Postprint: Communication Optimization for Large-Scale Clusters in MapReduce Computing

Authors: Cao Yunpeng, Wang Haifeng, Liu Haitao, He Shuqing

Date: 2019-01-28T00:00:00+00:00

Abstract

To optimize communication efficiency and reduce Shuffle data transmission volume in large-scale clusters running MapReduce jobs, this work proposes a three-fold approach: first, adopting a strategy of trading storage locality for communication locality to establish a distributed collaborative data mapping model; second, employing random sampling and machine learning methods to extract locality features of job data, thereby enabling effective deployment of map computation data; and finally, leveraging the global flexible control capability of Software-Defined Networking to preferentially select nodes with favorable communication links and map computational tasks to such nodes. Experimental results demonstrate that for intermediate data shuffle-intensive jobs, the proposed approach achieves favorable optimization effects, reducing communication latency by 4.3%–5.8%. This scheme can reduce Shuffle traffic and data migration latency, and is suitable for various scheduling strategies and network topologies.

Full Text

Communication Optimization for Large-Scale Clusters Aiming at MapReduce Computing

Cao Yunpeng[1,2], Wang Haifeng[1,2]†, Liu Haitao[1,2], He Shuqing[1,2] 1. School of Information Science & Engineering, Linyi University, Linyi Shandong 276002, China
2. Linda Institute, Shandong Provincial Key Laboratory of Network Based Intelligent Computing, Linyi Shandong 276002, China

Abstract

To optimize communication efficiency and reduce Shuffle data transmission in large-scale clusters running MapReduce jobs, we first propose a distributed col-

laborative data mapping model based on a strategy that trades storage locality for communication locality. Next, we extract job locality features through random sampling and machine learning methods to achieve effective deployment of map computation data. Finally, leveraging the global flexible control capabilities of Software-Defined Networking (SDN), we select nodes with high-quality communication links and schedule computing tasks onto these nodes. Experimental results demonstrate that our approach achieves favorable optimization effects for intermediate data shuffle-intensive jobs, reducing communication latency by 4.3%-5.8%. This solution can reduce Shuffle traffic and data migration delay while being suitable for various scheduling strategies and network topologies.

Keywords: data communication optimization; MapReduce; software-defined network; collaborative data mapping

0 Introduction

With the proliferation of big data analytics and real-time computing, large-scale clusters now run numerous MapReduce-style jobs. MapReduce divides computation into Map and Reduce phases, where input data is split into multiple blocks and mapped to various nodes for parallel processing in the Map phase. Intermediate results from Map computations must be relocated through data shuffling. During this Shuffle phase, data with identical key values is aggregated from multiple nodes to a single node for Reduce computation, creating an incast problem where numerous nodes simultaneously transmit data to one node.

Currently, network traffic from data shuffling accounts for 58.6% of total network traffic in data centers [1], making intermediate data communication a critical performance bottleneck in MapReduce computing. Reducing intermediate data transmission latency has become an urgent priority for improving big data computation performance.

Existing research on intermediate data transmission optimization for MapReduce computing falls into four main categories:

a) **Coarse-grained data block balancing mapping.** Hadoop employs simple hash functions for data block mapping, leading to unbalanced distribution. To address this, Ibrahim et al. proposed a fairness-aware key partitioning method that designs data block mapping schemes based on key frequency distribution [2]. Palanisamy designed a resource optimization allocation system to improve MapReduce performance by placing intermediate data at locations with storage locality to reduce data transmission [3]. The iShuffle system separates data shuffling from the Reduce process, remapping data blocks in an extracted Shuffle service to balance intermediate data skew and reduce Shuffle write latency [4].

b) **Fine-grained intermediate data aggregation optimization.** This approach aggregates correlated traffic in-network before transmission to

reduce intermediate data transmission latency [5]. Ke et al. added a data aggregation layer before the Reduce phase, converting intermediate data aggregation into a mixed-integer nonlinear programming problem and designing a distributed data distribution algorithm to optimize intermediate data transmission [6].

c) **Coarse-grained intermediate data placement optimization.** This research places intermediate data in RDMA-based intermediate storage layers to improve read/write efficiency and reduce network traffic [7], or proposes new similarity-based distance models to recalculate network distances between data center nodes, placing Reducer tasks on similar nodes to reduce Shuffle data volume through compute-near-data approaches [8].

d) **Optimization from the data center network architecture perspective.** For fat-tree network structures in data centers, methods have been proposed to optimize core network traffic to solve network congestion caused by Shuffle communication [9]. Software-Defined Networking (SDN), as a maturing technology, has also emerged as an approach to optimize Shuffle traffic by improving network links in computing clusters and optimizing link quality between compute nodes to reduce data transmission delay [11]; designing a bandwidth-aware task scheduling model using SDN' s global information capability to guarantee data locality across the data center network and improve job performance [12]; leveraging SDN' s flexible network control to complete Shuffle intermediate data merging on OpenFlow switches, reducing data traffic and transmission time [13]; and proposing a Shuffle traffic optimization routing algorithm based on SDN that selects efficient forwarding paths according to Shuffle flow characteristics, improving both computing performance and network stability [14].

In summary, while fine-grained data aggregation in the Shuffle phase offers high accuracy, it suffers from high algorithmic complexity. This paper adopts a coarse-grained data block balancing mapping scheme to improve data locality in the Shuffle phase and utilizes SDN' s capability to flexibly grasp global network links to schedule data blocks onto nodes with high-quality communication links in advance. By combining data locality optimization with MapReduce job scheduling algorithms, we reduce intermediate data transmission volume and communication delay. Our approach offers two advantages: since data mapping optimization is performed before actual scheduling, it is suitable for improving any MapReduce scheduling algorithm; and with SDN's flexible global network control, cluster communication links can be optimized without being constrained by network topology.

1 Problem Description

In the MapReduce model, Map tasks distributed across nodes must aggregate intermediate data as input for Reduce tasks—this is the Shuffle phase for inter-

mediate data. During Shuffle, the phenomenon of numerous nodes transmitting data to a single node can easily cause network performance bottlenecks. To improve network performance, we propose a collaborative data mapping model that uses data storage locality to optimize network communication. The collaborative data mapping model is a functional mapping between a computing task' s input dataset and the set of compute nodes, enabling rational and effective partitioning and deployment of big data inputs.

Let the input dataset for computing task T_i be $D_i = \{B_i, i = 1...m\}$, and the cluster be the set of compute nodes $\{N_j, j = 1..., n\}$. Then function f represents the collaborative data mapping model.

As shown in [Figure 1: see original paper], a job' s input data GA exists as a global array storing metadata about data distribution—that is, the distribution of each data block. A Data Block is a physical storage unit directly mapped to a distributed storage system and can be distributed across disks on various nodes. For example, $GA = (B1, B2, B3, B4, B5, B6)$ stored across 4 compute nodes [Node1, Node4] can be represented as $(B1(N1), B2(N1), B3(N2), B4(N3), B5(N4), B6(N4))$, satisfying equation (2). In equation (2), the data mapping for a computing task is mapped to a set of compute nodes with minimal network communication cost. In essence, the problem to be solved is achieving optimal data transmission cost among mapping nodes.

2 Collaborative Data Mapping Model

The collaborative data model addresses the identification of computing data distribution. The data distribution identification scheme is as follows: Let the data for a computing job consist of subsets $\{B1, B2, ..., Bi, Bj, ...Bn\}$, with $Cost$ as the communication cost function (including intra-node communication and inter-node communication, where inter-node communication is further divided into intra-rack and cross-rack communication costs). The constraints for data layout are shown in equation (3).

Under these constraints, we identify the data mapping functional relationship. Based on fundamental function mapping relationships, we adopt an inverse function solving approach. The input to the mapping function is a rational data block layout, and the output is a subset of nodes with high communication efficiency. According to inverse function solving, we first determine the range of the data layout mapping function—that is, identify compute nodes with communication locality, such as the node set $\{N1, N2, ...Nk\}$ located in the same rack $R1$. After determining the function' s dependent variables, we find the corresponding independent variable values $\{B1, B2, ..., Bt\}$. In the MapReduce computing model, computing task T_i is divided into subtasks $T_i = \{Ts1, Ts2, ...Tst\}$, with each subtask concurrently processing a subset of input data $D_i = \{B1, B2, ..., Bt\}$. Equation (4) implements the decision space transformation for the data mapping function:

After decision space transformation, the data mapping problem becomes one of finding compute subtasks with good communication locality. Thus, the decision space for input data is transformed into the feature space of computing tasks, from which we identify decision attributes to partition computing tasks into two categories: those with good communication locality and those with poor communication locality. We then map the node set with better communication locality to nodes with good network links, such as physical nodes within the same rack switch.

3.1 Job Classification Prediction

The main idea of computing job classification prediction is to run MapReduce jobs in heterogeneous clusters, pre-compute job features, record the network communication process of sample jobs, quantify data transmission volume, and establish classification metrics. A classifier is built through machine learning methods, and after training the classification model, online prediction is achieved. Based on this approach, we must address three issues: computing job feature extraction, classification metric quantification, and job classifier selection.

3.1.1 Computing Job Features

Selecting features for MapReduce computing jobs is a prerequisite for establishing a classification model. However, we cannot extract runtime performance parameters for MapReduce jobs [15], such as Map/Reduce execution times or hardware resource utilization rates across cluster nodes. This runtime information can only be obtained after job execution and cannot be applied to static job classification prediction. Therefore, we select information available before job deployment and execution as sample features, specifically:

Before submitting a MapReduce job to the computing platform, users provide two basic pieces of information: a) **Job basic type** T**.** MapReduce jobs fall into three categories: CPU-intensive, I/O and network-intensive, and CPU&I/O hybrid. We use three fuzzy levels—high, medium, and low—to measure these types. For example: CPU-intensive jobs {high=0.1, medium=0.2, low=0.3}, CPU&I/O hybrid jobs {high=0.6, medium=0.5, low=0.4}, and I/O and network-intensive jobs {high=0.9, medium=0.8, low=0.7}. b) **Input data scale** S**.** Input data scale is also quantified using three fuzzy levels—high, medium, and low—primarily based on expert experience to determine data scale ranges. High, medium, and low correspond to 0.9, 0.6, and 0.3, respectively.

Additionally, MapReduce computing jobs are pre-deployed to clusters by schedulers. We extract pre-execution deployment information as classification features. Let the cluster consist of several racks $G = \{R1, R2, ..., Rp\}$, with each rack containing compute nodes $R_i = \{n1, n2, ..., nq\}$. The scheduler partitions Map tasks based on the data blocks they process, as shown in equation (5).

This determinant' s rows represent racks and columns represent nodes within

racks. Here there are 6 racks, each containing 7 nodes. Job J' s Map task distribution forms a task matrix $J = [T_{ij}]_{p\times q}$, where T_{ij} represents the number of Map tasks assigned to node n_{ij}. If a node in the cluster fails or does not exist, T_{ij} is set to Null. For example, $T_{0,4} = 5$ indicates that the fifth node $n_{0,4}$ in rack $R0$ is allocated 5 tasks. All zero elements represent nodes not assigned any tasks, such as $T_{0,1} = 0$. Null nodes indicate non-existent or faulty nodes. Based on the job' s Map task matrix, we introduce two feature metrics:

a) **Map task distribution sparsity γ.** Let the total number of compute nodes in the cluster be n, and the number of nodes partitioned with tasks be m. Then distribution sparsity is calculated as in equation (6). Distributed sparsity 刻画了 Map 任务划分的广度，该值越大说明任务划分越细，涉及的计算节点越多，作业计算的中间数据传输越复杂。

b) **Map task dispersion λ.** Let the number of racks in the cluster be p, and the number of racks occupied by nodes assigned tasks be k. Then dispersion is calculated as in equation (7). Dispersion describes the degree of cross-rack partitioning of Map tasks. A larger value indicates more complex cross-rack data transmission.

3.1.2 Communication Activity

This section uses job communication activity metrics for classification, dividing computing jobs into communication-active and communication-inactive categories. We select cross-rack data transmission during the MapReduce Shuffle phase as observation data. Let job $J_i = \{n1, n2, ..., np\}$, where nodes communicate with each other, making J_i also a communication set. The data communication volume between cross-rack nodes in this communication set is denoted as d_{ij}, where R represents racks. The communication volume during job J_i' s Shuffle phase is D_i:

In equation (9), k is the number of cross-rack communicating node pairs. We define job J_i' s communication activity CA_j as the average cross-rack data communication volume involved in the job:

In equation (10), D_{max} is the maximum communication volume for a single node pair, normalizing CA_j to [0,1]. Next, we must determine the classification threshold. We identify the threshold through experimental analysis of samples. First, we select 50 sample jobs from MapReduce benchmark suites and existing big data analytics jobs, monitor network traffic during the data Shuffle phase, and count cross-rack communication volumes. Finally, we calculate each job' s communication activity using equation (10). The distribution of sample jobs' communication activity is shown in [Figure 2: see original paper].

[Figure 2: see original paper] Sample communication activity distribution

Observing the distribution reveals a clear demarcation between 0.3 and 0.4.

Using confidence interval analysis of sample communication activity, when accepting 30% of jobs as communication-active, the threshold can be set at 0.38.

3.1.3 Classification Predictor

The computing job type prediction model is a classification model that takes the job feature vector $< T, S, \gamma, \lambda >$ as input and predicts communication activity CA as output, dividing jobs into two categories based on the communication activity threshold. We adopt a BP neural network as the classifier—a traditional model using backpropagation to adjust network parameters, offering simplicity and ease of implementation [16]. The BP neural network topology uses a typical three-layer structure: input layer, hidden layer, and output layer, with neuron counts of 4, 6, and 1 respectively. The hidden layer neuron count is set empirically, with the transfer function being $g(x) = \frac{1}{1+e^{-x}}$. The network converges after 562 iterations, with a learning rate of 0.3 and sample size of 100.

3.2 Network Link Optimization Design

This section identifies node subsets with good communication locality in the cluster—nodes with high-quality transmission links between them. By deploying MapReduce tasks onto nodes with good communication locality, we reduce data communication volume during the Shuffle process for multiple computing tasks and improve cluster communication efficiency. In traditional clusters, nodes with good communication locality heavily depend on network topology. For example, in Fat-Tree structures, node subsets within the same Pod have good communication locality, and all nodes connected to the same rack switch have high communication efficiency. However, this topology-dependent approach suffers from severe drawbacks: (a) poor scalability—adding or removing nodes or node failures affect communication optimization schemes and require timely awareness of network topology changes; (b) lack of dynamic deployment capability—when user job loads change, the inability to grasp global network information in a timely manner prevents dynamic adjustment of network topology.

To address these issues in traditional cluster network structures, we adopt Software-Defined Networking (SDN). SDN is an emerging network architecture whose core idea is separating data forwarding and decision control functions in network hardware devices, implementing hardware-based data forwarding and software-based logical forwarding decisions [17]. SDN primarily consists of three layers: application layer, control layer, and data layer, as shown in [Figure 3: see original paper].

[Figure 3: see original paper] SDN architecture schematic diagram

The control layer comprises software-based SDN controllers that maintain a global network view and provide centralized control functions. The data layer includes physical or virtual switches and other data forwarding units that implement network data forwarding based on rules issued by the control layer. The

application layer consists of various network service applications, with MapReduce computing also running as an application supported by the SDN architecture.

We now detail how to obtain node sets with good communication locality in SDN-architected cluster networks. Our goal is to find a compute node subset $\{N1, N2, ...Nm\}$ whose network links $\{e1, e2, ...ek\}$ have optimal communication efficiency. Thus, the problem of finding the optimal communication locality node set is transformed into finding the optimal link subset in the cluster network. In SDN networks, we obtain the optimal link subset by measuring link available bandwidth and ranking. Let the traffic on link e_i during time interval t be l_i, and the total transmission capacity of e_i be c_i. Then the available bandwidth σ_i is:

This optimal link subset solving functional module is placed within the SDN controller. Since the SDN controller grasps global network topology and traffic information from a holistic perspective, it manages various OpenFlow switches (Open vSwitch, OVS) in a distributed manner. The SDN controller sends switch status query commands via the OpenFlow protocol. After the OVS switch' s Local port receives the SDN control message OFQueueStatsRequest, it generates a response message OFQueueStatsReply containing a tx_byte field that counts transmitted bytes. The query message time granularity is nanosecond-level. By initiating two requests within consecutive time intervals, we can 统计出该条链路的可用带宽，这里设置的间隔是 1s. After traversing the link set in the cluster network, we can obtain the optimal link subset and the corresponding compute node set.

3.3 Communication Energy Efficiency Optimization Deployment Algorithm

For the massive data communication process during the Shuffle phase of MapReduce computing, we reschedule task distribution based on job type prediction results. Based on the SDN architecture for cluster interconnection, the SDN controller obtains network topology and link information in real-time. Communication-active jobs are redeployed to nodes in the optimal link set, while communication-inactive jobs maintain their original deployment scheme. The algorithm's core idea is to control communication locality between tasks as much as possible, reducing performance loss from data transmission over poor-quality links in the cluster and thereby improving intermediate data communication efficiency. The specific algorithm is as follows:

Algorithm 1: Communication Optimization Deployment Algorithm

Input: Big data job queue $Q(J1, J2, ..., Jn)$, pre-deployment information for computing jobs, current SDN link set E_{cur}

Output: Deployment node set for job J_i' s Map tasks—the distribution of a job' s Map tasks across the cluster.

1. Initialize current link set E_{cur}

2. While queue(J) is not null
3. J_i = Deque(J)
4. S_i = Scheduling(J_i)
5. J_i.type = Prediction(S_i)
6. Switch(J_i.type)
7. Case $C0$: // Communication-inactive job
8. `Maintain original deployment scheme`
9. End case
10. Case $C1$: // Communication-active job
11. `$E_i$ = Sort($E_{cur}$) // Sort by available bandwidth`
12. `$E_{cur}$ = $E_{cur} - E_i$`
13. `Update($J_i.S$) // Update job deployment node set`
14. `Wait for job completion`
15. `$E_{cur}$ = $E_{cur} + E_i$ // Recycle SDN link resources`
16. End case
17. End while

Algorithm 1' s input includes the job queue, the scheduler' s pre-scheduling information for each job, and the current SDN link set. The algorithm outputs the deployment node set for a job' s Map tasks—the distribution of a job' s Map tasks throughout the cluster. The algorithm extracts the pre-deployment scheme generated by the original scheduling algorithm for each job in the queue (line 4), then uses a neural network prediction model to determine the job type (line 5). Based on the predicted job type (line 6), if it is a communication-inactive job, the original deployment scheme is maintained (lines 7-9). If it is a communication-active job, the SDN controller detects current link available bandwidth, sorts the current link set by available bandwidth, and extracts the optimal link subset (line 11), then updates the job' s deployment node set (line 13). When the job completes execution, SDN link resources are recycled (lines 14-15).

4 Simulation Experiments and Analysis

We established two different experimental testbed environments using a 32-node cluster. The first environment simulates a data center tree topology using three-layer switches, divided into 4 racks with 8 compute nodes per rack. The second environment deploys a Software-Defined Network using a fat-tree topology containing 50 switches and 32 compute nodes, with the controller and switches selected as Ryu and OpenSwitch respectively. Each compute node is configured

with an Intel Xeon E5620 2.4GHz dual-core CPU, 16GB DDR RAM, and a 2TB SATA hard drive. The software system uses Ubuntu 15.0, JDK 1.8, and Hadoop 1.2.1.

We selected Sort, WordCount, TeraSort, Bayesian Classification, and K-means Cluster from the HiBench benchmark suite as test jobs. Table 1 lists the data scales and characteristics of the experimental jobs [18].

** Test job characteristics**

Abbreviation	Map Input	Intermediate Data	Reduce Output
Sort	120 GB	120 GB	120 GB
WordCount	4.1 GB	4.6 KB	3.3 GB
TeraSort	43 GB	43 GB	43 GB
Bayesian Classification	330KB	4.6KB	330KB
K-means Cluster	66 GB	66 GB	0.5 GB

For convenience, we denote our Shuffle communication optimization deployment scheme as SCOD (Shuffle Communication Optimization Deployment), while the unoptimized MapReduce scheme using Hadoop' s default scheduling method is denoted as MR (MapReduce).

4.1 Communication Performance Comparison and Analysis

Since MapReduce job completion time consists of computation time and intermediate data communication time, and to reduce experimental complexity, we ignore scheduling-induced computation deviations and assume computation time is identical for jobs of the same data scale. Therefore, we use job completion time as a proxy for intermediate data communication time. For jobs of identical data scale running on different testbeds, longer execution time indicates longer intermediate data communication time, as shown in [Figure 4: see original paper].

[Figure 4: see original paper] Performance comparison between communication optimization schemes and benchmark methods

Benchmark jobs st, ts, and bc show approximately 4.7%–6.2% performance improvement, while wc and kc show no significant optimization. As seen in Table 1, the three jobs with large intermediate data volumes (st, ts, bc) achieve better communication optimization effects, while jobs with smaller intermediate shuffle data volumes (wc, kc) show insignificant optimization effects. Therefore, the communication optimization scheme works well for shuffle-intensive jobs.

To further verify communication performance optimization, we analyze the shuffle-intensive TeraSort job in greater detail. After the Map phase ends, the intermediate data Shuffle process occurs until the Reduce phase completes. Technically, we can monitor the end time of the Map phase and the job completion

time. As shown in [Figure 5: see original paper], with job data scales of 32GB, 64GB, 128GB, 512GB, and 1TB, the vertical axis represents the proportion of data shuffling and Reduce phase time to total job computation time. A relative decrease in this proportion indicates communication process optimization.

[Figure 5: see original paper] TeraSort communication optimization comparison of different data scales

As data scale increases, the proportion of SCOD' s data shuffling and Reduce phase to total computation time gradually decreases, indicating that communication optimization effects become more pronounced with larger data scales. This demonstrates significant performance improvement for big data-intensive computing.

4.2 System Scalability Analysis

As compute nodes increase, the intermediate data communication process becomes more complex, making adaptability to cluster expansion a critical concern. This section examines the impact of system scalability on communication optimization. To increase compute node count, we launch 32, 64, and 128 virtual machine nodes on the SDN fat-tree structure to execute TeraSort jobs, analyzing communication optimization effects by examining the proportion of data shuffling time to total runtime. As shown in [Figure 6: see original paper], as compute node count increases, the proportion of data shuffling gradually decreases, and this proportion also decreases as data scale increases.

[Figure 6: see original paper] TeraSort communication optimization comparison of different node scales

Experiments show that optimization effects become more significant as cluster nodes and data scale increase. When cluster and data scales are small, background network traffic masks the optimization effects of intermediate data communication, resulting in less noticeable improvements.

4.3 Job Execution Time Comparison

To enhance experimental validation, we selected four additional jobs from PUMA [19] and HiBench benchmark suites: self-join, inverted-index, pagerank, and term-vector. These jobs have intermediate data volumes between 32–300 GB, making them shuffle-intensive jobs. Other jobs like histogram-movies, histogram-ratings, and grep were not evaluated due to minimal data exchange. To simplify experimental operations, we directly compare job execution times to analyze communication optimization effects, normalizing execution times to Hadoop' s scheduling model as a baseline (value of 1), as shown on the y-axis in [Figure 7: see original paper].

[Figure 7: see original paper] Contrastive analysis of job execution time

SCOD reduces overall job execution time compared to the MR baseline scheduling model. Since Map and Reduce computation times are essentially identical for jobs with the same data scale, the reduced execution time corresponds to intermediate data shuffling communication time, further validating our optimization scheme. Although inverted-index and term-vector are shuffle-intensive jobs, their relatively small intermediate data volumes result in less pronounced communication optimization effects. In contrast, self-join and pagerank generate larger intermediate data scales, demonstrating significantly better communication optimization.

5 Conclusion

For shuffle-intensive jobs in MapReduce computing, we designed a scheduling scheme to optimize intermediate data communication. Using a BP neural network to build a job prediction model, we identify data communication-intensive jobs and deploy their tasks onto compute nodes with better network links, implementing an optimization strategy that trades storage locality for communication locality. To accommodate various network topologies, we employ SDN technology to identify nodes with optimal links, enabling flexible adaptation to diverse network structure expansion needs in data centers and compatibility with various MapReduce scheduling algorithms. Experiments demonstrate effective control of locality during the intermediate data shuffling phase, reducing intermediate data transmission delay and overall job execution time with favorable communication optimization effects. However, in real-time big data computing, jobs are submitted dynamically and network links exhibit temporal variability. Future work will deeply integrate SDN link planning with dynamic job scheduling, establishing an intermediate data communication optimization model suitable for complex multi-tenant application scenarios to further enhance practical value.

References

[1] Zhang Jiaxing, Zhou Hucheng, et al. Optimizing data shuffling in data-parallel computation by understanding user-defined functions [C]// Proc of the 9th USENIX Conference on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2012: 295-308.

[2] Shadi I, Jin Hai, Lu Lu, et al. Leen: locality/fairness-aware key partitioning for mapreduce in the cloud [C]// Proc of the 2nd IEEE International Conference on Cloud Computing Technology & Science. Piscataway, NJ: IEEE Press, 2010: 17-24.

[3] Palanisamy B, Singh A, Liu Ling, et al. Purlieus: locality-aware resource allocation for mapreduce in a cloud [C]//Proc of International Conference for High Performance Computing, Networking, Storage and Analysis. New York: ACM Press, 2011: 58.

[4] Guo Yanfei, Rao Jia, et al. iShuffle: improving hadoop performance with shuffle-on-write [J]. IEEE Trans on Parallel and Distributed System, 2017, 28 (6): 1649-1661.

[5] Lu Feifei, Guo Deke, Fang Xing, et al. Efficient data aggregation transfers in data center networks [J]. Chinese Journal of Computers, 2016, 39 (9): 1750-1762.

[6] Ke Huan, Li Peng, et al. On traffic-aware partition and aggregation in mapreduce for big data applications [J]. IEEE Trans on Parallel and Distributed System, 2016, 27 (3): 818-828.

[7] Rahman M W, Islam N S, et al. A comprehensive study of MapReduce over lustre for intermediate data placement and shuffle strategies on HPC clusters [J]. IEEE Trans on Parallel and Distributed System, 2017, 28 (3): 633-646.

[8] Wang Jihe, Wang Danghui, et al. Similarity-based node distance and locality-aware shuffle optimization for Hadoop MapReduce [C]// Proc of IEEE International Conference on Smart Cloud. Piscataway, NJ: IEEE, 2017: 103-108.

[9] Ye Yu, Qian Chen. Space Shuffle: a scalable, flexible, and high-performance data center network [J]. IEEE Trans on Parallel and Distributed System, 2016, 27 (11): 3351-3365.

[10] Wang Jihe, Wang Danghui, et al. A locality-aware shuffle optimization on fat-tree data centers [J]. Future Generation Computer Systems, 2018, 89: 31-43.

[11] Narayan S, Bailey S, Daga A. Hadoop acceleration in an openflow-based cluster [C]// Proc. of SC Companion: High Performance Computing, Networking Storage and Analysis. Piscataway, NJ: IEEE Press, 2012: 535-538.

[12] Qin Peng, Dai Bin, et al. Bandwidth-aware scheduling with SDN in Hadoop: a new trend for big data [J]. IEEE Systems Journal, 2017, 11 (4): 2337-2344.

[13] Yang Jun, Lyu Lu, Xu Guan, et al. Optimization design of MapReduce based on SDN [J]. Journal of Application Research of Computers, 2016, 33 (10): 3109-3113.

[14] Khaleel A, Al-Raweshidy H. Optimization of computing and networking resources of a Hadoop cluster based on software defined network [J]. IEEE Access, 2018, 6: 61351-61365.

[15] Zhang Fan, Sakr M, et al. Empirical discovery of power-law distribution in mapreduce scalability [J]. IEEE Trans on Cloud Computing, 2017, PP (99): 1.

[16] Zhang Fan, Cao Xibin, Zou Jingxiang. Cost estimating for small satellite using fuzzy BP neural networks [J]. Systems Engineering and Electronics, 2000, 22 (10): 75-78.

[17] Qi Chao, Wu Jiangxing, et al. An intensive security architecture with multi-controller for SDN [C]// Proc of IEEE Conference on Computer Communications. Piscataway, NJ: IEEE, 2016: 401-402.

[18] Huang Shengsheng, Huang Jie, et al. The Hibench benchmark suite: characterization of the mapreduce-based data analysis [C]// Proc of the International Conference on Data Engineering. Piscataway, NJ: IEEE, 2010, 74: 41-51.

[19] PUMA. Purdue MapReduce benchmark suite [EB/OL]. [2012]. http://web.ics.purdue.edu/fahmad/benchm

Note: Figure translations are in progress. See original paper for figures.