Probabilistic Network-Aware Task Placement for MapReduce Scheduling

Haiying Shen*, Ankur Sarker*, Lei Yu†, and Feng Deng*
*Department of Electrical and Computer Engineering, Clemson University, Clemson, SC 29634, USA
†College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280, USA
*{shenh, asarker, fengd}@clemson.edu, †lyu79@gatech.edu

***Abstract*—Maximizing data locality in task scheduling is critical for the performance of MapReduce job execution. Many existing works on MapReduce scheduling decide the placement of map and reduce tasks on a coarse granularity of locations measured by located machines and racks. They do not explicitly consider the network topology and data transmission cost, which may cause task straggling and degrade the job performance. In order to improve MapReduce job performance, in this paper, we consider the task placement with the goal of minimizing the overall data transmission cost for a job execution while balancing the transmission cost reduction and resource utilization. We propose a probabilistic network-aware scheduling algorithm that selects a task (map task or reduce task) to be scheduled on a given available task slot that leads to the minimum transmission cost among the task candidates, and then schedule the selected task on the slot with a probability determined by its transmission cost; a lower expected transmission cost leads to a higher probability and vice versa. We also propose a method to more accurately estimate the intermediate data size based on the progress of map tasks, which is needed to calculate the transmission cost of reduce tasks but is unknown at the time of reduce task scheduling. We implement our probabilistic network-aware scheduling algorithm on Apache Hadoop and conduct experiments on a high-performance computing platform. The experimental results show that our scheduling algorithm outperforms the previous approaches in terms of job completion time and cluster resource utilization.**

***Index Terms*—MapReduce, task scheduling, job scheduling**

I. Introduction

MapReduce [1], with its open-source implementation Hadoop [2], has become a popular paradigm for large-scale data processing in clusters. A MapReduce job is decomposed to map tasks and reduce tasks running in parallel. As shown in Figure 1, a job's input data is divided into blocks, which are stored in the cluster machines (called data nodes). Each machine has a specific number of map slots and reduce slots. Each map task processes an input data block using the user-defined *map* function and produces intermediate data in the local disk. The intermediate data is then shuffled to the reduce tasks that apply the user-defined *reduce* function to generate the final output.

In MapReduce, the task scheduler is centralized and a single node is responsible for scheduling tasks to different nodes. One key factor contributing to the efficient parallel computation of MapReduce is the principle of moving computation towards data, which is achieved by the task scheduler. Specifically, the task scheduler assigns a mapper to a machine to handle an input data block (which may or may not be in this machine), and assigns a reducer to a machine to handle shuffled data (which may or may not be in this machine). With each map task being scheduled on a machine as close to its handled input data as possible, the network bandwidth is saved. Similarly, with each reduce task being scheduled on a machine as close to its handled shuffled data as possible, the network bandwidth is saved. Therefore, maximizing data locality (i.e., scheduling map tasks and reduce tasks to their handled data) is a primary goal of task scheduling in MapReduce. Several schedulers have been proposed to improve the existing scheduler in Hadoop for data locality. Delay Scheduler [3] schedules map tasks, LARTS [4] schedules reduce tasks and Coupling Scheduler [5] schedules both map tasks and reduce tasks.

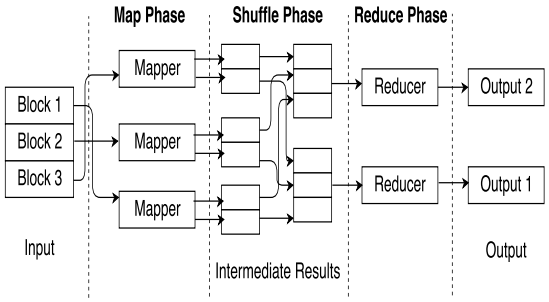


Fig. 1: Map, shuffle and reduce phases in MapReduce [6].

Although data locality is desired, it is not always possible to place all tasks on the machines where their handled data is stored. Due to the capacity constraints, the machines storing the handled data may not have enough resources to accommodate the tasks at the scheduling point. Delay Scheduler [3] extends the Fair Scheduler [7]. It improves the data locality by delaying the execution of map tasks for a period of time to find available machines storing their handled data for them. However, as pointed out in [5], such execution delay can lead to resource under-utilization, i.e., the number of map tasks running simultaneously can be far below a desired level. Also, it causes the number of running map tasks to change significantly over time when the input data is not uniformly distributed. Coupling Scheduling [5] shows that launching remote map tasks (i.e., scheduling tasks to machines not close to their handled data) to avoid delay can be beneficial. In this case, the task scheduled machine must fetch data from another machine that stores its handled data.

When it is necessary to schedule tasks on machines with remote data accesses, the placement of tasks is important for the performance of MapReduce. The distances between tasks and their remote handled data as well as the data size decide the data transmission cost and access latencies, which have great impact on the job completion time. Thus, it is critical to find optimal task placement that minimizes the transmission cost and access latencies to reduce job execution time. Existing works on task scheduling [3–5, 7–9] only decide the task placements on a coarse granularity (e.g., on local machines, on machines in the same rack and on off-rack machines) without explicitly considering the network topology and the resulting latencies. As a result, these methods may lead to task straggling and degrade the job performance when data replicas are distributed among different racks or stored in NAS or SAN devices located in a subset of the nodes in a cloud system [10].

In this paper, we aim to design a task scheduler with optimal task placements to minimize overall data transmission cost and delays and hence to reduce job execution time while balancing the transmission cost reduction and resource utilization. This task is faced with three challenges. First, because the limited resources are shared among multiple jobs, the available machines for running tasks dynamically change due to resource allocation and release over time. The tasks of a job are gradually scheduled in an on-line manner in the MapReduce clusters. Thus, it is not possible to obtain a global optimal placement of all tasks for a job. Second, the data fetching time of reduce tasks depends on not only the placement of reduce tasks but also the locations and sizes of the intermediate data produced by map tasks. The placement of reduce tasks must consider the distribution of the intermediate data in the cluster. However, at the time of reduce task scheduling, the complete information of intermediate data is not available. Third, the link load on the routing path also has a significant impact on the data access latency. In order to reduce the latency, the link status of the network must be considered in the scheduling decision, which however may greatly increase the complexity of the scheduler for achieving the optimality.

To address these challenges, we propose a probabilistic network-aware on-line scheduling algorithm for map task and reduce task. The algorithms select a map task or a reduce task to be scheduled on a given available mapper or reducer computing slot that leads to the minimum transmission cost (hence latency) for using this slot among the task candidates, and then schedule the selected task on the slot with a probability determined by its transmission cost; a lower expected transmission cost leads to a higher probability and vice versa. In this way, we can reduce the expected data transmission cost while enabling tasks to have fair opportunities to be allocated. Also, we propose a method to more accurately estimate the intermediate data size based on the progress of map tasks, which is needed to calculate the transmission cost of reduce tasks but is unknown at the time of reduce task scheduling. Further, we consider the network condition in calculating the transmission cost; data transmission in a link with a higher transmission rate generates a lower transmission cost and vice versa. Instead of using task execution delay [3], our probabilistic network-aware approach schedules the tasks on available computing slots immediately as long as the expected network cost is low with the calculated probability. We use the probabilistic approach rather than the deterministic approach in order to enable tasks to have fair opportunities to be allocated. We implement our probabilistic network-aware scheduling algorithm on Apache Hadoop version 1.2.1 and conduct various experiments on the Palmetto high-performance computing platform [11] which is located at Clemson University. The experimental results show that our scheduling algorithm achieves better job completion time, data locality and cluster resource utilization than the existing Fair Scheduler and Coupling Scheduler. The rest of this paper is organized as follows. Section II describes the problem and our scheduling algorithms. Section III presents our experimental results and evaluation. Section IV introduces the related work. Section V concludes the paper with remarks on our future work.

II. Probabilistic Network-Aware Task Scheduling

A. System Model and Problem Description

A MapReduce cluster uses slot-based resource management model. Each physical node is configured as multiple computing slots for map tasks and reduce tasks. With multiple jobs simultaneously running in the cluster, the MapReduce scheduler essentially implements two levels of scheduling: job-level scheduling and task-level scheduling. The job-level scheduling first decides for which job its tasks should be scheduled, and the task-level scheduling further decides which task of this job is assigned to which computing slot. The MapReduce scheduler makes scheduling decisions at the time of receiving a heartbeat from a node indicating slot availability. In this paper, we mainly focus on the task-level scheduling. For the job-level scheduling, we can use an existing scheduler such as FIFO Scheduler [2], or Fair Scheduler [7], or Capacity Scheduler [12]. Our method is focused on how to place the map and reduce tasks to available computing slots for a job of which tasks should be scheduled at this time. The main idea is to minimize the overall data transmission cost from map tasks to their input blocks and from reduce tasks to their intermediate data produced by map tasks, while balancing the transmission cost reduction and resource utilization. To solve this problem, we propose a probabilistic network-aware task scheduling scheme. In the following sections, we first present the computation of the transmission cost for a given task placement in Section II-B, and then present transmission cost based placement algorithms for map tasks and reduce tasks in Section II-C and Section II-D, respectively.

B. Transmission Cost Computation

The data transmission cost of a task is determined by the distance between the task and its handled data and the total size of the data to be transferred. We denote the data nodes in the cluster as a set, $\mathcal{D} = \{D_1, D_2, ..., D_k\}$. Below, we introduce how to calculate the transmission cost for map tasks

TABLE I: Notations.

Symbol	Definition
D_i	ith data node
J_i	ith job
M_i	ith available map task
H	Distance matrix
h_{ab}	The number hops between D_a and D_b
B_j	The size of data block for M_j
L_{ij}	D_i stores the data for M_j
R_i	ith reduce task
x_{ij}	M_i is assigned to D_j
I_{jf}	The amount of intermediate data M_j generates for R_f
C	Sum of transmission cost
d^i_{read}	The number of byes M_i has read
A_{ij}	The size of current intermediate data for R_j generated by M_i
P_{mj}	Probability M_j being assigned
P_{min}	Probability threshold
N_m	The number of data nodes available for map tasks

and for reduce tasks, respectively. For easy reference, Table I presents the main notations used in this paper.

1) Transmission Cost for Map Tasks: Let $M_1, M_2, ..., M_m$ be the available map tasks of a job to be assigned to the data nodes. Each map task processes a particular data block. We assume that the network topology is known and represented by a distance matrix H, where each element h_{ab} denotes the number of hops of the path between D_a and D_b. Suppose that the data block required by map task M_j is stored in data node D_l. If map task M_j is assigned to node D_i, the latency for map task M_j to fetch the data from data node D_l can be simply measured by the distance between local node D_i and data node D_l. We will introduce a more accurate method to measure the latency by considering network condition later. A local task (which is assigned to the data node, i.e., $D_i = D_l$) will have zero latency or no data transmission cost at all. Let B_j be the size of the data block for M_j. Based on the cost measurement for data transfer in [13, 14], we measure the cost of the task placement by the product of the data block size and the distance between D_i and D_l, that is, $B_j h_{il}$. The cost equals zero if $i = l$, which indicates assigning a map task to a node with local data. Generally, multiple nodes store the same data block (replica) required by M_j. For map task M_j, if data node D_l stores the data block which M_j requires, we let a binary variable $L_{lj} = 1$; otherwise $L_{lj} = 0$. Let $C_{m_{ij}}$ be the transmission cost for map task M_j placed on node D_i. Suppose M_j reads the data block from the nearest node that stores the data block, $C_{m_{ij}}$ can be calculated by

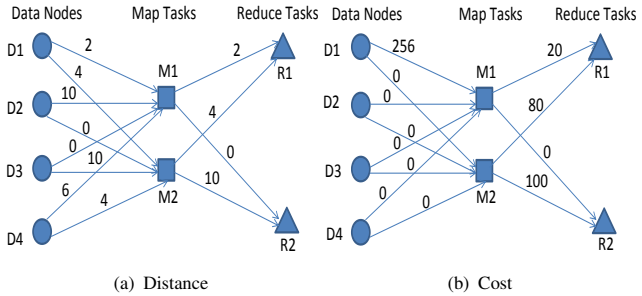
$$C_{m_{ij}} = B_j \min_{L_{lj}=1} h_{il} \tag{1}$$

2) Transmission Cost for Reduce Tasks: Let $R_1, R_2, ..., R_n$ be the reduce tasks. Each reduce task processes a different partition in the key space of the shuffled data, generally produced by several map tasks. The transmission cost of assigning a reduce task to a data node equals the sum of the costs to transfer the intermediate data in the reduce task's corresponding responsible partition from all map tasks to this node. When scheduling reduce tasks, the assignments of map tasks are already finished and all map tasks are placed for execution. We represent the map task placement by a binary matrix X where x_{jp} ($j \in \{1, ..., m\}, p \in \{1, ...k\}$) is set to 1 if map task M_j is assigned to data node D_p, or 0 otherwise.

Let I be a $m \times n$ matrix. Each element I_{jf} ($j \in \{1, ..., m\}, f \in \{1, ..., n\}$) represents the amount of intermediate data which the map task M_j generates for the reduce task R_f. For a map task M_j placed on the data node D_p and a reduce task R_f placed on the data note D_i, the transmission cost between map task M_j and reduce task R_f is $I_{jf}h_{pi}$. Then, the transmission cost for each reduce task R_f when it is assigned to the data node D_i, denoted by $C_{r_{if}}$, is the sum of the transmission cost from each of the map task to the reduce task, which can be represented by

$$C_{r_{if}} = \sum_{j=1}^{m} \sum_{p=1}^{k} x_{jp} h_{pi} I_{jf} \tag{2}$$

Estimation of Intermediate Data Size. Unlike map tasks, the size of complete intermediate data for each reduce task (i.e., I_{jf}) in the above formula, may be not available at the scheduling time, because the reduce tasks are normally launched before map tasks have completed processing the data. Coupling Scheduler [5] uses the current in-progress intermediate data size at the time of reduce task scheduling to compute the transmission cost [5] and decides the placement that minimizes the transmission cost. However, such a method may cause a large deviation from the optimal placement and leads to sub-optimal placement. For example, assume a map task M_2 has a data size of 10MB for the reduce task R_1 at the time of its completion, but at time t_1, it only has finished 10 percent of its work. Then, the data is approximately 1MB. At the same time, map task M_1 has already generated 5MB of data for R_1 and it has finished 90 percent of its work. If R_1 is to be assigned at time t_1 with the goal of lower transmission cost, it is most likely to be assigned to the node with M_1 or a node near it if we estimate the transmission cost based on current intermediate data size, even though the node with M_2 is actually a better location. To avoid this drawback, in our approach, when scheduling a reduce task, we use the current intermediate data size and the progress of the map tasks to estimate the size of completed map output for each reduce task. However, it is difficult to learn the progress of the map tasks. To handle this problem, we smartly use the progress of input data processing as the progress of a map task since a map task always aims to complete handling its input data. Specifically, each map task, M_j, reports its progress information to the scheduler periodically as a heartbeat [15], which includes the number of bytes it has already read from its input data, denoted by d^j_{read}, and the size of current intermediate data for reduce task R_f, denoted by A_{jf}. With the size of the input data that M_j needs to process (i.e., B_j), the scheduler predicts how many bytes M_j would produce for reduce task R_f using the model $A_{jf} \times \frac{B_j}{d^j_{read}}$. By replacing I_{jf}



(a) Distance (b) Cost

Fig. 2: Distance and transmission cost of task assignment.

in Formula (2) with $A_{jf} \times \frac{B_j}{d^j_{read}}$, then the transmission cost $C_{r_{if}}$ is computed by

$$C_{r_{if}} = \sum_{j=1}^{m} \sum_{p=1}^{k} x_{jp} h_{pi} A_{jf} \frac{B_j}{d^j_{read}} \tag{3}$$

An Example. To illustrate our proposed transmission cost function, we analyze a simplified cluster in Figure 2. In this example, there are four nodes D_1, D_2, D_3, D_4, two map tasks M_1, M_2 and two reduce tasks R_1, R_2. The data block of M_1 is on D_1 and that of M_2 is on D_2. The size of both data blocks is 128MB. The distance matrix H between data nodes is:

$$\begin{array}{c} \\ D_1 \\ D_2 \\ D_3 \\ D_4 \end{array} \begin{array}{c} \begin{array}{cccc} D_1 & D_2 & D_3 & D_4 \end{array} \\ \begin{bmatrix} 0 & 4 & 2 & 8 \\ 4 & 0 & 10 & 4 \\ 2 & 10 & 0 & 6 \\ 8 & 4 & 6 & 0 \end{bmatrix} \end{array}$$

Based on existing information, we could get a cost value for each task-slot assignment using our transmission cost function. Let we want to know the transmission cost of assigning M_1 to D_3 and M_2 to D_2. The distance between M_1 (i.e., D_3) and D_1, D_2, D_3 and D_4 is 2, 10, 0 and 6, respectively. By using Formula (1), we can get the transmission cost of each assignment of a map task to a node. The distance between between M_1 (i.e., D_3) and its data node D_1 is 2, the distance between M_2 (i.e., D_2) and its data node D_2 is 0. Therefore, the transmission cost for M_1 is $128 \times 2 = 256$ and the cost for M_2 is $128 \times 0 = 0$. Assume reduce task R_1 is assigned to D_1 and R_2 is assigned to D_3. Similarly, we can have the distance matrix from two mappers to the reducers:

$$\begin{array}{c} \\ M_1 \\ M_2 \end{array} \begin{array}{c} \begin{array}{cc} R_1 & R_2 \end{array} \\ \begin{bmatrix} 2 & 0 \\ 4 & 10 \end{bmatrix} \end{array}$$

As a result, Figure 2(a) shows the distance map for the cluster with this assignment. The size of intermediate data is represented in megabytes by matrix I, which is:

$$\begin{array}{c} \\ M_1 \\ M_2 \end{array} \begin{array}{c} \begin{array}{cc} R_1 & R_2 \end{array} \\ \begin{bmatrix} 10 & 5 \\ 20 & 10 \end{bmatrix} \end{array}$$

With the distance map, we can get the transmission cost for each assignment on each link. By using Formula (3), we can get the transmission cost of each assignment of a reduce task to a node. For the assignment shown in Figure 2(a), the transmission cost is the sum of all the link costs shown in Figure 2(b).

3) Considering Network Condition: We further increase the accuracy of the transmission cost measurement by considering network condition. As the network bandwidth is shared among multiple jobs and the links have varied available bandwidths, the number of hops between two nodes may not accurately reflect data access latency through a path. However, it is not easy to consider the network condition in calculating the transmission cost. We notice that the network condition of a path from node D_a to D_b can be reflected by the data transmission rate of this path. Also, there can be link status monitoring in the cluster or active measure for path throughput [16]. Then, we can replace each element h_{ab} in distance matrix H by the inverse of the transmission rate of the path from node D_a to D_b. In this way, we can take into account the bandwidth factor for computing transmission cost function, which helps to produce a more efficient task placement.

C. Map Task Placement

When a job is being scheduled, its map tasks are first assigned to the available computing slots. When a node sends a heartbeat to the scheduler indicating that it has available slots for placement, the transmission cost of assigning a map task to its slot can be calculated based on Formula (1). To assign a map task, we could use a similar approach as previous work [3] to wait for a period to capture the opportunity of running the task on the node with the smallest transmission cost, but the delay may lead to lower resource utilization and longer job completion time [5]. Thus, to avoid delay, we propose a transmission cost based probabilistic network-aware solution, which determines the probability of an assignment of a map task to an available slot as a function of data transmission cost under this assignment. Rather than assigning the task with the lowest transmission cost instantly (that improves resource utilization with degraded data locality), we use such a probability to determine whether a task can be assigned to the slot in order to achieve a balance between the transmission cost reduction and resource utilization. Though our approach and Coupling Scheduling [17] are both probabilistic for map task scheduling, Coupling Scheduling decides task placement based on a coarse granularity of locations that differentiates data locations by local machines, the same rack and different racks, while our approach considers a fine-grained granularity on network cost based on transmission cost for better scheduling placement.

Given a list of candidate map tasks to be launched and a list of data nodes that have available map slots, the algorithm first randomly selects a node D_i and then selects the map task that leads to the maximum transmission cost saving by assigning it instantly to D_i than assigning it to other nodes for scheduling. For the map task selection, based on the data transmission cost, our algorithm first determines the probability P_{m_j} of

each map task M_j being assigned to node D_i. Probability P_{m_j} reflects how much cost is saved by assigning M_j instantly to D_i rather than assigning it to other nodes with available map slots; higher P_{m_j} means that more cost is saved. Then, the algorithm picks the map task with the highest probability P_{m_j} among all the candidates in the list. If P_{m_j} is too low (lower than a threshold P_{min}), the algorithm skips the current node D_i and then randomly selects another node with available map slot for scheduling. Otherwise, the algorithm assigns the map task to node D_i with its probability P_{m_j}. P_{min} is determined based on historical data to ensure that a certain amount of cost can be saved in a map task scheduling. After the completion of map task scheduling on node D_i, the algorithm will randomly select another data node from the list and repeat the map task scheduling process for that data node. The pseudocode

Algorithm 1 Probabilistic network-aware map task placement algorithm.

Input: A list of map tasks of job J to be scheduled; Node D_i with available map slots; Distance matrix H

1: **if** D_i has an available map slot **then**
2: **for** each unassigned map task M_j in the list **do**
3: ▷ *Calculate the cost of assigning map task M_j to D_i using H*
4: $C_{m_{ij}} = B_j \min_{L_{lj=1}} h_{il}$
5: ▷ *Calculate the expected cost of assigning map task M_j uniformly to the available nodes*
6: $C_{m_{ave}} = \sum_{k=1}^{N_m} C_{m_{kj}}/N_m$
7: $P_{m_j} = 1 - e^{-\frac{C_{m_{ave}}}{C_{m_{ij}}}}$
8: **end for**
9: Choose the map task M_j with the largest P_{m_j}
10: **if** $P_{m_j} < P_{min}$ **then**
11: Return
12: **end if**
13: P = random(0,1)
14: **if** $P < P_{m_j}$ **then**
15: Assign M_j to D_i
16: **end if**
17: **end if**

of the map task scheduling algorithm on node D_i is shown in Algorithm 1. The algorithm is triggered when JobTracker receives a heartbeat and there is available map task to be scheduled. The inputs of the algorithm include a list of map tasks of a job to be scheduled, a data node D_i with available map slots, and distance matrix H. For each map task M_j in the list, the algorithm calculates its P_{m_j} (Lines 2-8). Specifically, the algorithm calculates $C_{m_{ij}}$, which represents the cost of the placement of M_j to D_i computed by Formula (1) (Line 4). It then calculates the expected cost $C_{m_{ave}}$ of assigning a map task M_j to the cluster by $\sum_{k=1}^{N_m} C_{m_{kj}}/N_m$, where N_m be the number of data nodes that have available map slots (Line 6). The probability P_{m_j} is determined by the ratio of the expected cost $C_{m_{ave}}$ to the data transmission cost for M_j being placed on D_i, and computed as:

$$\begin{aligned} P_{m_j} &= 1 - e^{-\frac{C_{m_{ave}}}{C_{m_{ij}}}} \\ &= 1 - e^{-\frac{\sum_{k=1}^{N_m} C_{m_{kj}}/N_m}{C_{m_{ij}}}} \end{aligned} \tag{4}$$

where we let $P_{m_j} = 1$ if $C_{m_{ij}} = 0$ (Line 7). The intuition behind this algorithm is to give a higher priority to the map task if assigning it instantly to D_i is expected to incur much lower cost than assigning it to other nodes. When the average placement cost is much less than the cost incurred by assigning the task to D_i, there should be nodes which have much lower cost than D_i for the assignment. When the ratio increases, the benefit of assigning the map task later to other nodes decreases. If the data is available in D_i to the map task, the task is always assigned to D_i with $P_{m_j} = 1$.

Our algorithm selects the map task with the highest probability P_{m_j} (Line 9), which is actually corresponding to the task with the minimum cost among all the map task candidates. Then, the algorithm decides whether to assign it based on the value of P_{m_j}. If $P_{m_j} < P_{min}$, the algorithm does not assign any map task to D_i due to large transmission cost (Lines 10-12). When the probability P_{m_j} is no less than threshold P_{min}, we have $C_{m_{ij}} \leq \frac{\sum_{k=1}^{N_m} C_{m_{kj}}/N_m}{-\ln(1-P_{min})}$ based on Formula (4). That is, the task is assigned to D_i only if the cost is no less than $\frac{1}{-\ln(1-P_{min})}$ of the expected cost. If $P_{m_j} \geq P_{min}$, M_j is assigned to D_i with probability P_{m_j} (Lines 13-16).

Algorithm 2 Probabilistic network-aware reduce task placement algorithm.

Input: A list of reduce tasks of job J to be scheduled; Node D_i with available reduce slots; Distance matrix H

1: **if** D_i has available reduce slots and has no running reduce tasks of J **then**
2: **for** each unassigned reduce task R_f in Job J **do**
3: Calculate the distance map for task R_f
4: ▷ *Calculate the cost of assigning reduce task R_f to D_i using H*
5: $C_{r_{if}} = \sum_{j=1}^{m} \sum_{p=1}^{k} x_{jp} h_{pi} A_{jf} \frac{B_j}{d^j_{read}}$
6: ▷ *Calculate the expected cost of assigning reduce task R_f uniformly to the nodes (matrix I)*
7: $C_{r_{ave}} = \sum_{k=1}^{N_r} C_{r_{kf}}/N_r$
8: $P_{r_f} = 1 - e^{-\frac{C_{r_{ave}}}{C_{r_{if}}}}$
9: **end for**
10: Choose the reduce task R_f with the largest P_{r_f}
11: **if** $P_{r_f} < P_{min}$ **then**
12: Return
13: **end if**
14: P= random(0,1)
15: **if** $P < P_{r_f}$ **then**
16: Assign R_f to D_i
17: **end if**
18: **end if**

D. Reduce Task Placement

For reduce task scheduling, we aim to assign each reduce task to a node that would incur the least transmission cost

of intermediate data. The reduce task scheduling algorithm shares the similar idea as the map task placement algorithm. Algorithm 2 shows the pseudocode of the reduce task scheduling algorithm on node D_i. The algorithm is triggered when a heartbeat is received and there is available reduce task to be scheduled. The algorithm also computes a probability P_{r_f} for each reduce task R_f of the job to be scheduled, considers the reduce task with the highest probability P_{r_f} among all the reduce tasks, and then assigns this task to the node D_i with its probability P_{r_f}. Besides, in the algorithm, we avoid running multiple reduce tasks of a job on the same node as in [5, 15] in order to decrease their I/O contention and avoid the downlink congestion on the node (Line 1).

The probability P_{r_f} is computed in a similar way as P_{m_j} (Line 8). Let N_r be the number of nodes that have available reduce task slots. The expected cost $C_{r_{ave}}$ of assigning the reduce task R_f to the cluster is calculated by $\sum_{k=1}^{N_r} C_{r_{kf}}/N_r$, where $C_{r_{kf}}$ represents the cost of the placement of R_f to D_k computed by Formula (3) (Line 7). The probability P_{r_f} for each reduce task launched on the slot can be calculated by

$$P_{r_f} = 1 - e^{-\frac{\sum_{k=1}^{N_r} C_{r_{kf}}/N_r}{C_{r_{if}}}} \quad (5)$$

If $P_{r_j} < P_{min}$, the algorithm does not assign any reduce task to D_i due to large transmission cost (Line 11-12). Otherwise, R_f is assigned to D_i with probability P_{f_j} (Lines 15-16). As we can see, the probabilistic strategy for reduce task assignment is the same as that for map task. The key difference between the assignments for map tasks and reduce tasks is the computation of the transmission cost for the assignment.

III. Performance Evaluation

To evaluate our probabilistic network-aware scheduling method, we implement our method and the coupling scheduling method [5] on Apache Hadoop version 1.2.1. Apache Hadoop version 1.2.1 uses the fair scheduling method [7], which uses the delay scheduling in the map task allocation. The fair scheduling method delays the map task assignment for data locality, achieve fairness among jobs and randomly selects a reduce task to be assigned to an available reduce slot. In the coupling scheduling method, for an available map task slot, a randomly picked map task is assigned to it with a probability that balances data locality and resource utilization; the reduce tasks can be postponed to be launched in order to be assigned to the data "centrality" nodes and can wait at most three rounds of heartbeats before being assigned. Here, the data "centrality" node means the data node that is approximately located in the middle of all data nodes and can greatly decrease the data transmission overhead if the reduce task is assigned to it. We conduct various experiments on the Palmetto high-performance computing platform [11] located at the Clemson University. The Palmetto cluster currently comprises of 1,978 slave nodes (the nodes that do the computational work), and a few service nodes that handle managing activities. The entire Palmetto cluster is connected to Internet2's 100 GbE (gigabit Ethernet) Advanced Layer 2 Service. All nodes are connected to a top of rack switch within a rack. Most top of rack switches are uplinked to the core switch at 10Gbps, and some switches are aggregated to a Z9000 switch that is uplinked to the brocade at 40Gbps. We chose 60 slave nodes from the Palmetto cluster with 4 map slots and 2 reduce slots per node. Each node has 16 cores (2300MHz), 16GB of memory and 37GB of disk. To find the appropriate value of P_{min}, we ran 10 Wordcount jobs together several times with different P_{min} values and picked the highest P_{min} value at the time when the all jobs finished successfully. Accordingly, we set P_{min} to 0.4. The input workload in the experiments was generated based on representative Hadoop applications including Wordcount, Terasort, and Grep.

TABLE II: The description of the 30 jobs.

JobID	Job	Map (#)	Reduce (#)
01	Wordcount_10GB	88	157
02	Wordcount_20GB	160	169
03	Wordcount_30GB	278	159
04	Wordcount_40GB	502	169
05	Wordcount_50GB	490	127
06	Wordcount_60GB	645	187
07	Wordcount_70GB	598	165
08	Wordcount_80GB	818	291
09	Wordcount_90GB	837	157
10	Wordcount_100GB	930	197
11	Terasort_10GB	143	190
12	Terasort_20GB	199	186
13	Terasort_30GB	364	131
14	Terasort_40GB	320	149
15	Terasort_50GB	490	189
16	Terasort_60GB	480	193
17	Terasort_70GB	560	178
18	Terasort_80GB	648	184
19	Terasort_90GB	753	171
20	Terasort_100GB	824	193
21	Grep_10GB	87	148
22	Grep_20GB	163	174
23	Grep_30GB	188	184
24	Grep_40GB	203	158
25	Grep_50GB	285	164
26	Grep_60GB	389	137
27	Grep_70GB	578	179
28	Grep_80GB	634	178
29	Grep_90GB	815	164
30	Grep_100GB	893	184

We compared our probabilistic network-aware scheduling method with the coupling scheduling method [5] and the fair scheduling method [7] in terms of job completion time, task completion time, cluster resource utilization, and data locality. Recall that our method focuses on the task-level scheduling, we used the default fair scheduling method of Hadoop for job scheduling. In our experiment, we created 3 batches of jobs, which consists of 10 Wordcount jobs, 10 TeraSort jobs, and 10 Grep jobs, respectively, and run these 3 batches separately on the Palmetto cluster. In the batches of the Wordcount, TeraSort, and Grep jobs, the input data size was varied from 10GB to 100GB. For the Wordcount and Grep jobs, we generated the input data by BigDataBench [18] based on the Wikipedia datasets. BigDataBench is an open-source big data benchmark

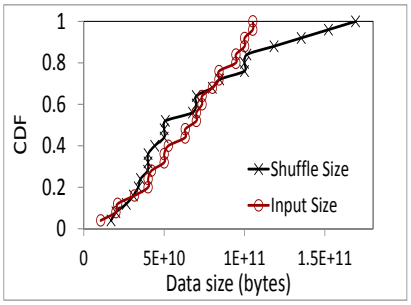


Fig. 3: CDF of data size.

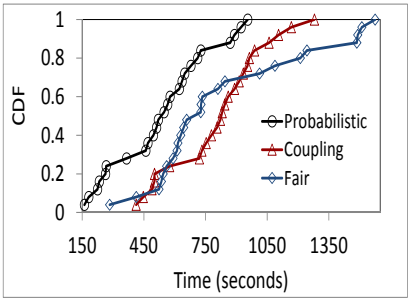


Fig. 4: CDF of job completion time.

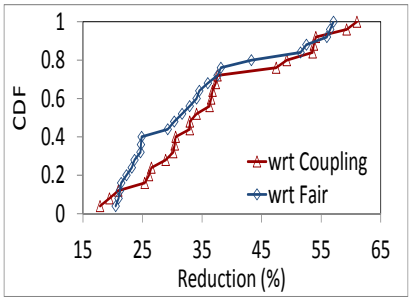


Fig. 5: Reduction of job processing time.

suite, which includes 14 real-world data sets and 33 big data workloads. For the Terasort jobs, we generated the input data using Teragen (another example job from Hadoop benchmark).

The generated files are stored in slave nodes with the replication factor being set to 2. We run each of the three batches at one time. Table II show the input data size, number of map task and reduce task of each job in the three batches in the experiments. To further illustrate the characteristics of the input workload, Figure 3 shows the Cumulative Distribution Function (CDF) of input data size and the shuffle data size from map tasks in the submitted jobs. We can see that about 60 percent of jobs have more than 50GB shuffle data size, and about 20 percent of jobs have more than 100GB shuffle data size. These jobs are considered as shuffle-intensive jobs. There are about 20 percent of jobs having less than 10GB shuffle data size, which can be considered as computing (or map) intensive jobs that mainly rely on map tasks to process data. For these map intensive jobs and shuffle intensive jobs, the total execution time is sensitive to the delay of map tasks and the placement for both map tasks and reduce tasks. Therefore, these jobs also give a better illustration of the performance of job completion time of different schedulers under heavy network bandwidth usage.

A. Job Completion Time

We first measured the job completion time using our proposed probabilistic network-aware scheduling method and compared it with the coupling scheduling method and fair scheduling method. Figure 4 shows the CDF of the job completion time when the replication factor equals to 2. Replication factor is the number of replicas of each file. We can see that given a running time t, the probabilistic network-aware scheduling method produces a higher percentage of jobs that complete within t time than other two methods. Actually, our experimental results show that using the probabilistic network-aware scheduling method, the completion time of each job is shorter than the job completion time using the coupling scheduling method or the fair scheduling method. To demonstrate this, we draw Figure 5 to show the reduction of the job processing time for replication factor 2 achieved by the probabilistic scheduling method compared with the coupling scheduling and fair scheduling methods. The x-axis is the reduction percentage of job processing time calculated by $((coupling - probabilistic)/coupling$ or $(fair - probabilistic)/fair)$, and the y-axis represents the CDF of the jobs versus the reduction percentage. The Figure 5(a) shows that, when the replication factor is 2, with the probabilistic network-aware scheduling method, the completion times of about 28 percent of jobs decrease by more than 47 percent compared with the coupling scheduling method and about 24 percent of jobs have more than 43 percent reduction compared with the fair scheduling method. On average, the probabilistic network-aware scheduling method decreases the job processing time by 17 percent and 46 percent compared with the coupling scheduling and fair scheduling, respectively.

The above results demonstrate that our proposed probabilistic network-aware scheduling method outperforms the coupling scheduling and fair scheduling methods in terms of job completion time. It is mainly caused by three reasons. First, unlike the other two methods, our method uses fine-grained transmission cost measurement instead of coarse-grained measurement that only considers the located machine and rack. It can more accurately measure the delay caused by the task placement and generate better task placement. Its consideration of network condition in transmission cost calculation further improves its performance. Second, the fair scheduling method delays the map task assignment for data locality and randomly selects a reduce task to be assigned to an available reduce slot. In the coupling scheduling method, the reduce tasks are postponed to be launched and can wait at most three rounds of heartbeats before being assigned. The delay strategy could increase the job completion time due to the delay though it achieves high data locality. In our method, a task is assigned probabilistically once a slot is

available, which increases the opportunity of its assignment to the cluster at an earlier time, as long as the resulted transmission cost is low enough. The third reason is the estimation of the data size of the intermediate data for reduce tasks in our probabilistic network-aware scheduling method. We use the progress information of map tasks to estimate the complete input data size for each reduce task. However, the coupling scheduling method uses the current intermediate data to compute the data "centrality" nodes for reduce tasks, which could be insufficiently accurate and results in sub-optimal placement.

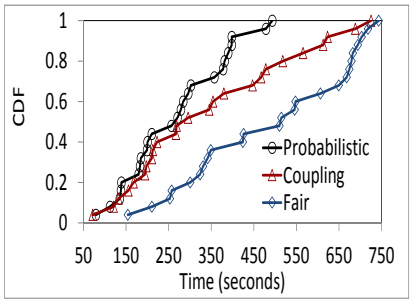
B. Task Running Time

To further examine the completion time of each job, we draw the CDF of the running time for map tasks and reduce tasks in Figure 6 for replication factor 2. The figures show that both map tasks and reduce tasks finish earlier by using our probabilistic network-aware scheduling method than using the coupling scheduling and fair scheduling methods. As shown in Figure 6(a), when the replication factor is 2, all map tasks can finish within 493 seconds in the probabilistic scheduling method, while only 76 percent of map tasks using the coupling scheduling method and only 48 percent of map tasks using the fair scheduling method can finish within 493 seconds. Figure 6(b) shows that all reduce tasks can finish within 574 seconds in the probabilistic scheduling method, while only around 65 percent of reduce tasks using the coupling scheduling method and only around 85 percent of reduce tasks using the fair scheduling method can finish within 574 seconds. The reasons for the higher performance of our probabilistic network-aware scheduling method are the same as explained previously.

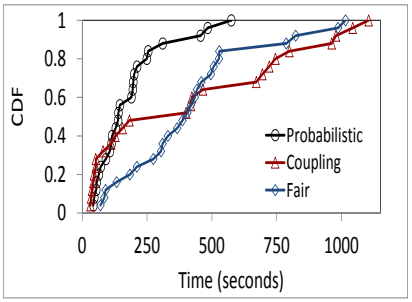
C. Percentage of Local Tasks

A map or reduce task that is assigned to a machine with data for that task is referred to as a *local task*. A map or reduce task that is assigned to a machine without local data but in the rack having the machine with local data is referred to as a *local rack task*, and other map or reduce tasks are referred to as *remote tasks*. Local tasks increase, then either local rack tasks or remote tasks decrease, and vice-versa.

Table III shows the percentage of the three types of tasks using three different scheduling methods. We can see that although the fair scheduling method tries to delay map tasks in order to execute them in the data nodes, both our method and the coupling scheduling method achieve higher percentage of local tasks than it. This is because the fair scheduling method randomly assigns reduce tasks to slots, while our method tries to assign both map tasks and reduce tasks to nodes that lead to low transmission costs and the coupling scheduling method also considers locality in assigning both map and reduce tasks. Our method generates a higher percent of local node tasks than the coupling scheduling method because the coupling scheduling method assigns a reduce task to a random slot if it is postponed for a certain time. A higher percentage of local node tasks leads to a lower percentage of local rack tasks. We



(a) Map tasks (2 replications)



(b) Reduce tasks (2 replications)

Fig. 6: CDF of task completion time.

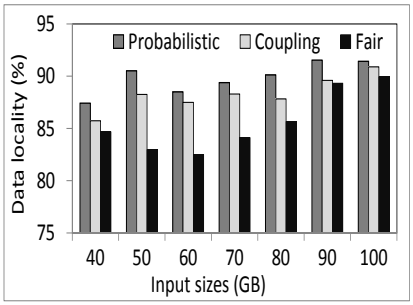


Fig. 7: The percentage of map tasks with local data.

do not find remote tasks in the experiment because the slave nodes we requested were all assigned to the same rack by Palmetto.

Figure 7 gives a further illustration for the performance of data locality with different input data sizes. The x-axis represents the input data size for jobs, the y-axis represents the percentage of local node tasks. The figure shows that our method constantly achieves better data locality among different scheduling methods under different input sizes due to the same reasons explained above. The coupling scheduling method considers data locality for both map task and reduce task assignment. The fair scheduling method does not consider the data locality for the reduce tasks. Therefore, the coupling scheduling method incurs a higher percentage of local node tasks than the fair scheduling method.

IV. Related Work

Three schedulers commonly used in the MapReduce cluster are FIFO Scheduler [2], Fair Scheduler [7] and Capacity Scheduler [12]. FIFO Scheduler is limited in its ability to run

TABLE III: Details on data locality using the three schedulers.

	Probabilistic	Coupling	Fair
% of local node tasks	89.84	88.30	85.59
% of local rack tasks	10.16	11.70	14.41
% of remote tasks	0	0	0

heterogeneous jobs from different users because it determines which job to schedule in the order of job arrivals without job priority mechanism. Fair Scheduler and Capacity Scheduler aim to manage and arbitrate the scheduling of jobs to satisfy the need of different jobs and achieve fairness among jobs by allocating equal share of the available resources among jobs and among users, respectively. In Fair Scheduler, jobs are organized into pools, and slot resources are fairly divided between these pools. In each pool, jobs can be scheduled using either Fair Scheduler or FIFO Scheduler. Capacity Scheduler is designed to run jobs as a shared, multi-tenant cluster to maximize the throughput and the utilization of the cluster. It gives a higher priority to a job that can achieve higher data locality when assigning available slot resources in the map task allocation and delays reduce tasks to achieve data locality in the reduce task allocation. These schedulers place all the tasks of each job to the nodes in the cluster with a primary goal of data locality by running the tasks on nodes where the required input blocks reside or as close to those nodes as possible.

A number of works have been proposed to improve the task scheduling for MapReduce jobs. Delay Scheduling [3] extends Fair Scheduler [7] to improve data locality by letting map tasks wait for the opportunities to run on a node with local data. LARTS [4] is a location-aware reduce task scheduler, which schedules the reduce tasks as close to their maximum amount of input data as possible and thus decreases the bandwidth cost during shuffling. In existing schedulers, either greedily launching reduce tasks or delaying the launching of map tasks to improve data locality can lead to resource under-utilization. Coupling Scheduler [5] addresses these problems by gradually launching the reduce tasks according to the progresses of map tasks, and randomly launching the map tasks with a probability of balancing data locality and resource utilization. Dominant Resource Fairness Scheduler [19] achieves a max-min fairness for multiple resources (e.g., CPU, memory and I/O resources). Quincy Scheduler [20] uses the min-cost flow algorithm to find out the optimal task placement by calculating the cost of map task assignment based on the data locality and fairness. All these scheduling schemes, however, do not explicitly take into account the network cost for deciding the placement of tasks, which may lead to excessive latency in shuffling and degrade the performance of job execution.

Considering that placing reduce tasks to slots with the lowest transmission cost for a job can prevent a subsequent job from being launched on these occupied slots with better data locality for it, Tan *et al.* [21] proposed a stochastic optimization framework to improve the data locality for reduce tasks within sequential map reduce jobs. Kondikoppa *et al.* [22] proposed a network aware scheduling in Hadoop to minimize the transmission cost of the data among clusters. However, they enable the Delay Scheduling [3] to take maximum use of data locality in scheduling tasks, which would lead to under-utilization of the cluster resources. Tang *et al.* [23] proposed a scheme for MapReduce task scheduling with deadline constraints in the Hadoop platform, in which nodes are classified by their computing capacity and tasks are scheduled to these nodes according to the map and reduce task slot requirements for meeting deadlines. Wang *et al.* [24] proposed a map task scheduling algorithm with a new queueing architecture, with the goal of striking an appropriate balance between data locality and load balancing to maximize throughput and minimize delay. Alicherry *et al.* [10] proposed a virtual machine placement algorithm to minimize overall data access latency for a job. However, it assumes static resource availability on nodes and one-time placement for all virtual machines. Another recent work [25] deploys a cluster scheduler, which packs tasks to machines based on their requirements along multiple resources by adaptively learning task requirement and monitoring available resources at machines to overcome the limitation of the existing schedulers based on only one resource (e.g., memory). Dogar *et al.* [26] demonstrated a decentralized task-aware network scheduling algorithm, which schedules a group of tasks together regarding their network flow by dynamically adapting the level of multiplexing in the network to reduce the average waiting time and tail task completion. The above works on MapReduce scheduling decide the placement of tasks on a coarse granularity of locations that differentiates data locations measured by located machines and ranks. Unlike these previous works, we consider fine-grained granularity on network cost based on transmission latency for better task placement.

V. Conclusion

In this paper, we focus on the problem of task placement in MapReduce task scheduling. Previous works on MapReduce scheduling consider the placement of map/reduce tasks on a coarse granularity of locations without taking into account the network topology and link bandwidth. As a result, they may lead to degraded job performance in cloud systems that have data replicas stored among different racks, in NAS or SAN devices located in a subset of nodes, or in a shared cluster with varied and dynamic bandwidth utilization of links. To address this issue, we proposed a transmission cost based probabilistic scheduling method. Based on the data transmission cost, our method decides the probability to assign a task to an available computing slot. The transmission cost computation takes into account the network condition and data size. A lower transmission cost leads to a higher probability of task placement and vice versa. The probabilistic approach relaxes the condition to assign a task to a computing slot, which helps avoid the delay while reducing the data transmission cost with a high probability. The experimental results conducted on the real testbed demonstrate that our method improves the job completion time and resource utilization compared with the coupling scheduling and fair scheduling methods.

Currently, we determine the probability of assigning a task to a slot based on an exponential model based on the ratio of the expected transmission cost to the transmission cost resulted from the corresponding assignment. However, the optimality of this model is not known. In the future, we will conduct a theoretical analysis for the performance of our probabilistic network-aware scheduling method. We will further explore various probabilistic computation models for the probability determination and study their impacts on the job performance. We will also evaluate the performance of our method under different network conditions (e.g., bandwidth utilization), and implement it in the most recent YARN [27] framework and evaluate its performance.

ACKNOWLEDGEMENTS

This research was supported in part by U.S. NSF grants NSF-1404981, IIS-1354123, CNS-1254006, IBM Faculty Award 5501145, and Microsoft Research Faculty Fellowship 8300751.

REFERENCES

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] "Hadoop," http://hadoop.apache.org/ [Accessed in February 2016].

[3] 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 Computer systems*, 2010.

[4] M. Hammoud and M. Sakr, "Locality-aware reduce task scheduling for mapreduce," in *Proc. of CloudCom*, 2011.

[5] J. Tan, X. Meng, and L. Zhang, "Coupling task progress for mapreduce resource-aware scheduling," in *Proc. of INFOCOM*, 2013.

[6] "Admin New Day HPC," http://www.admin-magazine.com/HPC/Articles/MapReduce-and-Hadoop [Accessed in February 2016].

[7] "Fair scheduler," http://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html [Accessed in February 2016].

[8] G. Liu, H. Shen, and H. Wang, "Computing load aware and long-view load balancing for cluster storage systems," in *Proc. of Big Data*. IEEE, 2015, pp. 174–183.

[9] Z. Li, H. Shen, W. B. L. III, and J. Denton, "An exploration of designing a hybrid scale-up/out hadoop architecture based on performance measurements," *IEEE Transactions on Parallel and Distributed Systems*, vol. P-P, no. 99, pp. 1–1, 2016.

[10] M. Alicherry and T. Lakshman, "Optimizing data access latencies in cloud systems by intelligent virtual machine placement," in *Proc. of INFOCOM*, 2013.

[11] "Clemson Palmetto HPC Cluster," http://citi.clemson.edu/palmetto/. [Accessed in February 2016].

[12] "Capacity scheduler," http://hadoop.apache.org/docs/r2.3.0/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html [Accessed in February 2016].

[13] A. Beloglazov and R. Buyya, "Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient dDnamic Consolidation of Virtual Machines in Cloud Data Centers." *CCPE*, 2011.

[14] C. Peng, M. Kim, Z. Zhang, and H. Lei, "VDN: Virtual Machine Image Distribution Network for Cloud Data Centers." in *Proc. of INFOCOM*, 2012.

[15] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in map-reduce clusters using mantri," in *Proc. of OSDI*, 2010.

[16] K. LaCurts, S. Deng, A. Goyal, and H. Balakrishnan, "Choreo: network-aware task placement for cloud applications," in *Proc. of IMC*, 2013.

[17] J. Tan, X. Meng, and L. Zhang, "Coupling scheduler for mapreduce/hadoop," in *Proc. of international symposium on High-Performance Parallel and Distributed Computing*, 2012.

[18] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang *et al.*, "Bigdatabench: A big data benchmark suite from internet services," in *Proc. of HPCA*, 2014.

[19] 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 NSDI*, 2011.

[20] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: Fair scheduling for distributed computing clusters," in *Proc. of SOSP*, 2009, pp. 261–276.

[21] J. Tan, S. Meng, X. Meng, and L. Zhang, "Improving reducetask data locality for sequential mapreduce jobs," in *Proc. of INFOCOM*, 2013, pp. 1627–1635.

[22] P. Kondikoppa, C.-H. Chiu, C. Cui, L. Xue, and S.-J. Park., "Network-aware scheduling of mapreduce framework on distributed clusters over high speed networks," in *Proc. of FederatedClouds*. ACM, 2012, pp. 39–44.

[23] Z. Tang, J. Zhou, K. Li, and R. Li, "A mapreduce task scheduling algorithm for deadline constraints," *Cluster computing*, vol. 16, no. 4, pp. 651–662, 2013.

[24] W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang, "Map task scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality," in *Proc. of INFOCOM*, 2013, pp. 1609–1617.

[25] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella., "Multi-resource packing for cluster schedulers," in *Proc. of SIGCOMM*. ACM, 2014, pp. 455–466.

[26] F. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron., "Decentralized task-aware scheduling for data center networks," in *Proc. of SIGCOMM*. ACM, 2014, pp. 431–442.

[27] "Hadoop NextGen MapReduce (YARN)," http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html [Accessed in February 2016].