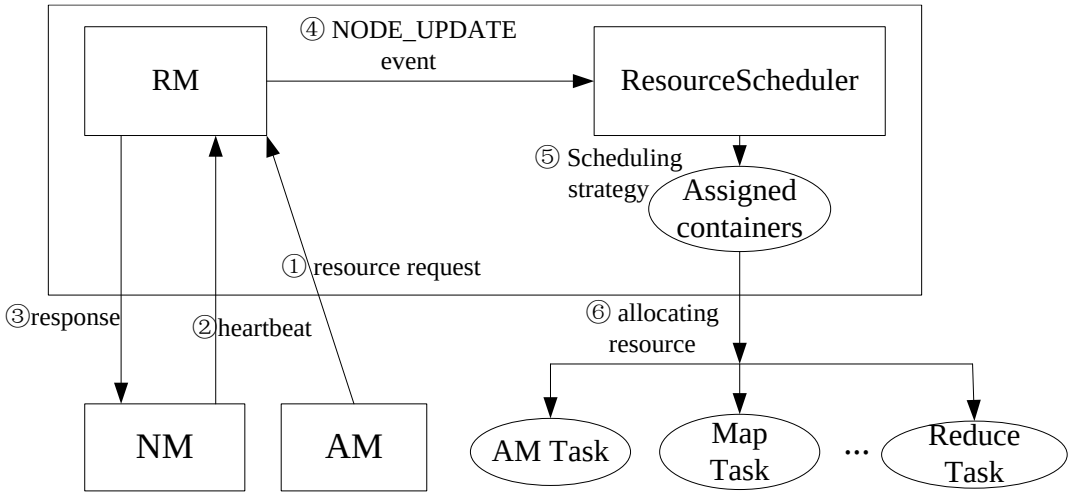

Fig 1. The process of resource request and scheduler resource allocation

When there is available resource from the node Rj, the data perception score localFScorei is calculated for each resource request ,as shown in the Eq.(4).

$$Local FScore_i = \begin{cases} a, & req_i.host = R_j \\ b, & req_i.host = \{rack \mid R_j \in rack\} \\ 0, & others \end{cases} \tag{4}$$

Where, a, b represent the weight of different data location preferences (normalized). Reqi.host represents the node location of the desired resource: The value "Rj" means the allocated container and the input data are on the same node, The value {rack|Rj∈rack) means that the allocated container and input data are on the same rack, the value "other" indicates that the assigned container and the input data are on different racks of the cluster. LocalFScorei is calculated based on the relationship between the resource request and the location of the available resources, and the resource requests satisfying the locality of the data (including the same node and the same rack) are preferentially allocated. When input data and containers with available free resources are on a rack or on different racks, the node with the highest computing power is selected to allocate resources for the tasks.

3.4 Scheduling Algorithm Design

The resource scheduler makes reasonable resource allocation based on node computing power and data location. When there are available containers and input data are on the same data node, the

container is preferentially assigned to the request. Otherwise, the container is allocated to the request based on the node's computational capabilities $Capable_j$.

The scheduler monitors the resource state changes in the cluster. When the cluster has the idle resource resMj from node Rj, the NCLDR scheduler first traverses the current resource request set Req of all jobs and selects the available request subset availReq of Req, which meets the following conditions Eq.(5):

$$\begin{cases} \text{req}_r.capacity.memory \le resM_j.memory \\ \text{req}_r.capacity.cpu \le resM_j.cpu \end{cases} \tag{5}$$

When the conditions are met, the scheduler traverses the elements in the availReq of the subset and computes the scheduling score totalScorer for each resource request reqr∈availReq, using the Eq.(6) and Eq.(7) to calculate.

$$\text{totalScore}_r = norm(Capable_j) + LocalFScore_r \tag{6}$$

$$norm(x) = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{7}$$

Eq.(7) represents the min-max algorithm, which is used to scores for the final scheduling decision. $LocalFScore_r$ represents the data location aware algorithm score of reqi according to equation (4).

Algorithm 1. NCLDR scheduling algorithm

```
Input: Resource resMj from Node Rj, A list of all resource requests Req
Output: A list of task requests to be allocated
availReq ß ←∅
foreach reqi in Req do
        if( resMj is enough for required resource of reqi)
        availReq = availReq ∪ {reqi}
         end if
while resMj > 0 && reqi in availReq do
         if(reqi is the request for ApplicationMaster) then
         allocateContainers(reqi)
         resMj -= reqi.capacity
        continue
        end if

/* Step 1: Calculate the total score for each reqi */
selReq <- null
maxScore<- 0
foreach reqi in availReq do
         totalScore[req] = Normalize (Capablej) + localScore[req]
if(totalScore >= maxScore) then
        maxScore = totalScore
        selReq = reqi
end if

/* Step 2: Allocate containers for selected selReq */
maxContainers= getMaxAllocatableContainers(selReq)
if (maxContainers > 0) then
        allocateContainersOnNode(selReq)
         resMj -= resMj .capacity
end if
end while
return A list of selReq
```

When a resource request is to request AM for the job, the resource is preferentially assigned to this request, because it is necessary to firstly run an AM task that manages a single job before the job starts running. The AM periodically communicates with the RM and requests resources to perform map/reduce tasks for the job. For other types of resource requests, the min-max algorithm scheduling algorithm will first calculate the scheduling score, and select the request with the highest score, then allocate the resource for the request, and at the same time, update the remaining idle resource status. Scheduling is stopped until the idle resource resMj can no longer be allocated or the cluster resource request is empty. The proposed scheduling algorithm is shown in Algorithm 1.

4. Experiments

4.1 Experimental Environment

To evaluate the overall performance of the NCLDR algorithm, a test environment was deployed. In heterogeneous environment, there are 6 computing nodes, 1 master node, and 5 slave nodes. In order to study the impact of data locality on resource scheduling, this experiment added rack awareness and set up 3 node racks. The specific configuration is shown in Table 1. The operating system version is Ubuntu16.04 and OpenJDK 1.7 is installed.

Table 1. Experiment specific configuration

Node	CPU/GHz	Memory/GB	Cores	Racks
Master	2.4	2	2	Rack0
Slave1	2.25	1	1	Rack1
Slave2	2.4	1	2	Rack2
Slave3	2.4	3	2	Rack2
Slave4	2.4	2	2	Rack2
Slave5	2.89	1	1	Rack1

This experiment runs a hybrid set of job sets with different workload types to validate the performance of the NCLDR scheduling algorithm. This paper uses the following three benchmarks.

TestDFSIO performs read and write operations concurrently and can be used to test the I / O performance of cluster storage systems. TestDFSIO is divided into two applications: Writing and Reading. Write program randomly generates a file with specified data size and writes it to the storage system, which is the input of Read program. TestDFSIO does not require a lot of calculations, but requires a lot of disk read and write operations, so it is a typical I/O-intensive application.

Sort application sorts the input data by using the MapReduce framework, and then directly places the sort results into the output directory. The input and output must be a serialized files. The keys and values must be written in bytes. RandomWriter generates a random file with specified data size as input to the Sort application. Sorting requires a certain amount of CPU resources, and there are a lot of I/O operations during the execution of the application, so Sort application is not a typical CPU-intensive, but a hybrid application.

WordCount statistics the number of the input file, which is a CPU-intensive application.

4.2 Experimental Results and Analysis

Using the Hadoop cluster FIFO, the fair scheduler, and the proposed NCLDR scheduling algorithm to test the same job set. Each method runs more than 10 times. The average execution time, the average CPU utilization, and the memory utilization obtained after the test are respectively shown in Fig.2, Fig.3 and Fig.4.

As shown in Fig.2, the average execution time of the FIFO is 570s, and the average execution time of the FAIR is 403s. The average time of the NCLDR is 340s. Known from the average completion time, compared with the two schedulers built in Hadoop, the scheduling algorithm proposed in this paper makes the running time of each program significantly shorter.

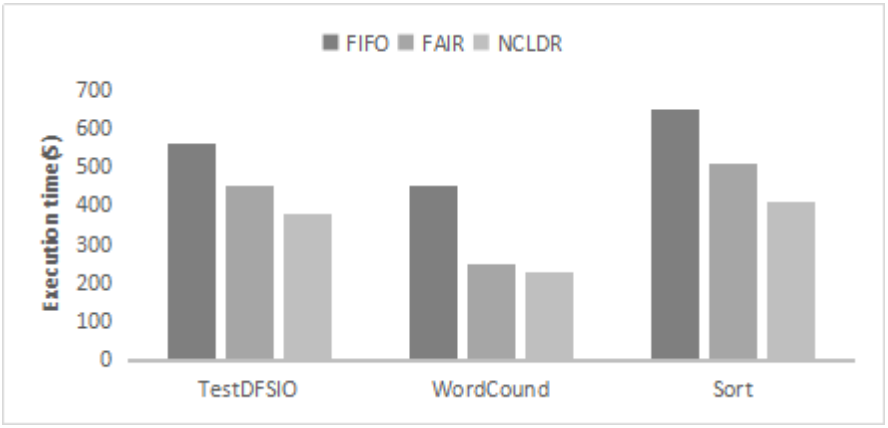

Fig 2. The average completion time of the same set of job sets under different scheduling algorithms

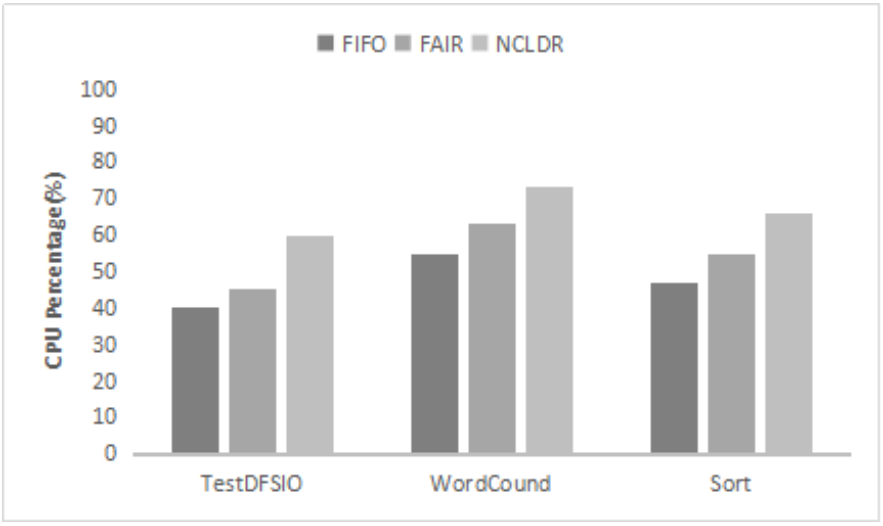

Fig 3. The average CPU utilization of the same set of job sets under different scheduling algorithms

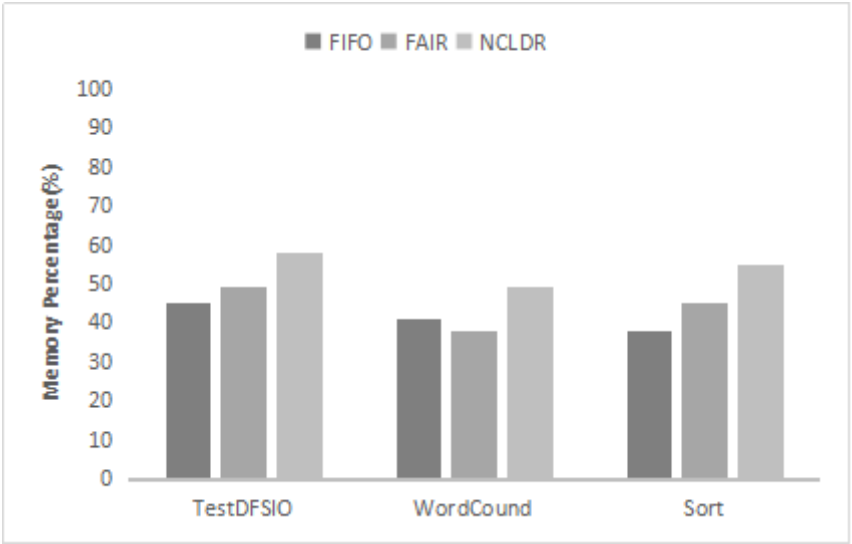

Fig 4. The average memory utilization of the same set of job sets under different scheduling algorithms

As shown in Fig.3 and Fig.4, under the three different benchmark tests, the average CPU utilization of FIFO, FAIR, NCLDR is 41.3, 44, 71.6, respectively. The average memory utilization is 38,43.54.

Table 2(Corresponding to Fig.2, Fig.3 and Fig.4) shows that average Completion time and average CPU utilization and average memory utilization for the same set of job sets under different scheduling algorithms.

Table 2. Completion time and average resource utilization of the same job set under different scheduling algorithms

Scheduler	Execution time(s)	Average CPU Usage (%)	Average memory Usage(%)
FIFO	553	45.7	38
FAIR	403	54.3	43
NCLDR	340	66.3	54

From Fig.2, Fig.3, Fig.4 and Table 2, we can see that compared with the FIFO scheduler, the proposed NCLDR scheduling algorithm improves CPU utilization rate by 45.1% and memory utilization rate by 42.1%. Compared with the Fair scheduling algorithm, the proposed NCLDR scheduling algorithm improves CPU utilization rate by 22.1% and improves memory utilization rate by 25.6%. In addition, the execution time of the proposed NCLDR scheduling algorithm is reduced by 38.6% compared with FIFO scheduling algorithm. Compared with FAIR algorithm, the execution time is reduced by 15.6%.

Due to the current Yarn scheduler is based on CPU and memory resources, so this paper only compares the completion time, CPU and memory utilization. The execution time is the total execution time of all job sets in the cluster. The shorter the execution time, the higher the utilization rate of cluster resources and the better the performance of the scheduling algorithm. The average resource utilization represents the CPU and memory usage of the nodes in the cluster. From the analysis of the above results, we can see that the NCLDR algorithm proposed in this paper is better than the default scheduler in Hadoop.

5. Conclusion and Future Work

In view of the following two issues, a resource allocation algorithm based on node computing capability and data locality is proposed in this paper. Two issues are: 1) The performance difference of different nodes in the cluster leads to the phenomenon of uneven busy and idle, which affects the efficiency of job execution; 2) It does not take fully consideration into the location of the data, which results in unnecessary Data transmission and takes up extra network bandwidth and time. The proposed algorithm first calculates the computational power of each node in the cluster and preferentially assigns the resources to the task request that is at the same node as the input data, according to the data location awareness score. Experimental results show that the proposed algorithm can effectively reduce the execution time of the job and improve the utilization of resources. In the future research work, more considerations will be added to this research method, such as the study of dynamic job scheduling. For example, some types of jobs may use more CPUs earlier but use I/O later. In many cases, the scheduling of this type of operation has yet to be further studied. More often than not, the scheduling of this type of operation has yet to be further studied. In the research direction of future work, the proposed model and resource allocation algorithm can also be migrated to other frameworks, such as Storm, Spark, and so on.

References

[1] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Communications of the ACM 2008;51(1):107–113.

[2] Chauhan J, Makaroff D, Grassmann W. Simulation and performance evaluation of the hadoop capacity scheduler[C]// International Conference on Computer Science and Software Engineering. IBM Corp. 2014:163-177.

[3] Li yuanzhen,Yang qun, et al. Research on a Resource Scheduling Method of Hadoop Yarn[J]. Chinese Journal of Electronics, 2016, 44(5):1017-1024.

[4] Hou X, Ashwin Kumar T K, Thomas J P, et al. Dynamic workload balancing for hadoop mapreduce[C] .Big Data and Cloud Computing (BdCloud) , 2014 IEEE Fourth International Conference on.IEEE, 2014: 56-62.

[5] Ibrahim S, Jin H, Lu L, et al. Handling partitioning skew in mapreduce using leen[J].Peer-to-Peer Networking and Applications, 2013, 6(4) : 409-424.

[6] Tao Yong-cai, Shi Lei. Optimization of map reduce performance in heterogeneous resource environments[J].Journal of Chinese Computer Systems 2013, 34(2) : 287-292

[7] Xie Jiong, Yin Shu, Ruan Xiao-jun, et al. Improving mapreduce performance through data placement in heterogeneous hadoop clusters[C] Proc.of 2010 IEEE International Symposium on Parallel &Distributed Processing, Workshops and Phd Forum, Atlanta, Georgia, 2010: 1-9

[8] Zaharia M, Konwinski A, Joseph A D, et al. Improving MapReduce performance in heterogeneous environments[C]// Usenix Conference on Operating Systems Design and Implementation. USENIX Association, 2008:29-42.

[9] Chen Q, Liu C, Xiao Z. Improving MapReduce Performance Using Smart Speculative Execution Strategy[J]. IEEE Transactions on Computers, 2014, 63(4):954-967.

[10] Liu Q, Cai W, Jin D, et al. Estimation Accuracy on Execution Time of Run-Time Tasks in a Heterogeneous Distributed Environment[J]. Sensors, 2016, 16(9):1386.

[11] Xu X, Cao L, Wang X. Adaptive Task Scheduling Strategy Based on Dynamic Workload Adjustment for Heterogeneous Hadoop Clusters[J]. IEEE Systems Journal, 2017, 10(2):471-482.

[12] He Xi, Zhang Xiangli, Zhang Hongmei, et al. Real-time job scheduling algorithm in heterogeneous Hadoop environment[J]. Computer Engineering and Applications, 2016, 52(16): 100-104.