2012 International Conference on Information and Computer Applications (ICICA 2012)
IPCSIT vol. 24 (2012) © (2012) IACSIT Press, Singapore

A Locality Enhanced Scheduling Method for Multiple MapReduce Jobs In a Workflow Application

Dongjin Yoo[+] and Kwang Mong Sim

Multi-Agent and Cloud Computing Systems Laboratory

School of information and Communication, Gwangju Institute of Science and Technology (GIST), Gwangju, Republic of Korea

Abstract. MapReduce is an emerging paradigm processing massive data over computing Clouds. It provides an easy programming interface, high scalability, and fault tolerance. For achieving better performances, there were many scheduling issues for map-reduce jobs in shared cluster environments. In particular, there is a need for scheduling workflow services composed of multiple MapReduce tasks with precedence dependency in shared cluster environments. By using the list scheduling approach, the issue of precedence constraints can be addressed. The major factor affecting the performances of map-reduce jobs is locality constraints for reducing data transfer cost in limited bisection bandwidth. Since multiple map-reduce jobs in a workflow are running in shared clusters, when placing data sets, concurrency also should be considered for locality enhancement. The proposed scheduling approach provides 1) a data pre-placement strategy for improvement of locality and concurrency and 2) a scheduling algorithm considering locality and concurrency.

Keywords: MapReduce, Scheduling, Workflow Application, Data Intensive Computing, Cloud Computing

1. Introduction

MapReduce [4] is a framework based on a functional programming model for data intensive application. The powerfulness of MapReduce enables distributed parallel programming for application involving the processing of a large amount of data with easy interface. Increasingly MapReduce is employed as the main model for applications requiring parallel resources of cloud computing infrastructures for processing a huge amount of data.

Many researchers have devised scheduling mechanisms for efficiently assigning map-reduce jobs to allow the sharing of multiple clusters for achieving better performance. For example, Hadoop [3] uses FairScheduler [2] to satisfy fairness criterion and CapacityScheduler [1] to guarantee a certain level of capacity for each job. Some researchers adopt an asynchronous processing approach to relieve synchronization overhead or serious I/O cost over networks. Another major issue of scheduling map-reduce jobs is minimizing the transmission cost over network.

However, there are few works that focused on optimizing mechanisms for scheduling workflows that consist of multiple map-reduce tasks. Directed Acyclic graphs (DAGs) are used to represent workflow applications with precedence dependency constraints. Since the DAG scheduling problem is NP-complete in general, several heuristics have been proposed. One of the good heuristics is the list scheduling algorithm [5]. The basic idea of list scheduling is making a list of jobs with priorities and assigning jobs by using a rank score function.

[+] Corresponding author.
E-mail address: dongjinyoo@gist.ac.kr

In addition, the data transmission cost is the major factor that affects the performances of data intensive computing applications due to limited bisection bandwidth. The data transfer cost is closely related to the concept of locality which is the distance between a node with input data and a task assigned node. Depending on how the data sets are placed over a shared cluster environment, the data transfer cost can be a major factor during the execution of tasks. Since there can be multiple map-reduce jobs running simultaneously in a shared cluster of nodes, when placing the data sets, concurrency should be considered so as to allow as many map-reduce jobs to be executed as possible.

This paper investigates a scheduling mechanism for a workflow service containing multiple map-reduce jobs in a shared cluster environment. In this proposed mechanism, both 1) the data placement strategy and 2) the scheduling algorithm consider locality enhancement with concurrency.

2. Scheduling Algorithm Considering Locality and Concurrency

In this section, the scheduling mechanism for multiple map-reduce jobs in a workflow applications considering precedence constraints among multiple map-reduce tasks and locality constraints is proposed. Basically, the scheduling mechanism adopts the list scheduling approach[5] to deal with precedence constraints by using a rank score function. The proposed scheduling algorithm considers locality when assigning avialable slots to the data set.

2.1 List Scheduling Method

The priority of each task is determined by following a recursive rank equation as follows:

$$rank(t_i) = 1 + \max_{t_j \in succ(t_i)} (rank(t_j)) \qquad (1)$$

where $succ(t_i)$ is the set of successors (children tasks) of task ti. A task has a higher priority for execution if it has a higher rank score. The rank score of an exit task of a workflow is 0, defined as

$$rank(t_{exit}) = 0 \qquad (2)$$

Using the rank score function to record the precedence constraints in a task workflow graph (DAG), the scheduler on the master node assigns the task with higher rank score.

2.2 Slot Allocation Algorithm

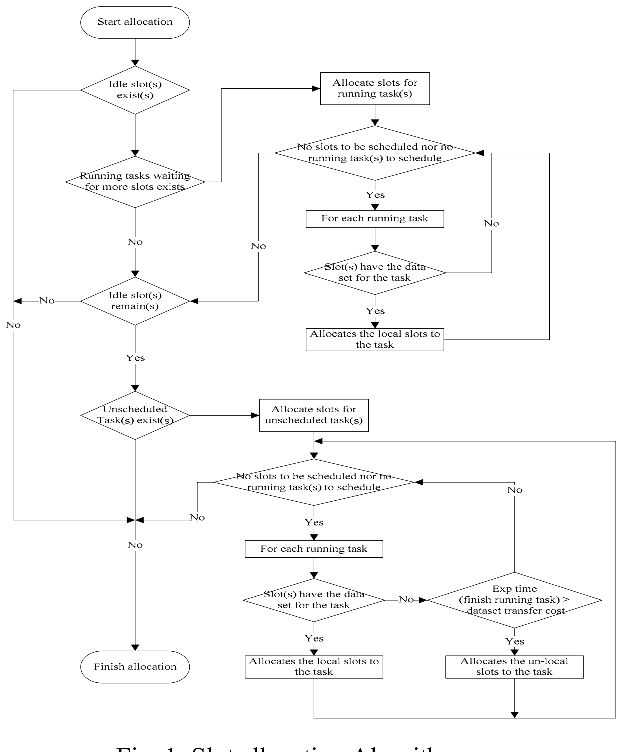

Start allocation
Idle slot(s) exist(s)
Running tasks waiting for more slots exists
No
Allocate slots for running task(s)
No slots to be scheduled nor no running task(s) to schedule
Yes
For each running task
No
Slot(s) have the data set for the task
Yes
Allocates the local slots to the task
No
Idle slot(s) remain(s)
No
No
Yes
Unscheduled Task(s) exist(s)
Allocate slots for unscheduled task(s)
No slots to be scheduled nor no running task(s) to schedule
No
Yes
For each running task
Slot(s) have the data set for the task
No
Exp time (finish running task) > dataset transfer cost
Yes
Yes
Allocates the local slots to the task
Allocates the un-local slots to the task
No
Finish allocation

Fig. 1: Slot allocation Algorithm

In addition to the mechanism of list scheduling, the proposed scheduling algorithm also considers both locality and concurrency constraints. The slot allocation mechanism of the master node is shown in Figure 1. The order of allocating tasks to idle slots is determined by the rank score. Initially, the master node schedules idle slots for running tasks. After allocating running tasks, the master allocates idle slots to unscheduled task(s). This scheduling algorithm basically assigns nodes with locality constraints. When allocating slots to the task, the master checks whether idle node(s) have the data set for the task and allocates the slot(s) to the task. For unscheduled task(s), if the data movement cost is lower than the waiting time for the slots which have the data sets for the unscheduled task(s), the master assigns un-local slots which do not have the data set to the task with additional file transmissions. To achieve concurrency, the master assigns a group of nodes (probably a rack or a data center in reality) to a map-reduce jobs.

3. Data Placement Strategy for Locality Enhancement with Concurrency

In this section, the data placement strategy for multiple map-reduce jobs in a workflow applications considering locality enhancement with concurrency is proposed. The proposed strategy employs the concept of data cohesion score to partition data sets into groups of nodes.

3.1 Data Cohesion score

Data cohesion score represents the number of common datasets that exist between the data sets (DTi) used by a task Ti and each data set (Dj) residing in each group of nodes (Gj) as described in following equation.

$$\text{Cohesion}(Ti\ ,\ Dj) = \text{count}(\ DTi \cap Dj) \tag{3}$$

If the cohesion score is higher than the cohesion score of the data sets in other group of nodes, placing the data sets DTi used by the task sets Ti to the group Gj.is more desirable for reducing the data transfer cost.

3.2 Data placement algorithm for enhancing locality and concurrency

In pre-data placement algorithm described in Figure 2, the data sets are distributed and partitioned into a group of nodes Gj. The data sets of the tasks with the same rank are distributed in different groups of nodes (No overlapping among the data sets in the same group of nodes between Ti and Tj with the same rank score is allowed). This enhances the concurrency of the tasks with the same rank. Two replica of each data set is maintained for concurrency and fault tolerance.

```
Input: $R_k$ : set of tasks with the rank k
Output: $D_j$ : the partitioned data sets of each group of nodes ($G_j$)
begin
k = root rank
// Step 1: allocation for the first copy of each data set
For (each set of tasks with rank k, $R_k$) // descending order from the root rank
   For (each task $T_i \in R_k$)
      For (each data set $D_j$ residing in $G_j$)
         $C_j$ = Cohesion ($T_i, D_j$)
           $C_f(T_i)$ = the highest cohesion score from D
                  (f = location to allocate the first replica, no overlapping with other
                  tasks in $R_k$)

       //allocate the first replica of datasets used by the task Ti to the group of nodes
       $G_f$
      $D_f \leftarrow D_{Ti}$ (the data set used by the task $T_i$)

// Step 2: allocation for the second copy of each data set
For (each set of tasks with rank k, $R_k$) // descending order from the rank of root
   For (each task $T_i \in R_k$)
      For (each data set $D_j$ residing in $G_j$)
         $C_j$ = Cohesion ($T_i, D_j$)
         $C_s(t_i)$ = the second highest cohesion score from D
                  (s=location to allocate the second replica, no overlapping with
                  other tasks in $R_k$)
                  and $f \neq s$ (first allocated group $\neq$ second group)

       // allocates the second replica of datasets used by the task $T_i$ to the group of
       nodes $G_s$
      $D_S \leftarrow D_{Ti}$ (the data set used by the task $T_i$)
```

Figure 2. Data Placement Strategy

4. Experiments and Evaluation

A sample set of workflow graphs is given from random graph generation for experimental evaluation. For each experiment, about 750 differrent workflow is generated and experiments were conducted with two environmental setups (N, T) = {50, 40}, {100, 80} as described in Tab. 1. Also, the makespan, concurrency, count of local allocation, and count of unlocal allocation are recorded as shown Table 1.

Tab.1 Performance Measure and Experimental Setting

Average concurrency	Average number of concurrent executing jobs
Average make span	Average elapsed time for a workflow application
Average count of local allocation	Average count of local allocation from experimental samples
Average count of un-local allocation	Average count of un-local allocation from experimental samples
The number of tasks (T)	{40, 80}
The number of nodes (N)	{50, 100}
Environmental Setup (N, T)	{(50, 40), (100, 80)}

Empirical results were obtained with two experimental settings ((N, T) = {(50, 40), (100, 80)}) as shown in Figures 3, 4, 5 and 6. To evaluate the performance of the data placement strategy, the experimental results using random data placement were obtained.

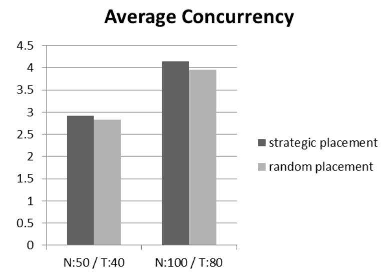


Figure 3. Average Concurrency

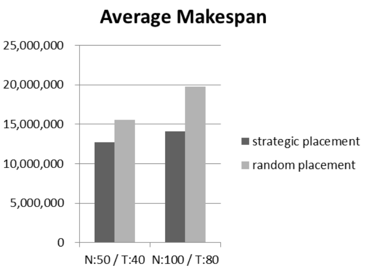
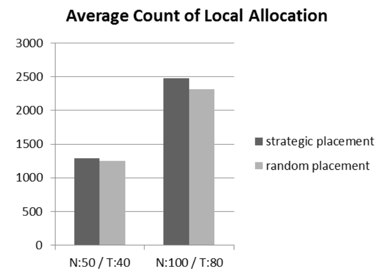


Figure 4. Average Makespan



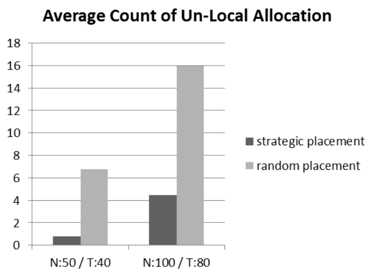


Figure 5. Average Count of Local Allocatio Figure 6. Average Count of Un-Local Allocation

From the results, the scheduling mechanism with the data placement strategy acheives a slightly better averge concurrency. Also, the scheduling mechanism with data placement strategy acheives better locality as shown in Figures 5 and 6. Thus, with a higher concurrency and a better locality of the data plecement strategy, data transfer cost is reduced, and consequently the makespan with strategic data placement is much shorter than the makespan with random data placement. From the results, it can be seen that the scheduling mechanism with data pre-placement strategy acheives shorter execution times by enhancing concurrency and locality.

6. Conclusion

The contribution of this paper is proposing a scheduling approach for a workflow application which has a set of multiple MapReduce jobs with precedence dependency. The proposed method adopts the list scheduling method to deal with precedence constraints, which can be represented as a DAG. Also the proposed mechanism takes into consideration of issues such as locality and concurrency. The scheduling mechanism with data placement strategy acheives better locality and concurrency, and this leads shorter execution times by reducing data transfer costs during execution.

7. Acknowledgement

This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MEST) (KRF-2009-220-D00092).

8. References

[1] Apache Software Foundation , Capacity Scheduler, http://hadoop.apache.org/common/docs/current/capacity_scheduler.html.

[2] Apache Software Foundation. FairScheduler, http://hadoop.apache.org/mapreduce/docs/r0.21.0/fair_scheduler.html

[3] Apache Software Foundation. Hadoop, http://hadoop.apache.org/core/.

[4] J. Dean and S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters. In proceeding of the 6th Symposium on Operating systems Design and Implementation (OSDI 2004), pp 137-150. USENIX Association, 2004.

[5] H. Topcuouglu, S. Hariri, M.Y. Wu, Performance-effective and lowcomplexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 260–274.