Network-Aware Locality Scheduling for Distributed Data Operators in Data Centers

Long Cheng, *Member, IEEE,* Ying Wang, *Member, IEEE,* Qingzhi Liu, Dick H.J. Epema, Cheng Liu, Ying Mao, *Member, IEEE,* and John Murphy, *Senior Member, IEEE*

Abstract—Large data centers are currently the mainstream infrastructures for big data processing. As one of the most fundamental tasks in these environments, the efficient execution of distributed data operators (e.g., join and aggregation) are still challenging current data systems, and one of the key performance issues is network communication time. State-of-the-art methods trying to improve that problem focus on either application-layer data locality optimization to reduce network traffic or on network-layer data flow optimization to increase bandwidth utilization. However, the techniques in the two layers are totally independent from each other, and performance gains from a joint optimization perspective have not yet been explored. In this paper, we propose a novel approach called NEAL (NEtwork-Aware Locality scheduling) to bridge this gap, and consequently to further reduce communication time for distributed big data operators. We present the detailed design and implementation of NEAL, and our experimental results demonstrate that NEAL always performs better than current approaches for different workloads and network bandwidth configurations.

Index Terms—data locality; coflow scheduling; distributed operators; data centers; big data; SDN; metaheuristic

1 Introduction

With the continuous growth of data volumes in various domains, large computing systems such as data centers have been built across the globe to store and process massive datasets. As the core operations in these systems, distributed data operators such as join and aggregation are widely used for big data computing. For example, analytical queries are always composed of a set of joins, and a log often needs to be joined with reference data such as information about users as a part of the large-scale log analysis [1].

Up to now, the efficient execution of distributed data operators are still challenging current systems. One of the main reasons is that the typical execution of a distributed data operator contains a data redistribution process. This process always involves transferring large amounts of data over networks [2], which can consume tremendous network resources and results in long communication time. Actually, in recent years, as the performance of CPUs has grown much faster than network bandwidth, the network has become a performance bottleneck to computation, even in a single data center [3]. Moreover, some recent works have shown that distributed data operators in expensive analytical queries can spend more than half of their completion time on data transfers [4]. In such conditions, effective optimization of the execution of the operators, which can minimize network communication time, becomes increasingly desirable.

Reducing the volume of data transferred over networks is an efficient way to speed up distributed data operators [5]. The main reason is that, for both low-end and high-end platforms, communication reduction would directly result in a faster execution [2], [4]. From that basis, various efficient approaches have been proposed in the data management domain [6], [7]. Their general idea is to move small data chunks rather than large ones in an execution using data locality scheduling. For instance, track-join [5] has adopted a fine-grained scheduling, which can search all possible opportunities for reducing network traffic in a join execution. Although all the approaches are shown to be efficient, an application-layer optimization of network traffic does not necessarily lead to optimal communication time. This is because when computing nodes use the network without any coordination, the utilization of network bandwidth can be very poor [8].

To improve network communication for big data applications, various network-aware scheduling frameworks have been proposed for data center environments [9], [10]. However, all the approaches mainly work on reducing or balancing the utilization of network resources of a system. Therefore, they actually cannot characterize the possible optimal communication time for an application. On the other hand, with the focus on improving communication time, data flow scheduling over the abstraction *coflow* [11], is being studied in the domain of data communications. Rather than individual flows, coflow scheduling focuses on minimizing the completion time of the slowest flow

- *L. Cheng is with the School of Control and Computer Engineering, North China Electric Power University, Beijing 102206, China, and also the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China. E-mail: lcheng@ncepu.edu.cn*
- *Y. Wang and C. Liu are with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China. E-mail: {wangying2009, liucheng}@ict.ac.cn*
- *Q. Liu is with the Information Technology Group, Wageningen University & Research, Netherlands. E-mail: qingzhi.liu@wur.nl*
- *D. Epema is with the Distributed Systems Group, Delft University of Technology, Netherlands. E-mail: d.h.j.epema@tudelft.nl*
- *Y. Mao is with the Department of Computer and Information Science at Fordham University in the New York City. E-mail: ymao41@fordham.edu*
- *J. Murphy is with the School of Computer Science, University College Dublin, Ireland. E-mail: j.murphy@ucd.ie*

Manuscript received May 24, 2019; revised December 17, 2020.
(Corresponding author: Ying Wang).

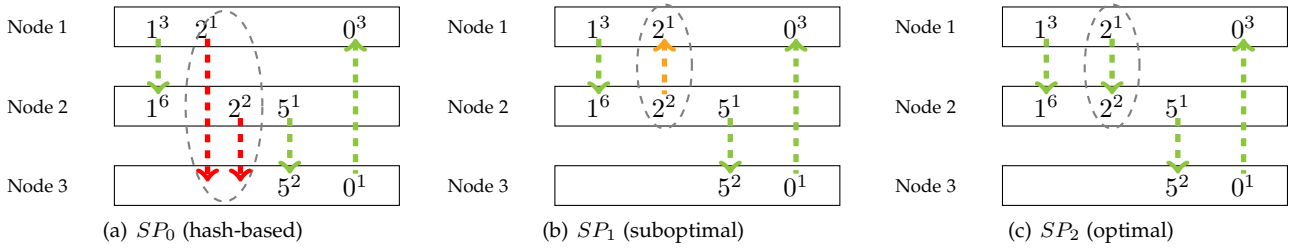


Fig. 1. Difference data locality scheduling plans (SP) for an aggregation over three nodes.

in a job. Since the data redistribution process of a distributed data operator can be modeled as a coflow, i.e., an *n-to-n* communication pattern, we can directly apply the relevant coflow scheduling techniques to accelerate its executions [11], [12]. However, almost all the current coflow scheduling approaches focus on optimizing communication time under the condition that the source and destination nodes of each data flow have been given. Namely, their designs are decoupled from application-layer locality scheduling. In such scenarios, the communication time of a distributed operator with a coflow scheduling will be still suboptimal (we demonstrate this problem with examples in Section 2).

In this work, we present a novel scheduling approach called NEAL (**NE**twork-**A**ware **L**ocality scheduling), which aims to reduce the communication time of distributed data operators such as join and aggregation as much as possible in data center environments. Generally, NEAL is a cross-layer scheduler which can seamlessly combine the application-layer locality scheduling and network-layer flow scheduling to optimize data flow parallelism. Moreover, different from existing network-aware schedulers, the optimization process in NEAL is driven directly by the possible communication time rather than the status of a network, such as network resource utilization. We provide the detailed design and implementation of NEAL and conduct extensive performance evaluation with a large number of emulations and simulations.

The main contributions of this paper are summarized as follows:

- We demonstrate that a distributed operator can achieve additional performance gains on communication time by considering coflow scheduling in its data locality scheduling process.
- We propose NEAL, an approach which seamlessly combines data locality scheduling and data flow scheduling for distributed operator execution. We introduce the performance model of NEAL and also present its system design with a detailed implementation based on a metaheuristic.
- We compare NEAL with the current approaches and experimentally show that NEAL can always achieve better network communication time for distributed data operators in the presence of different workloads and network bandwidth configurations.

The remainder of this paper is organized as follows. In Section 2, we introduce the background with a motivating example of this work. We present the design of NEAL in Section 3 and its detailed implementation in Section 4. We carry out extensive evaluation of our approach in Section 5. We report the related work in Section 6 and conclude this paper in Section 7.

2 Background and Motivation

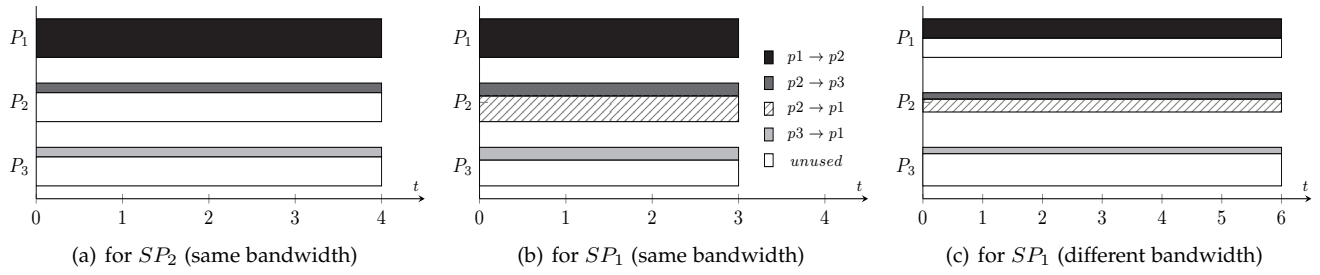
Distributed data operators, such as joins and aggregations, can be broadly decomposed into an initial data redistribution stage followed by a local computing process [6]. The latter process does not contain any inter-machine communication, for the purpose of this work, we will only study the former phase. Without loss of generality, we assume the input data is in the form of key-value pairs, and we will only focus on handling the keys with regarding to locality scheduling.

In the following, we first briefly introduce the data locality and coflow scheduling techniques. Then, we demonstrate the advantage of combining data locality and network communication through a motivating example.

2.1 Data Locality Scheduling

Data locality scheduling is mainly used to generate a plan including the node destinations of data tuples to be transferred over a network. To show the details of relevant techniques, an example of three possible scheduling plans for an aggregation (a general case without the consideration of any functions like sum, count, etc.) over a three-node system is demonstrated in Fig. 1. There, each tuple is represented by its key, and the superscript of each key means its appearing frequency. For instance, 1^3 means that there are three tuples with the key 1. Also, the dashed arrows mean the scheduled results, i.e., the destination node of each tuple.

Fig. 1(a) shows the details of the most commonly used hash-based method. There, a very simple hash function is used to assign the destination for each tuple, i.e., the hash value of a key is a modulus of the value of key and the number of nodes. Fig. 1(b) and Fig. 1(c) demonstrate two other scheduling plans for the agggregation operator respectively, and their only difference is on how to assign the desitination for the key 2. If we quantify the network traffic by the number of tuples moved to remote nodes, then the cost of three scheduling methods is 8, 7 and 6 respectively. From an application angle (e.g., [5], [7]), the schedule plan SP_2 will be considered as the optimal solution and chosen by underlying systems for the final execution, because it transfers less data than other two approaches.



(a) for SP_2 (same bandwidth) (b) for SP_1 (same bandwidth) (c) for SP_1 (different bandwidth)

Fig. 2. Optimal coflow scheduling for SP_1 and SP_2 with different network configurations. The vertical height of filled bars indicates the bandwidth.

2.2 Coflow Scheduling

The coflow abstraction is used to define a group of parallel data flows that are related to each other and also share a common performance goal (e.g., shuffle flows in MapReduce) [13]. To optimize the communication time for big data applications, we need to optimize their data communication at a coflow rather than an individual level. This is because the network communication time depends on the coflow completion time (CCT), instead of the time to complete an individual flow in the coflow [12].

An individual flow within a coflow can be defined by a tuple in the form of $[src, des, v]$, where src and des are the source and destination nodes, and $v > 0$ is the flow volume [14]. In fact, coflows have been shown to be able to express most communication patterns in data parallel applications including distributed data operators [13]. For example, the data flows generated by the plan SP_2 in Fig. 1(c) can be seen as a coflow with three individual data flows (the keys 1 and 2 moving from Node 1 to Node 2 will generate a single data flow in real executions because their source and destination are the same). In this case, we will be able to use coflow scheduling to improve the communication time for the parallel data flows generated in a distributed data operator. As demonstrated in Fig. 2(a), the optimal network communication time of SP_2 will be 4 using an optimal coflow scheduling.

2.3 Motivation of NEAL

To optimize communication time for a distributed data operator, we can first use data locality scheduling to reduce network traffic and then use coflow scheduling to optimize network communications. However, the communication time of a distributed data operator would be suboptimal within such a scheme. The main reasons are: (1) the data locality scheduling focuses on an application-layer optimization, which is actually not communication-time aware; and (2) the coflow scheduling assumes that the detailed information (i.e., $[src, des, v]$) of each data flow is known before a coflow starts [11], [12], [14]. Namely, the relevant network-layer optimization has not realized that the locality scheduling could actually impact the final communication time.

To illustrate the above problem, here we give an example on applying coflow scheduling to the plans SP_1 and SP_2 depicted in Fig. 1 respectively. We assume that each node (i.e., network port P_i) transfers one data tuple in one time unit, and we use a bandwidth-based model [12] to describe the coflow scheduling. Namely, bandwidth is assigned in a way that all data flows are ended at the same time point.

As shown in Fig. 2(a) and Fig. 2(b), using an optimal coflow scheduling, a sub-optimal application-layer plan SP_1 can even lead to better performance on network communications, i.e., the CCT of SP_1 is 3, which is smaller than the 4 achieved by SP_2. This means that the underlying data system should choose SP_1 as the data locality scheduling plan rather than the optimal plan SP_2. In fact, this decision could change again when the available network bandwidth between nodes is not the same, such as that part of bandwidth is consumed by the other services. As shown in Fig. 2(c), if the port P_2 can only transfer a half data tuple in one time unit for the aggregation, then the optimal communication time for SP_1 will be 6 while SP_2 will be still 4. Namely, we should choose SP_2 rather than SP_1 for the execution. In such scenarios, our question is: to achieve the best possible communication time for a distributed data operator, how should we perform the data locality scheduling? Actually, our examples have implied that, to minimize the CCT cost, data locality scheduling at application-layer should be jointly considered with the network-layer data flow scheduling, which motivates our design as below.

3 The NEAL Approach

In this section, we introduce the NEAL approach by leveraging the emerging coflow scheduling. Specifically, we present its system architecture design in an SDN (software-defined networking) data center environment [15].

In addition to that SDN becomes increasingly popular in data centers in recent years [16], we have two other main motivations for the choice of SDN in our design: (1) The coflow scheduling in NEAL can be easily realized in terms of system development. Namely, we can manually assign the bandwidth for each link with SDN, and we do not need to develop a complex system like Varys [11], in which the Varys master and daemons are required for coflow scheduling; and (2) The available network resources of a system like bandwidth are visible with SDN, which allows NEAL to work in a network with different bandwidth configurations.

3.1 Architecture

A system view of the proposed NEAL in an SDN data center is demonstrated in Fig. 3. The main components include a NEAL scheduler which contains a NEAL analyzer and a scheduling solver, a cluster with a large amount of computing nodes, and an SDN network which consists of an OpenFlow controller and some OpenFlow switches [15].

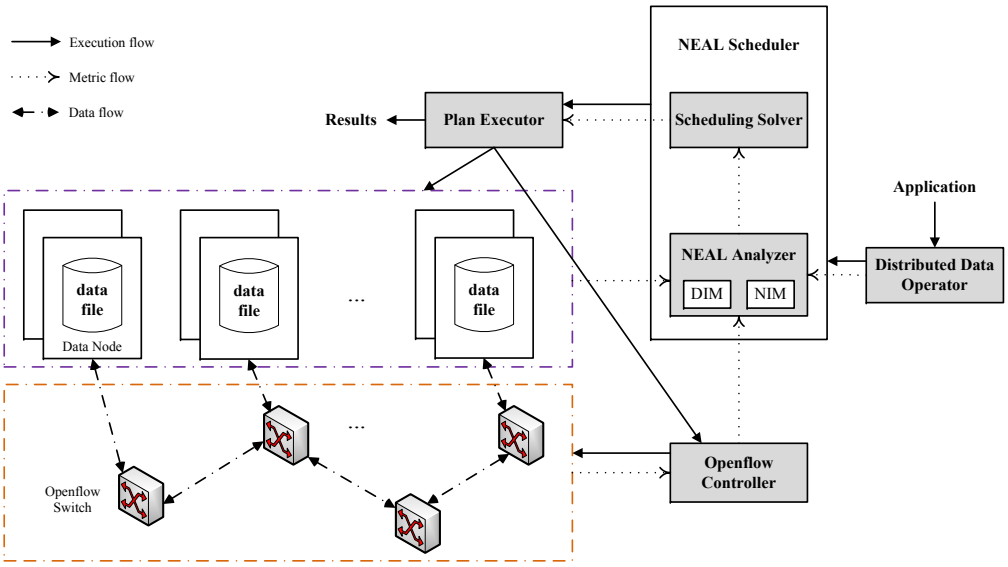


Fig. 3. System Architecture of NEAL in an SDN Data Center.

For a given data application, such as an analytical query, its data processing will be decomposed into a set of distributed data operators. For each operator, a performance model on optimizing its communication time is constructed by the NEAL Analyzer, based on the data and network information collected by its two components: (1) a data information manager (DIM) and (2) a network information manager (NIM). The same as a traditional distributed data management system, DIM maintains a global view of the meta-data of all data relations stored in the cluster, such as key appearing frequency on each node. For NIM, it is responsible for communicating with the OpenFlow controller to collect network resource usage information like the available bandwidth of each link.

The generated optimization model will be forwarded to a scheduling solver (described in Section 4) to calculate an optimized scheduling plan for data redistribution, which includes: (1) the destination node information for data tuples on each node, and (2) the assigned network bandwidth for each link of data flows generated by (1). Specifically, following the plan, the Openflow controller assigns the bandwidth for each communication link, and each computing node sends its local data chunks to the targeted remote nodes, and all the data transferring is done in parallel.

3.2 SDN in NEAL

SDN [15] is essentially a centralized networking paradigm, and the OpenFlow is a standard communication interface among the layers of an SDN architecture [17]. The network control functions of an SDN network are located at a set of control entities called OpenFlow controllers, and the data forwarding plane is located at packet-processing devices for operating the requests from OpenFlow controllers. Compared to traditional networking control approaches, SDN can provide per-flow control based on the attributes of packet headers.

The SDN architecture has many features, such as flow-based forwarding, dynamic flow rules, traffic monitoring [18], etc. In our system, the OpenFlow controller has to communicate with the NIM and the plan executor to update network information and receive network control commands respectively. In these processes, we mainly take the advantages of two key properties of SDN as follows.

Decoupled control plane and data plane. Network operators can be easily deployed, configured, and managed on OpenFlow controllers without setting any packets-processing devices. Therefore, bandwidth assignment plans generated by NEAL can be deployed in the Openflow controller in our system to simplify its network management. For example, in the case of a re-configuration requirement for the network, e.g., to execute a new data operator, we just need to adjust the controller rather than all the network devices.

Flow control. Quality of Service (QoS) [19] is the capability of a network to provide the required performance for a specific network traffic, such as bandwidth, delay, and loss. Most existing QoS solutions work in best-effort services, and their control on finer-granular traffic is limited. In comparison, the control plane of SDN architecture has an abstract view of the whole network, and it can obtain global network status, such as network resources, events, and packets [19] [20]. Based on these information, fine-grained policies can be then defined and applied per-flow to each networking entity. NEAL works on the basis of flow bandwidth assignment. Therefore, SDN provides NEAL a more flexible approach to control network flows and guarantee the bandwidth QoS.

3.3 Optimization Model

To describe the detailed optimization process of a distributed operator in the NEAL scheduler, we take the aggregation as an example and use the follow model: There are n computing nodes and the input tuples on each node have been partitioned into p parts. For a general case, we assume that the partitioning is based on the hash values of keys. The hash value of the k-th part is k, and the size of the data at node i is h_{ik}. We denote an individual data partition at a node as a *chunk* and a group of data chunks

TABLE 1
Table of notations

Notation	Meaning
τ	network communication time
n	number of computing nodes
p	number of data partitions
x_{jk}	decision variable whether the k-th partition is assigned to node j
h_{ik}	size of the k-th data chunk on node i
f_{ij}	data flow from node i to node j
b_{ij}	transmission bandwidth assigned to f_{ij}
v_{ij}	size of data flow f_{ij}
L_{ij}	link set of data flow f_{ij}
R_l	available bandwidth of link l
r_i	avail. bandwidth for the physical port of node i

with a same hash value (from all the nodes) as a *partition*. For example, in Fig. 1, the 1^3 is a data chunk on the Node 1, and the group of 1^3 and 1^6 is a partition of the input data. In an aggregation operator, tuples at each node with the same hash value will be assigned to a same node to implement the final local computing. Therefore, we can use a decision variable $x_{jk} \in \{0, 1\}$ to indicate whether a data partition k is assigned to the node j. Namely, $x_{jk} = 1$ represents that partition k is assigned to node j, and $x_{jk} = 0$ means not. Moreover, we use f_{ij} to indicate the data flow generated by data movement from node i to node j, the size of which is v_{ij}, the assigned bandwidth for the transmission is b_{ij}, and the set of its communication links is L_{ij}. Obviously, this model contains both application-layer scheduling for data locality assignment (i.e., x_{jk}) and network-layer scheduling for data communication (i.e., b_{ij}). For the convenience of our presentation, we use the notations as listed in Table 1.

We only compute the network communication cost for the data moved to a remote node, since a local movement will not consume any network resources. Based on a given data locality scheduling plan, we can get all the information of each individual data flow, i.e., its source, destination, volume and the network resource status (i.e., the available bandwidth for its link). In this case, the problem on minimizing network communication time τ for an aggregation can be represented as to minimize the CCT of the coflow composed by all the responsible individual data flows. Namely, we have the optimization model as below.

$$\mathcal{P}_1: \ \ minimize \ \ \tau \tag{1}$$

subject to:

$$\tau = \frac{v_{ij}}{b_{ij}} \tag{1.1}$$

$$v_{ij} = \sum_{k=1}^{p} h_{ik} x_{jk} \quad \forall i, \ i \neq j \tag{1.2}$$

$$\sum_{l \in L_{ij}} b_{ij} \leq R_l \quad \forall i, \ i \neq j \tag{1.3}$$

$$\sum_{j=1}^{n} x_{jk} = 1 \quad \forall k \tag{1.4}$$

$$x_{jk} \in \{0, 1\} \quad \forall j, k \tag{1.5}$$

As demonstrated in Fig. 2(b), using a bandwidth-based model, the communication time for each flow f_{ij} is the same in an optimal coflow scheduling. The value τ can be calculated by the flow size v_{ij} divided by the assigned bandwidth b_{ij} for its communication link, as presented in Eq. (1.1). The size of the flow f_{ij} is computed by summarizing the size of all the data chunks transferred from node i to j, i.e., Eq. (1.2). In the data communication process, the consumed bandwidth of a link (by all the individual data flows) should be smaller than its available bandwidth. This constraint is presented in Eq. (1.3).

The link set L_{ij} can be used to characterize the topology of the underlying network, and we can use such information to optimize the routing for all the data flows. However, such a scheduling will be outside the scope of this work. For our purpose, a star topology network is used in our design. Namely, in our network model, each link set contains two links in a node-switch-node way. From this basis, the constraint Eq. (1.3) can be represented by:

$$\sum_{j=1, j \neq i}^{n} b_{ij} \leq r_i \quad \forall i \tag{1.3.1}$$

$$\sum_{i=1, i \neq j}^{n} b_{ij} \leq r_j \quad \forall j \tag{1.3.2}$$

Namely, for each computing node, the consumed bandwidth of all its output and input links should not be larger than the available bandwidth of its physical port.

For a given τ, we know that the bandwidth b_{ij} for each flow is directly proportional to its volume v_{ij} from Eq. (1.1), i.e., $b_{ij} = \frac{v_{ij}}{\tau}$. If we apply b_{ij} and the statement of v_{ij} presented in Eq. (1.2) to Eq. (1.3.1) and (1.3.2), then we have:

$$\sum_{j=1, j \neq i}^{n} \frac{1}{\tau} \sum_{k=1}^{p} h_{ik} x_{jk} \leq r_i \quad \forall i \tag{1.3.3}$$

$$\sum_{i=1, i \neq j}^{n} \frac{1}{\tau} \sum_{k=1}^{p} h_{ik} x_{jk} \leq r_j \quad \forall j \tag{1.3.4}$$

The available bandwidth for each link in our network is collected (or assigned) by the Openflow Controller. Therefore, each r_i ($\forall i \in [1, n]$) is a constant for a given data operator. From the basis of Eq. (1.3.3) and (1.3.4), our optimization problem $\mathcal{P}_1$ can be then converted into the $\mathcal{P}_2$:

$$\mathcal{P}_2: \ \ minimize \ \ \tau \tag{2}$$

subject to:

$$\sum_{j=1, j \neq i}^{n} \sum_{k=1}^{p} \frac{h_{ik}}{r_i} x_{jk} \leq \tau \quad \forall i \tag{2.1}$$

$$\sum_{i=1, i \neq j}^{n} \sum_{k=1}^{p} \frac{h_{ik}}{r_j} x_{jk} \leq \tau \quad \forall j \tag{2.2}$$

$$(1.3) \quad and \quad (1.4)$$

It is hard to solve the optimization problem $\mathcal{P}_1$ directly, since the programming is not only nonlinear, but also has binary integer variables. Regardless, as the model $\mathcal{P}_2$ shows, our optimization problem actually can be transformed into a mixed integer linear programming (MILP) problem. This is because both the available bandwidth r_i of each node

(port) i is a constant and the size of each data chunk h_{ij} will be also a constant for a given data partitioning method. In this condition, we only have the binary integer variables x_{jk}, and we will be able to get an approximately optimal solution for the problem.

To speed up the parallel execution of a data operator in a distributed system, we have tried to minimize the time of the possible slowest data flow generated by the operator. This makes our optimization problem look like a load balancing (LB) problem. However, it should be noted that our work is a coflow-based problem rather than a LB-based problem. The main difference is that coflow scheduling aims to optimize the parallelism of data flows to minimize communication time, and LB focuses on balancing the resource utilization (e.g., bandwidth). Therefore, they are two different types of problems having different objectives.

Generally, in our performance model, x_{ij} is associated with the data locality and we have used the knowledge of underlying networks (i.e., available bandwidth and possible communication time) to help us to optimize the data locality scheduling. This also the reason why we call our method as **NE**twork-**A**ware **L**ocality scheduling.

3.4 Distributed Join within NEAL

The presented NEAL scheduling model is built on the basis of an aggregation process. We can apply the model to distributed joins. For a join between two relations $\mathcal{R}$ and $\mathcal{S}$, we denote $\mathcal{R}_i$ and $\mathcal{S}_i$ are the tuples of $\mathcal{R}$ and $\mathcal{S}$ on node i respectively. With a hash partitioning, the partitions with a hash value k are $\mathcal{R}_{ik}$ and $\mathcal{S}_{ik}$ respectively. Then, we can treat the sum of the $\mathcal{R}_{ik}$ and $\mathcal{S}_{ik}$ as h_{ik} in our optimization model $\mathcal{P}_1$ and on that basis to get an optimized locality scheduling plan. However, the performance improvement brought by such a plan could be limited, especially when data skew appears in a relation. The reason is that transferring skewed tuples will bring in heavry network traffic and also result in network hot spots [6], i.e., a large number of skewed tuples are flushed into a few nodes.

As skew occurs naturally in big data applications and joins [6], a large number of techniques have been proposed to against data skew in join executions [6], [7], [21], [22]. Among them, we have chosen the PRPD (*partial redistribution & partial duplication* [21]) in our approach. The reason is that PRPD is highly efficient and very simple to implementation, and the method has been adopted by many data systems (e.g., Teradata [21], Microsoft [7] and Oracle [23]). The core idea of PRPD is: the large number of skewed tuples in a relation are kept locally and not transferred at all, instead, just a very small number of non-skewed tuples from another relation are broadcast to all other nodes.

For a general case, we assume that only $\mathcal{S}$ is skewed here[1]. Following the PRPD, we extend our optimization model $\mathcal{P}_1$ for joins by replacing Eq.(1.2) with the two equations as below.

$$v'_{ij} = \sum_{k=1,k\in\mathcal{H}}^{p} |\mathcal{R}_{ik}| \quad \forall i,\ i \neq j \quad (1.2.1)$$

$$v_{ij} = v'_{ij} + \sum_{k=1,k\notin\mathcal{H}}^{p} (|\mathcal{R}_{ik}| + |\mathcal{S}_{ik}|)x_{jk} \quad \forall i,\ i \neq j \quad (1.2.2)$$

Here, $\mathcal{H}$ is the set of hash values of the skewed keys, $|\mathcal{R}_{ik}|$ and $|\mathcal{S}_{ik}|$ are the size of $\mathcal{R}_{ik}$ and $\mathcal{S}_{ik}$ respectively. In this condition, v'_{ij} means the size of the data flow generated by the duplication behavior from node i to j, and it will be considered as an initial value to calculate the v_{ij} for the flow f_{ij}, as presented by Eq. (1.2.2).

The above skew handling method focuses on the processing of skewed data, we can treat it as a pre-processing before our scheduling. It means that the value of each v'_{ij} will be a constant, and thus we can convert the optimization model of a join to a MILP problem similared to the problem $\mathcal{P}_2$, and consequently to get an optimal data locality scheduling plan. In fact, serveral efficient approaches have been proposed for the pre-processing (e.g., bifocal-sampling [24]), and various results in real big data applications have shown its overhead can be ignored, compared to the performance improvement it brings [25]. Moreover, we will apply the PRPD strategy to handle data skew in all the approaches in our evaluation. In such scenarios, we just assume that the skew is known for a given input and will not consider its detailed overhead in this work.

4 Implementation of NEAL

In this section, we describe the detailed implementation of NEAL based a metaheuristic approach.

4.1 Metaheuristics-based Formulation

As we have described, the data locality scheduling optimization for both the aggregation and join operator can be converted into the MILP problem as presented in the optimization model $\mathcal{P}_2$. It is obvious that our optimization is more complex than a single coflow scheduling, which actually can be mapped to an open-shop scheduling problem [11], [26], [27], of which the computational complexity is NP-hard[2]. From a theoretical perspective, our problem solving time is exponential time. Therefore, the scheduling process could bring in a heavy overhead for an application: (1) an analytical job always contains multiple distributed operators; and (2) for each operator, when the number of nodes n and the number of data partitions p are large, the problem instances will get too large to be solved in a timely manner.

Although we can use an optimizer such as Gurobi to solve the problem, we propose an implementation based on a metaheuristic algorithm in this work. We have two main considerations here: (1) Metaheuristic algorithms are shown to be able to provide better scalability compared to an optimizer, since they can solve much larger problem instances than an optimizer in reasonable time; and (2) We have plans to extend our model in more complex

1. Note that uniform-skewed joins are the core part of join executions, and the skewed-skewed joins can be processed in a similar way.

2. For three or more nodes, or three or more data partitions, with varying transmission times, in our problem.

environments (e.g., complex network topology) and more complex cases (e.g., with multiple objectives) in our future work. In this case, it will be hard to use an optimizer to solve the relevant problems. In comparison, we can simply extend a metaheuristics-based implementation to deal with them. Moreover, using a heuristics would make our solution simpler and easier to implement and deploy in a computing system. In fact, metaheuristics have been used in SDN data centers to solve various optimization problem [28], [29], [30]

To fit a metaheuristic, we reformulate the optimization model $\mathcal{P}_2$ as $\mathcal{P}_3$ in Eq. (3). There, F is a fitness function, $\mathcal{O}_i$ and $\mathcal{I}_i$ mean the communication time of each node on sending and receiving data respectively. In this case, our target becomes to minimize the value of F.

$$\mathcal{P}_3: \quad F = \max_{\forall i \in [1,n]} \{\mathcal{O}_i, \mathcal{I}_i\} \tag{3}$$

subject to:

$$\mathcal{O}_i = \sum_{j=1, j\neq i}^{n} \sum_{k=1}^{p} \frac{h_{ik}}{r_i} x_{jk} \quad \forall i \tag{3.1}$$

$$\mathcal{I}_j = \sum_{i=1, i\neq j}^{n} \sum_{k=1}^{p} \frac{h_{ik}}{r_j} x_{jk} \quad \forall j \tag{3.2}$$

To date, various metaheuristic algorithms have been proposed, such as the the genetic algorithm (GA) and the swarm intelligence algorithms like ACO and PSO (details see the survey [31]). We choose one of the latest metaheuristics – the whale optimization algorithm (WOA) [32] in our implementation. The algorithm is nature-inspired by the humpback hunting method (i.e., bubble-net predation). Because of this unique optimization mechanism, WOA can provide a good global search capability, which makes it become popular in various engineering problems such as cloud task scheduling [33].

4.2 An Overview of WOA

In the WOA algorithm, a whale (or search agent) in the search space is considered as a candidate solution, and the WOA utilizes a set of search agents to determine the global optimal solution. For a given optimization problem, the searching process of WOA starts with a set of random solutions, and each solution is updated by the optimization rules until the end condition is met. The WOA algorithm can be divided into three main stages: encircling preying, bubble-net attack and search for prey. There mathematical representations are given as below.

4.2.1 Encircling Preying

In the initial stage, search agents do not know the optimal location in the search space when the prey is surrounded. In WOA, the current best solution is considered as the target prey and the whale closest to the prey is considered as the best search agent. Then, other individual whales may approach the target prey and gradually update their locations. This behavior is represented as:

$$\vec{D} = |C \times \vec{X}^*(t) - \vec{X}(t)| \tag{4.1}$$

$$\vec{X}(t+1) = \vec{X}^*(t) - A \times \vec{D} \tag{4.2}$$

TABLE 2
Terminology Mapping between Whale Forage and NEAL Scheduling

Whale Forage	NEAL Scheduling
individual whale	scheduling problem
foraging process	optimal solution search process
whale position	a solution X for F_{opt}
leader whale	optimal solution X^* for F
fitness of whale	value of F

Here, $\vec{D}$ indicates the distance vector from the search agent to the target prey, t is the current iteration number, $\vec{X}^*$ is the local optimal solution and $\vec{X}$ is the position vector. C and A are the coefficient vectors and their calculations are defined as:

$$C = 2 \times r \tag{4.3}$$

$$A = 2a \times r - a \tag{4.4}$$

The r is a random number between 0 and 1, and a represents a linear decremented value from 2 to 0 based on the number of iteration t.

4.2.2 Bubble-net Attack (Exploitation Phase)

The behavior of whales' bubble-net attack is modeled based on two ideas: (1) *Shrinking encircling*. From Eq. (4.2), we can see that the whales will shrink their encircling when $|A| < 1$. This means that the individual whales will approach the whale in the current best position, i.e., swim around the prey in a gradual contraction of a circle. The larger the value of $|A|$ is, the bigger steps the whales will take, and vice verse; and (2) *Spiral position updating*. Each individual whale first calculates its distance from the current optimal whale and then moves in a spiral shaped path. The mathematical model of the position update process is described as:

$$\vec{X}(t+1) = \vec{D'} \times e^{lb} \times \cos(2\pi l) + \vec{X}^*(t) \tag{4.5}$$

where $\vec{D'} = |\vec{X}^*(t) - \vec{X}(t)|$ is a vector indicating the distance from the individual whale to the best whale (current best found), b is a constant and l is a random number with the value between -1 and 1.

In order to mimic the two behaviors in a simultaneous way, it is assumed that the possibility of a whale updating its location based on the contraction path and the spiral path is 0.5 respectively.

4.2.3 Search for Prey (Exploration Phase)

To ensure that a global optimal solution can be achieved, the search agents are pushed away from each other when $|A| > 1$. In this case, the position of the current optimal search agent will be replaced by a randomly selected search agent, and the responsible mathematical model is

$$\vec{X}(t+1) = \vec{X}_r - A \times |C \times \vec{X}_r - \vec{X}(t)| \tag{4.6}$$

where $\vec{X}_r$ is a position vector of the randomly selected search agent.

4.3 WOA-based Implementation

The NEAL optimization problem described in Section 4.1 can be translated to the whale foraging problem with preemption as summarized in Table 2: An individual whale corresponding to the given scheduling task and the whale foraging process is the optimal solution searching process. In a search, a whale has a position corresponding to the scheduling problem has a solution X_{pn}, which is a $p \times n$ matrix representing the values of all the decision variables x_{ij}. The position of the leader whale in a search means the current optimal solution, and the fitness value of the whale is the current optimal value of F. On this basis, we can get an optimized solution for our NEAL problem with the WOA algorithm.

The detailed implementation of NEAL scheduling using WOA is presented in Algorithm 1. The input information includes the number of nodes, the number of data partitions, the size of each data chunk and the available bandwidth for each network port (node). First, the position of each search agent is randomly generated (line 1), and each of the positions is then converted into a solution to the optimization problem (line 2). Then, the fitness value for each solution is computed based on Eq. (3), and the solution X^* with the minimal value will be stored (lines 3-4). After that, the iteration process is commenced to refine the optimal solution. In each iteration t, the position of each search agent is updated based on the mathematical equations (4.1) - (4.6) designed in the WOA algorithm (lines 7-17). The updated positions are then converted into solutions to calculate the fitness values in the current iteration (lines 19-20). The current optimal solution X^* will be updated based on the achieved minimal fitness value, and it will be used to update the positions for the search agents in the next iteration, when required. This process will be repeated until the final iteration is reached. The final optimal X^* is then output as the optimal data locality plan. With Eq. (1.1) and (1.2), we calculate the value of each b_{ij} (lines 24 -25). This optimal scheduling plan including the assignments of data locality and network bandwidth will be delivered to the underlying system for the execution of the given distributed data operator.

A specified optimization: In a conventional WOA implementation, all the initial positions of search agents are randomly generated. Different from that, we have employed a specified optimization in Algorithm 1 for our implementation: one of the positions is initialized with the solution from the hash-based scheduling. Namely, the k-th data chunk on each node is assigned to node k. Since the hash-based scheduling is the most commonly used approach, this initialization will provide useful knowledge for all the search agents and consequently to improve the accuracy of the optimal solution and enhance the global development capability of the algorithm, compared to a random one. Also, it should be noticed that we can get the hash-based plan directly without bringing in any overhead.

Algorithm 1 WOA-based implementation of NEAL

```
   Input: n, p, h_ik, r_i
   Output: x_jk, b_ij
 1: Randomly initialize position P^i for each agent i
 2: Convert each P^i into X_i
 3: Calculate F_i for each X_i by Eq. (3)
 4: Store X* as best search agent
 5: while t < Iter_max do
 6:     for each search agent do
 7:         Update the values of a, r, A, C, l, rand
 8:         if rand < 0.5 then
 9:             if |A| < 1 then
10:                 Update current position with Eq.(4.1)
11:             else|A| ≥ 1
12:                 Select a random agent X_r
13:                 Update current position by Eq.(4.6)
14:             end if
15:         else rand ≥ 0.5
16:             Update the position by Eq.(4.5)
17:         end if
18:     end for
19:     Convert each position to X_i
20:     Calculate fitness value of each agent by Eq. (3)
21:     Update the value of X*
22:     t++
23: end while
24: Output X* as x_jk, the fitness value of X* as τ
25: Calculate v_ij based on x_jk by Eq. (1.2)
26: Calculate b_ij based on τ by Eq. (1.1)
```

4.4 Overhead of Implementation

Compared to a conventional data locality scheduling approach, our WOA-based NEAL implementation could bring in more overhead for the execution of data operators. However, we argue that its overhead can be actually ignored. The main reason is that the communication of data operators is not short-lived. For instance, as we will demonstrate in the experimental results in Section 5, the communication may take hundreds or even thousands of seconds to complete. In this case, it is worth it to pay for the optimization to get more performance benefits. Moreover, one of the core motivations of metaheuristics is to get an approximate optimal solution in a reasonable time. From the Algorithm 1, we can see that the overhead of our implementation depends on the number of search agents and the number of iterations in the WOA algorithm as well as the the number of nodes and data partitions in our optimization model. Therefore, we can choose a small number of search agents and iterations in practice to reduce the overhead when required.

The WOA algorithm actually has demonstrated its performance advantages in a benchmark test including 29 mathematical optimization problems and 6 structural design problems [32]. For our implementation, we have recorded the convergence and overhead for each test we have conducted in our experiments in Section 5. In detail, we have set the number of search agents to 10 and perform our implementation using MatlabR2018b on a commodity laptop with an Intel i7-8550U CPU running at 1.80GHz (code link see Section 5). Since all the results have demonstrated a similar characteristic, we just report some typical ones in Fig. 4 as a reference for readers. For the convergence results, the normalized value is the communication time achieved by NEAL dividing by that achieved by the hash-based scheduling. We have set 100 iterations in our implementations and here we report the first 25 for the readability of the results. We can see that the converge of our searching process is very quick and can be done in less than 10 iterations. Moreover, the runtime of each iteration is

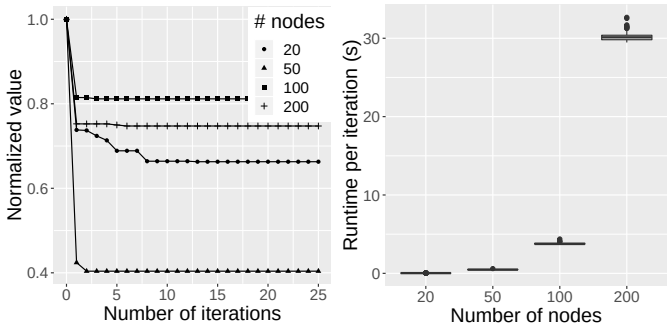


Fig. 4. The best performance (normalized values) achieved by increasing the number of iterations and the runtime of each iteration in optimal solution searching.

around 0.039 sec for 20 nodes (60 data partitions) and about 3 secs for 100 nodes, and these overheads can be ignored compared to achieved performance improvements (e.g., up to 60%).

From the results, we can also observe that the overhead is about 30 secs for each iteration when the number of nodes reaches 200 (6000 data partitions). The overhead is significant but still reasonable. This is because the number of the possible locality scheduling plans is 200^{6000}, and the search space of which is large. To reduce the overhead for such kind of large problems, we can apply various strategies to speed up the WOA implementation. For example, we can easily parallelize the WOA following the BSP model [34], since the computation of each search agent is totally independent from each other in each iteration. Specifically, the execution of WOA is compute-bound and the computing tasks are represented by the mathematical equations (4.1)-(4.6). In comparison, the size of the used arrays and matrices in memory will be fixed, since we only need to update the related values (e.g., lines 7, 10, 13 and 16 in Algorithm 1) during the execution. For a straightforward parallelism, in each iteration, we can assign the computing tasks of each agent (lines 7-17 in Algorithm 1) to a core (or a machine). Moreover, advanced approaches with parallelism over GPU and FPGA [35], [36] as well as effective convergence strategies [37] are also available for metaheuristics. Although we can integrate these approaches in NEAL, the details will be outside the scope of this work. In addition, it should be noticed that, in real cases, a system optimizer will be able to get the possible cost of scheduling and network communication for each candidate approach, and then choose the best one for executions.

In fact, with a low enough overhead, NEAL can be also applied in a dynamic network environment, i.e., the available bandwidth varies with time. In this condition, we can let the SDN controller periodically check the available bandwidth and forward the information to NEAL, to guarantee that the data transmission is optimal in each time period. In this work, the same as almost all the research works on distributed joins [5], [6], [7], we are concentrated on optimizing communication time in a static network, i.e., the available bandwidth for a data operator has been given by an SDN controller at a global level, and the detailed results of which will be presentation in the following section.

5 Evaluation

In this section, we evaluate the performance of NEAL through approximate testbed and simulation-based experiments. The code we have used in this section is available at https://github.com/longcheng11/NEAL. A detailed guidance on how to generate data and how to run the code is also provided there.

5.1 Experimental Framework

We compare the performance of NEAL with the commonly used solutions on the executions of distributed data operators. The results of aggregations demonstrate the similar characteristics as distributed joins. For simplicity of presentation, we just report the results of distributed joins in this section. As summarized in Table 3, we divide the solutions into five schemes, based on the used scheduling techniques.

1. In relation to locality scheduling, the *Hash* there means the most commonly used hash-based approach [21]. Namely, after the hash partitioning, each data chunk is assigned to a node based on its responsible hash value. The *Min* is a scheduling approach focusing on reducing network traffic. Namely, we will examine all the possible destinations for each data partition and choose the one that can make the network traffic minimal. The detailed scheduling is similar to the locality-aware operation described in the work [2]. Namely, after data partitioning, we explore data locality at a per-partition level. For each data chunk in a partition, we set its destination to a node, which contains the most number of tuples.
2. For network scheduling, the *free flow* means that data flows compete the available bandwidth in a fair way. The *Coflow* means that we apply optimal coflow scheduling to the generated parallel data flows and guarantee that their CCT is minimized.

Moreover, we have applied the commonly used skew handling technique PRPD [21] to all the five approaches, to guarantee a fair comparison in the presence of data skew.

TABLE 3
Commonly used approaches and comparison to NEAL

Scheme	Locality Scheduling	Network Scheduling
Hash-1	Hash	free flow
Min-1	Min	free flow
Hash-2	Hash	Coflow
Min-2	Min	Coflow
NEAL	NEAL	Coflow

Generally, the schemes Hash-1 and Min-1 represent the techniques studied the data management domain (e.g., [21], [5]), which seldom consider underlying network communications. The schemes Hash-2 and Min-2 represent the approaches studied in the area of data communications. They focus on optimize CCT time for given data flows, generated by application-layer data locality scheduling. Specifically, Min-2 represents the current approaches focusing on optimizing both network traffic and network communication time for given data operators, but in a *decoupled* way. In contrast, to improve communication time, our approach adopts

a *co-optimization* way, i.e., NEAL seamlessly combines data locality and data communication by formulating it as a joint optimization problem.

5.1.1 Methodology

For a given workload and network bandwidth configuration, we measure the communication time of a distributed join based on an approximate testbed emulation and a set of large simulations. (1) Our emulations are based on the Mininet [38], which is a network emulator for OpenFlow based software defined networks. (2) In terms of coflow simulations, the *CoflowSim* used in Varys [11] and Aalo [39] can not be applied to our case, because we can not set bandwidth values in the software. In such a condition, we have developed a simulator using Matlab based on the bandwidth-based model [12], which can provide theoretical communication time for a coflow.

There are three inputs for each test in our experiments: (1) the volume of data flow between each node, (2) the assigned bandwidth for each data flow, and (3) the available bandwidth for each port. The former two inputs are in the form of an $n \times n$ matrix respectively. They can be generated by a given scheduling scheme as described in Table 3. The third input is in the form of an array with size n. Moreover, we have used the widely used TPC-H benchmark [40] to generate test workloads, and evaluated three aspects that may affect the performance of NEAL: (1) the number of computing nodes (hosts), (2) the data distribution over computing nodes, and (3) the data skew of a dataset.

5.1.2 Workloads

We select two large relations `CUSTOMER` and `ORDER` from TPC-H and perform the join operators based on their `CUSTKEY`. The scaling factor of TPC-H is set to 600, and the number of tuples in the two relations is 90 millions and 900 millions correspondingly. To get the volume of a data flow easily in a scheduling process, we set the payload of each tuple to a fixed value. In this case, we can get the volume by counting the number of tuples. We set the value to 10 Bytes to in our testbed experiments and 100 Bytes in our large simulations, which leads to around 10GB and 100GB input size respectively. The generated data is uniformly distributed, and the size of data chunks in each partition would be very closed to each other for a given hash partitioning. To evaluate our approach in more complex conditions, we make the difference of the size more obvious: for each partition, we let the size of included data chunks follow the Zipfian distribution over the n nodes based on their node id (referred to as *zipf*). Moreover, as data skew (referred to as *skew*) is quite common in data applications, in order to control the skewness in our tests, similar to many current works such as [21], we randomly choose a portion of data and change their `CUSTKEY` to a specified value. For example, we randomly choose 10% of the tuples and set their key to 1, which will make the skewness to 10%. In this way, we can easily identify on-going experiments and capture the essence of data skew.

5.1.3 General Setup

There are two parameters for the test datasets, we set *zipf* to 0.8 and *skew* to 20% as default. Moreover, without loss of generality, we just use a simple hash function $f(k) = k \ mod \ p$ to partition data tuples, and set p to a value which is 3 times the number of used nodes in each test. To demonstrate that NEAL can always perform better than other approaches in various network bandwidth configurations, we randomly generate an array to set the available bandwidth of each port assigned for a single data operator to a value between 0 and 1Gbps in each test[3]. For the parameters used in the WOA-based implementation, as we have discussed in Section 4.4, we have set the number of search agents to 10 and the maximum number of iterations to 100 for NEAL .

5.2 Approximate Testbed Emulations using Mininet

5.2.1 Emulation Setup

We deploy Mininet on the virtual Ubuntu 64-bit OS of a local machine. The virtual machine runs on on VMWare Workstation 12 Player. To gurantee the emulation procedures are not affected by the performance of the machine and the results are close to real hardware experiments, we set the resources of the virtual machine, including bandwidth, processors and RAM, much larger than the total resources required in the experiment [41] [42]. The virtual machine is set to 16G RAM and 8 Intel(R) Core(TM) i7-6700HQ CPUs on 2.60GHz. The internal bandwidth of the virtual Ubuntu system is around 50.0 Gbps. We use random data to transfer between nodes, which does not involve any data processing and storage before and after data transmission.

The aim of the experiment is to test the scheduling of data flow among hosts in SDN networks. Following our model, we use a star topology, in which all the hosts link to a central switch. We use an Open vSwitch in the network, which is a kernel-level virtual software switch. An OpenFlow reference controller is connected to the switch. In the Coflow experiments, to set up the flow among hosts according to the optimal values, we do not limit the bandwidth of physical link between the switch and each host. Instead, we make TCP link for each data flow and set the bandwidth for each TCP link based on iPerf3 library [43]. In the free flow experiments, we set the bandwidth of physical link between the switch and each host to a generated value. After that, we make TCP link between each pair of hosts without assigning the bandwidth.

Although the emulated environment cannot fully catch all the properties of a real-world network system, we believe the resulting error between our emulation and a real-world system is quite limited. We explain the reasons from the software and hardware aspect respectively as follows.

Software: Mininet [38] is a networking emulator providing a network testbed for developing OpenFlow application prototypes. The emulated entities by Mininet include virtual hosts, switches, controllers, links, etc. The entities use Linux kernel, real network stack, and standard Unix/Linux network applications. Specifically, the hosts in Mininet use Linux network software, and thus each of them can be

3. The bandwidth arrays we have generated in our test are available at the link of our code. Readers can also set the bandwidth range or distribution in the source code based on their own interests.

TABLE 4
Performance comparison with different network bandwidth configurations

Exp./Alg.	Network Traffic (GB)			Mininet time (s)					Matlab time (s)		
	HASH	Min	NEAL	HASH-1	MIN-1	HASH-2	MIN-2	NEAL	HASH-2	MIN-2	NEAL
Exp-1	6.64	5.21	6.28	985.01	2575.29	877.15	2374.04	699.15	866.31	2318.28	682.65
Exp-2	6.64	5.21	6.57	1011.17	117.04	981.55	65.45	30.16	958.49	63.81	29.34
Exp-3	6.64	5.21	6.31	2018.44	5359.83	1876.74	5022.05	1480.25	1832.65	4904.26	1445.45
Exp-4	6.64	5.21	6.69	88.06	167.57	59.36	71.94	54.95	57.82	70.18	53.54
Exp-5	6.64	5.21	6.52	37.57	99.06	18.5	44.04	15.26	17.94	42.88	14.8

considered as a Linux machine. Moreover, the used program, for each OpenFlow controller, switch, and host, can be directly deployed to a real hardware system.

Hardware: The hardware resources in a real-world system are much more powerful than the emulated hardware in a virtual machine. However, the emulated network bandwidth and computing speed will not lead to significant performance difference, compared to a real system. There are two main reasons: (1) The bandwidth allocated in the experimental network is smaller than the internal bandwidth of the virtual system (50 Gpbs). In the meantime, we use SDN architecture to set bandwidth for each flow. Therefore, the available bandwidth for each traffic in our emulation environment is the same as a real-world system; and (2) Although the CPU speed of the virtual machine is lower than a real-world server, the speed is enough for performing bandwidth assignment in a controller. In addition, we focus on optimizing communication time. The relevant operations on computing such as data partitioning have been preprocessed. Therefore, the CPU speed of the virtual machine does not affect the result validation.

5.2.2 Experimental Results

As a proof of concept, we set the number of computing nodes to 10 and use the 10GB dataset with the default parameters. We run our experiments five times with five different network bandwidth configurations, and the results of network traffic and communication time for each approach are presented in Table 4. It can be observed that Hash and Min transfer the same number of tuples over five different networks respectively. The reason is their data locality scheduling is independent from underlying networks. In comparison, the network traffic of NEAL is different in the presence of different network conditions, demonstrating that our method is indeed network-aware and its data locality plan can adjust the changes of the network. Moreover, we can see that Min transfers less data than other approaches for a given workload, which shows its advantages on network traffic reduction.

For the communication time over Mininet, we can see that Hash-2 and Min-2 can always perform faster than Hash-1 and Min-2 respectively. This means that a network-layer optimization data flow scheduling can indeed speed up data communications. Compared to Hash-2, Min-2 sometimes performs much faster (i.e., Exp-2) and sometimes performs much slower (i.e., Exp-1 and Exp-3). Considering that network traffic of Min is smaller than Hash, these results imply that minimizing network traffic will not necessary lead to an optimal communication time, and we should consider underlying networks when we perform data locality scheduling. Compared to Hash-2 and Min-2, we can see that NEAL always performs the best. Specifically, it achieves $1.3\times$ and $3.4\times$ speedups over Hash-2 and Min-2 respectively in Exp-1, and $32.5\times$ and $2.2\times$ in Exp-2.

5.3 Large-scale Simulations using Matlab

For our purpose of this work, we will only compare the performance of Hash-2, Min-2 and NEAL in the following. Before that, we first compare the results we have got by Matlab with Mininet for the tests in Section 5.2. As shown in Table 4, the emulated time for transmitting data flow is a little longer than the theoretical value by Matlab. The main reason is that, in Mininet, packets on data plane are exchanged among switches and hosts. Also, the controllers exchange packets on control plane with switches. These process will consume extra time in data transmission. From another perspective, these results also demonstrate the effectiveness of our simulation using Matlab.

We use 100GB data and evaluate the three approaches in three different scenarios as described in Section 5.1.1. In each scenario, we perform five tests with five randomly generated network bandwidth configurations.

Varying number of nodes. We compare the performance of Hash-2, Min-2 and NEAL by varying the number of nodes, from 20 to 200. The results of the network traffic and network communication time are presented in Figure 5. There, the Min-2 approach transfers the least amount of data over networks, because it focuses on reducing network traffic in join executions. Moreover, we can see that NEAL sometime has more network traffic while sometime has less, compared to Hash-2. The reason is that Hash-2 just simply redistributes all the data chunks, while NEAL is able to explore part of data locality based on the constraints in our optimization model, and such data locality could even lead more network traffic due to the different bandwidth configuration of each port.

Looking at the communication time, it can been that NEAL always performs the best by varying the number of computing nodes, and its performance advantage sometimes is significant. For example, in Exp-A with 50 nodes, the communication time of NEAL is 667 secs while Min-2 is 4105 secs. In Exp-B, NEAL uses 4236 secs while Hash-2 takes 10489 secs on data communication. The main reason is that NEAL generates a data locality plan based on the network bandwidth information, and thus the network congestion can be efficiently decreased. In contrast, Min-2 just transfers data chunks to a node with the largest size for each data partition while Hash-2 simply assigns data chunks based on their hash values. Both the approaches

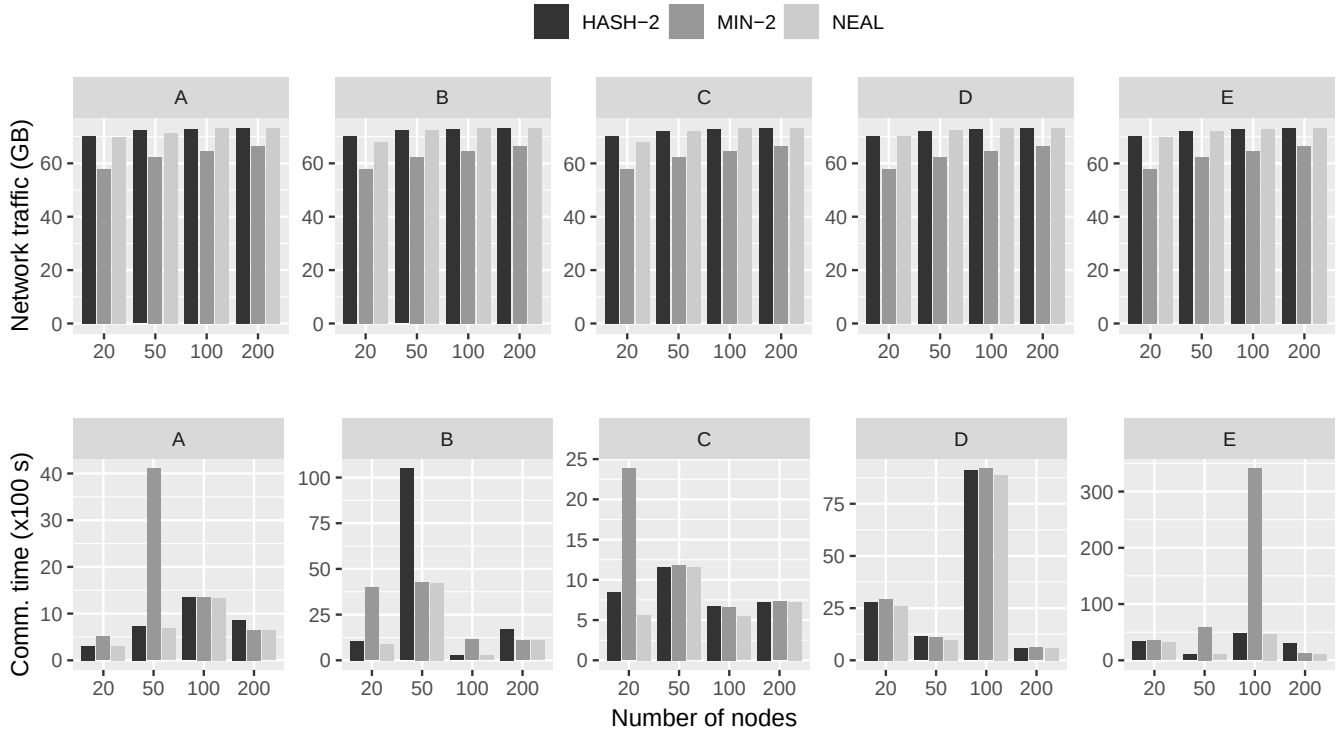


Fig. 5. Performance comparison by varying the number of nodes with different network bandwidth configurations (*zipf*=0.8, *skew*=20%).

could results in significant network congestion for the ports with a small bandwidth. Based on these results, we can see that focusing on network traffic reduction sometimes could result in a very poor join performance. In comparison, we have considered optimizing data locality based on the possible network communication time, and thus our NEAL can always perform better than the other two techniques.

With different Zipf factors. We examine the efficiency of each method over 100 nodes with increasing the parameter *zipf* from 0 to 1. As shown in Figure 6, similar to the above results, Min-2 transfers less data than the other two approaches. Moreover, its network traffic is decreasing with increasing the value of the Zipf factor. The reason is that the large data chunks become even larger with the increment of *zipf*, and such large chunks are kept locally in Min-2. In terms of network communication time, it can be observed again that Min-2 performs sometimes better and sometimes worse than Hash-2, and the NEAL approach always performs the best in all the cases. In some experiments, NEAL can achieve significant speedups compared to the Min-2 and Hash-2 methods. For example, in Exp-H, NEAL is $5.3\times$ faster than Min-2 when the Zipf factor is 0, and $4.8\times$ faster than Hash-2 when the factor is 1.

Over various skews. We evaluate the performance of the three approaches over 100 nodes with various skews, increasing from 0 to 40%. The results are presented in Figure 7. It can be seen that the network traffic of the three approaches decreases with increasing the skew. The reason is that the skew handling technique we have adopted in the three methods can effectively reduce the network traffic. Moreover, Min-2 has less network traffic than the other two approaches, demonstrating its strong capability on network traffic reduction once again. The results of the network communication time are also very similar to the previous ones, i.e., NEAL performs the best all the time, and it can achieve obvious speedups in many cases. For instance, in Exp-N, it performs $1.9\times$ faster than Hash-2 and $3.9\times$ than Min-2 when the skewness is 10%.

5.4 Brief summary and discussion

Based on all the above results, we can see that application-layer optimization techniques on data locality scheduling can efficiently reduce or even minimize network traffic. However, because the approaches have not considered the underlying network communication, their communication performance could be very poor in distributed systems, such as the Min approach we have studied here. In contrast, we propose a network-aware optimization for locality scheduling, thus NEAL can always perform better on network communications than current techniques for different network bandwidth configurations.

We focus on optimizing the parallel execution of data operators in this work. Besides the join and aggregation as we have described, NEAL can be used to speed up the communication of many other distributed data operators, e.g., groupby, sortby, outer joins, etc. Since all these data operators are the fundamental units for big data processing, it is expected that we can use NEAL to speed up various data applications. For example, in a data warehouse, users would frequently want to execute a distributed join over large datasets to combine and analyse the datasets collected from different sources [44].

Generally, NEAL can be applied directly to the applications, in which data operators can be performed in a

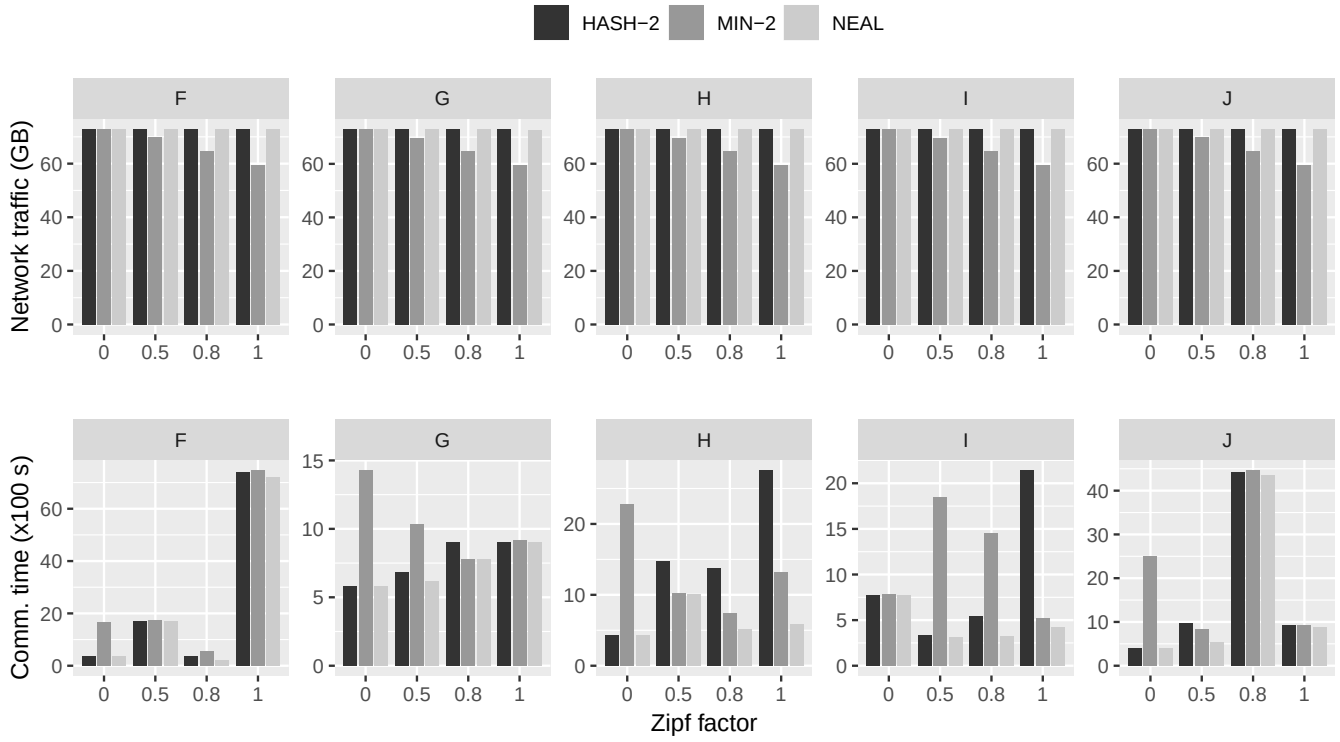


Fig. 6. Performance comparison by varying the Zipf factor with different network bandwidth configurations (100 nodes, *skew*=20%).

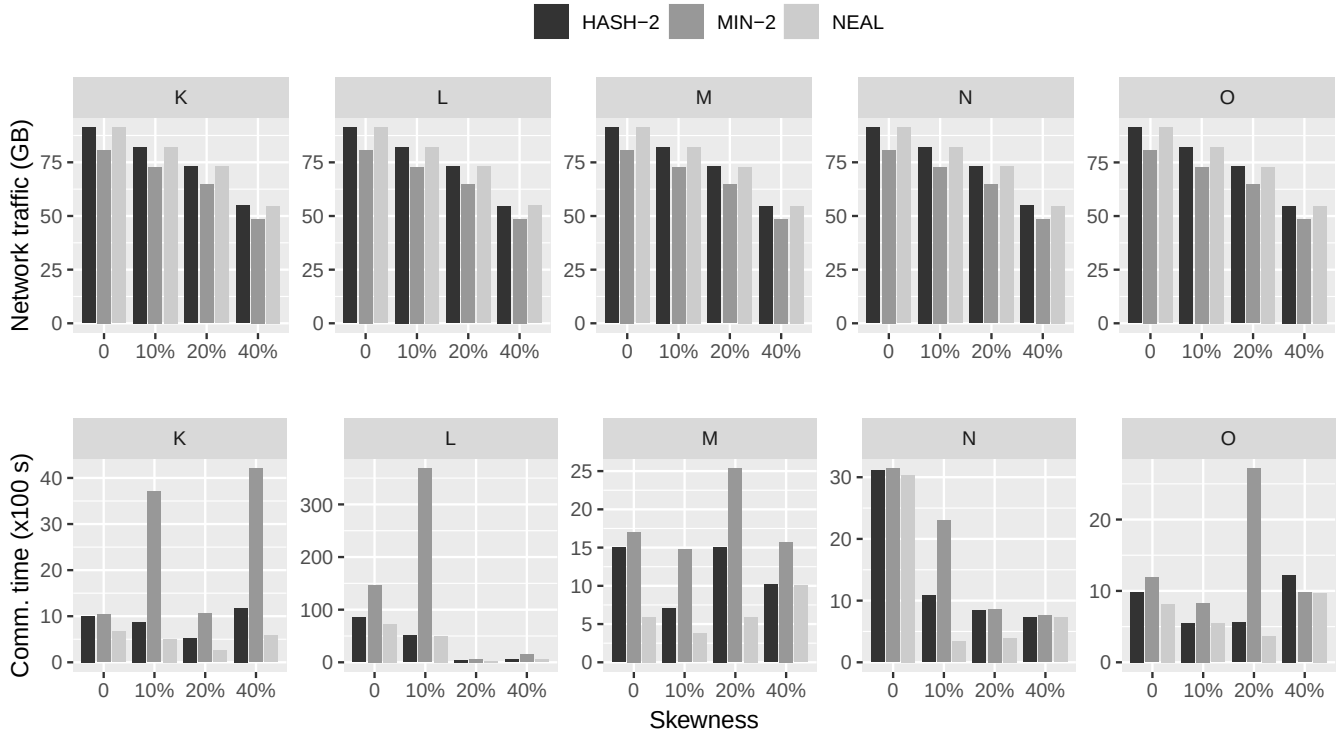


Fig. 7. Performance comparison with different network bandwidth configurations by varying the data skewness (100 nodes, *zipf*=0.8).

sequential way. For instance, a data pipeline or a query with a left/right-deep tree execution plan. The reason is that each data operator under the NEAL scheme is executed independently, i.e., each operator can use and maintain its own link bandwidth. For more complex workloads, such as performing multiple concurrent coflows generated by different data operators, NEAL does not guarantee the globally optimal performance. Moreover, we have limited

our research on network communications and have not considered the CPU time on each node in this work, such as that data flows actually can start at different time points in real applications. However, the model we have proposed in this work can be easily extended to the cases using the existing coflow scheduling techniques (e.g., the online multi-coflow model in [14]). In such scenarios, we believe that NEAL has laid a solid theoretical and practical foundation for the development of a network-aware optimization system for big data processing.

6 Related Work

Distributed join processing: As the most essential tasks in data center environments, distributed data operators such as joins and aggregations can incur significant time costs and hence improving their execution efficiency would have a significant impact on the overall performance of big data processing and analytics [6].

The research on optimizing the parallel execution of data operators is mainly done in the fields of data management and data engineering. To reduce network traffic and/or improve load-balancing, a large number of methods, mainly on distributed joins, have been proposed in the past years [5], [6], [7], [21], [22], [45]. For example, some recent works focus on optimizing network traffic by careful data partitioning and placement [4], [5], [8], [46]. Although these approaches have been shown to be efficient, their designs are totally independent from networks. As we have studied in this work, when network is unaware for an approach, the performance of the method could be poor, even in the condition that an optimal coflow scheduling has been employed. On the other hand, NEAL works under the condition that data partitioning has been given (such as the hash partitioning we have used in this work). In this scenario, NEAL is orthogonal to those approaches that they can improve the partitioning used as input for NEAL. Additionally, although several works [44] have realized that networks should be considered when performing distributed joins, the proposed approach focuses on dynamical data partitioning to handle the skew of networks rather than how to effectively utilize network resources as we have studied in this work.

Coflow scheduling: A lot of work has been done on flow-level optimization for data communications in data centers prior to the concept of coflow. However, because of their agnostic on the existence of coflows, many of them have been shown to be actually harmful to the performance of data applications [14]. Current research on coflow scheduling aims to minimize the average CCT cost or meet coflow deadlines [11]. Advanced approaches such as [39], [12], [14] have been proposed to handle online coflows and complex network topology. All of them are based on the assumption that the source and destination of each data flow have been given. Namely, their schemes are totally independent from application-layer optimization. As we have demonstrated in this work, this kind of design has reduced the possible performance gains for distributed data operators, which can actually benefit from turning data flows with data locality scheduling. Compared to this, NEAL has considered the impact of data locality, and thus it can always achieve better communication time. From another perspective, NEAL is based on the coflow model. Therefore, advanced coflow techniques such as multi-coflow scheduling can be applied to our approach, to enhance its applicability on handling more complex workloads and network configurations.

Network-aware scheduling: To speed up big data processing, various network-aware scheduling frameworks have been proposed in recent years [9], [10], [47], [48]. Their main idea is to coordinate the placement of data and tasks so that the system resource utilization (e.g., server or network) can be reduced, such as that the freed up bandwidth can be used by other running jobs in a cluster. For instance, the Map Task Scheduling (MTS) method proposed in the work [10] aims to balance cross-node network load in a data shuffle process, which is very similar to the data redistribution process in a join. Its main idea is to make sure that the total size of shuffled data from a node is not higher than a threshold. Obviously, MTS can improve network communication time because of the balancing of the utilization of network links. Although NEAL is a also network-aware scheduler, it is different from all these techniques in two aspects. (1) Optimization objective: NEAL aims to *optimize* (i.e., minimize) but not to *reduce* the communication time, which can be actually a byproduct of other optimization such as the reduction of network traffic or resource utilization of a system. (2) Methodology: the locality scheduling in NEAL is based on the possible communication time, i.e., the optimal communication time will be fixed for a give locality scheduling plan. In comparison, other approaches are based on the status of a network such as network traffic or network resource utilization, and they actually have not reached the level of coflow or data flow parallelism for communication time optimization.

SDN for data management: Software-defined networking (SDN) [15] is widely considered as the technology for network management of the next generation networks. SDN aims to provide an open interfaces for efficiently developing network control programs [49]. Compared with traditional networking structure, SDN has more flexibility and less complexity in networking flow management [50]. Given these properties and advantages, more and more implementation of data-center networks and wide-area networks utilizes SDN architecture [16] [51]. Currently, SDN has been applied to big data management [52], [53]. For example, the work [52] presents a method that be able to adaptively select an optimal query plan based on the information provided by the network before a query execution. However, these techniques just focus on using SDN to move data in a distributed way. Namely, at each specified time point, they just move data from a node to another node, but not like the problem we have studied in this work, in which the data from different nodes moves in a parallel way. We should notice that parallel cases (e.g., coflows) are very common in large-scale data applications, since data sets from different resources are always loaded in computation nodes in a parallel way so that the loaded data can undergo further downstream processing as quickly as possible.

Metaheuristics algorithms: To find an approximately optimal solution for our optimization problem, we have adopted a metahuersitc in our implementation. In fact,

metaheuristics algorithms have received increasing attention in recent years on optimization problems [54]. The reason is that they are shown to be highly effective and can find optimal solutions in polynomial time rather than exponential time, compared to conventional methods [55], [37]. Currently, some commonly used metahuristics mainly include the genetic algorithm (GA) [56] and swarm intelligence algorithms, such as the ant colony optimization (ACO) [57] and the particle swarm optimization (PSO) [58]. These optimization algorithms are derived from the simulations of biological population evolutions, and they can solve complex global optimization problems through cooperation and competition among individuals [59], [60]. We have used the latest WOA algorithm [32] in our implementation with a very lightweight optimization, and our experiments have shown that our method can always achieve better solutions on data locality scheduling compared to the current techniques.

Differences from our previous work: This paper builds upon our earlier work [61]. Besides the rephrasing, we have introduced a new research problem with four main new innovations: (1) We have extended our problem to a case with more complex network configurations, i.e., the available bandwidth for each network link is different rather than the same in [61]. (2) To handle the new problem, we have provided a system solution in a software-defined networking data center environment. (3) Because of the complexity of the new problem, the idea of our previous heuristic algorithm cannot be applied to the optimization model in this work. Therefore, we have proposed a new metaheuristic-based implementation for NEAL with a specified optimization. This makes our method easy to implement and deploy in data center systems and also guarantees that an approximately optimal data locality scheduling plan can be achieved in reasonable time. (4) For the new research problem with newly proposed implementation, we have conducted a totally new experimental evaluation over a emulator and a simulator with complex network bandwidth configurations. Moreover, we also have applied an efficient data skew handling technique to all the competitors for a more fair comparison.

7 Conclusion

In this paper, we have proposed NEAL, a novel cross-layer optimization approach which aims to improve the communication time for distributed data operators in data center systems. We have presented the detailed design and implementation of NEAL, and conducted a detailed performance evaluation using approximate testbed emulations with Mininet and large-scale simulations with Matlab. Our experimental results have shown that the proposed NEAL can indeed outperform current approaches on network communications, in the presence of different workloads and network bandwidth configurations.

To our knowledge, this is the first work to analyze the joint optimization opportunity of applying coflow scheduling to data locality scheduling for the execution optimization of distributed data operators. We believe that various big data analytics applications will benefit from our designs. Our future work mainly lies in extending our optimization model to more complex workloads (e.g., online queries), more complex networks (e.g., Fat-tree topology [62]) by integrating routing, and more complex computing environments (e.g., Cloud [22]). Our long term goal is to develop a network-aware query execution system which can be always highly efficient and adaptive in the presence of different query workloads and network configurations in large-scale distributed scenarios.

Acknowledgments

Part of this work was supported by the Beijing Municipal Science & Technology Commission (Z181100005118016), the National Natural Science Foundation of China under Grant 61874124, Grant 61876173, and the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 799066. The authors also thank the reviewers for their insightful comments and suggestions, which have greatly helped in improving the work.

References

[1] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian, "A comparison of join algorithms for log processing in MapReduce," in *Proc. 2010 ACM SIGMOD Int. Conf. Management of Data*, 2010, pp. 975–986.

[2] L. Cheng, J. Murphy, Q. Liu, C. Hao, and G. Theodoropoulos, "Minimizing network traffic for distributed joins using lightweight locality-aware scheduling," in *European Conf. Parallel Processing*, 2018, pp. 293–305.

[3] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: a scalable and flexible data center network," *Comm. ACM*, vol. 54, no. 3, pp. 95–104, 2011.

[4] O. Polychroniou, W. Zhang, and K. A. Ross, "Distributed joins and data placement for minimal network traffic," *ACM Trans. Database Systems*, vol. 43, no. 3, p. 14, 2018.

[5] O. Polychroniou, R. Sen, and K. A. Ross, "Track join: distributed joins with minimal network traffic," in *Proc. 2014 ACM SIGMOD Int. Conf. Management of Data*, 2014, pp. 1483–1494.

[6] L. Cheng, S. Kotoulas, T. E. Ward, and G. Theodoropoulos, "Robust and skew-resistant parallel joins in shared-nothing systems," in *Proc. 23rd ACM Int. Conf. Information and Knowledge Management*, 2014, pp. 1399–1408.

[7] N. Bruno, Y. Kwon, and M.-C. Wu, "Advanced join strategies for large-scale distributed computation," *Proc. VLDB Endowment*, vol. 7, no. 13, pp. 1484–1495, 2014.

[8] W. Rödiger, T. Muhlbauer, P. Unterbrunner, A. Reiser, A. Kemper, and T. Neumann, "Locality-sensitive operators for parallel main-memory database clusters," in *Proc. IEEE 30th Int. Conf. Data Engineering*, 2014, pp. 592–603.

[9] J. Jiang, S. Ma, B. Li, and B. Li, "Symbiosis: Network-aware task scheduling in data-parallel frameworks," in *Proc. 35th IEEE Int. Conf. Computer Communications*, 2016, pp. 1–9.

[10] Z. Li, H. Shen, and A. Sarker, "A network-aware scheduler in data-parallel clusters for high performance," in *Proc. 18th IEEE/ACM Int. Symp. Cluster, Cloud and Grid Computing*, 2018, pp. 1–10.

[11] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with Varys," in *Proc. Conference of the ACM Special Interest Group on Data Communication*, 2014, pp. 443–454.

[12] Y. Zhao, K. Chen, W. Bai, M. Yu, C. Tian, Y. Geng, Y. Zhang, D. Li, and S. Wang, "RAPIER: Integrating routing and scheduling for coflow-aware data center networks," in *Proc. 34th IEEE Int. Conf. Computer Communications*, 2015, pp. 424–432.

[13] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proc. 11th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2012, pp. 31–36.

[14] Y. Li, S. H.-C. Jiang, H. Tan, C. Zhang, G. Chen, J. Zhou, and F. C. Lau, "Efficient online coflow routing and scheduling," in *Proc. 17th ACM Int. Symp. Mobile Ad Hoc Networking and Computing*, 2016, pp. 161–170.

[15] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[16] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 7–12, 2013.

[17] W. Li, W. Meng, and L. F. Kwok, "A survey on OpenFlow-based Software Defined Networks: Security challenges and countermeasures," *Journal of Network and Computer Applications*, vol. 68, pp. 126–139, 2016.

[18] T. Y. Cheng and X. Jia, "Compressive traffic monitoring in hybrid SDN," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 12, pp. 2731–2743, 2018.

[19] M. Karakus and A. Durresi, "Quality of service (QoS) in software defined networking (SDN): A survey," *Journal of Network and Computer Applications*, vol. 80, pp. 200–218, 2017.

[20] R. Durner, A. Blenk, and W. Kellerer, "Performance study of dynamic QoS management for OpenFlow-enabled SDN switches," in *Proc. 23rd IEEE Int. Symp. Quality of Service*, 2015, pp. 177–182.

[21] Y. Xu, P. Kostamaa, X. Zhou, and L. Chen, "Handling data skew in parallel joins in shared-nothing systems," in *Proc. 2008 ACM SIGMOD Int. Conf. Management of Data*, 2008, pp. 1043–1052.

[22] L. Cheng and S. Kotoulas, "Efficient skew handling for outer joins in a cloud computing environment," *IEEE Trans. Cloud Computing*, vol. 6, no. 2, pp. 558–571, 2018.

[23] S. Bellamkonda, H.-G. Li, U. Jagtap, Y. Zhu, V. Liang, and T. Cruanes, "Adaptive and big data scale parallel execution in Oracle," *Proc. VLDB Endowment*, vol. 6, no. 11, pp. 1102–1113, 2013.

[24] S. Ganguly, P. B. Gibbons, Y. Matias, and A. Silberschatz, "Bifocal sampling for skew-resistant join size estimation," *ACM SIGMOD Record*, vol. 25, no. 2, pp. 271–281, 1996.

[25] S. Kotoulas, J. Urbani, P. Boncz, and P. Mika, "Robust runtime optimization and skew-resistant execution of analytical SPARQL queries on Pig," in *Proc. 11th Int. Semantic Web Conf.*, 2012, pp. 247–262.

[26] M. Chowdhury, S. Khuller, M. Purohit, S. Yang, and J. You, "Near optimal coflow scheduling in networks," in *Proc. 31st ACM Symp. Parallelism in Algorithms and Architectures*, 2019, pp. 123–134.

[27] T. Gonzalez and S. Sahni, "Open shop scheduling to minimize finish time," *Journal of the ACM*, vol. 23, no. 4, pp. 665–679, 1976.

[28] X. Li, H. Wang, S. Yi, and X. Yao, "Receiving-capacity-constrained rapid and fair disaster backup for multiple datacenters in SDN," in *Proc. 2017 IEEE Int. Conf. Communications*, 2017, pp. 1–6.

[29] M. Tanha, D. Sajjadi, R. Ruby, and J. Pan, "Traffic engineering enhancement by progressive migration to SDN," *IEEE Communications Letters*, vol. 22, no. 3, pp. 438–441, 2018.

[30] C.-W. Tsai, H.-H. Cho, T. K. Shih, J.-S. Pan, and J. J. Rodrigues, "Metaheuristics for the deployment of 5G," *IEEE Wireless Communications*, vol. 22, no. 6, pp. 40–46, 2015.

[31] I. BoussaïD, J. Lepagnot, and P. Siarry, "A survey on optimization metaheuristics," *Information Sciences*, vol. 237, pp. 82–117, 2013.

[32] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Advances in Engineering Software*, vol. 95, pp. 51–67, 2016.

[33] X. Chen, L. Cheng, C. Liu, Q. Liu, J. Liu, Y. Mao, and J. Murphy, "A WOA-based optimization approach for task scheduling in cloud computing systems," *IEEE Systems Journal*, 2020.

[34] M. Goudreau, K. Lang, S. Rao, T. Suel, and T. Tsantilas, "Towards efficiency and portability: Programming with the BSP model," in *Proc. 8th ACM Symp. Parallel Algorithms and Architectures*, 1996, pp. 1–12.

[35] T. Van Luong, N. Melab, and E.-G. Talbi, "GPU computing for parallel local search metaheuristic algorithms," *IEEE Trans. Computers*, vol. 62, no. 1, pp. 173–185, 2013.

[36] Q. Jiang, Y. Guo, Z. Yang, and X. Zhou, "A parallel whale optimization algorithm and its implementation on FPGA," in *2020 IEEE Congress on Evolutionary Computation*, 2020, pp. 1–8.

[37] S. K. Gavvala, C. Jatoth, G. Gangadharan, and R. Buyya, "QoS-aware cloud service composition using eagle strategy," *Future Generation Computer Systems*, vol. 90, pp. 273–290, 2019.

[38] B. Lantz and B. O'Connor, "A mininet-based virtual testbed for distributed SDN development," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, 2015, pp. 365–366.

[39] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," in *Proc. Conference of the ACM Special Interest Group on Data Communication*, 2015, pp. 393–406.

[40] Transaction Processing Performance Council, "TPC-H benchmark specification," *available at http://www.tpc.org/tpch/*, 2019.

[41] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proc. 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, no. 19, 2010.

[42] J. Yan and D. Jin, "A lightweight container-based virtual time system for software-defined network emulation," *Journal of Simulation*, vol. 11, no. 3, pp. 253–266, 2017.

[43] "Iperf," 2019. [Online]. Available: https://iperf.fr/

[44] L. Rupprecht, W. Culhane, and P. Pietzuch, "Squirreljoin: network-aware distributed join processing with lazy partitioning," *Proc. VLDB Endowment*, vol. 10, no. 11, pp. 1250–1261, 2017.

[45] D. DeWitt and J. Gray, "Parallel database systems: the future of high performance database systems," *Comm. ACM*, vol. 35, no. 6, pp. 85–98, Jun. 1992.

[46] W. Rödiger, S. Idicula, A. Kemper, and T. Neumann, "Flow-join: Adaptive skew handling for distributed joins over high-speed networks," in *Proc. 32nd IEEE Int. Conf. Data Engineering*, 2016.

[47] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. Vijaykumar, "Shufflewatcher: Shuffle-aware scheduling in multi-tenant mapreduce clusters," in *Proc. 2014 USENIX Annual Technical Conf.*, 2014, pp. 1–13.

[48] V. Jalaparti, P. Bodik, I. Menache, S. Rao, K. Makarychev, and M. Caesar, "Network-aware scheduling for data-parallel jobs: Plan when you can," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 407–420, 2015.

[49] "Software-defined networking: The new norm for networks." 2019. [Online]. Available: https://www.opennetworking.org

[50] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[51] L. Cui, F. R. Yu, and Q. Yan, "When big data meets software-defined networking: SDN for big data and big data for SDN," *IEEE Network*, vol. 30, no. 1, pp. 58–65, 2016.

[52] P. Xiong, H. Hacigumus, and J. F. Naughton, "A software-defined networking based approach for performance management of analytical queries on distributed data stores," in *Proc. 2014 ACM SIGMOD Int. Conf. Management of Data*, 2014, pp. 955–966.

[53] P. Qin, B. Dai, B. Huang, and G. Xu, "Bandwidth-aware scheduling with SDN in Hadoop: A new trend for big data," *IEEE Systems Journal*, vol. 11, no. 4, pp. 2337–2344, 2017.

[54] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Computing Surveys*, vol. 47, no. 4, p. 63, 2015.

[55] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Computing Surveys*, vol. 35, no. 3, pp. 268–308, 2003.

[56] M. Gen and R. Cheng, *Genetic algorithms and engineering optimization*. John Wiley & Sons, 2000, vol. 7.

[57] M. Dorigo and G. Di Caro, "Ant colony optimization: a new meta-heuristic," in *Proc. 1999 Congress on Evolutionary Computation*, vol. 2, 1999, pp. 1470–1477.

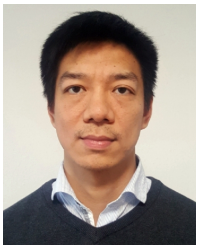
[58] J. Kennedy, "Particle swarm optimization," in *Encyclopedia of machine learning*. Springer, 2011, pp. 760–766.

[59] W.-z. Sun, J.-s. Wang, and X. Wei, "An improved whale optimization algorithm based on different searching paths and perceptual disturbance," *Symmetry*, vol. 10, no. 6, p. 210, 2018.

[60] A. Arunarani, D. Manjula, and V. Sugumaran, "Task scheduling techniques in cloud computing: A literature survey," *Future Generation Computer Systems*, vol. 91, pp. 407–415, 2019.

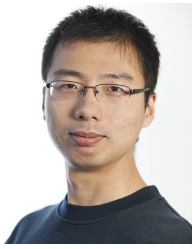
[61] L. Cheng, Y. Wang, Y. Pei, and D. Epema, "A coflow-based co-optimization framework for high-performance data analytics," in *Proc. 46th Int. Conf. Parallel Processing*, 2017, pp. 392–401.

[62] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 63–74, 2008.



Long Cheng is a Professor in the School of Control and Computer Engineering at North China Electric Power University in Beijing. He received the B.E. from Harbin Institute of Technology, China in 2007, M.Sc from University of Duisburg-Essen, Germany in 2010 and Ph.D from National University of Ireland Maynooth in 2014. He was an Assistant Professor at Dublin City University, and a Marie Curie Fellow at University College Dublin. He also has worked at organizations such as Huawei Technologies Germany, IBM Research Dublin, TU Dresden and TU Eindhoven. His research focuses on high-performance data analytics, distributed systems, cloud computing, and process mining.

Ying Wang is an Associate Professor at Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS). He received the B.E. and M.S. degrees in Electrical Engineering from Harbin Institute of Technology, in 2007 and 2009 respectively, and the Ph.D degree of computer science from ICT, CAS, Beijing, in 2014. His research interests includes computer architecture and VLSI design, specifically memory system, on-chip interconnects, resilient and energy-efficient architecture, and machine learning accelerators. He is a member of the IEEE.



Qingzhi Liu is a Lecturer at the Information Technology Group, Wageningen University & Research, The Netherlands. He received a B.E. and a M.Eng. from Xidian University, China in 2005 and 2008 respectively. He received a M.Sc. (with cum laude) and a Ph.D. from Delft University of Technology, The Netherlands in 2011 and 2016 respectively. He was a Postdoctoral Researcher at the System Architecture and Networking Group, Eindhoven University of Technology, The Netherlands from 2016 to 2019. His research interests include Internet of Things and machine learning.

Dick H.J. Epema is Full Professor of Distributed Systems at Delft University of Technology. His research interests are in the areas of resource management in distributed systems and in blockchain technology. He has authored over 140 scientific papers, and has been on numerous program committees in grids, clouds, and P2P computing. He was an associate editor of the IEEE TPDS and IEEE TCC. He was General Co-Chair of EuroPar 2009 and IEEE P2P 2010, and he was General Chair of HPDC 2012 and CCGrid 2013. He was Program Committee Co-Chair of HPDC 2013.

Cheng Liu is an Associate Professor of Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing. He received his B.E. and M.E. degree in Microelectronic Engineering from Harbin Institute of Technology in 2009 and his Ph.D. degree in Computer Engineering from The University of Hong Kong in 2016. His research focuses on FPGA based reconfigurable computing, and domain-specific computing systems.

Ying Mao is an Assistant Professor in the Department of Computer and Information Science at Fordham University in the New York City. He received his Ph.D. in Computer Science from the University of Massachusetts Boston in 2016. His research interests mainly focus on the fields of cloud computing, virtualization, resource management, and data-intensive platforms. He is a member of the IEEE.

John Murphy is a Professor at University College Dublin (UCD). He is an IBM Faculty Fellow, a Fellow of the IET, a Senior Member of the IEEE, a Fellow and Chartered Engineer with Engineers Ireland, and a Fellow of the Irish Computer Society. For many years he held an academic part-time position at the Jet Propulsion Laboratory in Pasadena, and acted as a consultant to the US Department of Justice. He has published over 200 peer-reviewed journal articles or international conference full papers in performance engineering of networks and distributed systems. He has supervised 24 Ph.D. students to completion and been awarded over 30 competitive research grants (in excess of €9.5 million direct costs).