This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2419671, IEEE Transactions on Parallel and Distributed Systems

On Traffic-Aware Partition and Aggregation in MapReduce for Big Data Applications

Huan Ke, *Student Member, IEEE*, Peng Li, *Member, IEEE*,
Song Guo, *Senior Member, IEEE*, and Minyi Guo, *Senior Member, IEEE*

Abstract—The MapReduce programming model simplifies large-scale data processing on commodity cluster by exploiting parallel map tasks and reduce tasks. Although many efforts have been made to improve the performance of MapReduce jobs, they ignore the network traffic generated in the shuffle phase, which plays a critical role in performance enhancement. Traditionally, a hash function is used to partition intermediate data among reduce tasks, which, however, is not traffic-efficient because network topology and data size associated with each key are not taken into consideration. In this paper, we study to reduce network traffic cost for a MapReduce job by designing a novel intermediate data partition scheme. Furthermore, we jointly consider the aggregator placement problem, where each aggregator can reduce merged traffic from multiple map tasks. A decomposition-based distributed algorithm is proposed to deal with the large-scale optimization problem for big data application and an online algorithm is also designed to adjust data partition and aggregation in a dynamic manner. Finally, extensive simulation results demonstrate that our proposals can significantly reduce network traffic cost under both offline and online cases.

1 Introduction

MapReduce [1] [2] [3] has emerged as the most popular computing framework for big data processing due to its simple programming model and automatic management of parallel execution. MapReduce and its open source implementation Hadoop [4] [5] have been adopted by leading companies, such as Yahoo!, Google and Facebook, for various big data applications, such as machine learning [6] [7] [8], bioinformatics [9] [10] [11], and cybersecurity [12] [13].

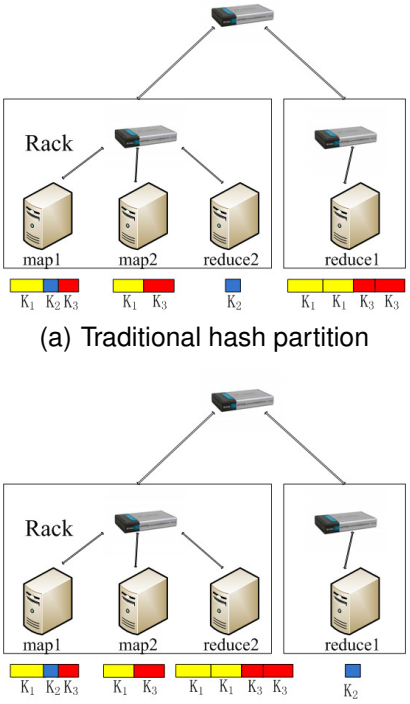
MapReduce divides a computation into two main phases, namely map and reduce, which in turn are carried out by several map tasks and reduce tasks, respectively. In the map phase, map tasks are launched in parallel to convert the original input splits into intermediate data in a form of key/value pairs. These key/value pairs are stored on local machine and organized into multiple data partitions, one per reduce task. In the reduce phase, each reduce task fetches its own share of data partitions from all map tasks to generate the final result. There is a shuffle step between map and reduce phase. In this step, the data produced by the map phase are ordered, partitioned and transferred to the appropriate machines executing the reduce phase. The resulting network traffic pattern from all map tasks to all reduce tasks can cause a great volume of network traffic, imposing a serious constraint on the efficiency of data analytic applications. For example, with tens of thousands of machines, data shuffling accounts for 58.6% of the cross-pod traffic and amounts to over 200 petabytes in total in the analysis of SCOPE jobs [14]. For shuffle-heavy MapReduce tasks, the high traffic could incur considerable performance overhead up to 30-40 % as shown in [15].

By default, intermediate data are shuffled according to a hash function [16] in Hadoop, which would lead to large network traffic because it ignores network topology and data size associated with each key. As shown in Fig. 1, we consider a toy example with two map tasks and two reduce tasks, where intermediate data of three keys K_1, K_2, and K_3 are denoted by rectangle bars under each machine. If the hash function assigns data of K_1 and K_3 to reducer 1, and K_2 to reducer 2, a large amount of traffic will go through the top switch. To tackle this problem incurred by the traffic-oblivious partition scheme, we take into account of both task locations and data size associated with each key in this paper. By assigning keys with larger data size to reduce tasks closer to map tasks, network traffic can be significantly reduced. In the same example above, if we assign K_1 and K_3 to reducer 2, and K_2 to reducer 1, as shown in Fig. 1(b), the data transferred through the top switch will be significantly reduced.

To further reduce network traffic within a MapReduce job, we consider to aggregate data with the same keys before sending them to remote reduce tasks. Although a similar function, called combiner [17], has been already adopted by Hadoop, it operates immediately after a map task solely for its generated data, failing to exploit the data aggregation opportunities among multiple tasks on different machines. As an example shown in Fig. 2(a), in the traditional scheme, two map tasks individually send data of key K_1 to the reduce task. If we aggregate the data of the same keys before sending them over the top switch, as shown in Fig. 2(b), the network traffic will be

H. Ke, P. Li and S. Guo are with the School of Computer Science and Engineering, the University of Aizu, Japan. Email: {m5172105, pengli, sguo}@u-aizu.ac.jp
M. Guo is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. Email: guo-my@cs.sjtu.edu.cn

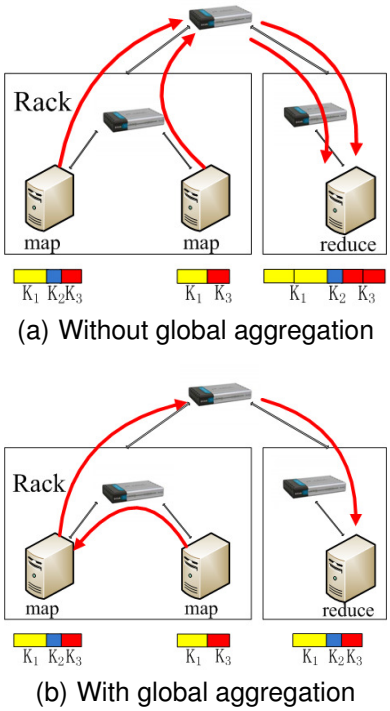
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2419671, IEEE Transactions on Parallel and Distributed Systems



(a) Traditional hash partition

(b) Traffic-aware partition

Fig. 1. Two MapReduce partition schemes.



(a) Without global aggregation

(b) With global aggregation

Fig. 2. Two schemes of intermediate data transmission in the shuffle phase.

reduced.

In this paper, we jointly consider data partition and aggregation for a MapReduce job with an objective that is to minimize the total network traffic. In particular, we propose a distributed algorithm for big data applications by decomposing the original large-scale problem into several subproblems that can be solved in parallel. Moreover, an online algorithm is designed to deal with the data partition and aggregation in a dynamic manner. Finally, extensive simulation results demonstrate that our proposals can significantly reduce network traffic cost in both offline and online cases.

The rest of the paper is organized as follows. In section II, we review recent related work. Section III presents a system model. Section IV develops a mixed-integer linear programming model for the network traffic minimization problem. Sections V and VI propose the distributed and online algorithms, respectively, for this problem. The experiment results are discussed in section VII. Finally, Section VIII concludes the paper.

2 RELATED WORK

Most existing work focuses on MapReduce performance improvement by optimizing its data transmission. Blanca et al. [18] have investigated the question of whether optimizing network usage can lead to better system performance and found that high network utilization and low network congestion should be achieved simultaneously for a job with good performance. Palanisamy et al. [19] have presented Purlieus, a MapReduce resource allocation system, to enhance the performance of MapReduce jobs in the cloud by locating intermediate data to the local machines or close-by physical machines. This locality-awareness reduces network traffic in the shuffle phase generated in the cloud data center. However, little work has studied to optimize network performance of the shuffle process that generates large amounts of data traffic in MapReduce jobs. A critical factor to the network performance in the shuffle phase is the intermediate data partition. The default scheme adopted by Hadoop is hash-based partition that would yield unbalanced loads among reduce tasks due to its unawareness of the data size associated with each key. To overcome this shortcoming, Ibrahim et al. [20] have developed a fairness-aware key partition approach that keeps track of the distribution of intermediate keys' frequencies, and guarantees a fair distribution among reduce tasks. Meanwhile, Liya et al. [21] have designed an algorithm to schedule operations based on the key distribution of intermediate key/value pairs to improve the load balance. Lars et al. [22] have proposed and evaluated two effective load balancing approaches to data skew handling for MapReduce-based entity resolution. Unfortunately, all above work focuses on load balance at reduce tasks, ignoring the network traffic during the shuffle phase.

In addition to data partition, many efforts have been made on local aggregation, in-mapper combining and in-network aggregation to reduce network traffic within

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2419671, IEEE Transactions on Parallel and Distributed Systems

MapReduce jobs. Condie et al. [23] have introduced a combiner function that reduces the amount of data to be shuffled and merged to reduce tasks. Lin and Dyer [24] have proposed an in-mapper combining scheme by exploiting the fact that mappers can preserve state across the processing of multiple input key/value pairs and defer emission of intermediate data until all input records have been processed. Both proposals are constrained to a single map task, ignoring the data aggregation opportunities from multiple map tasks. Costa et al. [25] have proposed a MapReduce-like system to decrease the traffic by pushing aggregation from the edge into the network. However, it can be only applied to the network topology with servers directly linked to other servers, which is of limited practical use.

Different from existing work, we investigate network traffic reduction within MapReduce jobs by jointly exploiting traffic-aware intermediate data partition and data aggregation among multiple map tasks.

3 System Model

MapReduce is a programming model based on two primitives: map function and reduce function. The former processes key/value pairs $\langle k, v\rangle$ and produces a set of intermediate key/value pairs $\langle k', v'\rangle$. Intermediate key/value pairs are merged and sorted based on the intermediate key k' and provided as input to the reduce function. A MapReduce job is executed over a distributed system composed of a master and a set of workers. The input is divided into chunks that are assigned to map tasks. The master schedules map tasks in the workers by taking into account of data locality. The output of the map tasks is divided into as many partitions as the number of reducers for the job. Entries with the same intermediate key should be assigned to the same partition to guarantee the correctness of the execution. All the intermediate key/value pairs of a given partition are sorted and sent to the worker with the corresponding reduce task to be executed. Default scheduling of reduce tasks does not take any data locality constraint into consideration. As a result, the amount of data that has to be transferred through the network in the shuffle process may be significant.

In this paper, we consider a typical MapReduce job on a large cluster consisting of a set N of machines. We let d_{xy} denote the distance between two machines x and y, which represents the cost of delivering a unit data. When the job is executed, two types of tasks, i.e., map and reduce, are created. The sets of map and reduce tasks are denoted by M and R, respectively, which are already placed on machines. The input data are divided into independent chunks that are processed by map tasks in parallel. The generated intermediate results in forms of key/value pairs may be shuffled and sorted by the framework, and then are fetched by reduce tasks to produce final results. We let P denote the set of keys contained in the intermediate results, and m_i^p denote the data volume of key/value pairs with key $p \in P$ generated by mapper $i \in M$.

A set of δ aggregators are available to the intermediate results before they are sent to reducers. These aggregators can be placed on any machine, and one is enough for data aggregation on each machine if adopted. The data reduction ratio of an aggregator is denoted by α, which can be obtained via profiling before job execution.

The cost of delivering a certain amount of traffic over a network link is evaluated by the product of data size and link distance. Our objective in this paper is to minimize the total network traffic cost of a MapReduce job by jointly considering aggregator placement and intermediate data partition. All symbols and variables used in this paper are summarized in Table 1.

TABLE 1
Notions and Variables

Notations	Description
N	a set of physical machines
d_{xy}	distance between two machines x and y
M	a set of map tasks in map layer
R	a set of reduce tasks in reduce layer
A	a set of nodes in aggregation layer
P	a set of intermediate keys
A_i	a set of neighbors of mapper $i \in M$
δ	maximum number of aggregators
m_i^p	data volume of key $p \in P$ generated by mapper $i \in M$
$\phi(u)$	the machine containing node u
x_{ij}^p	binary variable denoting whether mapper $i \in M$ sends data of key $p \in P$ to node $j \in A$
f_{ij}^p	traffic for key $p \in P$ from mapper $i \in M$ to node $j \in A$
I_j^p	input data of key $p \in P$ on node $j \in A$
M_j	a set of neighboring nodes of $j \in A$
O_j^p	output data of key $p \in P$ on node $j \in A$
α	data reduction ratio of an aggregator
α_j	data reduction ratio of node $j \in A$
z_j	binary variable indicating if an aggregator is placed on machine $j \in N$
y_k^p	binary variable denoting whether data of key $p \in P$ is processed by reducer $k \in R$
g_{jk}^p	the network traffic regarding key $p \in P$ from node $j \in A$ to reducer $k \in R$
z_j^p	an auxiliary variable
ν_j^p	Lagrangian multiplier
$m_j^p(t)$	output of m_j^p at time slot t
$\alpha_j(t)$	α_j at time slot t
$\Psi_{jj'}$	migration cost for aggregator from machine j to j'
$\Phi_{kk'}(\cdot)$	cost of migrating intermediate data from reducer k to k'
$C_M(t)$	total migration cost at time slot t

4 Problem Formulation

In this section, we formulate the network traffic minimization problem. To facilitate our analysis, we construct an auxiliary graph with a three-layer structure as shown in Fig. 3. The given placement of mappers and reducers applies in the map layer and the reduce layer, respectively. In the aggregation layer, we create a potential aggregator at each machine, which can aggregate data from all mappers. Since a single potential aggregator is sufficient at each machine, we also use N to denote all potential aggregators. In addition, we create a shadow

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2419671, IEEE Transactions on Parallel and Distributed Systems

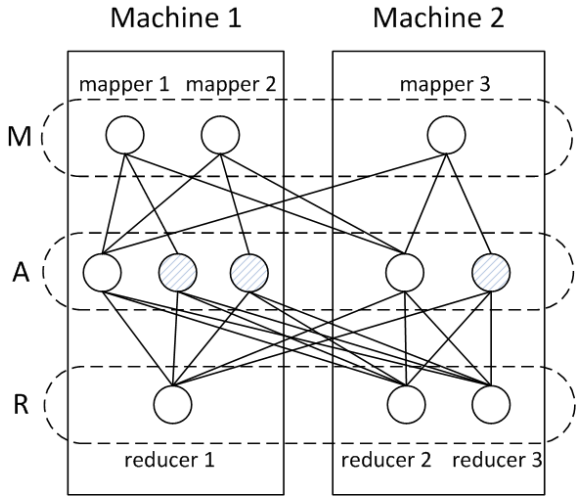


Fig. 3. Three-layer model for the network traffic minimization problem.

node for each mapper on its residential machine. In contrast with potential aggregators, each shadow node can receive data only from its corresponding mapper in the same machine. It mimics the process that the generated intermediate results will be delivered to a reduce directly without going through any aggregator. All nodes in the aggregation layers are maintained in set A. Finally, the output data of aggregation layer are sent to the reduce layer. Each edge (u, v) in the auxiliary graph is associated with a weight $d_{\phi(u)\phi(v)}$, where $\phi(u)$ denotes the machine containing node u in the auxiliary graph.

To formulate the traffic minimization problem, we first consider the data forwarding between the map layer and the aggregation layer. We define a binary variable x_{ij}^p as follows:

$$x_{ij}^p = \begin{cases} 1, \text{ if mapper } i \in M \text{ sends data of key } p \in P \\ \quad \text{to node } j \in A; \\ 0, \text{ otherwise.} \end{cases}$$

Since all data generated in the map layer should be sent to nodes in the aggregation layer, we have the following constraint for x_{ij}^p:

$$\sum_{j \in \mathcal{A}_i} x_{ij}^p = 1, \forall i \in M, p \in P, \tag{1}$$

where A_i denotes the set of neighbors of mapper i in the aggregation layer.

We let f_{ij}^p denote the traffic from mapper $i \in M$ to node $j \in A$, which can be calculated by:

$$f_{ij}^p = x_{ij}^p m_i^p, \forall i \in M, j \in A_i, p \in P. \tag{2}$$

The input data of node $j \in A$ can be calculated by summing up all incoming traffic, i.e.,

$$I_j^p = \sum_{i \in M_j} f_{ij}^p, \forall j \in A, p \in P, \tag{3}$$

where M_j denotes the set of j's neighbors in the map layer. The corresponding output data of node $j \in A$ is:

$$O_j^p = \alpha_j I_j^p, \forall j \in A, p \in P, \tag{4}$$

where $\alpha_j = \alpha$ if node j is a potential aggregator. Otherwise, i.e., node j is a shadow node, we have $\alpha_j = 1$.

We further define a binary variable z_j for aggregator placement, i.e.,

$$z_j = \begin{cases} 1, \text{ if a potential aggregator } j \in N \text{ is activated} \\ \quad \text{for data aggregation,} \\ 0, \text{ otherwise.} \end{cases}$$

Since the total number of aggregators is constrained by δ, we have:

$$\sum_{j \in N} z_j \leq \delta. \tag{5}$$

The relationship among x_{ij}^p and z_j can be represented by:

$$x_{ij}^p \leq z_j, \forall j \in N, i \in M_j, p \in P. \tag{6}$$

In other words, if a potential aggregator $j \in N$ is not activated for data aggregation, i.e., $z_j = 0$, no data should be forwarded to it, i.e., $x_{ij}^p = 0$.

Finally, we define a binary variable y_k^p to describe intermediate data partition at reducers, i.e.,

$$y_k^p = \begin{cases} 1, \text{ if data of key } p \in P \text{ are processed by} \\ \quad \text{reducer } k \in R, \\ 0, \text{ otherwise.} \end{cases}$$

Since the intermediate data with the same key will be processed by a single reducer, we have the constraint:

$$\sum_{k \in R} y_k^p = 1, \forall p \in P. \tag{7}$$

The network traffic from node $j \in A$ to reducer $k \in R$ can be calculated by:

$$g_{jk}^p = O_j y_k^p, \forall j \in A, k \in R, p \in P. \tag{8}$$

With the objective to minimize the total cost of network traffic within the MapReduce job, the problem can be formulated as:

$$\min \sum_{p \in P} \Big(\sum_{i \in M} \sum_{j \in A_i} f_{ij}^p d_{ij} + \sum_{j \in A} \sum_{k \in R} g_{jk}^p d_{jk} \Big)$$
$$\text{subject to:} \qquad (1) - (8).$$

Note that the formulation above is a mixed-integer nonlinear programming (MINLP) problem. By applying linearization technique, we transfer it to a mixed-integer linear programming (MILP) that can be solved by existing mathematical tools. Specifically, we replace the nonlinear constraint (8) with the following linear ones:

$$0 \leq g_{jk}^p \leq O_j^p, \forall j \in \mathcal{A}, k \in R, p \in P, \tag{9}$$
$$O_j^p - (1 - y_k^p)\bar{O}_j^p \leq g_{jk}^p \leq \bar{O}_j^p, \forall j \in \mathcal{A}, k \in R, p \in P, \tag{10}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2419671, IEEE Transactions on Parallel and Distributed Systems

where constant $\bar{O}_j^p = \alpha_j \sum_{i \in M_j} m_i^p$ is the upper bound of O_j^p. The MILP formulation after linearization is:

$$\min \sum_{p \in P} \Big(\sum_{i \in M} \sum_{j \in \mathcal{A}_i} f_{ij}^p d_{ij} + \sum_{j \in \mathcal{A}} \sum_{k \in R} g_{jk}^p d_{jk} \Big)$$
$$\text{subject to:} \quad (1)-(7), (9), \text{and } (10).$$

Theorem 1. *Traffic-aware Partition and Aggregation problem is NP-hard.*

Proof: To prove NP-hardness of our network traffic optimization problem, we prove the NP-completeness of its decision version by reducing the set cover problem to it in polynomial time.

The set cover problem: given a set $U = \{x_1, x_2, \ldots, x_n\}$, a collection of m subsets $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$, $S_j \subseteq U$,$1 \leq j \leq m$ and an integer K. The set cover problem seeks for a collection C such that $|C| \leq K$ and $\bigcup_{i \in C} S_i = U$.

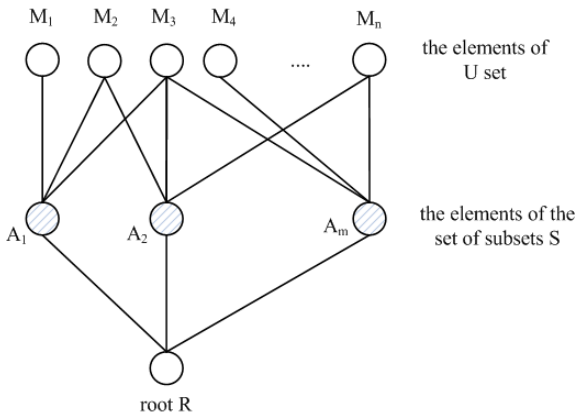


Fig. 4. A graph instance.

For each $x_i \in U$, we create a mapper M_i that generates only one key/value pair. All key/value pairs will be sent to a single reducer whose distance with each mapper is more than 2. For each subset S_j, we create a potential aggregaor A_j with distance 1 to the reducer. If $x_i \in S_j$, we set the distance between M_i to A_j to 1. Otherwise, their distance is greater than 1. The aggregation ratio is defined to be 1. The constructed instance of our problem can be illustrated using Fig. 4. Given K aggregators, we look for a placement such that the total traffic cost is no greater than $2n$. It is easy to see that a solution of the set cover problem generates a solution of our problem with cost $2n$. When we have a solution of our problem with cost $2n$, each mapper should send its result to an aggregator with distance 1 away, which forms a solution of the corresponding set cover problem. □

5 Distributed Algorithm Design

The problem above can be solved by highly efficient approximation algorithms, e.g., branch-and-bound, and fast off-the-shelf solvers, e.g., CPLEX, for moderate-sized input. An additional challenge arises in dealing with the MapReduce job for big data. In such a job, there are hundreds or even thousands of keys, each of which is associated with a set of variables (e.g., x_{ij}^p and y_k^p) and constraints (e.g., (1) and (7)) in our formulation, leading to a large-scale optimization problem that is hardly handled by existing algorithms and solvers in practice.

In this section, we develop a distributed algorithm to solve the problem on multiple machines in a parallel manner. Our basic idea is to decompose the original large-scale problem into several distributively solvable subproblems that are coordinated by a high-level master problem. To achieve this objective, we first introduce an auxiliary variable z_j^p such that our problem can be equivalently formulated as:

$$\min \sum_{p \in P} \Big(\sum_{i \in M} \sum_{j \in A_i} f_{ij}^p d_{ij} + \sum_{j \in A} \sum_{k \in R} g_{jk}^p d_{jk} \Big)$$
$$\text{subject to:} \quad x_{ij}^p \leq z_j^p, \forall j \in N, i \in M_j, p \in P, \quad (11)$$
$$z_j^p = z_j, \forall j \in N, p \in P, \quad (12)$$
$$(1)-(5), (7), (9), \text{and } (10).$$

The corresponding Lagrangian is as follows:

$$\begin{aligned} L(\boldsymbol{\nu}) &= \sum_{p \in P} C^p + \sum_{j \in N} \sum_{p \in P} \nu_j^p (z_j - z_j^p) \\ &= \sum_{p \in P} C^p + \sum_{j \in N} \sum_{p \in P} \nu_j^p z_j - \sum_{j \in N} \sum_{p \in P} \nu_j^p z_j^p \\ &= \sum_{p \in P} \big(C^p - \sum_{j \in N} \nu_j^p z_j^p \big) + \sum_{j \in N} \sum_{p \in P} \nu_j^p z_j \quad (13) \end{aligned}$$

where ν_j^p are Lagrangian multipliers and C^p is given as

$$C^p = \sum_{i \in M} \sum_{j \in A_i} f_{ij}^p d_{ij} + \sum_{j \in A} \sum_{k \in R} g_{jk}^p d_{jk}.$$

Given ν_j^p, the dual decomposition results in two sets of subproblems: intermediate data partition and aggregator placement. The subproblem of data partition for each key $p \in P$ is as follows:

$$\textbf{SUB\_DP:} \quad \min \quad (C^p - \sum_{j \in N} \nu_j^p z_j^p)$$
$$\text{subject to:} (1)-(4), (7), (9), (10), \text{and } (11).$$

These problems regarding different keys can be distributed solved on multiple machines in a parallel manner. The subproblem of aggregator placement can be simply written as:

$$\textbf{SUB\_AP:} \ \min \quad (\sum_{j \in N} \sum_{p \in P} \nu_j^p z_j) \qquad \text{subject to: } (5).$$

The values of ν_j^p are updated in the following master problem:

$$\min L(\boldsymbol{\nu}) = \sum_{p \in P} \hat{C}^p + \sum_{j \in N} \sum_{p \in P} \nu_j^p \hat{z}_j - \sum_{j \in N} \sum_{p \in P} \nu_j^p \hat{z}_j^p$$
$$\text{subject to: } \nu_j^p \geq 0, \forall j \in A, p \in P, \quad (14)$$

where $\hat{C}^p$, $\hat{z}_j^p$ and $\hat{z}_j$ are optimal solutions returned by subproblems. Since the objective function of the master

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2419671, IEEE Transactions on Parallel and Distributed Systems

Algorithm 1 Distributed Algorithm

1: set $t = 1$, and $\nu_j^p (j \in A, p \in P)$ to arbitrary nonnegative values;
2: **for** $t < T$ **do**
3: distributively solve the subproblem **SUB_DP** and **SUB_AP** on multiple machines in a parallel manner;
4: update the values of ν_j^p with the gradient method (15), and send the results to all subproblems;
5: set $t = t + 1$;
6: **end for**

problem is differentiable, it can be solved by the following gradient method.

$$\nu_j^p(t+1) = \left[\nu_j^p + \xi\left(\hat{z}_j(\nu_j^p(t)) - \hat{z}_j^p(\nu_j^p(t))\right)\right]^+, \quad (15)$$

where t is the iteration index, ξ is a positive step size, and '+' denotes the projection onto the nonnegative orthants.

In summary, we have the following distributed algorithm to solve our problem.

5.1 Network Traffic Traces

In this section, we verify that our distributed algorithm can be applied in practice using real trace in a cluster consisting of 5 virtual machines with 1GB memory and 2GHz CPU. Our network topology is based on three-tier architectures: an access tier, an aggregation tier and a core tier (Fig. 6). The access tier is made up of cost-effective Ethernet switches connecting rack VMs. The access switches are connected via Ethernet to a set of aggregation switches which in turn are connected to a layer of core switches. An inter-rack link is the most contentious resource as all the VMs hosted on a rack transfer data across the link to the VMs on other racks. Our VMs are distributed in three different racks, and the map-reduce tasks are scheduled as in Fig. 6. For example, rack 1 consists of node 1 and 2; mapper 1 and 2 are scheduled on node 1 and reducer 1 is scheduled on node 2. The intermediate data forwarding between mappers and reducers should be transferred across the network. The hop distances between mappers and reducers are shown in Fig. 6, e.g., mapper 1 and reducer 2 has a hop distance 6.

We tested the real network traffic cost in Hadoop using the real data source from latest dumps files in wikimedia (http://dumps.wikimedia.org/enwiki/latest/). In the meantime, we executed our distributed algorithm using the same data source for comparison. Since our distributed algorithm is based on a known aggregation ratio α, we have done some experiments to evaluate it in Hadoop environment. Fig. 5 shows the parameter α in terms of different input scale. It turns out to be stable with the increase of input size, and thus we exploit the average aggregation ratio 0.35 for our trace.

To evaluate the experiment performance, we choose the wordcount application in Hadoop. First of all, we

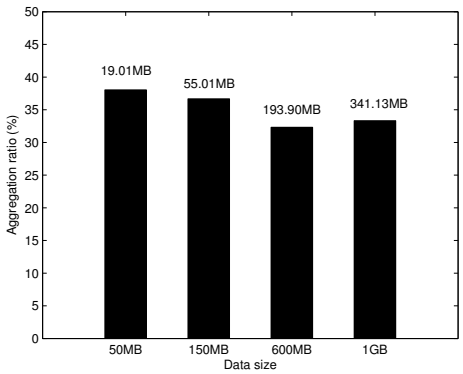


Fig. 5. Ratio evaluation.

tested inputs of $213.44M$, $213.40M$, $213.44M$, $213.41M$ and $213.42M$ for five map tasks to generate corresponding outputs, which turn out to be $174.51M$, $177.92M$, $176.21M$, $177.17M$ and $176.19M$, respectively. Based on these outputs, the optimal solution is to place an aggregator on node 1 and to assign intermediate data according to the traffic-aware partition scheme. Since mappers 1 and 2 are scheduled on node 1, their outputs can be aggregated before forwarding to reducers. We list the size of outputs after aggregation and the final intermediate data distribution between reducers in Table 2. For example, the aggregated data size on node 1 is $139.66M$, in which $81.17M$ data is for reducer 1 and $58.49M$ for reducer 2.

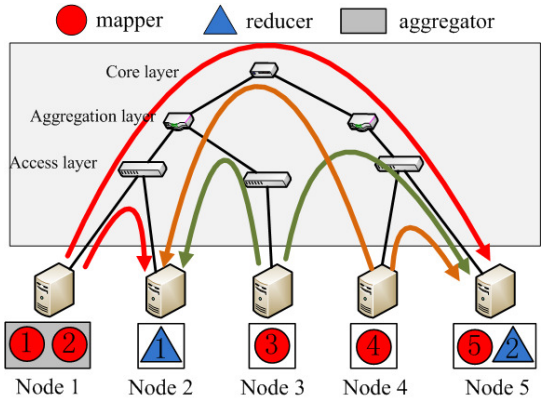


Fig. 6. A small example.

The data size and hop distance for all intermediate data transfer obtained in the optimal solution are shown in Fig. 6 and Table 2. Finally, we get the network traffic cost as follows:

$$81.17 \times 2 + 58.49 \times 6 + 96.17 \times 4 + 80.04 \times 6 + 98.23 \times 6 + 78.94 \times 2 + 94.17 \times 6 + 82.02 \times 0 = 2690.48$$

Since our aggregator is placed on node 1, the outputs of mapper 1 and mapper 2 are merged into $139.66M$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2419671, IEEE Transactions on Parallel and Distributed Systems

TABLE 2
Practical v.s Simulated Cost

mappers / data size	Node 1		Node 2	Node 3	Node 4	Node 5
	mapper 1	mapper 2	—	mapper 3	mapper 4	mapper 5
Before aggregation	174.51M	177.92M	—	176.21M	177.17M	176.19M
After aggregation	139.66M		—	176.21M	177.17M	176.19M
reducer 1	81.17M		—	96.17M	98.23M	94.17M
reducer 2	58.49M		—	80.04M	78.94M	82.02M
Practical cost	2690.48					
Simulated cost	2673.49					

The intermediate data from all mappers is transferred according to the traffic-aware partition scheme. We can get the total network cost 2690.48 in the real Hadoop environment while the simulated network cost is 2673.49. They turn out to be very close to each other, which indicates that our distributed algorithm can be applied in practice.

6 Online Algorithm

Until now, we take the data size m_i^p and data aggregation ratio α_j as input of our algorithms. In order to get their values, we need to wait all mappers to finish before starting reduce tasks, or conduct estimation via profiling on a small set of data. In practice, map and reduce tasks may partially overlap in execution to increase system throughput, and it is difficult to estimate system parameters at a high accuracy for big data applications. These motivate us to design an online algorithm to dynamically adjust data partition and aggregation during the execution of map and reduce tasks.

In this section, we divide the execution of a MapReduce job into several time slots with a length of several minutes or an hour. We let $m_j^p(t)$ and $\alpha_j(t)$ denote the parameters collected at time slot t with no assumption about their distributions. As the job is running, an existing data partition and aggregation scheme may not be optimal anymore under current $m_j^p(t)$ and $\alpha_j(t)$. To reduce traffic cost, we may need to migrate an aggregator from machine j to j' with a migration cost $\Psi_{jj'}$. Meanwhile, the key assignment among reducers is adjusted. When we let reducer k' process the data with key p instead of reducer k that is currently in charge of this key, we use function $\Phi_{kk'}(\sum_{\tau=1}^{t}\sum_{j\in A}\sum_{k\in R} g_{jk}^p(\tau))$ to denote the cost migrating all intermediate data received by reducers so far. The total migration cost can be calculated by:

$$C_M(t) = \sum_{k,k'\in R}\sum_{p\in P} y_k^p(t-1)y_{k'}^p(t)\Phi_{kk'} \cdot \Big(\sum_{\tau=1}^{t}\sum_{j\in A}\sum_{k\in R} g_{jk}^p(\tau)\Big) + \sum_{j,j'\in N} z_j(t-1)z_{j'}(t)\Psi_{jj'}. \quad (16)$$

Our objective is to minimize the overall cost of traffic and migration over a time interval $[1,T]$, i.e.,

$$\min \sum_{t=1}^{T}\Big(C_M(t) + \sum_{p\in P} C^p(t)\Big), \text{subject to:}$$
$$(1)-(7),(9),(10),\text{and } (16), \forall t = 1,...,T.$$

An intuitive method to solve the problem above is to divide it into T one-shot optimization problems:

$$\textbf{OPT\_ONE\_SHOT:} \qquad \min C_M(t) + \sum_{p\in P} C^p(t)$$
$$\text{subject to:} \quad (1)-(7),(9),(10),\text{and } (16), \text{for time slot } t.$$

Unfortunately, the algorithm of solving above one-shot optimization in each time slot based on the information collected in the previous time slot will be far from optimal because it may lead to frequent migration events. Moreover, the coupled objective function due to $C_M(t)$ introduces additional challenges in distributed algorithm design.

In this section, we design an online algorithm whose basic idea is to postpone the migration operation until the cumulative traffic cost exceeds a threshold. As shown in Algorithm 2, we let $\hat{t}$ denote the time of last migration operation, and obtain an initial solution by solving the OPT_ONE_SHOT problem. In each of the following time

Algorithm 2 Online Algorithm

```
t = 1 and t̂ = 1;
solve the OPT_ONE_SHOT problem for t = 1;
while t ≤ T do
    if Σ_{τ=t̂}^{t} Σ_{p∈P} C_t^p(τ) > γ C_M(t̂) then
        solve the following optimization problem:
            min Σ_{p∈P} C^p(t)
        subject to:(1) − (7), (9), and (10), for time slot t.
        if the solution indicates a migration event then
            conduct migration according to the new solution;
            t̂ = t;
            update C_M(t̂);
        end if
    end if
    t = t + 1;
end while
```

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2419671, IEEE Transactions on Parallel and Distributed Systems

slot, we check whether the accumulative traffic cost, i.e., $\sum_{\tau=\hat{t}}^{t}\sum_{p\in P} C_t^p(\tau)$, is greater than γ times of $C_M(\hat{t})$. If it is, we solve an optimization problem with the objective of minimizing traffic cost as shown in line 5. We conduct migration operation according to the optimization results and update $C_M(\hat{t})$ accordingly as shown in lines 6 to 10. Note that the optimization problem in line 5 can be solved using the distributed algorithm developed in last section.

7 Performance Evaluation

In this section, we conduct extensive simulations to evaluate the performance of our proposed distributed algorithm DA. We compare DA with HNA, which is the default method in Hadoop. To our best knowledge, we are the first to propose the aggregator placement algorithm, and compared with the HRA that focuses on a random aggregator placement. All simulation results are averaged over 30 random instances.

- **HNA:** Hash-based partition with No Aggregation. It exploits the traditional hash partitioning for the intermediate data, which are transferred to reducers without going through aggregators. It is the default method in Hadoop.
- **HRA:** Hash-based partition with Random Aggregation. It adds a random aggregator placement algorithm based on the traditional Hadoop. Through randomly placing aggregators in the shuffle phase, it aims to reducing the network traffic cost in the comparison of traditional method in Hadoop.

7.1 Simulation results of offline cases

We first evaluate the performance gap between our proposed distributed algorithm and the optimal solution obtained by solving the MILP formulation. Due to the high computational complexity of the MILP formulation, we consider small-scale problem instances with 10 keys in this set of simulations. Each key associated with random data size within [1-50]. There are 20 mappers, and 2 reducers on a cluster of 20 machines. The parameter α is set to 0.5. The distance between any two machines is randomly chosen within [1-60].

As shown in Fig. 7, the performance of our distributed algorithm is very close to the optimal solution. Although network traffic cost increases as the number of keys grows for all algorithms, the performance enhancement of our proposed algorithms to the other two schemes becomes larger. When the number of keys is set to 10, the default algorithm HNA has a cost of 5.0×10^4 while optimal solution is only 2.7×10^4, with 46% traffic reduction.

We then consider large-scale problem instances, and compare the performance of our distributed algorithm with the other two schemes. We first describe a default simulation setting with a number of parameters, and then study the performance by changing one parameter while fixing others. We consider a MapReduce job with 100 keys and other parameters are the same above.

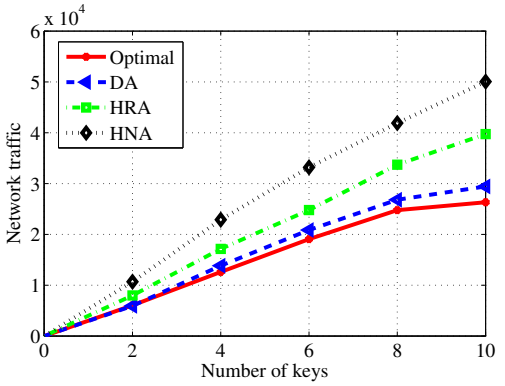


Fig. 7. Network traffic cost versus number of keys from 1 to 10

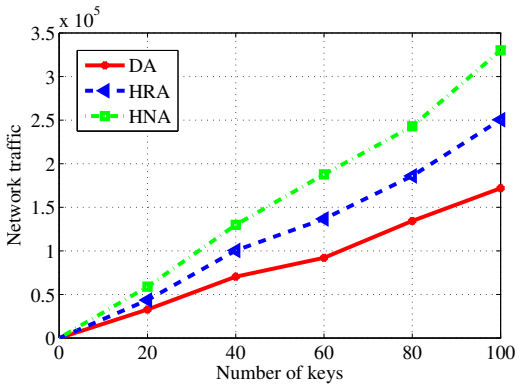


Fig. 8. Network traffic cost versus different number of keys from 1 to 100.

As shown in Fig. 8, the network traffic cost shows as an increasing function of number of keys from 1 to 100 under all algorithms. In particular, when the number of keys is set to 100, the network traffic of the HNA algorithm is about 3.4×10^5, while the traffic cost of our algorithm is only 1.7×10^5, with a reduction of 50%. In contrast to HRA and HNA, the curve of DA increases slowly because most map outputs are aggregated and traffic-aware partition chooses closer reduce tasks for each key/value pair, which are beneficial to network traffic reduction in the shuffle phase.

We then study the performance of three algorithms under different values of α in Fig. 9 by changing its value from 0.2 to 1.0. A small value of α indicates a lower aggregation efficiency for the intermediate data. We observe that network traffic increases as the growth of α under both DA and HRA. In particular, when α is 0.2, DA achieves the lowest traffic cost of 1.1×10^5. On the other hand, network traffic of HNA keeps stable because it does not conduct data aggregation.

The affect of available aggregator number on network traffic is investigated in Fig. 10. We change aggregator number from 0 to 6, and observe that DA always outperforms other two algorithms, and network traffics

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2419671, IEEE Transactions on Parallel and Distributed Systems

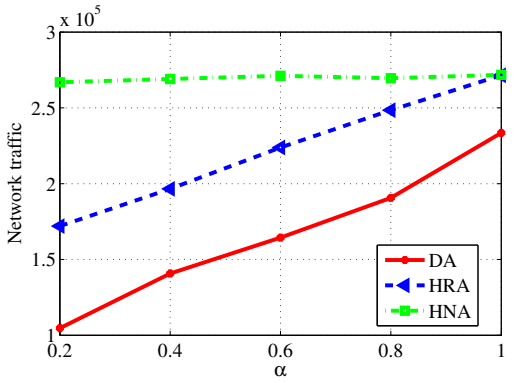


Fig. 9. Network traffic cost versus data reduction ratio α.

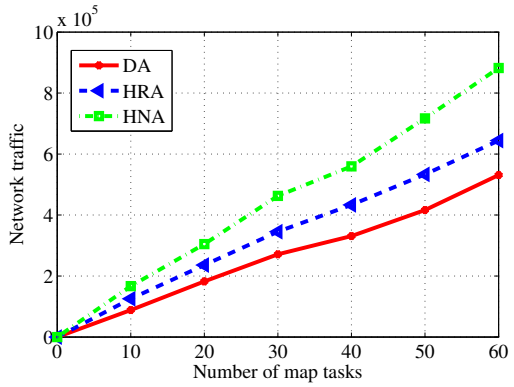


Fig. 11. Network traffic cost versus number of map tasks.

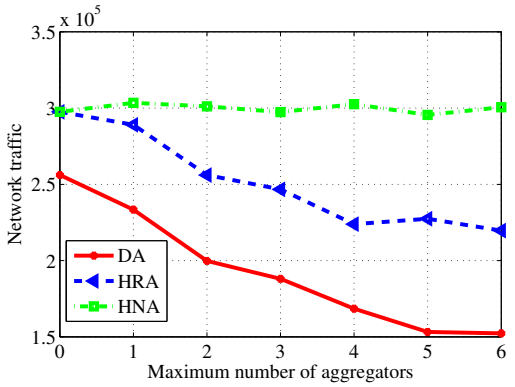


Fig. 10. Network traffic cost versus number of aggregators.

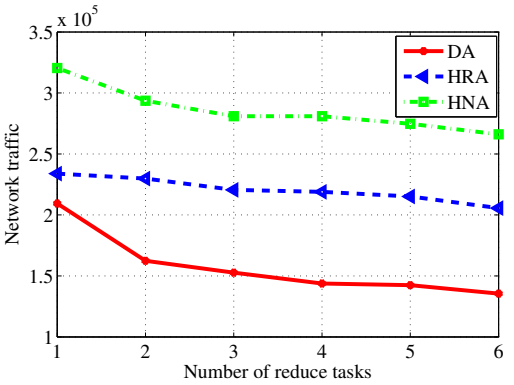


Fig. 12. Network traffic cost versus number of reduce tasks.

decrease under both HRA and DA. Especially, when the number of aggregator is 6, network traffic of the HRA algorithm is 2.2×10^5, while of DA's cost is only 1.5×10^5, with 26.7% improvement. That is because aggregators are beneficial to intermediate data reduction in the shuffle process. Similar with Fig. 9, the performance of HNA shows as a horizontal line because it is not affected by available aggregator number.

We study the influence of different number of map tasks by increasing the mapper number from 0 to 60. As shown in Fig. 11, we observe that DA always achieves the lowest traffic cost as we expected because it jointly optimizes data partition and aggregation. Moreover, as the mapper number increases, network traffic of all algorithms increases.

We shows the network traffic cost under different number of reduce tasks in Fig. 12. The number of reducers is changed from 1 to 6. We observe that the highest network traffic is achieved when there is only one reduce task under all algorithms. That is because all key/value pairs may be delivered to the only reducer that locates far away, leading to a large amount of network traffic due to the many-to-one communication pattern. As the number of reduce tasks increases, the network traffic decreases because more reduce tasks share the load of intermediate data. Especially, DA assigns key/value pairs to the closest reduce task, leading to least network traffic. When the number of reduce tasks is larger than 3, network traffic decreasing becomes slow because the capability of intermediate data sharing among reducers has been fully exploited.

The affect of different number of machines is investigated in Fig. 13 by changing the number of physical nodes from 10 to 60. We observe that network traffic of all the algorithms increases when the number of nodes grows. Furthermore, HRA algorithm performs much worse than other two algorithms under all settings.

7.2 Simulation results of online cases

We then evaluate the performance of proposed algorithm under online cases by comparing it with other two schemes: OHRA and OHNA, which are online extension of HRA and HNA, respectively. The default number of mappers is 20 and the number of reducers is 5. The maximum number of aggregators is set to 4 and we also vary it to examine its impact. The key/value pairs with random data size within [1-100] are generated randomly in different slots. The total number of physical machines is set to 10 and the distance between any two machines is randomly choose within [1-60]. Meanwhile, the default parameter α is set to 0.5. The migration cost $\Phi_{kk'}$ and $\Psi_{jj'}$ are defined as constants 5 and 6. The initial migration cost $C_M(\hat{0})$ is defined as 300 and γ is set to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2419671, IEEE Transactions on Parallel and Distributed Systems

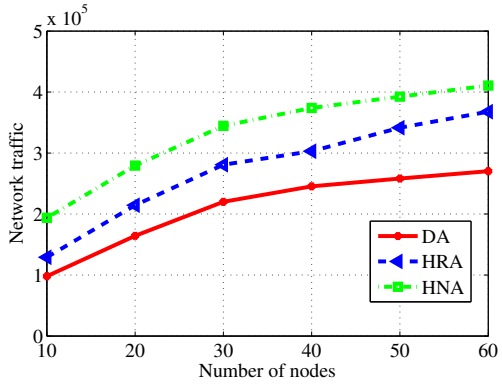


Fig. 13. Network traffic cost versus number of machines.

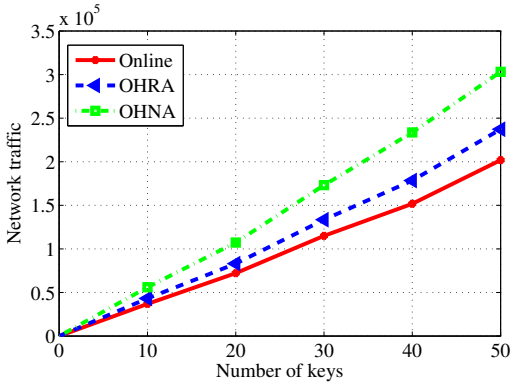


Fig. 15. Network traffic cost versus number of keys

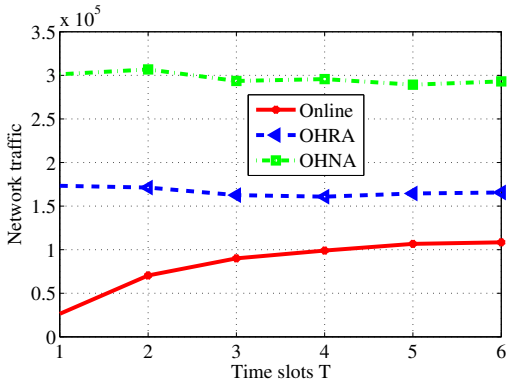


Fig. 14. Network traffic cost versus size of time interval T

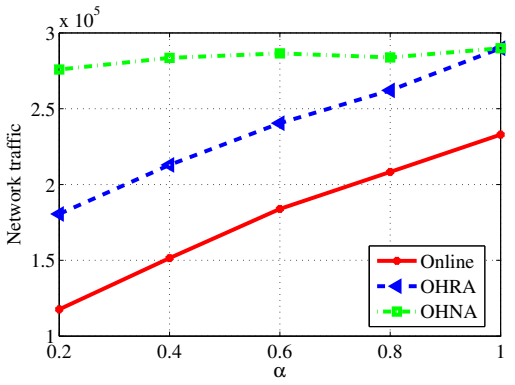


Fig. 16. Network traffic cost versus data reduction ratio α

1000. All simulation results are averaged over 30 random instances.

We first study the performance of all algorithm under default network setting in Fig. 14. We observe that network traffic increases at the beginning and then tends to be stable under our proposed online algorithm. Network traffics of OHRA and OHNA always keep stable because OHNA obeys the same hash partition scheme and no global aggregation for any time slot. OHRA introduces slightly migration cost due to $\Psi_{jj'}$ is just 6 . Our proposed online algorithm always updates migration cost $C_M(\hat{t})$ and executes the distributed algorithm under different time slots, which will incur some migration cost in this process.

The influence of key numbers on network traffic is studied in Fig. 15. We observe that our online algorithm performs much better than other two algorithms. In particular, when the number of keys is 50, the network traffic for online algorithm is about 2×10^5 and the traffic for OHNA is almost 3.1×10^5, with an increasing of 35%.

In Fig. 16, we compare the performance of three algorithms under different values of α. The larger α, the lower aggregation efficiency the intermediate data has. We observe that network traffics increase under our online algorithm and OHRA. However, OHNA is not affected by parameter α because no data aggregation is conducted. When α is 1, all algorithms has similar performance because $\alpha = 1$ means no data aggregation. On the other hand, our online algorithm outperforms OHRA and OHNA under other settings due to the jointly optimization of traffic-aware partition and global aggregation.

We investigate the performance of three algorithms under different number of aggregators in Fig. 17. We observe the online algorithm outperforms other two schemes. When the number of aggregator is 6, the network traffic of the OHNA algorithm is 2.8×10^5 and our online algorithm has a network traffic of 1.7×10^5 , with an improvement of 39%. As the increase of aggregator numbers, it is more beneficial to aggregate intermediate data, reducing the amount of data in the shuffle process. However, when the number of aggregators is set to 0, which means no global aggregation, OHRA has the same network traffic with OHNA and our online algorithm always achieves the lowest cost.

8 Conclusion

In this paper, we study the joint optimization of intermediate data partition and aggregation in MapReduce to minimize network traffic cost for big data applications. We propose a three-layer model for this problem and formulate it as a mixed-integer nonlinear problem, which is then transferred into a linear form that can be solved by mathematical tools. To deal with the large-scale formulation due to big data, we design a distributed algorithm

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2419671, IEEE Transactions on Parallel and Distributed Systems

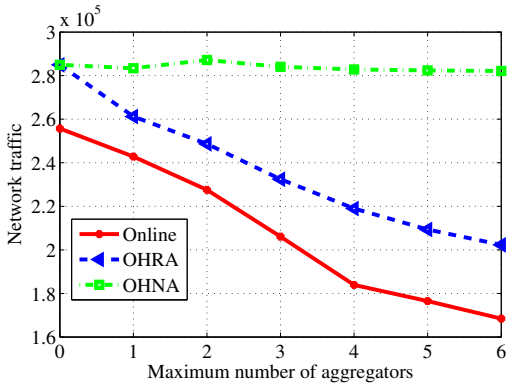


Fig. 17. Network traffic cost versus number of aggregators

to solve the problem on multiple machines. Furthermore, we extend our algorithm to handle the MapReduce job in an online manner when some system parameters are not given. Finally, we conduct extensive simulations to evaluate our proposed algorithm under both offline cases and online cases. The simulation results demonstrate that our proposals can effectively reduce network traffic cost under various network settings.

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] 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 *INFOCOM, 2013 Proceedings IEEE*. IEEE, 2013, pp. 1609–1617.

[3] F. Chen, M. Kodialam, and T. Lakshman, "Joint scheduling of processing and shuffle phases in mapreduce systems," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 1143–1151.

[4] Y. Wang, W. Wang, C. Ma, and D. Meng, "Zput: A speedy data uploading approach for the hadoop distributed file system," in *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1–5.

[5] T. White, *Hadoop: the definitive guide: the definitive guide*. " O'Reilly Media, Inc.", 2009.

[6] S. Chen and S. W. Schlosser, "Map-reduce meets wider varieties of applications," *Intel Research Pittsburgh, Tech. Rep. IRP-TR-08-05*, 2008.

[7] J. Rosen, N. Polyzotis, V. Borkar, Y. Bu, M. J. Carey, M. Weimer, T. Condie, and R. Ramakrishnan, "Iterative mapreduce for large scale machine learning," *arXiv preprint arXiv:1303.3517*, 2013.

[8] S. Venkataraman, E. Bodzsar, I. Roy, A. AuYoung, and R. S. Schreiber, "Presto: distributed machine learning and graph processing with sparse matrices," in *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 2013, pp. 197–210.

[9] A. Matsunaga, M. Tsugawa, and J. Fortes, "Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications," in *eScience, 2008. eScience'08. IEEE Fourth International Conference on*. IEEE, 2008, pp. 222–229.

[10] J. Wang, D. Crawl, I. Altintas, K. Tzoumas, and V. Markl, "Comparison of distributed data-parallelization patterns for big data analysis: A bioinformatics case study," in *Proceedings of the Fourth International Workshop on Data Intensive Computing in the Clouds (DataCloud)*, 2013.

[11] R. Liao, Y. Zhang, J. Guan, and S. Zhou, "Cloudnmf: A mapreduce implementation of nonnegative matrix factorization for large-scale biological datasets," *Genomics, proteomics & bioinformatics*, vol. 12, no. 1, pp. 48–51, 2014.

[12] G. Mackey, S. Sehrish, J. Bent, J. Lopez, S. Habib, and J. Wang, "Introducing map-reduce to high end computing," in *Petascale Data Storage Workshop, 2008. PDSW'08. 3rd*. IEEE, 2008, pp. 1–6.

[13] W. Yu, G. Xu, Z. Chen, and P. Moulema, "A cloud computing based architecture for cyber security situation awareness," in *Communications and Network Security (CNS), 2013 IEEE Conference on*. IEEE, 2013, pp. 488–492.

[14] J. Zhang, H. Zhou, R. Chen, X. Fan, Z. Guo, H. Lin, J. Y. Li, W. Lin, J. Zhou, and L. Zhou, "Optimizing data shuffling in data-parallel computation by understanding user-defined functions," in *Proceedings of the 7th Symposium on Networked Systems Design and Implementation (NSDI), San Jose, CA, USA*, 2012.

[15] F. Ahmad, S. Lee, M. Thottethodi, and T. Vijaykumar, "Mapreduce with communication overlap," pp. 608–620, 2013.

[16] H.-c. Yang, A. Dasdan, R.-L. Hsiao, and D. S. Parker, "Map-reduce-merge: simplified relational data processing on large clusters," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM, 2007, pp. 1029–1040.

[17] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, J. Gerth, J. Talbot, K. Elmeleegy, and R. Sears, "Online aggregation and continuous query support in mapreduce," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 1115–1118.

[18] A. Blanca and S. W. Shin, "Optimizing network usage in mapreduce scheduling."

[19] B. Palanisamy, A. Singh, L. Liu, and B. Jain, "Purlieus: locality-aware resource allocation for mapreduce in a cloud," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2011, p. 58.

[20] S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and L. Qi, "Leen: Locality/fairness-aware key partitioning for mapreduce in the cloud," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. IEEE, 2010, pp. 17–24.

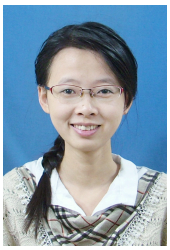
[21] L. Fan, B. Gao, X. Sun, F. Zhang, and Z. Liu, "Improving the load balance of mapreduce operations based on the key distribution of pairs," *arXiv preprint arXiv:1401.0355*, 2014.

[22] S.-C. Hsueh, M.-Y. Lin, and Y.-C. Chiu, "A load-balanced mapreduce algorithm for blocking-based entity-resolution with multiple keys," *Parallel and Distributed Computing 2014*, p. 3, 2014.

[23] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "Mapreduce online." in *NSDI*, vol. 10, no. 4, 2010, p. 20.

[24] J. Lin and C. Dyer, "Data-intensive text processing with mapreduce," *Synthesis Lectures on Human Language Technologies*, vol. 3, no. 1, pp. 1–177, 2010.

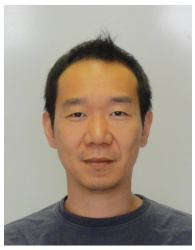
[25] P. Costa, A. Donnelly, A. I. Rowstron, and G. O'Shea, "Camdoop: Exploiting in-network aggregation for big data applications." in *NSDI*, vol. 12, 2012, pp. 3–3.



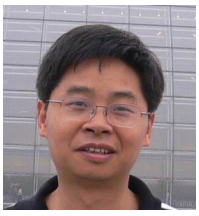
Huan Ke is a graduate student in the Department of Computer Science at University of Aizu. Her research interests include cloud computing, big data, network and RFID system. Huan got her bachelor degree from Huazhong University of Science and Technology.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2015.2419671, IEEE Transactions on Parallel and Distributed Systems

Peng Li received his BS degree from Huazhong University of Science and Technology, China, in 2007, the MS and PhD degrees from the University of Aizu, Japan, in 2009 and 2012, respectively. He is currently an Associate Professor in the University of Aizu, Japan. His research interests include networking modeling, cross-layer optimization, network coding, cooperative communications, cloud computing, smart grid, performance evaluation of wireless and mobile networks for reliable, energy-efficient, and cost-effective communications. He is a member of IEEE.



Song Guo (M'02-SM'11) received the PhD degree in computer science from the University of Ottawa, Canada in 2005. He is currently a Full Professor at School of Computer Science and Engineering, the University of Aizu, Japan. His research interests are mainly in the areas of wireless communication and mobile computing, cyber-physical systems, data center networks, cloud computing and networking, big data, and green computing. He has published over 250 papers in refereed journals and conferences in these areas and received three IEEE/ACM best paper awards. Dr. Guo currently serves as Associate Editor of IEEE Transactions on Parallel and Distributed Systems, Associate Editor of IEEE Transactions on Emerging Topics in Computing for the track of Computational Networks, and on editorial boards of many others. He has also been in organizing and technical committees of numerous international conferences. Dr. Guo is a senior member of the IEEE and the ACM.



Minyi Guo received the BSc and ME degrees in computer science from Nanjing University, China; and the PhD degree in computer science from the University of Tsukuba, Japan. He is currently Zhiyuan Chair professor and head of the Department of Computer Science and Engineering, Shanghai Jiao Tong University (SJTU), China. Before joined SJTU, Dr. Guo had been a professor and department chair of school of computer science and engineering, University of Aizu, Japan. Dr. Guo received the national science fund for distinguished young scholars from NSFC in 2007, and was supported by "1000 recruitment program of China" in 2010. His present research interests include parallel/distributed computing, compiler optimizations, embedded systems, pervasive computing, and cloud computing. He has more than 250 publications in major journals and international conferences in these areas, including the IEEE Transactions on Parallel and Distributed Systems, the IEEE Transactions on Nanobioscience, the IEEE Transactions on Computers, the ACM Transactions on Autonomous and Adaptive Systems, the Journal of Parallel and Distributed Computing, INFOCOM, IPDPS, ICS, ISCA, HPCA, SC, WWW, PODC, etc. He received 5 best paper awards from international conferences. He is on the editorial board of IEEE Transactions on Parallel and Distributed Systems and IEEE Transactions on Computers. Dr. Guo is a senior member of IEEE, member of ACM, IEICE IPSJ, and CCF.