Towards Network-aware Query Execution Systems in Large Datacenters

Long Cheng, *Senior Member, IEEE,* Ying Wang, *Member, IEEE,* Rutvij H. Jhaveri, *Senior Member, IEEE,* Qingle Wang, Ying Mao, *Member, IEEE*

***Abstract*—How to efficiently process concurrent data tasks such as online analytical queries in datacenter environments is still a big challenge for current computing techniques. One of the fundamental reasons is that their task execution normally involves large numbers of distributed data operators, which are always expensive in terms of communication time. To improve the general performance, various advanced approaches on the execution optimization of data operators have been proposed in the past years. However, most of them focus on application-level optimization, such as using data locality scheduling to reduce network traffic. Moreover, few of them has considered the optimization opportunities for concurrent execution of multiple data operators. In this paper, we propose a novel coflow-based scheduling system called CoFlop, which aims to improve network communication time for multiple distributed operators at a query level, and on that basis to lay a solid foundation for the development of a network-aware query execution system in datacenter networks. We introduce the detailed system design of CoFlop and conduct a simulation-based evaluation with large concurrent distributed join operations. Compared to existing methods, the experimental results show that CoFlop can perform better in the presence of different large workloads.**

***Index Terms*—query data operator; coflow scheduling; network communication; performance optimizations; datacenters**

I. Introduction

To support business intelligence (BI) over big data, large computing systems such as datacenters have become the mainstream infrastructures to store and analyze massive datasets [1]. Generally, a typical BI application aims to answer multidimensional analytical queries to provide supports for decision-making like for sales, and the performance of which is always time sensitive. However, how to perform complex queries in a highly efficient way in a distributed environment is still challenging current techniques [2]. One of the main reasons is that an analytical query, such as in an online analytical processing (OLAP) scenario as shown in Fig. 1(a), always contains a set of distributed data operators (e.g., join and aggregation). These operators often involve a large amount of data transmission over networks, which is expensive, in terms of network resource consumption and network communication time.

L. Cheng and Q. Wang are with the School of Control and Computer Engineering, North China Electric Power University, Beijing 102206, China. E-mail: lcheng@ncepu.edu.cn, wqle519@gmail.com

Y. Wang is with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China. E-mail: wangying2009@ict.ac.cn

Rutvij H. Jhaveri (Corresponding author) is with the Department of Computer Science and Engineering, School of Technology, Pandit Deendayal Energy University, India. E-mail: rutvij.jhaveri@sot.pdpu.ac.in

Y. Mao is with the Department of Computer and Information Science at Fordham University in the New York City. E-mail: ymao41@fordham.edu

Data locality scheduling is an effective way to improve communication time for the execution of data operators. As illustrated in Fig. 1(b), data chunks with the same join key should be allocated to the same computing node in a distributed join, and the locality scheduling decides which chunk goes to which node. The output of the scheduling process is a data distribution plan, which will be implemented by the underlying execution system. Normally, the less network traffic in data size a plan brings, the better the plan will be. This is because a data operator can get direct performance benefits from network traffic reduction in both low-end and high-end platforms [3].

Currently, various locality scheduling approaches have been proposed and have shown to be effective on reducing network traffic for data operator executions [4]. However, their network communication time is actually suboptimal. The main reason is that locality scheduling has not considered the performance of data flows over networks, which is associated with communication time directly. Additionally, current approaches generally perform locality scheduling for each data operator individually, and additional performance gains have been ignored for the scheduling of concurrent data operators, which is common in complex analytical query scenarios.

To optimize communication performance for applications in datacenters, data flow scheduling aims to minimize flow completion times based on prior knowledge of flows [5]. Specifically, coflow scheduling is one of the state-of-the-art techniques which can minimize the communication time for a group of parallel flows which are related to each other [6]. In fact, we can model data flows generated by the distribution plan of a data operator as a coflow [6]. For instance, the two parallel data flows of the join in Fig. 1 can be treated as a coflow, and thus coflow scheduling can be applied to optimize its performance.

With the above two scheduling techniques, for the execution of distributed data operators, we can first perform locality scheduling to minimize their network traffic and then use coflow scheduling to optimize the generated data flows. However, such kind of processing will still lead to a suboptimal performance from a system point of view. The main reason is that locality scheduling is only aware of the size of the data to be transferred over networks rather than the possible network communication time. For a case with 10 computing nodes and 150 data partitions, the number of possible data distribution plans is 10^{150}, and it will be hard to search which plan will be

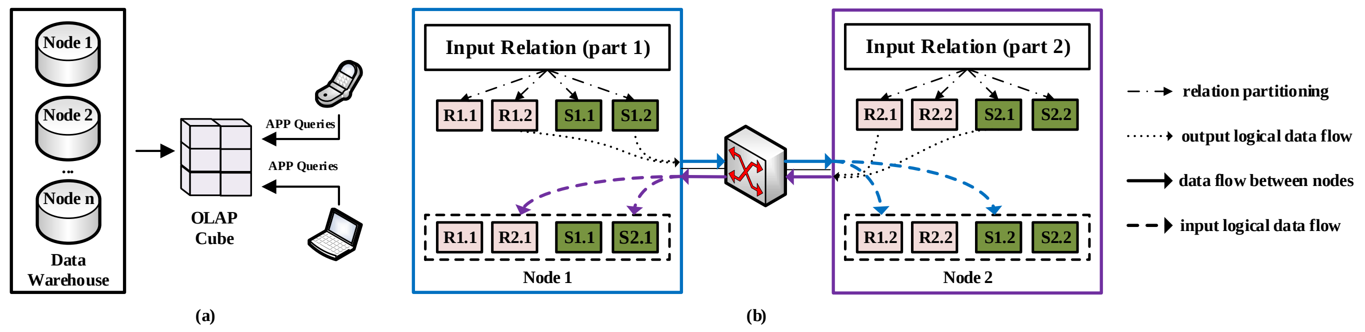


Fig. 1. (a) An OLAP application scenario with queries. The pre-calculated data (such as using joins and aggregations) is stored in an OLAP cube or OLAP database. (b) A simple example of a distributed join between two relations R and S over two computing nodes. Here, Ri.j (or Si.j) means the tuple set (or chunk) of the i-th part of relation R (or S) with the key j. The data distribution plan generated by locality scheduling is to move R1.2 (and S1.2) to Node 2 and R2.1 (and S2.1) to Node 1. Following the plan, the two parallel data flows from Node 1 to 2 and Node 2 to 1 can be abstracted as a coflow.

optimal in communication time following a coflow scheduling.

To optimize the communication performance for the executions of concurrent operators in datacenter networks and lay a practical foundation for the development of a network-aware query execution system, in this paper, we introduce a novel coflow-based scheduling approach, called CoFlop (**CoFl**ow-aware **op**timization for concurrent data operator execution). Specifically, we have tried to seamlessly combine data locality scheduling and coflow scheduling together in concurrent data operator execution. Different from existing scheduling works which have considered both data locality and underlying networks in parallel data processing [7], the optimization of CoFlop is driven by the possible network communication time directly (i.e., based on coflows).

In general, the main contributions of this work are summarized as follows:

- We demonstrate that communication performance for the execution of multiple distributed operators can be further improved through the joint optimization of locality scheduling and coflow scheduling.
- We introduce CoFlop with its detailed system design and optimization model. To achieve the best possible communication performance for different workloads, we also present an effective implementation for CoFlop in datacenter environments.
- We conduct a simulation-based evaluation of CoFlop with large concurrent distributed join operations from benchmarks. Compared to existing approaches, the experimental results show that CoFlop can always perform better in terms of network communication time.

The rest of this manuscript is structured as follows. The background of this work with a motivating example are given in Section II. In Section III, we report the related work. We present the detailed design of CoFlop and its implementation in Section IV. We carry out extensive evaluation of CoFlop in Section V. We present the limitation and further research directions in Section VI and conclude this paper in Section VII.

II. Background and Motivation

In this section, we introduce the execution of data operators and coflow scheduling in datacenters. Moreover, we provide a motivating example for CoFlop as proposed in this work.

A. Execution of Distributed Data Operators

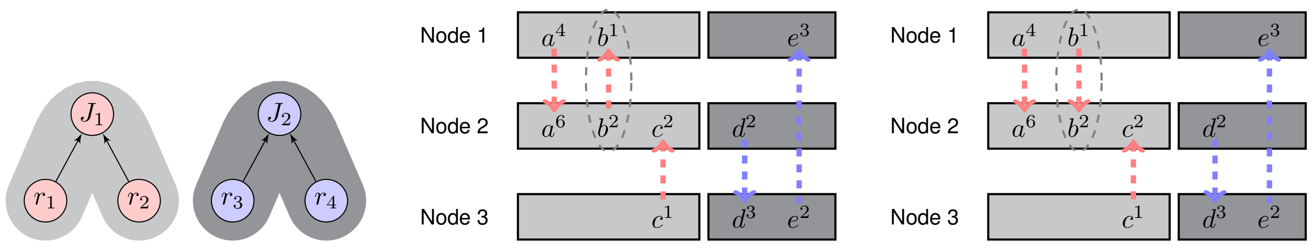
Similar to query execution in database systems, the plan for an analytical query is represented as a tree, in which internal nodes are operators and leaves are input relations. For the purpose of this work, we focus on a set of joins which can be executed in parallel in the tree structure. Specifically, following the example in Fig. 1, we take a given workload with two concurrent joins J_1 and J_2 over three-node system as a sample, to present the details of the relevant scheduling methods in this work. The proposed technique will applicable to other data operators such as outer joins and aggregations.

As demonstrated in Fig. 2, different data distribution plans can be generated using different locality scheduling strategies. There, we use join keys to represent data tuples respectively, and use dashed arrows to indicate the movements of tuples over networks. In detail, from LS_1 and LS_2, it can be seen that we can either schedule to move b^1 from Node 1 to 2 or b^2 from Node 2 to 1 to complete the two joins. This because that both the movements guarantee that tuples with the same join key are allocated on the same node. If we quantify the network cost by counting the number of transferred tuples over networks, then the cost of LS_1 and LS_2 will be 11 and 10, respectively. Since the plan LS_2 transfers less data (e.g., in Bytes) than LS_1, the underlying data system will choose LS_2 as an optimal solution and execute the joins following the plan.

B. Basics of Coflow Scheduling

The coflow abstraction [6] is used to describe a set of parallel data flows. Different from a general case, data flows in coflow share the same performance goal. Specifically, coflow scheduling aims to allocate network resources to shorten the completion time of coflows [8]. For a distributed operator like joins, to optimize its network performance, we need to perform the scheduling for the relevant parallel data flows in the scope of coflow. The reason is that the network communication time of the data operator depends on the completion of its entire coflow rather than individual flows (i.e., coflow completion time (CCT) [9]), or in other words, the time cost that all the transferred data reaches the destination nodes.

The metric average CCT is widely used to quantify the performance for multiple coflows, and several efficient solutions



(a) Two concurrent joins (b) LS_1 (suboptimal) (c) LS_2 (optimal)

Fig. 2. Two different data Locality Scheduling (LS) for the executions of the two concurrent distributed joins (J_1 and J_2) in a three computing node system. Each of the joins contains two input relations (i.e., r_i). The letters a, b and c represent the join keys in the relations r_1 and r_2 while d and e are for the relations r_3 and r_4. The superscript of each key means the number of the tuples having the key.

such as Varys [10] and Aalo [11] have been proposed on multi-coflow scheduling. Since data flows generated by concurrent data operators can be abstracted as multiple individual coflows, we can use existing multi-coflow scheduling techniques to optimize communication time for concurrent operators.

As shown in Fig. 3, the two coflows generated by the two joins in Fig. 2 can be scheduled in different ways. We assume that the time to transfer one tuple is 1. For the plan LS_2, as illustrated in Fig. 3(a) and 3(b), a multi-coflow scheduling can indeed improve the communication performance, compared to an approach which schedules each coflow independently. Moreover, Fig. 3(c) shows that performing multi-coflow scheduling actually can further optimize network performance, i.e., reducing the average CCT from 4 to 3.5, compared to the method in Fig. 3(b).

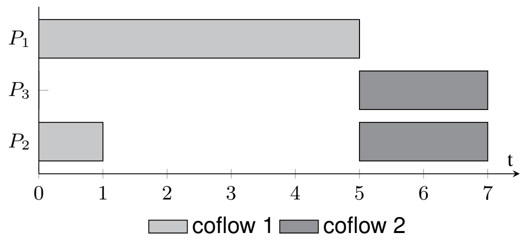
C. A Motivating Example of CoFlop

Like the case in Fig. 2, almost all current works on data operator executions focus on optimizing network traffic in data size while ignoring the impacts of underlying data flow controls [12], [4], [13]. In fact, following an optimal data locality scheduling, although we can use multi-coflow scheduling to optimize communication time, this kind of processing will be suboptimal. As demonstrated in Fig. 3(d), for the suboptimal locality scheduling plan LS_1 in Fig. 2, it can even achieve better average CCT than the optimal plan LS_2 (i.e., 3 instead of 3.5). Therefore, how should we perform scheduling for concurrent data operators in a distributed system, if we want to maximize their communication performance. The conclusion is that, the joint optimization of data locality scheduling and multi-coflow scheduling should be considered. For instance, we can achieve minimal average CCT if we combine the scheduling in Fig. 2(b) and in Fig. 3(d) together, which motivates the design of the proposed CoFlop.

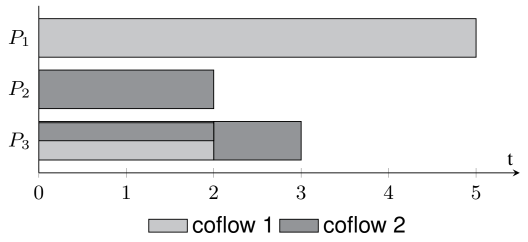
III. Related Work

Query execution systems (e.g., OLAP systems [2]) enables users to easily extract business intelligence information from the data with different points of view. However, performing complex queries in distributed environments in a highly performant way is always challenging.

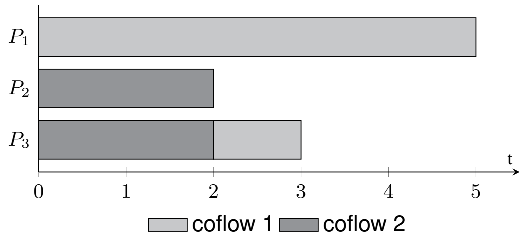
Generally, to improve the performance of query executions, as shown in Table I, we can summarize the main features of some typical works from five perspectives. Specifically,



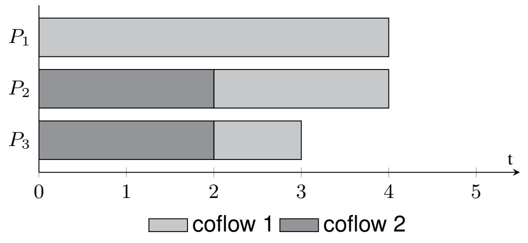
(a) sequential scheduling for LS_2



(b) per-flow fairness flow scheduling for LS_2



(c) optimal coflow scheduling for LS_2



(d) optimal coflow scheduling for LS_1

Fig. 3. Different flow scheduling approaches for the data flows generated by the locality scheduling presented in Fig. 2.

some studies focus on reducing network traffic with a data size-aware approach like using data locality scheduling, while some aim to balance the utilization of networks or optimize communication time through the scheduling of generated data flows such as using flow or coflow scheduling. Moreover, some of them are mainly designed to handle a single data operator and some can support the parallelism of multiple operators. In comparison to all of them, CoFlop can use the potential communication time (generated by an optimal coflow scheduling) to guide the optimization process of data locality scheduling. Additionally, it is designed to support multiple data operators in parallel at a query-level on the basis of multi-coflow scheduling.

TABLE I
MAIN FEATURES OF TYPICAL APPROACHES IN LITERATURE

Ref.	**Main features**				
	data size-aware	data flow-aware	comm. time-aware	operator-level support	query-level support
[13]	✓	✗	✗	✓	✗
[7]	✓	✓	✗	✓	✓
[14]	✗	✓	✓	✓	✓
[15]	✓	✓	✓	✓	✗
CoFlop	✓	✓	✓	✓	✓

For data size-aware approaches, the main motivation is that the network traffic incurred in the execution of distributed data operators like joins always impact the general query performance [16]. Currently, different strategies and methods have been introduced to optimize distributed joins through the optimization of the size of data transferred over networks [4], [13], [17], [18], [19]. However, almost all of them just focus on the optimization of data locality for individual joins, with the targets for data skew handling and network traffic reduction, and have ignored the underlying parallel data communications. A typical example is the track-join [13], which has adopted a four-phase locality scheduling to search all the possible opportunities to reduce network traffic. Different from that, in this paper, we focus on handling concurrent operators. Moreover, we try to minimize their communication time from both a data locality and data communication angle.

In relation to improving communication performance for distributed data operators, there are also several flow scheduling approaches in current literature [7], [20], [21], which have considered both data locality and underlying parallel data flows. Although CoFlop also adopts a data flow-aware method, its core technique has significant differences compared to existing approaches. Generally, CoFlop is communication time aware (from coflow scheduling), and it performs locality scheduling on that basis. In comparison, communication time is not aware for other methods while they focus on balancing the utilization of network links and improving network resource utilizations. For example, the method [7] aims to make sure that cross-node network load is not higher than a threshold. Obviously, such a method can avoid network congestions, and the freed up bandwidth can be used by other running tasks to improve the whole system performance.

In terms of optimizing network communication time, a lot of works have been done on data flow scheduling before the introduction of the concept of coflow [22]. However, since the approaches are unaware of coflows, they sometimes even reduce the performance for data applications [23]. In comparison, coflows can capture application-level semantics, and thus can be used to accelerate job completion [24].

Current coflow scheduling based methods mainly focus on optimizing average CCT or deadlines for multiple concurrent jobs in datacenter environments [10], [25]. To handle more complex workloads and networks, various advanced approaches and implementations, such as online coflow scheduling [11], [14] and coflow routing [9], [23], have been developed in the past years. Since data flows generated by distributed operators can be modelled as coflows, all the relevant scheduling approaches can be used to accelerate the execution of data operators. However, due to the isolation from locality optimization, as we have shown in our motivating example, applying coflow scheduling directly to distributed operators will result in suboptimal performance.

Additionally, compared to some latest works which focus on performing locality scheduling on the basis of coflow scheduling for each data operator individually [15], [26], this work aims to further push the boundary of network-aware data processing techniques to query-level rather than operator-level. Namely, the proposed approach supports the parallelism of multiple operators through the scheduling of concurrent distributed data operators.

IV. THE DESIGN OF COFLOP

In this section, we introduce the detailed system architecture and implementation of CoFlop in a datacenter system.

A. System Architecture and Model

The proposed CoFlop aims to optimize the average CCT for concurrent operators in datacenters by coordinating the data locality scheduling and coflow scheduling. Given an input workload and a network configuration, CoFlop determines the destinations of tuples, the time point to start and with how much network bandwidth to transfer them. The target of the optimization is to minimize the average CCT for all input data operators in the system.

Fig. 4 shows the system architecture of CoFlop. There, the system is decomposed into two layers. For the concurrent data operators to be executed in a query, the control layer is responsible to generate an execution plan including the detailed locality and coflow scheduling information. Following the plan, the execution layer will perform the transmission of the involved data over the networks in a similar way demonstrated in Fig. 1(b) in the computing system. As the cornerstone of the system, the CoFlop scheduler in the control layer aims to implement the joint optimization between the locality scheduling and coflow scheduling as illustrated in Fig. 2(b) and Fig. 3(d). In the optimization process, the CoFlop scheduler will collect data information such as data chunk sizes before performing locality scheduling from a system module in the execution layer (e.g., data information manager). In the meantime, in a data center environment, it will also get network information such as the available bandwidth of each communication link from the execution layer (e.g., network

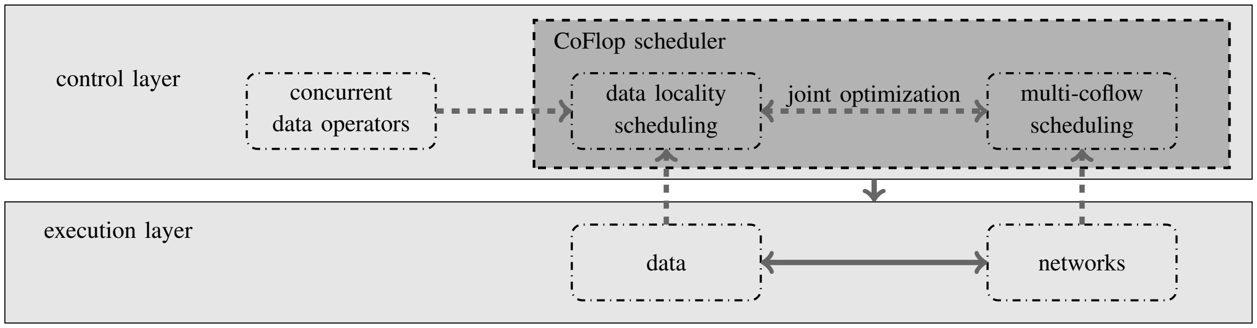


Fig. 4. The logical architecture of CoFlop in datacenter networks. The execution flow is indicated by solid lines and dashed lines means the metric flow.

information manager). With the above system architecture, for a set of joins which can be executed in parallel, the scheduled plan generated by CoFlop will be fixed. The process is online and CoFlop will schedule another set of joins once the current one is done. In this process, the coflow dependencies will be tracked by the logical execution plan of the query, which is in the form of a tree structure as mentioned.

To illustrate how CoFlop works in details, we use a system model as follows. There are M computing nodes and N concurrent joins. Moreover, the same as the current works [10], [27], we set the bandwidth between nodes to B and assume that the bandwidth competition during data redistribution only appears at the network ports in a non-blocking switch scenario. In the meantime, we assume that CoFlop has the required data information for each operator already, such as that a histogram with all join key statistics has been built in the system. For the most common condition, all the tuples are hash-partitioned into P chunks on each node, and the hash value of the p-th partition is p. For the operator k, the number of tuples in the p-th partition at node j is $s_{pj}^{(k)}$. For a distributed join, data partitions in the system with the same hash value should be allocated to the same node. Here, we use decision variables $x_{jp}^{(k)} \in \{0,1\}$ to indicate that data partition p in join k is assigned to node j or not, and $x_{jp}^{(k)} = 1$ means yes while $x_{jp}^{(k)} = 0$ means not. In the execution phase, each join i will generate a coflow $D^{(i)}$. Here, $d_{ij}^{(k)}$ is used to represent the data flow size from node i to j in the coflow, $b_{ij}^{(k)}(t)$ is the bandwidth assigned at time t, and C_k is the complete time of the coflow. For readability, we have summarized the notations we used in Table II.

Based on the above model, we formulate our optimization problem on minimizing the average CCT of N concurrent join operators as the problem $\mathcal{P}$ as follows.

$$\mathcal{P}: \quad min \quad \frac{1}{N}\sum_{k=1}^{N} C_k \tag{1}$$

subject to:

$$\int_0^{C_k} b_{ij}^{(k)}(t)dt = d_{ij}^{(k)} \quad \forall k, i \neq j \tag{1a}$$

$$d_{ij}^{(k)} = \sum_{p=1}^{P} s_{pi}^{(k)} x_{jp}^{(k)} \quad \forall k, i \neq j \tag{1b}$$

$$\sum_{j=1}^{M} x_{jp}^{(k)} = 1 \quad \forall k, p \tag{1c}$$

$$x_{jp}^{(k)} \in \{0,1\} \quad \forall j, k, p \tag{1d}$$

$$\sum_{k=1}^{N}\sum_{i=1}^{M} b_{ij}^{(k)}(t) \leq B \quad \forall t, j \neq i \tag{1e}$$

$$\sum_{k=1}^{N}\sum_{j=1}^{M} b_{ij}^{(k)}(t) \leq B \quad \forall t, i \neq j \tag{1f}$$

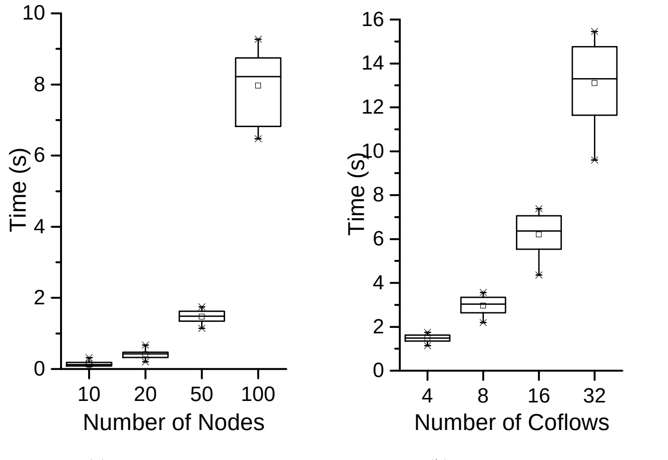
$$0 \leq b_{ij}^{(k)}(t) \leq B \quad \forall k, t, i, j \tag{1g}$$

Here, the formula (1) is aiming to minimize the average CCT of N concurrent join operators which appear in an execution phase of a query, C_k is the coflow time of the k-th join, and N is the number of the joins. Moreover, the equality (1a) and (1b) calculate the size of the data flow from node i to j in operator k from an application and a network angle respectively. The equality (1c) guarantees that the tuples which have the same hash value are co-located on the same node after data distribution. The inequalities (1e) and (1f) are the capacity constraints on ingress and egress ports for each node respectively. It can be seen that our approach is different from conventional optimization methods, as it contains not only the application-level optimization on data locality scheduling ($x_{jp}^{(k)}$), but also the network-level optimization ($b_{ij}^{(k)}(t)$), and seamlessly joint them together.

Solve the above problem $\mathcal{P}$ directly actually is hard. The reason is that the problem is nonlinear with binary integer variables. In fact, as proved in [10], the problem on minimizing the average CCT for multiple coflows is NP-hard even when data flow related information, such as the source and destination node with the data volume, is known beforehand. For the problem $\mathcal{P}$, we do not have any data flow information before our scheduling. Therefore, our joint optimization problem is actually harder than a multiple coflow scheduling problem, which means that our joint optimization problem is NP-hard.

TABLE II
NOTATIONS USED IN SYSTEM MODEL

Notation	Meaning
M	number of computing nodes
N	number of concurrent operators
B	bandwidth of ingress and egress ports of each node
P	number of data chunks of each relation on each node
D^k	the k-th coflow by data operator k
C_k	CCT for coflow D^k
C	total CCT for all coflows
$x_{jp}^{(k)}$	decision variable indicating the p-th chunk is assigned to node j or not in operator k
$s_{pj}^{(k)}$	size of the p-th data chunk of node j which is to be redistributed in data operator k
$f_{ij}^{(k)}$	data flow from node i to j in coflow D^k
$b_{ij}^{(k)}(t)$	transmission rate assigned to $f_{ij}^{(k)}$ at time t
$d_{ij}^{(k)}$	volume of data flow $f_{ij}^{(k)}$



(a) with 4 coflows (b) over 50 nodes

Fig. 5. The overhead of our implementation by varying the number of nodes and coflows.

B. Implementation Design

We are aiming to execute concurrent data operators as soon as possible when they arrive at a system. Therefore, an online algorithm is required for the scheduling. Given the complexity of the above problem of CoFlop, rather than introducing an optimal algorithm, we are aiming to provide an approximately optimal solution with an efficient heuristics, which can perform the scheduling in a quick time. Similar to many current approaches [9], [23], we first compute the minimum CCT for a single data operator, by considering both the locality scheduling and coflow scheduling, and then on that basis to tackle the problem of multiple concurrent data operators.

Following the above guidance, we assume that the minimum CCT C'_k of a single data operator k in the data system is known. Then, based on the inequalities (1e) and (1f) in the above model, there are:

$$\sum_{i=1}^{M}\int_0^{C'_k} b_{ij}^{(k)}(t)dt \le BC'_k \quad \forall j \neq i$$

$$\sum_{j=1}^{M}\int_0^{C'_k} b_{ij}^{(k)}(t)dt \le BC'_k \quad \forall i \neq j$$

Following the $d_{ij}^{(k)}$ in equalities (1a) and (1b), $\sum_{p=1}^{P} s_{pi}^{(k)} x_{jp}^{(k)}$ can be then used to replace $\int_0^{C'_k} b_{ij}^{(k)}(t)dt$ in the above two inequalities. In this case, we get the following two constraints:

$$\frac{1}{B}\sum_{i=1}^{M}\sum_{p=1}^{P} s_{pi}^{(k)} x_{jp}^{(k)} \le C'_k \quad \forall j \neq i \tag{2a}$$

$$\frac{1}{B}\sum_{j=1}^{M}\sum_{p=1}^{P} s_{pi}^{(k)} x_{jp}^{(k)} \le C'_k \quad \forall i \neq j \tag{2b}$$

Here, B is the bandwidth of ingress and egress ports of each node, which will be a constant for a given network. In the meantime, each $s_{pi}^{(k)}$ is also known for a given system. In this condition, we will only have the binary integer variables in the problem. Therefore, the optimization problem on minimizing C'_k is actually a mixed integer linear programming (MILP) problem. In this case, we can get an optimized solution $x_{jp}^{(k)}$ on the data locality scheduling by using an optimizer. In our evaluation in Section V, we have used the Gurobi version 8 in our implementations.

With the relevant locality information for all data operators, the detailed information of data flow in each coflow can then be obtained. In this case, we have the required coflow information for all the data operators and then can use exsiting heuristics to implement the multi-coflow scheduling. Specifically, we choose the SEBF (smallest effective bottleneck first) method as proposed in Varys [10] in our implementation. The approach has been shown to be very efficient compared to other heuristics in various experiments. This also makes our joint optimization be able to run on the top of existing coflow systems such as Varys and Aalo as well as the *CoflowSim* simulator [10] directly.

C. Discussion of Implementations

As mentioned above, we have used an optimizer (i.e., Gurobi) in our implementation. In fact, based on personal programming requirements, we can also use many other implementations (e.g., meta-heuristics [28] and customized heuristics [29]) to get approximate optimal solutions for our optimization problem if required. As a reference for the details of our implementation, we have provided the source code at https://github.com/longcheng11/CoFlop-Solver.

In relation to the overhead of for the presented implementation, the complexity of our optimization problem is generally related to three numbers: the number of nodes, data chunks, and joins (coflows). It is obvious that the problem solving time would increase with increasing these numbers. To give some details, we have evaluated the overhead of our method for the evaluation we have conducted in Section V. Since the chunk numbers are set to be proportional to the node numbers in our tests and all the results have presented a similar characteristic, we just report the two typical cases in Fig. 5. From the results, we can see that the overhead of our implementation is quite small. Although the cost is increasing with increasing the number of nodes and coflows, the increase is still located in an acceptance range. Specifically, for the two most complex cases, the overhead is around 8 sec when the number of nodes reaches 100, and is about 13 sec for the case with 32 coflows.

It is assumed that the overhead of our implementation will become obvious for large-scale problems, such as for the case with hundreds of computing nodes. To remedy this issue, we can use a powerful machine to reduce the overhead in practice. In the meantime, we can use the parallel version of Gurobi, and also assign execution parameters to make sure that a solution can be returned in reasonable time. On the other hand, our work limits the research on modeling and providing an optimized execution plan for concurrent data operators, rather than proposing a perfect system solution which is always optimal in the presence of different workloads and computing infrastructures. Generally, the optimizer of a query system will evaluate the potential performance benefits and the possible overhead at runtime, and then makes a decision on using the scheduling or not.

V. Evaluation

In this section, we present the evaluation of the proposed CoFlop through simulation-based experiments with large distributed join workloads.

A. Experimental Framework

We have performed a detailed replay of coflow traces over the *CoflowSim* simulator [10]. The simulator is one of most popular tools to evaluate the performance of coflow related schedulers (e.g., [30], [31]. We use SEBF for the multi-coflow scheduling, and compare the average CCTs of the traces generated by CoFlop with the following two schemes.

- **Hash**: The same as the most commonly used hash-based approach [12], in which the data distribution follows the hash values of the input tuples. Namely, after the hash partitioning, each data chunk is assigned to a node based on its responsible hash value.
- **Locality**: Similar to the locality-aware scheduling described in the work [3], [13], each partitioned chunk is allocated to the node with the maximum number of tuples with the same join key, through examining all the possible destinations for each chunk. This approach is widely used in databases to optimize network traffic.

For a given data distribution plan generated by each of the above two schemes, coflow scheduling is used to optimize the network communication time respectively. In this condition, the former scheme represents the approaches which only focus on network-layer optimization. In the meantime, the latter one represents the methods which perform optimization for both application and network-layer scheduling, but in an independent way.

The workloads used in our experiments are generated by TPC-H benchmark [32], which consists of a suite of business oriented ad-hoc queries and is widely used to evaluate the capability of a system to process queries. The same as many current works, we choose the joins between two relations *customer* and *order*. We set the scaling factor of the benchmark to 600 (with 90 million and 900 million tuples for the two relations) and generate 600 files for each relation. To generate different workloads, we select different number of files (referred to as *nf*) from the two relations in an ordered way, and treat each relevant join as an independent operator. As a default, we use four joins in our tests by setting *nf* to 50, 100, 200 and 400 respectively. To get the volume of each data flow in a simple way, the size of each tuple is set to a fixed value. We have set the value to 100 Bytes in our evaluation, then we can count the number of transferred tuples to get the flow size. This also leads to the size of the input data for the four default joins is around 8, 16, 32 and 64GB, respectively.

The join keys in the generated data by TPC-H is generally uniformly distributed, which will result in the size of data chunks being closed to each other in a hash partitioning scenario. To make the workloads more complex, we vary the data distribution of the table *order* over each node: for each hash value, we set the size of each data chunk to a value following Zipfian distribution (referred to as *zipf*). The distribution is based on node id and starts with the node with the largest part. We set *zipf* to 0.8 as default. Moreover, we partition data tuples on each node with a simple hash function $f(k) = k \ mod \ P$. Here, P is set to a number which is 15 times to the number of computing nodes M for each test.

We focus on the metric CCT in this section, because we are aiming to minimize the average CCT of multiple joins in a query execution in this paper. Besides that, in our results we also report the metric network traffic (in GB), which is always highly related to data locality scheduling. Moreover, coflow scheduling used by all the evaluated approaches replies on solving optimization problems, the results of which will be not deterministic. Therefore, we run each test five times and record the average values.

B. Performance Results

Varying number of nodes. To compare the performance of Hash, Locality and CoFlop in different computing environments, we vary the number of used computing nodes from 10 to 100 for the four default joins. The results of the CCT and network traffic of each join in each approach are presented in Fig. 6. It can be observed that Locality always transfers less data over networks than other methods. In the meantime, CoFlop performs better on network traffic than Hash. The main reason is that Locality is designed to optimize network traffic for the executions of joins. In comparison, CoFlop mainly focuses on its performance model and can only explore data locality based on the communication time constraints, while Hash does not consider data locality and just redistributes all data chunks over all the computing nodes. With growing the number of computing nodes, the network traffic of each join in each approach increases slightly in general due to the data locality decreases.

For the CCTs, we can see that Locality generally performs the worst in all the cases, although it has the least network traffic. One possible reason is that its tuples have not been efficiently spread out over networks, and thus, the network

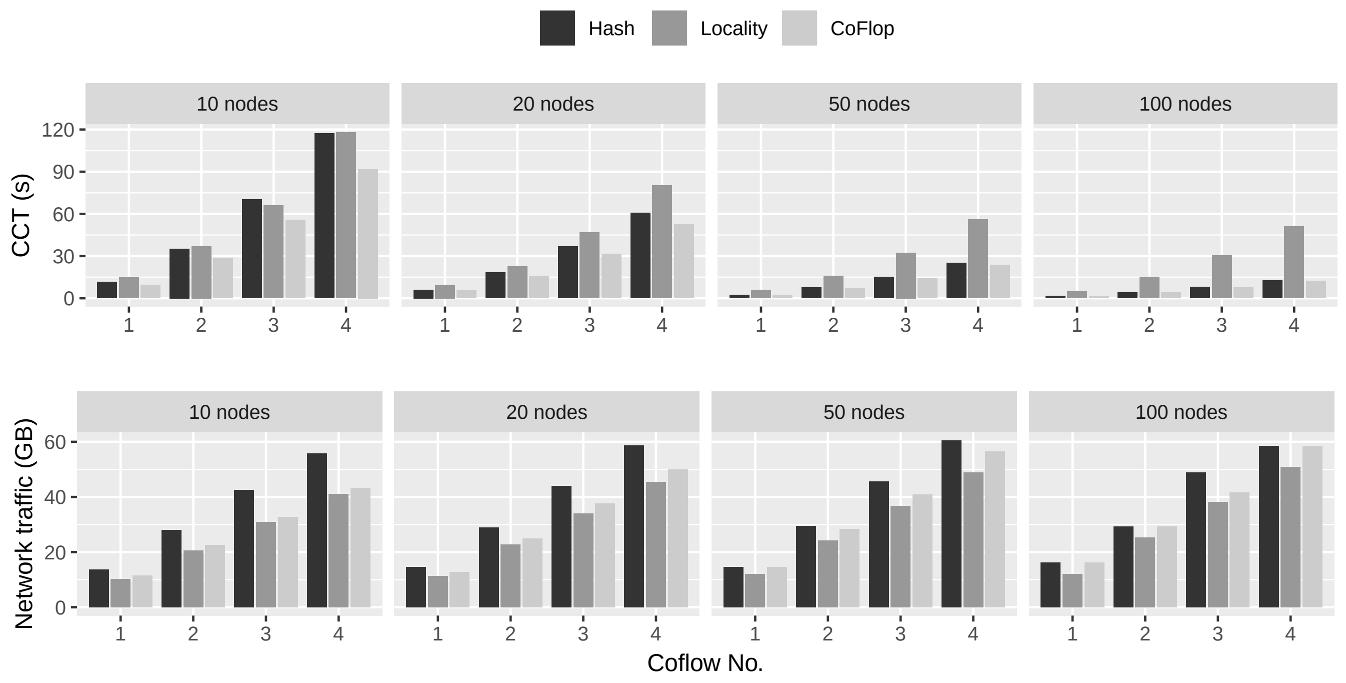


Fig. 6. Performance comparison of CCT and network traffic with varying the number of computing nodes, with 4 concurrent distributed joins (coflows) and *zipf*=1.

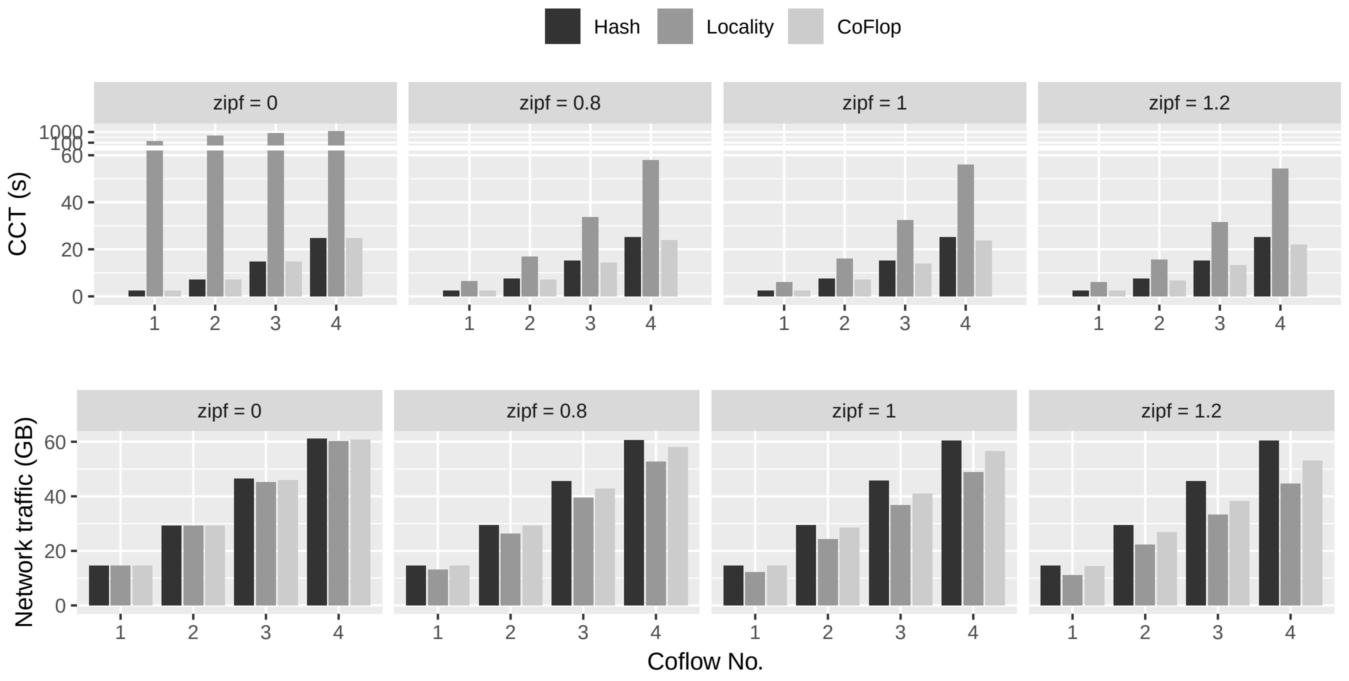


Fig. 7. Performance comparison of CCT and network traffic with varying the value of Zipf factor, with 4 concurrent distributed joins (coflows) and 50 nodes.

resources cannot be utilized in an efficient way. In comparison, CoFlop always performs the best. Specifically, its average CCT performs about $1.26\times$ better than the other two approaches when using 10 nodes, demonstrating the advantages of our approach on combining the optimizations of data locality and communications together. Moreover, we can see that all the CCTs decrease obviously with increasing the number of nodes. This is because the available network resources increase.

Varying value of zipf. To evaluate the performance of each approach in the presence of different workloads, by increasing *zipf* from 0 to 1.2 over 50 computing nodes, we report the results of network traffic and CCTs in Fig. 7. Similar to the

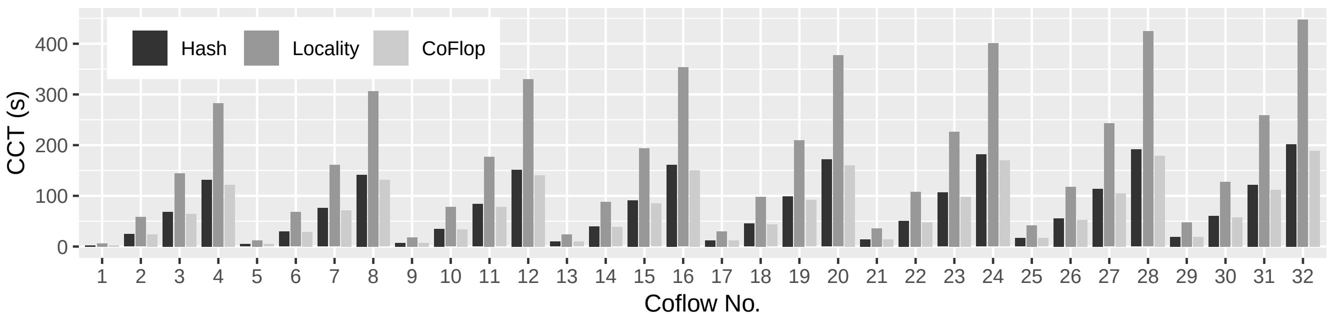


Fig. 8. The performance comparison of CCT in the case with 32 concurrent distributed joins (coflows), with 50 nodes and *zipf*=1.

above results, Locality still performs the best on network traffic reduction, but always the worst in terms of CCTs. Specifically, in the case of uniform distribution (i.e., *zipf*=0), each of its CCTs is much larger than Hash and CoFlop. This is because that the data chunk size is the same for a given hash value on each node. As a default, Locality sets the last node as the node having the largest number of tuples. In this condition, all the tuples are flushed to the final node and this results in serious network congestion and long communication time. In contrast, Hash and CoFlop still distribute tuples over all the computing nodes, and CoFlop can perform the best with its coflow-aware optimization. With increasing *zipf*, Locality shows an obvious reduction on network traffic while Hash and CoFlop do not change much. The reason is that the largest data chunk of each hash value becomes larger, and Locality achieves a higher data locality on that basis. The CCTs show the similar characteristics as the network traffic. Specifically, for the case *zipf* equals 1.2, CoFlop performs $1.14\times$ and $2.42\times$ better on the average CCT, compared to Hash and Locality respectively.

Varying number of concurrent distributed joins. To evaluate the performance of CoFlop in the presence of larger workloads, we increase the number of concurrent joins by duplicating the default workload, from 4 joins to 32 joins. In this process, we renumber the joins in a sequential way by keeping their original order. For the network traffic, because Locality still performs the best and Hash performs the worst, we only report the CCTs here. Specifically, since the results of 4, 8 and 16 concurrent joins demonstrate the same characteristics as the case with 32 joins, we only present the results for the 32 joins in Fig. 8. There, we can clearly observe that small joins are processed prior to large joins. This is done by the underlying coflow scheduling with the SEBF strategy for reducing the average CCT. Moreover, CoFlop performs better than Hash and Locality for each join, and thus its can achieve better performance on average CCT, which highlights again the performance advantages of our approach on network communication time.

VI. Limitation and further research directions

Based on the above experimental results, we believe that we have proposed a new solution for communication optimization for the execution of data queries in large datacenters. The method has laid a solid foundation for the development of a network-aware query execution system. Namely, we can strength the capability of query executions by combining data locality scheduling and data flow scheduling in a seamless way, based on the existing coflow scheduling techniques. In the meantime, we are also aware of two key points for the further improvement of our approach as follows.

- From the perspective of networks, it will be challenging for our method to handle a dynamic datacenter network or even a more special case with the breakdown of some links [33]. The main reason is that the execution of large number of data operators is time cost and the available bandwidth resources at some points would not meet the requirements of a scheduled plan. A possible solution is to monitor the networks and reserve the bandwidth for query applications using software-defined networking switches, which are actually commonly used in current data centers. In this case, we can feed the reserved bandwidth to CoFlop and guarantee that the optimal CCT can be achieved during query executions.
- From the perspective of workloads, it will be worth to apply advanced data partitioning scheme for each query workload in our framework. In this work, we have only used a simple hash-based method to partition the data on each node for all query data operators, which is kind of coarse-grained and thus the final CCT would be not the best possible time. However, perform scheduling for each individually key is complex and time consuming. As a representative of the current self-adaptive methods, machine learning (e.g., deep reinforcement learning) based data partitioning approaches have been applied for query optimization [34]. Therefore, it will be interesting to see whether combining the relevant techniques with CoFlop can further improve the query performance.

VII. Conclusion

In this work, we propose a network-aware scheduling system CoFlop to optimize the communication performance for query executions with distributed data operators in datacenter networks. Specifically, CoFlop can optimize the data locality assignment for input workloads and optimize the network communication time through multiple coflow scheduling. We have presented the system architecture and performance model of CoFlop, and our experiments have shown that the proposed CoFlop can indeed accelerate network communications for queries with large number of distributed joins, compared to the current approaches.

As future works, we plan to extend our approach to handle the complex cases as we have described in the above research directions. Moreover, we will also investigate decentralized approaches to scale up our scheduling process in large distributed scenarios. The long term goal of our research is to develop a highly efficient network-aware analytical query execution system in large datacenter environments.

REFERENCES

[1] N. Bruno, S. Jain, and J. Zhou, "Continuous cloud-scale query optimization and processing," *Proc. VLDB Endowment*, vol. 6, no. 11, pp. 961–972, 2013.

[2] J. Song, C. Guo, Z. Wang, Y. Zhang, G. Yu, and J.-M. Pierson, "HaoLap: A Hadoop based OLAP system for big data," *Journal of Systems and Software*, vol. 102, pp. 167–181, 2015.

[3] 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.

[4] 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.

[5] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-agnostic flow scheduling for commodity data centers," in *Proc. 12th USENIX Symposium on Networked Systems Design and Implementation*, 2015, pp. 455–468.

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

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

[8] C.-H. Chiu, D. K. Singh, Q. Wang, K. Lee, and S.-J. Park, "Minimal Coflow routing and scheduling in OpenFlow-based cloud storage area networks," in *Proc. 10th IEEE Int. Conf. Cloud Computing*, 2017, pp. 222–229.

[9] 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. 2015 IEEE Conf. Comput. Commun.*, 2015, pp. 424–432.

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

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

[12] L. Cheng, S. Kotoulas, T. E. Ward, and G. Theodoropoulos, "Improving the robustness and performance of parallel joins over distributed systems," *Journal of Parallel and Distributed Computing*, vol. 109, pp. 310–323, 2017.

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

[14] S. Wang, J. Zhang, T. Huang, T. Pan, J. Liu, and Y. Liu, "Multi-attributes-based coflow scheduling without prior knowledge," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1962–1975, 2018.

[15] L. Cheng, Y. Wang, Q. Liu, D. H. Epema, C. Liu, Y. Mao, and J. Murphy, "Network-aware locality scheduling for distributed data operators in data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 6, pp. 1494–1510, 2021.

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

[17] 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.

[18] 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 Inter. Conf. Management of Data*, 2010, pp. 975–986.

[19] L. Cheng, I. Tachmazidis, S. Kotoulas, and G. Antoniou, "Design and evaluation of small–large outer joins in cloud computing environments," *Journal of Parallel and Distributed Computing*, vol. 110, pp. 2–15, 2017.

[20] 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.

[21] K. Slagter, C.-H. Hsu, Y.-C. Chung, and G. Yi, "SmartJoin: a network-aware multiway join for MapReduce," *Cluster computing*, vol. 17, no. 3, pp. 629–641, 2014.

[22] G. Carofiglio and L. Muscariello, "On the impact of TCP and per-flow scheduling on internet performance," *IEEE/ACM Trans. Netw.*, vol. 20, no. 2, pp. 620–633, 2012.

[23] 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.

[24] T. Zhang, R. Shu, Z. Shan, and F. Ren, "Distributed bottleneck-aware coflow scheduling in data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 7, pp. 1565–1579, 2019.

[25] S. Wang, J. Zhang, T. Huang, J. Liu, T. Pan, and Y. Liu, "A survey of coflow scheduling schemes for data center networks," *IEEE Communications Magazine*, vol. 56, no. 6, pp. 179–185, 2018.

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

[27] S. Luo, H. Yu, Y. Zhao, S. Wang, S. Yu, and L. Li, "Towards practical and near-optimal coflow scheduling for data center networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 11, pp. 3366–3380, 2016.

[28] 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*, vol. 14, no. 3, pp. 3117–3128, 2020.

[29] A. A. Nasr, A. T. Chronopoulos, N. A. El-Bahnasawy, G. Attiya, and A. El-Sayed, "A novel water pressure change optimization technique for solving scheduling problem in cloud computing," *Cluster Computing*, vol. 22, no. 2, pp. 601–617, 2019.

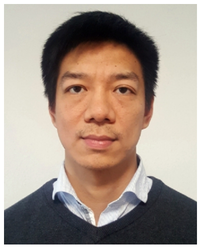
[30] W. Li, X. Yuan, K. Li, H. Qi, X. Zhou, and R. Xu, "Endpoint-flexible coflow scheduling across geo-distributed datacenters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 10, pp. 2466–2481, 2020.

[31] P. Sun, Z. Guo, J. Wang, J. Li, J. Lan, and Y. Hu, "Deepweave: Accelerating job completion time with deep reinforcement learning-based coflow scheduling," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, 2020.

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

[33] M. Jemmali, M. Denden, W. Boulila, R. H. Jhaveri, G. Srivastava, and T. R. Gadekallu, "A novel model based on window-pass preferences for data-emergency-aware scheduling in computer networks," *IEEE Transactions on Industrial Informatics*, 2022.

[34] G. Campero Durand, R. Piriyev, M. Pinnecke, D. Broneske, B. Gurumurthy, and G. Saake, "Automated vertical partitioning with deep reinforcement learning," in *European Conference on Advances in Databases and Information Systems*, 2019, pp. 126–134.



Long Cheng is a Full 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. He has published more than 80 papers in journals and conferences like TPDS, TON, TC, TSC, TASE, TCAD, TVT, TCC, TBD, TITS, TSMC, TVLSI, JPDC, IEEE Network, HPCA, CIKM, ICPP, CCGrid and Euro-Par etc. His research focuses on distributed systems, deep learning, cloud computing and process mining. Prof Cheng is a Senior Member of the IEEE and Co-Chair of Journal of Cloud Computing.

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 from Harbin Institute of Technology, in 2007 and 2009 respectively, and the Ph.D degree from ICT in 2014. His research interests includes computer architecture and VLSI design, specifically memory system, on-chip interconnects, resilient and energy-efficient architecture, machine learning accelerators, and parallel data systems. He is a member of the IEEE.

Rutvij H. Jhaveri is an experienced researcher working in the Department of CSE at Pandit Deendayal Energy University, India. He conducted his Postdoctoral Research at Delta-NTU Corporate Lab for Cyber-Physical Systems, Nanyang Technological University, Singapore. He completed his PhD in 2016, Master's degree in 2012 and Bachelor's degree in 2002. In 2017, he was awarded with prestigious Pedagogical Innovation Award by Gujarat Technological University. Currently, he is co-investigating a funded project from GUJCOST. He was ranked among top 2% scientists around the world in 2021. He has 2000+ Google Scholar citations with h-index 22. He authored 100+ articles published by prominent publishing houses. He is a member of the Research Advisory Board in Symbiosis Institute of Digital and Telecom Management. He is editorial board member in several journals of repute. He also served as a committee member in ”Smart Village Project” - Government of Gujarat, at district level during the year 2017. His research interests are network resilience, SDN for addressing CPS/IoT challenges, secure routing.

Qingle Wang received her Ph.D. degree in cryptography from Beijing University of Posts and Telecommunications in 2017. From 2015 to 2017, she was a visiting scholar in Quantum Sciences and Technologies Group, Department of Physics and Astronomy, Louisiana State University. Currently, she is an Associate Professor with North China Electric Power University. She has published more than 20 papers in international journals. Her research interests include information security, quantum cryptography, quantum information and high-performance 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.