Cluster Frameworks for Efficient Scheduling and Resource Allocation in Data Center Networks: A Survey

Kun Wang, *Senior Member, IEEE*, Qihua Zhou, *Student Member, IEEE*, Song Guo, *Senior Member, IEEE*, and Jiangtao Luo, *Senior Member, IEEE*

***Abstract*—Data centers are widely used for big data analytics, which often involve data-parallel jobs, including query and web service. Meanwhile, cluster frameworks are rapidly developed for data-intensive applications in data center networks (DCNs). To promote the performance of these frameworks, many efforts have been paid to improve scheduling strategies and resource allocation algorithms. With the deployment of geo-distributed data centers and data-intensive applications, the optimization in DCNs regains pervasive attention in both industry and academia. Many solutions, such as the coflow-aware scheduling and speculative execution, have been proposed to meet various requirements. Therefore, we present a solid starting ground and comprehensive overview in this area to help readers quickly understand state-of-the-art technologies and research progress. We observe that algorithms in cluster frameworks are implemented with different guidelines and can be classified according to scheduling granularity, controller management, and prior-knowledge requirement. In addition, mechanisms for conquering crucial challenges in DCNs are discussed, including providing low latency and minimizing job completion time. Moreover, we analyze desirable properties of fault tolerance and scalability to illuminate the design principles of distributed systems. We hope that this paper will shed light on this promising land and serve as a guide for further researches.**

Manuscript received February 21, 2018; revised June 13, 2018; accepted July 14, 2018. Date of publication July 20, 2018; date of current version November 19, 2018. This work was supported in part by NSFC under Grant 61872195 and Grant 61572262, in part by the China Post-Doctoral Science Foundation under Grant 2017M610252, in part by the China Post-Doctoral Science Special Foundation under Grant 2017T 100297, in part by the Open Research Fund of the Jiangsu Engineering Research Center of Communication and Network Technology, NJUPT, in part by the National Engineering Research Center of Communications and Networking (Nanjing University of Posts and Telecommunications) under Grant TXKY17014, in part by the Shenzhen Basic Research Funding Scheme under Grant JCYJ20170818103849343, and in part by the Chongqing Municipal Project under Grant cstc2015jcyjBX0009 and Grant CSTCKJCXLJRC20. *(Corresponding author: Kun Wang.)*

K. Wang is with the Jiangsu Engineering Research Center of Communication and Network Technology, Nanjing University of Posts and Telecommunications, Nanjing 210003, China, and also with Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China (e-mail: kwang@njupt.edu.cn).

Q. Zhou is with the National Engineering Research Center of Communications and Networking, Nanjing University of Posts and Telecommunications, Nanjing 210003, China (e-mail: kimizqh@foxmail.com).

S. Guo is with the Department of Computing, Hong Kong Polytechnic University, Hong Kong (e-mail: song.guo@polyu.edu.hk).

J. Luo is with the Electronic Information and Networking Research Institute, Chongqing University of Posts and Telecommunications, Chongqing 400065, China (e-mail: Luojt@cqupt.edu.cn).

Digital Object Identifier 10.1109/COMST.2018.2857922

***Index Terms*—Scheduling, cluster frameworks, data center networks, big data, data-parallel jobs, resource allocation, coflow, distributed systems.**

ACRONYMS

DCNs	Data Center Networks
HPC	High Performance Computing
FCT	Flow Completion Time
CCT	Coflow Completion Time
JCT	Job Completion Time
TCP	Transmission Control Protocol
RDD	Resilient Distributed Dataset
GFS	Google File Systems
DAG	Directed Acyclic Graph
SFF	Shortest Flow First
FIFO	First In First Out
FCFS	First Come First Served
SRPT	Shortest Remaining Processing Time
WSS	Weighted Shuffle Scheduling
PFF	Per Flow Fairness
PFP	Per Flow Priority
SEBF	Smallest Effective Bottleneck First
MADD	Minimum Allocation for Desired Duration
CLAS	Coflow-Aware Least-Attained Service
EDF	Earliest Deadline First
PDQ	Preemptive Distributed Quick
MRTF	Minimum Remaining Time First
PFC	Priority Flow Control
DBSCAN	Density Based Spatial Clustering Applications with Noise
BSP	Bulk Synchronous Parallel
RCP	Rate Control Protocol
ECMP	Equal Cost Multi Path
TC	Transfer Controller
ITC	Inter Transfer Controller
DCTCP	Data Center TCP
D^{2}TCP	Deadline Aware Data Center TCP
ECN	Explicit Congestion Notification
OLDI	Online Data Intensive

I. INTRODUCTION

WITH the rapid development of information technology, our society has stood at doorstep of big

data [1], [2]. To handle the rapid increment of computation requirements [3], [4], data centers are widely used [5]–[7] for data-intensive applications, including query, web service and big data analytics [8]–[12]. Computing in data center networks (DCNs) provides unparalleled mobility and ubiquitous data accessibility, which can make service more acceptable. Data centers are warehouses that host a large number of servers, providing data-processing service [13] and enabling communications among large amounts of computing resources [14]. Many high performance computing (HPC) clusters have been implemented to process jobs in DCNs. For example, MapReduce [15] is a model using the re-execution technology for fault tolerance. Meanwhile, Dryad [16] is a system that executes data-parallel applications in coarse-grained granularity. In addition, to promote the iteration procedure in data-intensive applications, Spark [17] chooses a coarse-grained division to task stages instead of a fine-grained scale.

Although the diversity exists in different cluster frameworks, some common patterns can be extracted. We observe that the transmission between multi-stages in tasks has a significant impact on job-level performance and contributes large parts of the completion time. The deficiency of network transmission incurs many problems, which inspire the performance improvement in DCNs. As data-parallel applications generate gigantic amount of network traffic, most cluster frameworks [15]–[18] concentrate on the transmission optimization of cluster frameworks. Meanwhile, determining the scheduling strategy and resource allocation algorithm is one of the most crucial issues [19]–[27]. To the best of our knowledge, existing researches [13], [28]–[32] do not cover coordination principles, desirable properties (e.g., data distribution, prior knowledge and control scheme), and corresponding evaluation metrics. Considering the above absences, we are encouraged to conduct this survey. As one of the most desirable metrics for evaluation, task throughput of a data center can be reflected by the average job completion time (JCT) [33]. The data transmission in DCNs generates the traffic called *flows*. A flow is defined as a sequence of packets sent from a source to a destination [34], and the flow completion time (FCT) directly influences JCT. Flows need to be transmitted in time to meet the deadlines, so as to guarantee user experience [35]. Motivated by this fact and the defects in transmission control protocol (TCP), many frameworks are designed to guarantee predictable communication time [36]. Therefore, minimizing completion time and meeting deadlines become the main optimization objectives.

Moreover, the data transmission can be categorised into different granularities (e.g., packet, flow, coflow, task and job). By exploiting the traffic patterns of big data jobs, Chowdhury and Stoica [37] proposed an abstraction of *coflow* which represents a collection of semantic-related flows to convey requirements of job-specific communications. A coflow is defined as a set of flows associated with the same computing stage, e.g., the shuffling stage in MapReduce jobs. A coflow completes only when the transmission of all associated flows finish. In other words, the completion time of a coflow is determined by the slowest flow. Coflow has been extensively studied with different objectives under various scenarios [38]–[44]. Note that minimizing completion time and meeting deadlines can be formulated as the problem of minimizing the corresponding coflow completion time (CCT) [37]. However, Appuswamy *et al.* [45] proposed a task-aware mechanism, arguing that tasks are the most crucial objective in scheduling, while coflow-aware algorithms allow applications to expose their semantics to network layer. To the length of flows, the few but long flows generate most traffic in DCNs [33], [46], [47]. This phenomenon is called the heavy-tailed distribution [48]. Users' feedback tasks associated with the *Partition/Aggregation* operations create short flows, which are in size of few kilobytes. Long flows are in megabyte scales, generated from information storage or maintenance [49]. In addition to flow length, the output size, input size and number of tasks in production also follow the heavy-tailed distribution [38]. We observe that the *Shortest-Flow-First* (SFF) is a provable optimized algorithm in such distribution. On the contrary, light-tailed distribution means that flows are nearly in the same size. In this case, the *First-In-First-Out* (FIFO) outperforms other algorithms. Therefore, making the algorithms adapted to both heavy-tailed and light-tailed distributions is reasonable [49].

To implement these algorithms, most schedulers require complete prior knowledge in terms of flow size, flow number and arrival time. Unfortunately, these characteristics cannot be predictable in many cases, such as pipelining, speculative executions and task failures [50]. Thus, scheduling without prior knowledge is proposed [39]. Note that multi-stage jobs transfer data between successive computation stages using pipelining [16], [39], [51], [52] and data from a single stage can be divided into multiple waves [39], [53]. Moreover, to fault tolerance and scalability, redundant flows can be generated by task failures and speculations [15], [16], [39], [50]. We also summarize the control schemes, which are one of the most significant coordination principles of algorithm design. Some frameworks [45] choose decentralized schedulers to avoid the problems such as scalability and fault tolerance. As allocating resources and offering Internet services are the goals of distributed systems [5], the decentralized design can provide absorbing scale and available capabilities, while a centralized design has a restriction on the throughput and availability. However, many systems [42], [54]–[56] still choose centralized framework due to the superiority of earning prior knowledge (e.g., flow size and arrival time).

As shown in Fig. 1, we classify existing frameworks according to the objective entities (e.g., disk, network, straggler) and compare different optimization aspects (e.g., resource allocation, scheduling mechanisms, utilization). In switch-level, we exhibit cluster frameworks in terms of packet and router. Meanwhile, the optimizations based on cloud and virtualization are subject to application-level patterns. Moreover, the network-level strategies mostly focus on designing a suitable coordination principle (e.g., control scheme, granularity, prior knowledge) and improving communication metrics (e.g., CCT and deadlines).

This paper takes the first step to survey the cluster frameworks and scheduling strategies in DCNs. Covering the

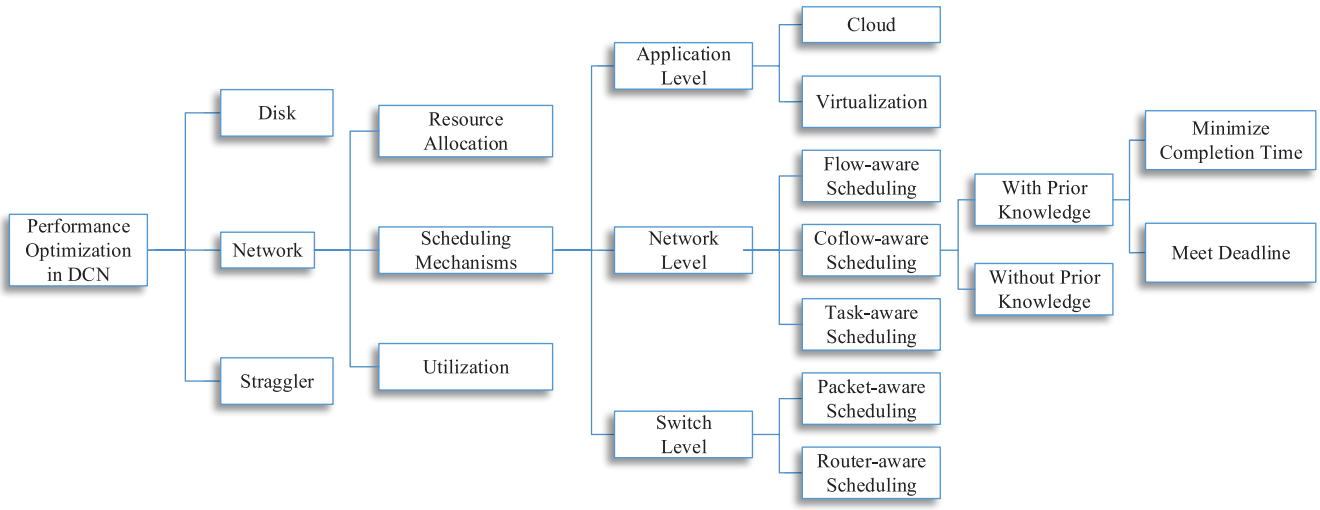


Fig. 1. The classification of different cluster frameworks for efficient scheduling.

absence of researches, we summarize the contributions of our work as follows:

- We provide a detailed discussion of recent achievements in algorithms. Moreover, by providing a review and giving a comparison in different metrics, we straighten out the development course of existing frameworks.
- We use coordination principles (e.g., data distribution, prior knowledge and control scheme) for scheduling strategy analysis to help readers understand the research progress of cluster frameworks.
- We summarize the patterns of cluster frameworks and illuminate the state-of-the-art assistive technologies for them.

The rest of our paper is organized as follows. We introduce the background in Section II. An architectural overview of coordination principle is given in Section III. Then, Section IV compares different cluster frameworks and we analyse the metrics for performance evaluation in Section V. Moreover, some desirable properties in system designs are demonstrated in Section VI. Finally, we discuss the open issues and give the summary in Section VII.

II. Background

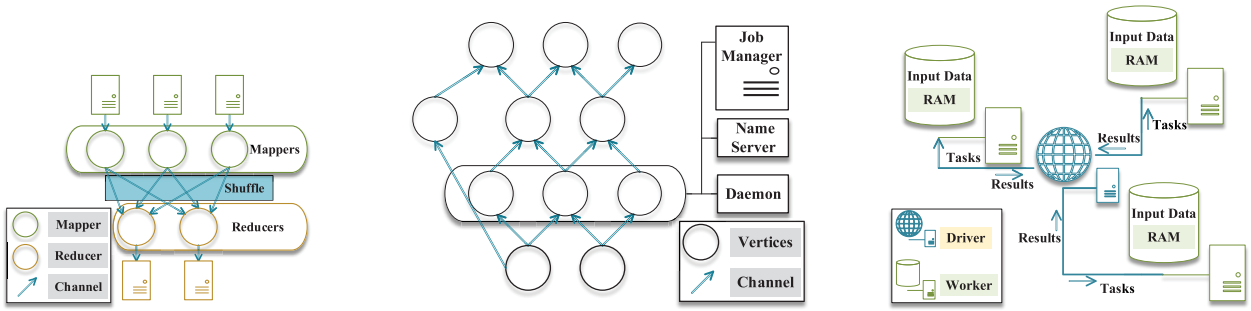
Computer systems are undergoing a revolution in the recent decades, from the single machine architecture to distributed cluster frameworks. The development of powerful microprocessors (e.g., CPU and GPU) and invention of high-speed networks promote this process forward. A distributed cluster framework is a collection of independent computers that works as a single coherent system. To design an applicable cluster framework, various important goals should be met, such as accessibility, transparency, scalability, consistency, reliability, and fault tolerance, etc. Exploiting the parallelism in big data applications for sequential programs has a long history. Many cluster frameworks are developed to handle these applications [15]–[17], [57]. As shown in Fig. 2, we summarize existing frameworks into three typical paradigms: (1) MapReduce [15] is a widely-used model in distributed systems, making full use of both network and disk resources. (2) Meanwhile, Dryad [16] proposed by Microsoft is an alternative for efficiently processing data-parallel applications in a coarse-grained scheme. (3) Subliming the design principle in MapReduce, Spark [17] provides a good performance in iteration-intensive operations (e.g., logistic regression) by using resilient distributed dataset (RDD) and guarantees the fault tolerance with the abstraction called *lineage*. To better understand the research progress in this area, we will provide more details about these frameworks.

A. MapReduce and Hadoop

MapReduce [15] is a model for tackling data-parallel applications and designing distributed systems. As shown in Fig. 2(a), the user-specified *mappers* generate the *key/value*-based pairs from large amounts of raw data. Then, the MapReduce library combines the same key and shuffles the intermediate data via network transmission. Finally, the *reducers* aggregate the data from last step and output the results. MapReduce is widely used for system design, the renowned Hadoop [57], [58] is an efficient implementation of MapReduce, using re-execution for fault tolerance and parallel operations.

B. Dryad

Dryad [16] is a distributed execution engine, processing data-parallel applications in a coarse-grained scheme. Similar to Condor [59], Dryad is an example in a distributed setting. The derivation of Dryad is inspired by graphics processing units (GPUs) [60], [61], Google file system (GFS) [15], [62] and parallel databases [63]. As shown in Fig. 2 (b), Dryad uses vertices (the circles) and channels (the arrows) to create several data flow graphs for job segmentation. The operations of message transmission guarantee a property that the resulting graph is acyclic. A basic definition of the graph can be described as $G = \{V_G, E_G, I_G, O_G\}$, where G represents the entire graph, containing a collection of vertices V_G and directed edges E_G. Meanwhile, I_G and O_G represent the *input* and



(a) MapReduce (b) Dryad (c) Spark

Fig. 2. (a) MapReduce works with two reducers (yellow circles) and three mappers (green circle). Note that the shuffle stage is highlighted via blue arrows; (b) Dryad uses vertices (circles) and channels (arrows) to form a data flow graph for job segmentation; (c) Spark provides a shared memory to tackle applications in a *master-worker* scheme. To accelerate job executions, workers read data blocks from a distributed file system and store intermediate results in memory.

output of the vertices, respectively. Note that data are transmitted through the channels, each of which is constituted with two vertices. Without thread creation and lock mechanism, Dryad can manage the available resources more efficiently.

C. Spark and RDD

Spark is a system which accelerates interactive operations and uses a general-purpose programming language for in-memory data mining on clusters [64]. Note that Scala is implemented as a language-integrated programming interface in Spark, similar to DryadLINQ [65]. To perfect the aspect of distribution and fault tolerance, Spark chooses coarse-grained updates, instead of fine-grained transformations [17]. As shown in Fig. 2 (c), a shared memory with restricted form can decrease application execution overheads (e.g., data replication, disk I/O) caused by writing the reuse data in an external system. In the history of applications reuse, HaLoop [66] is a system which offers an iterative interface in MapReduce. Pregel [67] keeps intermediate data in memory during iterative graph computation process. The core module in Spark is the RDD, which is a common reuse abstraction. Users can store intermediate results in memory explicitly, optimizing data placement by having charge of result partition. As shown in Fig. 3, RDD appropriately chooses coarse-grained transformation interfaces, allowing a system to provide fault tolerance with lineage.

Online services often have a partition part that requests are partitioned amongst workers and a aggregate part which generates response [68]. As a common feature, flows past through a sequence of stages to generate the final result. As shown in Fig. 4, there are six communication patterns: (a) single flow is the simplest pattern, where data is transmitted from a source to a destination in a single link, (b) parallel flows is a collection of independent single flows, each of which is transmitted at the same time, (c) broadcast is also a collection of flows, which follows the *one-to-many* scheme, (d) aggregation is an opposite pattern of broadcast, following the *many-to-one* scheme, (e) shuffle is a common communication patterns in MapReduce, where each mapper needs to communicate with all of the reducers, and (f) all-to-all is a full-connected network, where each node communicates

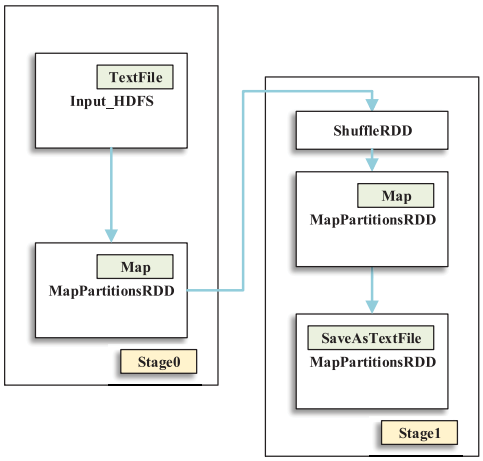


Fig. 3. To tackle a job, Spark launches a number of RDDs to divide the whole procedure into two stages: stage0 and stage1. Outputs are aggregated as a text file with the operations of *partition*, *map*, *shuffle* and *reduce*.

with other nodes. To express all of these patterns, the abstraction of coflow is proposed [37]. Based on it, Varys [38] and Aalo [39] are implemented for efficient coflow scheduling. Moreover, the CODA [40] provides an automatic coflow identification method to make previous coflow schedulers become applicable in real productions. Regardless of the different models of diverse applications, common flows can be grouped into a few common patterns.

III. Coordination Principles

The purpose of the cluster frameworks is to offer communication services for users and coordinate resource allocation efficiently [69]–[71]. Thus, it is necessary to carefully design cluster frameworks according to a number of coordination principles, which can be classified into four main categories: control scheme, distribution, prior knowledge and objective granularity.

A. Control Schemes

A crucial problem that should be considered first is to determine the control scheme, which contains two types: the centralized scheme and the decentralized scheme. In the architecture design of cluster frameworks, researchers have not

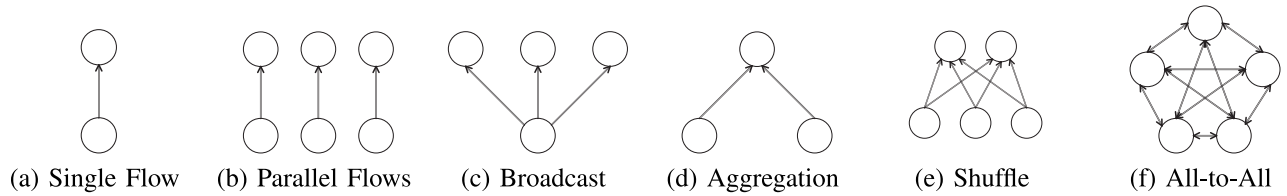


Fig. 4. Common patterns in data-intensive applications.

TABLE I
SUMMARY OF CENTRALIZED CLUSTER FRAMEWORKS

	Orchestra [55]	**Varys [38]**	**Aalo [39]**	**CODA [40]**	**Adia [42]**	**OMCoflow [72]**
On-Arrival Knowledge	Prior Knowledge	Prior Knowledge	Without Prior Knowledge	Without Prior Knowledge	Prior Knowledge	Prior Knowledge
Queue Scheduling	Not Applicable	Strict Priority	Weighted	Weighted	Priority	Not Applicable
Coflow/ Job Scheduling in Each Queue	FIFO	Not Applicable	FIFO	FIFO	Not Applicable	Weighted sharing
Flow/ Task Scheduling	WSS	SEBF and MADD	Max-Min fairness	Max-Min fairness	Largest Load First and SRTF	Optimization
Work Conservation	Coflow	Queue	Weighted Among Queues	Not Applicable	Coflow	Coflow
Metric	JCT	Average Completion Time and Deadline	Average CCT	Average CCT	Average link utilization and Average Completion Times	Average CCT

reached a consensus on the selection of control scheme. Both two schemes own their superiority and shortage.

1) The Centralized Scheme: We observe that the centralized scheme is widely employed by a central scheduler to control all the data (e.g., flows, coflows, tasks, and jobs) in the DCNs. This scheme usually achieves an acceptable performance because they can globally coordinate the communications. A number of cluster frameworks are designed in the centralized scheme. To show significant features of this scheme, we compare a number of typical centralized frameworks in Table I.

Hedera [54] is designed in a centralized manner. The scheduler in Hedera can meet the requirements of fault tolerance and scalability, using periodical updates to synchronize the network communications. Hedera desires to dispatch flows to the paths which are not conflicting. Specifically, it attempts to send numerous flows in an uplink or a downlink with accommodating the united bandwidth requirements. Thus, Hedera can outperforms (39% better bisection bandwidth) the *Equal-Cost-Multi-Path* (ECMP) scheme.

Orchestra [55] is also a centralized scheduler, implemented for controlling the actions of both intra- and inter- transmission. In order to achieve this purpose, the system architecture is designed in a multi-level controller manner. Each transfer controller (TC) is able to select different transmission devices according to the information about flow size and node status.

Meanwhile, Varys [38] claims that the FIFO strategy [5] may miss the deadlines when small coflows are blocked by a large one. However, the FIFO strategy is often used in decentralized system architecture [5], [55]. From the above, Varys presents a framework which combines *Smallest-Effective-Bottleneck-First* (SEBF) heuristic with *Minimum-Allocation-for-Desired-Duration* (MADD) algorithm. With these two algorithms, the centralized architecture of Varys is robust and the coflows can be efficiently scheduled.

Although the framework of Baraat is powerful in several distributions, especially the *light-tailed* distribution, Aalo [38] still argues that Baraat may not perform well in the conditions without global coordination. Therefore, Aalo uses a central-controlled architecture, which has two control manners: the global control with chronic coordination and the local control with temporary prioritization.

CODA [40] is also a centralized cluster framework, inspired by a number of successful implementations [15], [38], [50], [62], [73]. It employs a central master to control numerous devices in a *master-agent* structure. The *master* node plays a major role in coflow recognition and scheduling. To an *agent* node on each end-point, the flow-level information is collected for assisting the *master* to wisely make scheduling strategies. In order to handle numerous *agent* nodes, the central *master* should be designed to meet two desirable properties: the scalability and the fault tolerance.

Adia [42] considers that the centralized scheme is necessary because the controller needs to handle all the information about coflows. Note that some coflow scheduling mechanisms require all the knowledge [38], [72]. Similar to CODA [40], coflow information on each end host is collected and transported to a central scheduler. Furthermore, Adia concentrates on the link utility and reports the status on each machine to the master node, i.e., the controller. Meanwhile, the collector receives all relevant information from the daemons of senders. Thus, the prior knowledge of each coflow can be obtained in advance. From the above, the controller needs to make two decisions: (1) the rate for flow transmission and the host-based rate for each daemon. However, the fine-grained scheduling may incur side-effects on system scalability. In order to conquer this challenge, Adia delivers the fine-grained workloads to each daemon for alleviating the huge pressure on the central controller. Therefore, the controller just needs

to handle overall link status and collect prior knowledge of coflows. Meanwhile, each daemon can decide the flow rate by querying the controller regularly. Moreover, coflows are in the *heavy-tailed* distribution [38] and the majority of traffic is only generated by a handful of coflows. Thus, Adia claims that concentrating on the large coflows can further promote the efficiency of centralized frameworks.

Furthermore, Chronos [56] also chooses the centralized scheme to collect the prior knowledge (e.g., coflow size and arrival time). Chronos only schedules a coflow when all data belonging to it are ready for transmission. If Chronos cannot obtain enough coflow information, the coflow-level scheduling will be degraded to the flow-level mode.

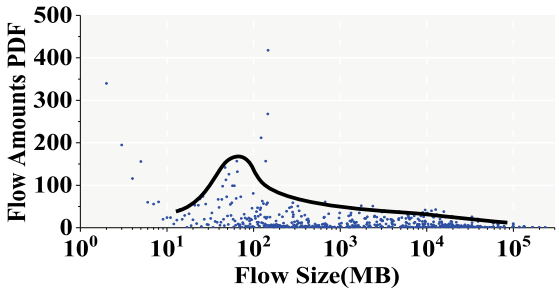
2) The Decentralized Scheme: Frameworks designed in the decentralized scheme can leverage network protocols to avoid the huge overheads in the centralized scheme. Moreover, the decentralized scheme makes frameworks more robust in fault tolerance and scalability. To have a better understand on this scheme, we illustrate several typical decentralized frameworks in Table II.

Sparrow [74] is implemented in a decentralized manner to provide a near-optimal performance. Ousterhout *et al.* [74] claimed that a decentralized design can provide an absorbing capability in job management, while a centralized design has a limitation on the throughput and scalability. To achieve good performance, Sparrow follows two basic policies: (1) A constraint exists where a per-job or a per-task is launched. (2) In terms of resource allocation, strict priorities and *Weighted-Fair-Sharing* are enabled when the resource requirements exceed the cluster capacity.

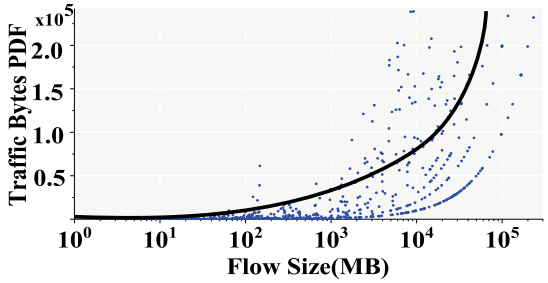
pFabric [75] is another decentralized scheduler, which owns a near-optimal performance in job scheduling and flow transmission. Meanwhile, Baraat [5] also employs a decentralized task-aware scheduler to prevent a number of problems (e.g., scalability and fault tolerance). No matter when the flows begin and in which portion to travel, each task owns an exclusive priority to classify the flows in it. In this way, all the flows belonging to a task can be aggregated as a group for scheduling together, regardless of time and space characteristics. Thus, Baraat outperforms both pFabric [75] and Orchestra [55] because heavy tasks can be efficiently handled in a multiplexed-level scheme. Moreover, Baraat improves the scheduling performance in various workloads, reducing both total and tail completion time.

TABLE II
SUMMARY OF DECENTRALIZED CLUSTER FRAMEWORKS

	Sparrow [74]	pFabric [75]	Baraat [5]	Optas [76]
On-Arrival Knowledge	Without Prior Knowledge	Prior Knowledge	Without Prior Knowledge	Prior Knowledge
Queue Scheduling	Priority	Priority	Not Applicable	Priority
Coflow/ Job Scheduling in Each Queue	FIFO	PDQ	FIFO	FIFO-based
Flow/ Task Scheduling	Fair-sharing or Priority-based	SRPT	Max-Min fairness	FIFO-based
Work Conservation	Not Applicable	Not Applicable	Coflow	Not Applicable
Metric	Response Time	FCT	Average and the Tail Task Completion Time	Flow-level



(a) PDF of flow amounts



(b) PDF of traffic bytes

Fig. 5. According to the flow distribution, 89.49% flows are smaller than 10 GB and most flows are in length of [10MB, 10GB]. Meanwhile, more than 93.03% traffic bytes are created by the flows larger than 10 GB.

B. Distribution

Two major distributions exist in data-intensive applications: the *light-tailed* distribution and the *heavy-tailed* distribution. Most jobs follow the heavy-tailed distribution in their output size, input size and number of tasks [38]. As shown in Fig. 5, a handful of large flows generate the majority of the traffic data in heavy-tailed distribution. On the contrary, flow sizes are fairly homogeneous in light-tailed distribution. In existing researches, scheduling disciplines which work well in light-tailed (exponential) distribution distributions may not perform well in heavy-tailed (power) distribution, and vice-versa [49]. In other words, it is hard to implement a framework that can perform perfectly in both two distributions. Thus, analysing the distribution in terms of flow, coflow, task, job and applications is a crucial issue in algorithm design. We observe that *First-Come-First-Served* (FCFS) scheduling performs well in minimizing JCT when the job size distribution is light-tailed [6]. Meanwhile, if the job size follows the heavy-tailed distribution, *Shortest-Remaining-Processing-Time* (SRPT) [49], *Processor Sharing* (PS) [49], and many other policies (e.g., all SMART policies [48]) have a good performance. In realistic deployment with distribution consideration, Chowdhury and Stoica [39] proposed the *Coflow-Aware Least-Attained Service* (CLAS) algorithm that straightforwardly leads to fine-grained sharing for light-tailed distribution. Note that the fine-grained sharing is suboptimal in terms of minimizing average CCT [5], [6], [77]. Fortunately, Chowdhury and Stoica addressed this dilemma and proposed a solution by discretizing coflow priorities.

C. Prior Knowledge

In existing scheduling mechanisms, it is common that the network condition and job characteristics are unknown in advance. Fortunately, some state-of-the-art researches [37], [78], [79] claim that prior knowledge can be obtained using prediction techniques.

Orchestra [55] employs an algorithm called *Weighted-Shuffle-Scheduling* (WSS). WSS assigns the weight of each flow by globally coordinating task information. Prior knowledge about data size and node number are used to help transfer controller (TC) choose a most feasible transmission mechanism. The scheduler in Varys [38] also requires complete prior knowledge, including flow distribution, flow size and cluster status. Note that flows can be aggregated into coflows when the centralized scheduler have full knowledge of all network traffic. Thanks to prior knowledge, Cho *et al.* [80] used credit packets to alleviate network congestion during transmission. In this scheme, a sender can safely transfer data and avoid congestion and reacting.

From the above, the consideration of prior knowledge is a crucial property in scheduling algorithm design. Actually, most of schedulers [5], [6], [38], [41], [43], [55], [81] are designed based on the preliminary that the prior knowledge can be gathered in advance. Prior knowledge contains the flow information, such as flow arrival time, size, numbers and length [37], [38]. Meanwhile, the status of cluster resources, including available bandwidth and computational capacity, also dominates prior knowledge. It is relatively readily to design an efficient scheduler when prior knowledge can be collected. However, it is not always suitable to know the prior knowledge. In many scenarios, flow and coflow characteristics are hard to gather. As the pipelining techniques are widely used to accelerate multi-stage jobs [16], [17], [50], the quickly-transferred data makes it unreasonable to collect flow characteristics. Moreover, the multi-waves [53] prevent the flows within a coflow starting together, making it hard to aggregate coflows. Finally, stragglers in task failures and speculative execution [16], [50], [82]–[85] also hamper the prior knowledge collection. The existence of re-started flows [15], [16], [50] makes the complete flow information cannot be obtained before completion. Thus, schedulers based on prior knowledge may be inapplicable in real use cases. Fortunately, some efforts have bee paid to conquer this dilemma. FIFO and its variants [5], [55] can be adopted into schedulers to conquer this problem, requiring no prior knowledge. For example, Baraat [5] employs FIFO for task scheduling, so as to change multiplexing layers to avoid *head-of-line* blocking. Moreover, based on updating priority class dynamically, Aalo [39] can also schedule coflows without any prior knowledge.

D. Objective Granularity

Data-intensive applications operated via cluster frameworks have multiple requirements [37]. Different with the proposals concentrate on the bandwidth [33], [86]–[88], Orchestra captures the job-level semantics and improves the transfer-level scheduling of tasks, each of which consists of a number of flows from a stage to the next. Capturing communication patterns is important to further promote the optimization of data-parallel jobs [5], [38], [54], [55], [89]. However, researchers maily focus on network-level metrics, overlooking application-level requirements [54], [75], [90], [91]. Fortunately, Chowdhury and Stoica [37] proposed the abstraction of *coflow*, which is a collection of semantic-related flows, catching communication patterns between two group of machines. By introducing coflow, the underlying attributes in both network and application layer can be detected. Moreover, researchers can further improve the performance of job scheduling in DCNs with the perspective of coflow-level optimization.

Meanwhile, various of tasks (or jobs) are executed in DCNs, including uploading user social news-feed and answering a search query. These tasks contain numerous components, each of which owns a deadline about task completion, making scheduling base on task-aware fashion popular. As shown in Fig. 6, the *all-or-nothing* property [53] has a significant influence on task-aware scheduling. Tasks are operated in parallel, only when all the components are optimized can tasks be accelerated. In realistic production, MapReduce [15] and BSP [67] also evidence this phenomenon, i.e., a stage cannot start until all the data from the previous stage is received.

IV. Cluster Frameworks

In this section, we will give a detailed discussion on existing cluster frameworks, comparing them in terms of scheduling strategies, online algorithms, network protocols, memory caching, resource allocation and straggler phenomenon. Moreover, the assistive technologies used for promoting the system performance are also included. We observe that scheduling strategy is the kernel in both algorithm design and framework implementation. More precisely, different strategies have their own superiority and shortage according to coordination principles.

A. Scheduling Strategies

In order to have a basic understanding on different algorithms and strategies implemented in these frameworks, we show five sequence diagrams in Fig. 7, including *First-In First-Out* (FIFO) [5], [55], *Per-Flow Fairness* (PFF) [55], *Per-Flow Prioritization* (PFP) [6], [75], *Weighted Shuffle Scheduling* (WSS) [55] and *Smallest-Effective-Bottleneck-First* (SEBF) [38]. Note that FIFO may lead to *head-of-line* blocking, showing randomness of average FCT and CCT in a macro view. Meanwhile, PFF guarantees the max-min fairness principle among flows, with average CCT of 2.75 time units. Also, PFP — the optimal scheduling algorithm for minimizing average FCT in a single link may not perform well in terms of minimizing average CCT. Moreover, WSS reduces the average FCT and CCT compared with PFF. Finally, SEBF is an effective coflow scheduling algorithm, with average CCT of 2.5 time units. This subsection can help readers follow the latest research progress in scheduling algorithm design and cluster framework implementation.

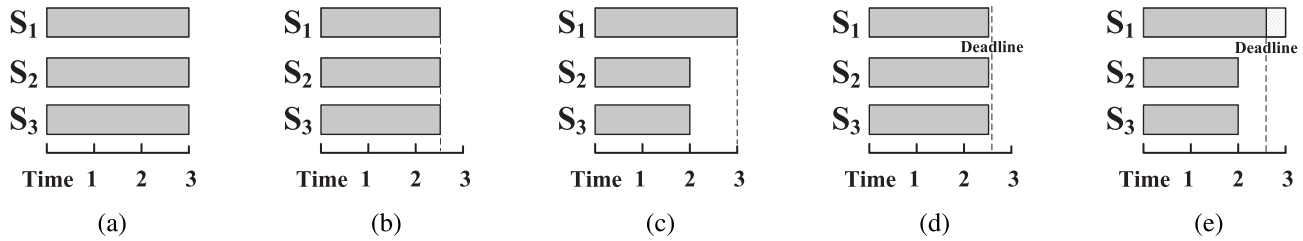


Fig. 6. The *all-or-nothing* property: (a) we show an example of a coflow which is constituted with three flows; (b) In order to reduce CCT, three flows should be optimized together; (c) Although the average FCT is reduced, the average CCT still cannot be optimized; (d) Under the *soft-real-time* constraints, all flows can finish before their deadlines; (e) The delay of one flow may make the whole coflow miss its deadline.

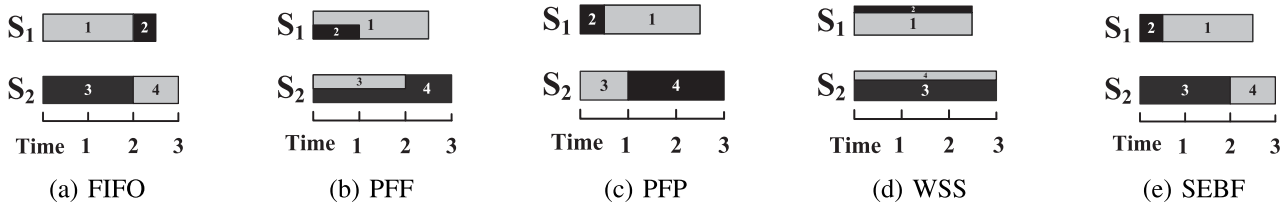


Fig. 7. Two coflows (coflow1 in dark/black and coflow2 in light/gray) are prepared for schduling. Note that each port can transfer one unit of data in one time unit. Average FCT and CCT (in time units) achieved by different mechanisms: (a) 2.375 and 2.75 by FIFO which allocates bandwidth to the flow arriving first. (b) 2.125 and 2.75 by *per-flow fairness* (PFP) which shares bandwidth fairly to each flow; (c) 1.75 and 2.75 by *per-flow prioritization* (PFF) which assign a higher priority to a short flow; (d) 2.75 and 3 by WSS which allocates bandwidth to flows based on their weights; and (e) 2 and 2.5 by SEBF which optimizes coflow-level scheduling.

There are three general aspects for summarizing improvements in existing frameworks.

- Breaking the bottleneck of network is important. Amounts of efforts have been paid to improve network conditions, including scheduling semantic-related flows, improving link utilization and aggregating data for network traffic reduction [37], [38], [54], [55], [92]–[108].
- Breaking the limitation of disk is also a useful aspect, including improving disk utilization [109] and employing cache for data storage [50], [53], [110], [111].
- Alleviating the impacts of stragglers is a novel issue, because stragglers will delay completion time severely. Moreover, the generation cause of stragglers is still not fully known [82]–[84], [89], [112]–[120]. Fortunately, it is reasonable to employ task speculation and load balancing to solve the staggler phnomenon [121].

Adia [42] tries to maximize the utility of available links while not impacting coflow-level performance. To acquire a high link utility, Adia needs to have a coordination of coupled bandwidth between uplinks and downlinks. Adia employs a layer-based scheduling mechanism to control both inter- and intra- link scheduling. To reduce the average CCT, uplinks are treated as a unity and inter-links are scheduled based on their priority in network layer. Adia focuses on the coflows which are all in uplinks. Therefore, it executes the intra-link scheduling in uplinks. The priority of scheduling mechanism is based on the transmission speed of coflows.

Meanwhile, Corral [122] has an *offline* planner which decides the assemble of racks. This planer controls which task should be operated and when a coflow should be transmitted according to job features (e.g., input data, CPU utilization and memory requirements). However, the planner cannot make a distinct determination on job execution. Thus, outputs should be treated as a guideline on placing data.

Orchestra [55] improves transmission performance by allowing scheduling policies to be implemented in transfer layer and capturing communication patterns. Previous researches mainly focus on decreasing JCT [89], [123], [124] in interactive workloads [123], [124] and optimizing resource utilization. However, Orchestra is implemented via arranging network resource. In order to follow network-level distributions, Chowdhury *et al.* [55] analysed application traces which are collected from Facebook, claiming that transferring data in network activity constitutes the majority of JCT. Traditional frameworks [33], [54], [86]–[88] for managing activities and optimizing network utility cannot take semantic-related flows into consideration, due to the lack of job-level semantics. Orchestra realizes the importance of job-level semantics and optimizes the transfers between intra-tasks. Each transfer consists of a number of flows from a stage to the next. Orchestra coordinates the data movement with a number of TCs, each of which guides the transfer continuously and updates the network status of both sources and destinations. Thus, global coordination is the key idea in Orchestra for coordinating the data movements in both inter- and intra-transfers. Orchestra breaks the limitation of optimizing the performance in data transfers using a global management architecture, i.e., inter-transfer controller (ITC), which can determine scheduling policies and manage numerous TCs to select an appropriate mechanism. To reduce the average time consumption in multi-transfer workloads, Orchestra employs a FIFO-based heuristic, using WSS for allocating bandwidth according to weight proportions. To conquer the challenges

of both managing hardwares and capturing network topology, Orchestra is implemented in application layer, similar to a number of previous frameworks [17], [57]. Therefore, Orchestra can be deployed without modifications on routers or switches. Note that Chowdhury *et al.* [55] argued that the application-level coordination still has an improvement restriction in networks. Thus, it is natural to adopt a *sweet-spot* to make a tradeoff between utility and scalability. Based on above architecture, Orchestra is a transfer-level optimal scheduling framework, which efficiently dispatches network resource for minimizing task communication time and guaranteeing predictable deadlines.

Furthermore, observing that previous scheduling approaches are often restricted by limited network resources. Swallow [125] leverages the superiority of coflow-level coordination and employs a fine-grained scheduling manner to alleviate the shortage of bandwidth. Swallow jointly considers traffic data compression and coflow scheduling, i.e., conducting coflow compression to accelerate the entire scheduling progress. The proposed scheduling heuristic *Fastest-Volume-Disposal-First* in Swallow can efficiently minimizes FCT, CCT and JCT, while providing work conservation and starvation freedom.

B. Online Algorithms and Network Protocols

Online applications with service level agreements (SLAs) in their operations [68] often have requirements of workers and aggregators. Reducing latency is a crucial problem in interactive web services [126]. The latency of user interactions reflects the time expires, i.e., a response must be aggregated before its deadline. Note that network bandwidth will be wasted if the response quality is impacted (e.g., a flow misses the deadline). Thus, all stages belonging to an online application should meet their deadlines. Due to the lack of consideration of latency, existing technologies (e.g., congestion control, flow scheduling, FIFO queuing) cannot perfectly maximize network throughput because the fairness in network-level metrics is missing. To design a efficient framework, two properties should be considered:

- As deadlines are associated with each flow, not network packets, flow-level optimization should be concerned.
- Requirements of meeting deadlines are significantly crucial in various applications. Therefore, responses should be operated in time.

Meanwhile, in the implementation of D^3 [127], flow deadlines are in an exponential (heavy-tailed) distribution. Existing frameworks may miss significant fraction of flow deadlines if they use the measurements [128] based on FCT. Wilson *et al.* [127] observed that flow properties in interactive applications can be predicted in many cases. Hence, prior knowledge can be collected in advance. Wilson *et al.* [127] also proposed an *Earliest-Deadline-First* (EDF) heuristic [129], which is the first deadline-aware solution in scheduling. The implementation of EDF employs priority class based on *per-pop packet* deadlines and allocates all the bandwidth to a flow with the earliest deadline. EDF is an optimal scheduling which meets flow deadlines. Thus, deadline-aware solutions outperform the counterparts based on fair sharing, in terms of flow distribution and deadlines. Note that D^3 [127] is a deadline-driven *delivery proactive* protocol, using an explicit rate to control network congestion [130], [131]. The allocation rate from a router to a destination is based on the information of flow initiation time. Routers allocate bandwidth greedily based on *First-Come-First-Served* (FCFS) strategy. Instead of employing work reservation, D^3 assigns rate in a lease scheme to avoid side effects (e.g., router failure) in greedy algorithms. Unfortunately, greedy allocation in D^3 may also impact job performance. Thus, preemptive *Deadline-Aware-Data-Center-TCP* (D^2TCP) [132] is proposed to solve this problem. Furthermore, to handle all the requests of line rate, D^3 needs to switch numerous chips while avoiding high costs in hardwares.

Moreover, as *Online-Data-Intensive* (OLDI) applications are widely executed under *soft-real-time* constraints, satisfying requirements in these applications is also significant. Therefore, D^2TCP also needs to improve performance of burst-processes and makes no change to hardwares and protocols. To achieve this purpose, D^2TCP implements a reactive bandwidth allocation, using a congestion window with *Explicit-Congestion-Notification* (ECN) feedbacks. More precisely, D^2TCP modulates the window size and adopts deadline-awareness based on *Data-Center-TCP* (DCTCP) [128], leveraging the extent of congestion and deadline information. To have a better understand on this approach, we will give some typical formulae to describe core technologies. First, given η to represent the fraction of sent bytes, we have:

$$\eta = (1 - k) \cdot \eta + k \cdot F, \tag{1}$$

where $k, k \in [0, 1]$, is a constant weight and F is the fraction of packets marked in the last window of one *Round-Trip-Time* (RTT). A larger η indicates a higher level of congestion. Once updated, the congestion window is adjusted as:

$$c = c \cdot (1 - \eta/2). \tag{2}$$

Therefore, D^2TCP can meet OLDI deadlines, considering *fan-in-burst-induced* congestion. Moreover, it achieves a higher bandwidth utility than D^3. Alizadeh *et al.* [75] claimed that D^2TCP can estimate flow rates precisely for minimizing FCT and ensuring network utilization. Moreover, collaborating with the technology of RTT, a sender can calculate the fraction of packets marked by ECN. The average fraction of ECN feedbacks can be updated using Eq. (1). Similarly, D^2TCP employs exponential smoothing equation to estimate these packets. As a reaction to congestion, a penalty function d can be defined as:

$$d = \eta^{W_C}, \tag{3}$$

where W_C is the current flow weight. To increase the congestion window size, a sender calculates the ratio θ of current flow weight and the maximum flow weight. This ratio can be defined as:

$$\theta = W_C / W_{\max}, \tag{4}$$

where W_C is a subset of $W_C \in [W_{\min}, W_{\max}]$. Note that $W_{\max}$ and $W_{\min}$ represent the maximum and minimum of assignable weight, respectively.

Different from fair-sharing protocols [128], [130], [133], which are mainly focused on minimizing FCT and meeting deadlines, Hong *et al.* [6] proposed *Preemptive-Distributed-Quick* (PDQ) where flows in high priority can be operated first by pausing others. Thus, PDQ indicates minimizing FCTs using preemptive flow scheduling. This flow scheduling mechanism in PDQ employs an explicit rate control, which is based on switches for assigning rates to individual flows. PDQ also uses an *early termination* to terminate the flow when a flow cannot finish in time in following cases:

- The flow has missed its deadline.
- The remaining time is not enough for flow transmission.
- The transmission operation cannot work smoothly during the rest processing.

To make switches meet *book-keeping* requirements, each flow must be terminated under control. Therefore, many latency-sensitive flows which consume just one RTT may suffer from the extra round-trip [33], [134]. Moreover, two crucial factors should be considered in the implementation of PDQ: (1) PDQ may lead to high latency due to the centralized-control scheduler. To conquer this dilemma, PDQ makes switches cooperatively gather flow workloads information; (2) the scheduler may pause existing tasks for supporting the distributed environments. Thus, PDQ preemptively assigns bandwidth to critical flows in distributed scheduling layer.

C. Memory Caching and Resource Allocation

As plenty of memory is deployed in data centers, in-memory caching of inputs can be employed to accelerate job processes [135], [136]. I/O-intensive phases which read raw data and write parsed output are common in data-intensive jobs. Ananthanarayanan *et al.* [53] proposed Pacman, a service coordinating approach for accessing distributed caches, using memory locality to store input data.

As a near-optimal transport, pFabric [75] implements the kernel algorithm with flow-level scheduling and rate control. By keeping queues empty, pFabric accepts ECN-based feedback and employs adaptive congestion control to reduce FCT. Meanwhile, other frameworks [6], [127] determine explicit transmission rates with detailed flow status and identify the bottleneck to improve the utilization among switches. Compared with these approaches, pFabric is more feasible for implementation. When a new packet arrives, the switches need to decide whether it can be accepted, because the buffer capacity is small (e.g., dozens of kilobytes in per-port evaluation). When the buffer is full, packets with lower priority in buffer will be dropped. Across the entire fabric, an approximately optimal scheduling algorithm containing local and greedy strategy is employed. Thus, pFabric conducts rate calculation in switches without any requirements of flow information [134] to guarantee deadlines and minimize average FCT.

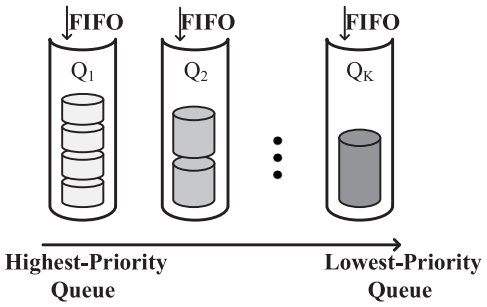


Fig. 8. Discretized Coflow-Aware Least-Attained Service: consecutive queues handle coflows with exponentially larger size. The priority of a queue decreases from left to right. In each queue, cylinders represent the coflows prepared for scheduling. Note that coflows are scheduled in a FIFO order.

Moreover, avoiding starvation is also a tough problem. In existing solutions, FIFO-LM can dynamically change the multiplexing level to make heavy tasks not block the small ones arriving later. Focusing on task awareness, FIFO-LM optimizes the bursty arrivals. Thus, a task only influences the one arrives next. Meanwhile, Dogar *et al.* proposed Barrat, a task-aware decentralized system, which assigns an unique priority to all flows in a task. Moreover, *Smart-Priority-Class* (SPC) technology is employed in Barrat, leveraging both rate control and work conservation.

Furthermore, Rapier [79] formulates a joint optimization model, which combines routing and scheduling to minimize CCT. Zhao *et al.* [79] proposed a heuristic which focuses on the relaxation of this non-linear programming model. Inspired by Orchestra [55] and Varys [38], Rapier works in a centralized manner. Information of network resources and coflow status are collected to estimate time consumption. Note that Rapier uses *Equal-Cost-Multi-Path* (ECMP) to send small coflows. The remaining bandwidth is assigned with work conservation. Thus, coflow scheduling is controlled by an algorithm called *Minimum-Remaining-Time-First* (MRTF) [6], [38] when a coflow arrives or finishes. All unfinished coflows are managed in a preemptive scheme. In this case, a small collow can pause other coflows and acquire bandwidth first. Zhao *et al.* [79] claimed that Rapier can tackle various requirements, especially for latency-sensitive applications.

Chowdhury and Stoica [37] proposed the abstraction of *coflow*, each of which involves multiple parallel flows. They implement SEBF heuristic that is based on network bottleneck and greedily schedules coflows. Meanwhile, MADD algorithm is designed to allocate flow rates. MADD slows down other flows to match the longest one in a same coflow. Considering scheduling coflows without prior knowledge, Chowdhury and Stoica [39] proposed Aalo, which uses multiplexing scheduling strategy with FIFO in each queue. As shown in Fig. 8, the priority of each coflow will decrease when network traffic of sent bytes exceeds predefined thresholds. Furthermore, Zhang *et al.* [40] combined the machine learning techniques in CODA without modifying any applications manually. In CODA, the identification of a coflow can be operated via *distance metric learning*. This contribution significantly promotes the developments of coflow-aware scheduling.

D. Straggler Phenomenon

1) Definition: A straggler is defined as a task which has particularly long completion time comparing with normal ones [117], [137]–[139]. Stragglers will significantly increase time consumption in a stage because this stage can only complete when all tasks finished. The straggler phenomenon occurs when there is an unexpected situation during job processing, such as a slow task [140] or a slow worker [85]. In previous work, stragglers are considered as a crucial bottleneck. The processing speed will be slow when stragglers exist: they delay 47% completion time in Hadoop and 29% completion time in Dryad [16]. The *data skew* may incur stragglers [141], but it cannot explain all the causes. To alleviate the impact of stragglers, many strategies try to schedule a speculative copy when a task is delayed.

Thus, analysing the causes of stragglers is meaningful because it can provide ideas for conquering this problem. Existing achievements describing attributes and potential causes of stragglers can be summarized into three aspects: (1) Stragglers are impacted by output skew instead of input data in terms of progress rate. (2) Stragglers can be detected when the first task of one stage belongs to a specific application. For example, the *just-in-time* compilation of Java may cause straggler, because the code is optimized during Java running time but may be performed beyond thresholds [142]. (3) Hardware malfunction and bad configurations are also potential causes of straggler [112], [112]. For example, JVM garbage [143] collection and I/O operations (e.g., disk and network) may incur staggers.

2) Speculation Technology: Many strategies employ task speculation [140], [144]–[148] in order to alleviate the impact of straggler. Previous works show that speculation makes up a quarter of entire tasks. As producing speculative copies of tasks may affect resource usage, speculation needs to be carefully adopted. If speculation is too aggressive, it will have a serious side effect on other jobs. Fortunately, some speculation strategies are independent of job scheduling in both design and operation. In these cases, schedulers will dispatch duplicate speculative tasks in available slots as far as possible [15], [82], [89], [149].

Hopper [140] is also a system which mitigates the impact of stragglers well. It can be implemented in both centralized and decentralized scheme. In centralized scheme, Hopper processes jobs in an ascending order of their virtual sizes, giving each job a desired allocation until all slots are exhausted. Meanwhile, in decentralized scheme, a Sparrow-based [74] architecture which consists multiple schedulers and workers is employed. Fig. 9 displays the advantages of Hopper in terms of making coordinated decision. By using speculation technology, Hopper adopts an idle slot for re-execution.

E. Assistive Technologies

1) Congestion and Convergence: To minimize FCT in TCP, Hong *et al.* [6] proposed PDQ to make a flow with higher priority run faster by pausing other flows with lower priority. In many cases, multiplexing strategy can avoid *head-of-line* blocking. For example, Baraat [140] schedules tasks in a FIFO order, dynamically changing the priority level in multiplexing.

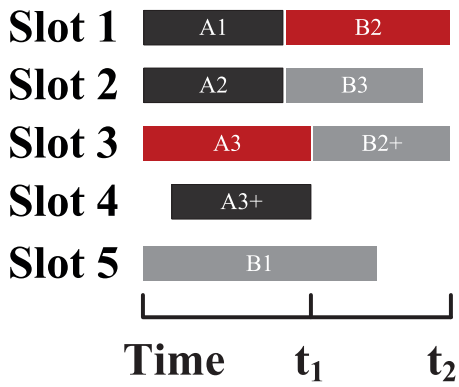


Fig. 9. Timing diagrams of two jobs in Hopper: job A (dark/black) and job B (light/gray) are operated in different slots. Two stragglers (red) are relaunched. Tasks with "+" suffix (B2+ and A3+) are the copies of speculation.

To avoid missing deadlines, a slow task will not be waited by its parent task. A network protocol which allows tighter network budgets can give more computation time to produce higher-quality responses. Note that a *fan-in* burst may appear when a children node responds to a parent node [127], [128], [150]. A burst where varied flows are sharing network may lead to TCP retransmission and congestive packet drops. Alizadeh *et al.* [128] claimed that the *fan-in* bursts are common in OLDI applications. Solutions for alleviating the burst can be summarized into two approaches: (1) The first approach is to absorb the burst with more cost with over-provision network bandwidth; (2) The second one is to increase the number of machines to add network time budget, but leaving less time for computation. Note that these two approaches may incur a worse situation due to the increment of *fan-in* degree.

To alleviate side effects of *fan-in* bursts, technologies of congestion control are widely employed. As a typical congestion control algorithm, the *Rate-Control-Protocol* (RCP) [130], [151] can be described in two phases: (1) In measurement phase, RCP must balance the noise of congestion signals via receiving multiplexing packets; (2) In reaction phase, RCP moves towards the optimal direction using *gradient descent* algorithm to tune received signals. In order to accelerate above procedures, an iteration step will be set in a large scale. Moreover, perpetual overshooting and system instability should also be considered, i.e., making a trade off between efficiency and robustness. In terms of working mechanism, RCP assigns a rate R (τ) to every concurrent flow and can be defined as:

$$\mathrm{R}(\tau) = \mathrm{R}(\tau - p_0) + \frac{\left[\eta(C - \zeta(\tau)) - \gamma \frac{S(\tau)}{p_0}\right]}{\hat{N}(\tau)}, \qquad (5)$$

where p_0 is average RTT of all packets, $\mathrm{R}(\tau - p_0)$ is previously updated rate, C is link capacity, $\zeta(\tau)$ is the input traffic rate during last update interval, $S(\tau)$ is instantaneous queue size, and $\hat{N}(\tau)$ is the estimated number of ongoing flows, calculated as $\hat{N}(\tau) = \frac{C}{\mathrm{R}(\tau - p_0)}$. Note that system stability and

performance parameters may influence the convergence of R(τ), where $0 < \eta$ and $\gamma < 1$. Note that $C - \zeta(\tau)$ represents available bandwidth and $\gamma \frac{S(\tau)}{p_0}$ is the remaining bandwidth in a queue.

Therefore, we can draw a conclusion that link speed, i.e., RTT may have a significant influence of job performance. Fortunately, over the past decade, the link speed have been rapidly improved. Network transfers can be completed with fewer RTTs. Hence, solving end-to-end latency and queuing delay have become two dominant issues. Allocating proper rates to each flow quickly is also significant. Network-level metrics (e.g., FCT) are affected by the convergence time of congestion control. Moreover, with higher link rates, most flows can be transmitted with fewer RTTs. If a flow consumes a long time due to congestion, other flows may also spend more time in transmission. As existing protocols (e.g., DCTCP [128], RCP [130], XCP [131]) iteratively react to congestion signals, the link speed may significantly impact flow performance.

However, detecting congestion signals requires a huge time consumption. Traditional congestion control algorithms converge slowly because routers have to calculate a current rate to estimate the flow-path before sending the feedback information. As flows on different paths share congested links interdependently, the final converged rate cannot be obtained precisely from routers in advance. Hence, routers have to update the rate and react to latest information which is passively gained. Furthermore, the robustness and scalability in algorithm design should also be taken into consideration by adjusting rates cautiously.

To implement congestion control, introducing proactive algorithms is a natural idea. Most proactive algorithms handle flow transmission rate according to global information because a global control requires fewer iterations. Typically, algorithms based on *Max-Min* fairness allocation are suitable to a given set of flows and links. These algorithms will converge if all flows are transmitted with a fair-sharing rate. In proactive algorithms, two properties are contained in congestion control: (1) With the congestion signals (e.g., queues, traffic volume), rates can be calculated independently. (2) There is no need to employ gradual adjustment for algorithm convergence; Based on active flows, rates can be calculated explicitly. Thus, proactive algorithms converges quickly by avoiding measurement of gradient descent phases. To flows with small RTTs, some challenges (e.g., flow busty, small RTTs) need to be conquered in congestion control. An effective congestion control should meet job demands (e.g., no data loss, quick convergence, high utilization, low buffer occupation) and prevent conflicts.

Moreover, in deployment of congestion control, ExpressPass [80] is a typical delay-bouned framework based on end-to-end scheduling, which uses credit packets to control data packets. In addition, *Remote-Direct-Memory-Access* (RDMA) is implemented to guarantee the requirements with a strict latency. Reacting in accurate fashion of early congestion signals is also a reasonable approach, which uses ECN-based algorithms [128], [131], [133], [152] to prevent network latency [153]–[155]. Moreover, in terms of avoiding data loss, *Priority-Flow-Control* (PFC) focuses on providing

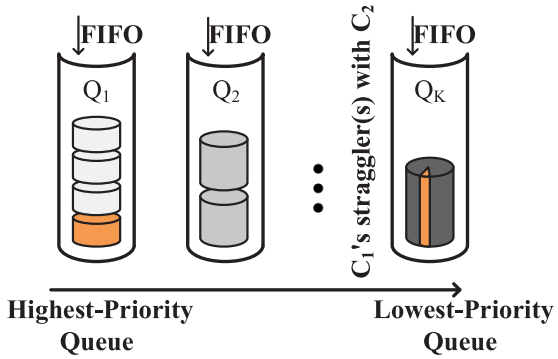


Fig. 10. CODA with machine learning: as an improvement on Aalo, CODA uses machine learning technologies to identify coflows. Two identical coflows (C_1 and C_2) sharing the same bottleneck link arrive at the same time. Note that C_1 (orange) is in a high-priority queue and C_2 (black) is in a low-priority queue. The orange cylinder in the bottom of Q_1 represents C_1 and the orange sectors in the cylinder of Q_K are the stragglers in C_1. Stragglers in C_1 are blocked by C_2, while other lower-priority coflows (grey) can complete earlier.

an aggressive increment. However, these rate-control technologies may still face a number of challenges [80]. Thus, an alternative is to make an explicit decision using a distributed controller [151].

2) Machine Learning in Scheduling: The rapidly developed machine learning technologies pioneer a new direction for promoting scheduling performance [145], [147], [156]–[160].

As shown in Fig. 10, CODA uses a machine learning algorithm called *Rough-Density Based Spatial Clustering Applications with Noise* (R-DBSCAN) [161] to identify coflow. Rought-DBSCAN is a variant of DBSCAN, which achieves a balance between training accuracy and convergence speed. R-DBSCAN can automatically control the cluster dimension according to a radius parameter. Thus, it is suitable to employ R-DBSCAN for incremental classification with the information of dynamic flow arrival and completion. Note that R-DBSCAN is deployed in a multi-level scheme and users do not need to modify applications. Meanwhile, CODA explores both explicit and implicit attributes. More precisely, explicit attributes represent flow characteristics containing traffic size and arrival time, while implicit attributes capture transmission patterns for designing data-parallel frameworks. Zhang *et al.* claimed that traditional traffic identification algorithms [24], [162]–[170] are not suitable for the situation associated with coflows. Because a coflow is a collection of semantic-related flows that cannot be predicted in advance and needs periodical identification. Therefore, DBSCAN is adopted as a fundamental algorithm for meeting the requirements in design. Moreover, to recognize a coflow, CODA requires improving accuracy and speed. Thus, R-DBSCAN is proposed which has an obvious superiority over DBSCAN in evaluation. The procedure of R-DBSCAN can be described in three steps: (1) Scanning dataset to derive leaders and their followers; (2) Running an algorithm with the same radius in DBSCAN, but adopting the set of leaders to derive clusters; (3) Deriving the cluster of flows from identified cluster of leaders, based on leader-follower relationships. Note that R-DBSCAN has not only a much lower complexity, but also fewer loss of accuracy compared with DBSCAN.

TABLE III
SUMMARY OF METRICS IN CLUSTER FRAMEWORKS

Approaches	Addressed Issue	Proposed solution	Pros	Cons
Hedera [54]	Enable near-optimal bisection bandwidth	Periodical refresh of network-wide communication requirements	Fail-over and scalability	High energy requirements
Orchestra [55]	Allocate computation and network resource	ITC Global coordination FIFO	Broadcast completion times completion time of transfers	Far from optimation
D^3 [127]	Serve users in a timely fashion	Deadline-Driven Delivery control protocol	Short flow latency Burst tolerance Large peak load	Hurt latter requests Hard to coexist
Chronos [56]	Make coflows meet their deadlines	Capture the relationship of flows among a coflow and allocate the resource	Well performance in deadlines	Prior knowledge acquiring
D^2TCP [132]	Deadline-aware Bursty Coexist with legacy TCP	Distributed and reactive bandwidth allocation ECN feedback congestion control	Well performance in deadlines	Hurt latter requests Fundamentally constrained for estimate flow rate
PDQ [6]	Deadline-aware Optimal FCT	Enables flow preemption	Well performance in deadlines Stable to packet loss	Complex to implement
Sparrow [74]	Scheduling highly parallel jobs in short time	Batch sampling Late binding Policies and Constraints	Scalability and low latency	Under optimal coordination
pFabric [75]	Optimal flow completion times for enormous short flows and large flows	Decouple flow scheduling from rate control	Near-optimal performance	Agnostic coflow
Adia [42]	Maximizing link utilization reasonably	Coordinate bandwidth between downlink and uplink consider link and coflow efficiency	Well performance in coflow complete time	Prior knowledge acquiring
Varys [38]	Application-level requirements	SEBF based on bottlenecks MADD	Maintain network utilization Starvation freedom	Prior knowledge acquiring
Barrat [140]	Treat flows as part of a task	FIFO-LM or FIFO Limited multiplexing	Outperforms pFabric, Orchestra	Not Applicable
Rapier [79]	Routing and scheduling for coflow	Combines routing and scheduling Minimum remaining time first	Implementable Significantly reduces CCT	Not Applicable
Aalo [39]	Scheduling without prior knowledge Different coflow distributions	Coflow-Aware Least-Attained Service Multi-level scheduler	High performance in presence of cluster dynamics	Not Applicable

Moreover, technologies of machine leaning can be adopted to predict time consumption of workloads. Ousterhou *et al.* [171] claimed that the mean error between estimation time and realistic time is within 9%. Based on this technology, MonoSpark [171] is proposed. The estimation controller in MonoSpark employs a sequence of matrix multiplications to solve the least-square problems in a 15-machine cluster where each machine has two SSD disks. Every multiplication contains a huge matrix which has multiple rows and columns. There are three aspects to explain why these workloads are different from formers: (1) MonoSpark improves the performance of CPU [172]. Matrices are considered as patterns of doubles which can be serialized rapidly; (2) Numerous data is sent between different stages. This can be combined into CPU optimization to improve the performance of data-intensive applications; (3) The storage of workloads is in memory instead of disks.

Moreover, VideoStorm [173] also uses machine learning technologies to analyse the streams. VideoStorm estimates each configuration which is identified as *F1* score [174], making a tradeoff between precision and recall.

V. EVALUATION METRICS

To evaluate the performance of diverse cluster frameworks, adopting appropriate metrics is important. In this section, we will compare a number of typical metrics which are employed in existing frameworks. As shown in Table III, we summarize existing cluster frameworks and compare them in different metrics.

A. Existing Metrics

As cluster frameworks are widely deployed in DCNs, data-parallel jobs can be efficiently processed via distributed systems, which manipulate huge amounts of data. Due to the high cost of hardware, it is natural to make full use of resources (e.g., CPU and bandwidth). Moreover, the application-level performance should also be optimized when cluster frameworks are processing big data analytics (e.g., Web service and user interactions). To achieve these goals, several optimization metrics have been proposed, including minimizing average time consumption and maximizing resource utilization. However, many efforts paid to promote the performance of data-intensive applications meet an optimization bound, i.e., network-level improvements are agnostic to job-specific communication requirements. This restriction often hurts application-level performance, even though the network-oriented metrics like FCT and fairness are improved. Fortunately, the abstraction of *coflow* [37] is proposed to conquer this mismatch. A coflow is a collection of parallel flows sharing a number of common performance goals. This abstraction can be introduced into framework archtecture and algorithm design to further minimize time consumption and guarantee predictable deadlines. However, existing flow-level scheduling schemes are insufficient to optimize the

coflow-level performance, because a coflow cannot complete until all associated flows are finished. Note that reducing the average CCT in a job can decrease the corresponding JCT. Meanwhile, making more coflows meet their deadlines can guarantee the purpose of application-level prediction. We observe that the coflow-level metrics can be classified into two main categories: (1) decreasing communication time (e.g., minimizing average CCT), and (2) guaranteeing predictable communication time (e.g., meeting deadlines).

Unfortunately, two metrics above are conflicting in essence. The former needs to avoid *head-of-line* blocking in preemptive implementations. Meanwhile, the latter requires *admission control* to ensure applications meet their deadlines. Thus, it is hard to satisfy these two requirements simultaneously. We need to make a tradeoff between them and take some desirable factors (e.g., starvation freedom and work conservation) into algorithm design. We enumerate two typical implementations: (1) Varys adopts the *Shortest-Remaining-Time-First* (SRTF) to handle coflow scheduling in a data center fabric with multiple ports and channels. Note that the *preemption* scheme in SRTF may lead to starvation in the worst case. (2) Based on Varys, Chen *et al.* [43] proposed a scheduler to minimize CCT and maximize resource utility, using the *Max-Min* Fairness. Moreover, a concept called *Completion-Time-Dependent-Utility* is introduced in this scheduler to describe time-sensitive attributes in coflows. Note that utility can be divided into three categories: completion time critical, completion time sensitive and completion time insensitive, each of which can reflect the final profits over time. Based on this concept, network resource is fully used and average CCT is reduced.

1) Decreasing Communication Time: In terms of decreasing communication time, minimizing average CCT is a natural idea that should be considered first. Varys employs the SEBF heuristic to schedule coflows and uses the MADD algorithm to make full use of bandwidth [38]. In algorithm design, Varys adopts the preemptive *Shortest-First* order, which may result in starvation. To estimate this side effect, Varys combines scheduling with two types of tunable time slice T and δ ($T > \delta$), ensuring that all coflows can acquire service at least once in every ($T+\delta$) interval. Note that Varys requires a prior information of coflow characteristics. Meanwhile, Aalo presents CLAS to minimize average CCT without the requirements of complete prior knowledge. According to coflow sizes, CLAS classifies coflows into multiple queues, each of which owns a specific *priority class* during scheduling. Moreover, the *weight* of a coflow decreases with the increment of traffic bytes that have been sent. CLAS prefers small coflows to large ones, so as to reduce average CCT as much as possible. Therefore, Aalo can minimize average CCT and guarantee starvation freedom but requiring no prior knowledge.

2) Guaranteeing Predictable Communication Time: Varys applies MADD with admission control to enable coflows to meet their deadlines. More precisely, MADD slows down other coexisting coflows, so as to fit the progress of the slowest coflow which will consume the longest time for finishing. As a result, all coflows can make progress and average CCT is reduced.

B. Comparison

We compare existing metrics mainly in two aspects: mechanism guideline and deployment performance.

1) Mechanism Guideline: In terms of reducing communication time, Baraat [5] and Orchestra [55] make compromises between preemption and multiplexing to avoid *head-of-line* blocking via FIFO order. Instead of emploing FIFO, Varys optimizes coflow scheduling by adopting SEBF heuristic and MADD algorithm, but it requires complete prior knowledge of coflow characteristics (e.g., the number of flows, traffic sizes and cluster status). Meanwhile, to guarantee the predictable communication times, Chen *et al.* [43] proposed the idea that meeting task deadlines should be jointly considered with optimizing resource utility, using *Max-Min* fairness.

2) Deployment Performance: As long response time can result in significant hurt job performance, different metrics are presented to reflect the feasibility of both systems and algorithms. In terms of decreasing communication time, Orchestra schedules flows by up to 4.5× faster than Hadoop. Note that Baraat outperforms pFabric and Orchestra in various workloads. To guranteee predictable deadline, Varys operates communication stages by up to 3.16× faster than the per-flow mechanisms on average. Meanwhile, Varys makes up to 2× more coflows meet their deadlines. Moreover, Aalo also completes jobs by up to 1.93× faster on average, compared with per-flow mechanisms. Finally, Chronos ensures 1.6× more coflows meet their deadlines compared with existing flow-level schemes.

VI. Desirable Properties

To design an efficient cluster framework, a number of desirable properties should be taken into consideration, including robustness and efficiency. We classify above properties into five categories: (1) fault tolerance, (2) scalability, (3) starvation freedom, (4) work conservation, and (5) resource prediction.

A. Fault Tolerance

Failures pervasively exist during the executions of data-intensive applications [16]. The main solution to fault tolerance is to restart the master and other slave nodes [175], [176].

In Dryad, failure policy in default can apply to the normal situation in which each vertex program is deterministic. As the communication pattern can be aggregated as a *Directed-Acyclic-Graph* (DAG), it is natural to ensure each job with immutable inputs to fetch the same results, regardless of network and disk failures during executions. Dryad introduces non-deterministic vertices into the framework to make their strategy feasible. A scalable mechanism [16] is to allow non-standard applications for user interactions.

RDD is a *distributed memory data* structure widely used in Spark [17]. RDD allows programs to be executed in memory without the consumption of I/O-bound processes (e.g., disk operation and network transmission), so as to accelerate data-intensive applications. Different from the *distributed shared*

memory technologies, RDD is produced via coarse-grained transformations, avoiding operations on an arbitrary memory location. Although RDD is restricted to accomplish bulk inputs due to these coarse-grained transformations, RDD still has a robust performance on fault tolerance. In Spark, failures of partial tasks can be restored and the global process still can be executed smoothly. All *in-flight* phases will be re-launched by the central handler instantly when a distributed scheduler fails. Due to the fact that tasks may be executed repetitively, RDD needs to ensure the idempotent property in operations for re-launching tasks.

Meanwhile, in Hedera, each scheduler should be responsible for both link and switch failures during resource allocation and job execution. In order to design a simple architecture, the PortLand-based routing and fault tolerance protocols are employed in the implementation of Hedera [54]. Moreover, the standard PortLand mechanisms in Hedera can catch failures during scheduling and re-route flows which are mapped to failures.

Although some frameworks adopt the decentralized schedulers to alleviate the issues of scalability and fault tolerance, failures in job execution still exist. For instance, Sparrow [74] employs a decentralized scheduler, elaborating the design to guarantee fault tolerance. As the schedulers in Sparrow do not have any logically centralized feature, the failure on one scheduler will not impact others. In order to restore a broken scheduler, the framework needs to recognize the failure and find a backup scheduler to connect. To implement the property of fault tolerance, Sparrow owns a Java client that handles failures between Sparrow schedulers. The client accepts a list of schedulers associated with all applications and connects to the first scheduler in the list. The client needs to notify the scheduler when to send messages properly. If the scheduler is not working regularly, the client will link to the next scheduler. In this way, Sparrow can make correct strategies to handle in-flight tasks if a scheduler fails. Thus, Sparrow will restart failure jobs to promote the processes of different applications, instead of acquiring the information of jobs which are launched by failure schedulers.

Moreover, in terms of system robustness, Aalo considers three failure scenes: (1) A daemon will not block job execution when it is failed, because the client libraries will haul off until the daemon is restarted. Thus, the daemon is still asynchronous to the later coordination procedure. (2) Client libraries pay attention to flow size and will attempt to reconnect to the coordinator regularly when meeting failures. (3) To restart a failed task, related flows are re-launched by matching job schedulers.

Furthermore, CODA [40] solves failures on master node via re-execution. The status of re-executed master can be rebuilt with latest updated wave. Only after the later scheduling are prepared, restored agents can coordinate coflows with each other. Due to the requirements of high precision, it is necessary to correctly identify and detect failures in reality. Note that, if these failures frequently occur, the performance of coflow scheduling will be significantly influenced. As a result, the kernel idea in CODA is to develop a robust scheduler for fault tolerance.

B. Consistency

As the machine learning applications dominate the big data applications, existing distributed systems trend to provide machine learning features. In the design of these distributed machine learning systems (DMLS), consistency-control is a crucial problem [177], [178]. The inconsistency may potentially retard the convergence speed in many machine learning (ML) algorithms, e.g., the Stochastic Gradient Decent (SGD) and Latent Dirichlet Allocation (LDA). Thus, various consistency mechanism are proposed to mitigate the impact of inconsistency. There are three main categories of consistency machenisms: Bulk Synchronous Parallel (BSP) [67], [179]–[181], Stale Synchronous Parallel (SSP) [182], and Asynchronous Parallel (ASP) [177], [183].

SSP is widely accepted (SSP is better than BSP and ASP in most pratical scenarios) by many DMLS [85], [184], [185]. The key idea in SSP is to make fast workers "wait" for slow ones, ensuring the gap of iteration progress between the fastest one and the slowest within s steps. Unfortunately, the stale policy in SSP may cause the waste of computation resource on fast workers. More seriously, in the scenario of cluster computing and datacenter networks, this phenomenon will be more ubiquitous. In addition, the heterogeneity of hardware resources (e.g., CPU and GPU) and network topology may also aggravate the difficulty of SSP deployment, even making SSP inapplicable in real productions.

Consequently, the key of designing an efficient DMLS is to make a trade-off between system efficiency and algorithm convergence speed in iteration procedures. It can be translated into a goal about making full use of system resource (both in computation and communication), while not hurting the performance of consistency-control when synchronizing (updating and sharing) parameters across the network.

C. Scalability

To achieve the requirements of scalability in frameworks, overheads of job rescheduling and information aggregation cannot be ignored due to their immediate influence on system performance [73], [186]. Meanwhile, it is important that cluster frameworks should provide HPC capacity to tackle a large scale of data-intensive applications. Thus, a tradeoff should be made between robustness and efficiency.

Orchestra [55] coordinates data movement with source TCs to guide transfer continuously. For a global coordination, an ITC manages numerous TCs, each of which selects a most appropriate mechanism for data transfer. An ITC only has to notify the cluster status to relevant TCs, thus it is feasible to achieve the requirements of fault tolerance and scalability. If an ITC is down, other existing transfers can finish their work continually. Even the crashes in reconnection are pervasive, a standby TC still can recover the state quickly.

Sparrow also demonstrates the superiority of distributed systems in terms of scalability. Considering a scenario that thousands of data-intensive applications need to be executed with short deadlines in DCNs. Such environment will generate millions of network traffic. Thus, Sparrow aims to improve job throughput. The core idea in Sparrow is to provide

scalability on distributed worker nodes, because executing low latency workloads is the main requirement on decentralized frameworks, compared with centralized ones.

Furthermore, Adia claims that a fine-grained scheduler cannot guarantee enough scalability. The flow maintenance in a specific coflow may slow down entire progress of job execution. To avoid huge overheads in a centralized framework, Adia is implemented to dispatch fine-grained tasks to each daemon through network.

D. Starvation Freedom

Preemptive scheduling algorithms, usually implemented in a priority-based manner, may suffer from starvation in the worst case. Flows with lower priority may persistently wait for the ones with higher priority in terms of resource (e.g., bandwidth) allocation and scheduling. Adia ensures starvation freedom by prioritizing links according to traffic load and time consumption dynamically. As a result, the priority of a link will rise if the transmission has been executed for a long time. Moreover, each link softly reserves a portion of bandwidth (donated as a α) for flows, which belong to a low priority coflow. On the contrary, a flow belonging to a high priority coflow can acquire at most $(1-\alpha)$ of uplink bandwidth if other flows on the same link are pending. Note that the pending flows will fairly share the reserved bandwidth. By combining *aging* and *multiplexing*, Adia ensures no flow or coflow will be perpetually suspended. Meanwhile, the FIFO-LM algorithm in Barrat [5] can dynamically change the level of multiplexing and ensure heavy tasks will not block light ones. Furthermore, employing user-defined thresholds is also a feasible solution [79]. In this approach, a coflow which has waited for a long time will be assigned with a higher priority, thus will get more opportunities for scheduling.

E. Work Conservation

Work conservation guarantees that a task in lower priority can get opportunity for scheduling, if the highest priority task cannot make full use of the resources (e.g., the network bandwidth). In priority-based scheduling, bandwidth is allocated to flows belonging to the fastest coflow on the *in-flight* links. The remaining bandwidth is assigned according to the priorities of flows and the procedure of rate allocation can finish when all the links are saturated and no flow is suspended. Employing work conservation can help to backfill the unsaturated resources into priority queues, making the remaining bandwidth allocated to coflows as much as possible, so as to facilitate the transmission progress.

Taking Adia [42] as an example, Adia schedules coflows based on priority classes. The fastest coflow in a congested link has the highest priority to obtain bandwidth allocation. Meanwhile, the remaining bandwidth is allocated to other coflows based on their priorities. Adia ensures that available bandwidth is fully used with integrated backfilling, so as to guarantee work conservation.

F. Resource Prediction and Measurement

In a framework design, guaranteeing the predictability on resource is also an important issue. At present, many features can be estimated through mathematical methods. For example, the link status in Adia is estimated by daemons when the corresponding ports are ready to receive messages. In addition, Chowdhury *et al.* [93] estimated the available bandwidth using periodic updates from slave nodes in each time interval.

As a great amount of business-critical jobs may be produced repeatedly in DCNs, Corral [122] needs to locate data accurately, using the pre-defined submission time and predictable resource demands to improve local property of network. Thus, Corral is a predictable framework which takes the advantages of job prediction to optimize data transmission and computation location. Note that the prediction accuracy in Corral is associated with a number of features (e.g., input data size and job demands). Considering these features together can help to make the error ratio in a low level. Based on prediction, Corral can figure out the input data size of each job in the recent one-month period. In addition, Corral can predict the data location in an arbitrary time with the job information in past few days.

Moreover, Mantri [89] can estimate the time consumption for data-intensive applications executed by cluster frameworks in DCNs. Ananthanarayanan *et al.* [89] claimed that reading input data in cluster can provide the possibility for assessment. In the architecture of Mantri, the information of each task will be informed to schedulers. More precisely, Mantri can analyse the status of each scheduler for predicting time consumption. Thus, Mantri makes use of the remaining time to manipulate data. Moreover, Mantri can figure out the time consumption which is estimated via the actions of previous tasks.

In addition, TopCluster [187] can estimate the cost of tasks by dispatching them to reducers. In existing distributed monitoring approaches, as adaptive load balancing algorithms are based on practical cost estimations which acquire data from mappers, the cardinality of clusters is the exclusive variable for estimation cost. Therefore, it is critical to calculate this cardinality accurately. The estimation needs to consider data skew with the column diagram of cluster frameworks. More precisely, TopCluster can figure out the *separation cost* estimations when data is in a *high-skew* distribution. Furthermore, TopCluster does not need to care about the scale of each cluster, but only aims to figure out the average scale of entire clusters for conducting cost estimation.

VII. Open Issues and Summary

We list three typical issues which are deserved for further researches.

Network Fabric: Schedulers can be designed in a non-intrusive manner, similar to the SPC mechanism in Baraat. This manner only allows light-weight changes to the switches. In the future, proposals of programmable switches [188] can be introduced into network fabric design.

Bottleneck: As CPU and network can be the optimization bottlenecks in the worst cases, it is natural to employ the pipeline-based technologies in DCNs. Moreover, the work conservation also should be taken into consideration.

Machine Learning: Technologies in machine learning can be adopted into the implementation of cluster frameworks. Taking

CODA as an example, it employs the *distance metric learning* to identify a coflow. We believe that a number of challenges can be conquered via machine leaning technologies. Moreover, as traditional distributed systems are evolving to DMLS, the desirable properties such as data consistency and computation efficiency should be taken into consideration. It is natural to achieve a balance between these factors, so as to guarantee the synchronization effectiveness while not deteriorating the convergence accuracy.

Energy Efficiency: In real productions, the powerful computation capacity of data centers requires huge amount of energy. The energy consumption is also a crucial issue which should be taken into consideration. An applicable cluster framework needs to make a trade-off between processing performance and energy efficiency [189]–[191]. As the abstraction of *green* [192], [193] has been widely used in many researches, it is high time that data centers should support the *green* feature.

Heterogeneous Computing: A modern data center is comprised of thousands of independent machines to provide sufficient computation capacity. However, these machines may equipped with various hardwares. For instance, for processing ML applications, some machines will equip with extra powerful GPUs, while the others may only own commodity CPUs. Meanwhile, the network topology may also be heterogeneous [194]: some machines are connected through gigabyte Ethernet, while the others are located in a wireless cellular network [195], [196]. This heterogeneity will significantly hamper the data consistency and job scheduling. Consequently, conquering the heterogeneity between machines is also a crucial issue that needs to be further researched.

Summary: Data-parellel applications in data centers have attracted much attention in both academia and industry. In this article, we provide an overview of distributed frameworks and summarize a number of significant guidelines in algorithm design and system implementation. In previous researches, efforts paid to improve system performance can be classified into three aspects: network, disk and straggler. To design an efficient scheduling algorithm, many basic coordination principles need to be taken into consideration. Thus, making an appropriate strategy is a key to develop a robust distributed system. We exposit different algorithms and compare the metrics, which are adopted in state-of-the-art frameworks. Moreover, some desirable properties such as fault tolerance, scalable and starvation freedom should also be guaranteed in algorithm design. Moreover, predictability and estimability are also widely used to build HPC clusters, due to the rapid development of applications in deep learning.

We hope this survey will illuminate a promising and ponderable research branch, providing solid basic knowledge for readers to further explore this area.

REFERENCES

[1] J. Wu, S. Guo, J. Li, and D. Zeng, "Big data meet green challenges: Big data toward green applications," *IEEE Syst. J.*, vol. 10, no. 3, pp. 888–900, Sep. 2016.

[2] J. Wu, S. Guo, J. Li, and D. Zeng, "Big data meet green challenges: Greening big data," *IEEE Syst. J.*, vol. 10, no. 3, pp. 873–887, Sep. 2016.

[3] J. Wu, S. Guo, H. Huang, W. Liu, and Y. Xiang, "Information and communications technologies for sustainable development goals: State-of-the-art, needs and perspectives," *IEEE Commun. Surveys Tuts.*, to be published.

[4] H. Li, K. Ota, M. Dong, A. Vasilakos, and K. Nagano, "Multimedia processing pricing strategy in GPU-accelerated cloud computing," *IEEE Trans. Cloud Comput.*, to be published.

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

[6] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," in *Proc. SIGCOMM*, Helsinki, Finland, Oct. 2012, pp. 127–138.

[7] G. Luo, Z. Qian, M. Dong, K. Ota, and S. Lu, "Network-aware re-scheduling: Towards improving network performance of virtual machines in a data center," in *Proc. ICA3PP*, Dalian, China, Aug. 2014, pp. 255–269.

[8] P. Li *et al.*, "Traffic-aware geo-distributed big data analytics with predictable job completion time," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 6, pp. 1785–1796, Jun. 2017.

[9] X. Zhou, K. Wang, W. Jia, and M. Guo, "Reinforcement learning-based adaptive resource management of differentiated services in geo-distributed data centers," in *Proc. IWQoS*, 2017, pp. 1–6.

[10] H. Li, M. Dong, K. Ota, and M. Guo, "Pricing and repurchasing for big data processing in multi-clouds," *IEEE Trans. Emerg. Topics Comput.*, vol. 4, no. 2, pp. 266–277, Apr./Jun. 2016.

[11] X. He, K. Wang, H. Huang, and B. Liu, "QoE-driven big data architecture for smart city," *IEEE Commun. Mag.*, vol. 56, no. 2, pp. 88–93, Feb. 2018.

[12] K. Wang *et al.*, "Wireless big data computing in smart grid," *IEEE Wireless Commun.*, vol. 24, no. 2, pp. 58–64, Apr. 2017.

[13] R. Rojas-Cessa, Y. Kaymak, and Z. Dong, "Schemes for fast transmission of flows in data center networks," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 3, pp. 1391–1422, 3rd Quart., 2015.

[14] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. SIGCOMM*, Seattle, WA, USA, Aug. 2008, pp. 63–74.

[15] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. OSDI*, San Francisco, CA, USA, Dec. 2004, pp. 137–150.

[16] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proc. EuroSys*, Lisbon, Portugal, Mar. 2007, pp. 59–72.

[17] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. HotCloud*, Boston, MA, USA, Jun. 2010, pp. 1–7.

[18] W. Chen, J. Rao, and X. Zhou, "Preemptive, low latency datacenter scheduling via lightweight virtualization," in *Proc. USENIX ATC*, Santa Clara, CA, USA, Jul. 2017, pp. 251–263.

[19] L. Chen, B. Li, and B. Li, "Surviving failures with performance-centric bandwidth allocation in private datacenters," in *Proc. IC2E*, Berlin, Germany, Apr. 2016, pp. 52–61.

[20] P. Bodík *et al.*, "Surviving failures in bandwidth-constrained datacenters," *SIGCOMM Comput. Commun. Rev.*, vol. 42, pp. 431–442, Aug. 2012.

[21] L. Popa *et al.*, "ElasticSwitch: Practical work-conserving bandwidth guarantees for cloud computing," in *Proc. SIGCOMM*, Hong Kong, Aug. 2013, pp. 351–362.

[22] P. X. Gao *et al.*, "Network requirements for resource disaggregation," in *Proc. OSDI*, Savannah, GA, USA, 2016, pp. 249–264.

[23] J. E. Gonzalez *et al.*, "GraphX: Graph processing in a distributed dataflow framework," in *Proc. OSDI*, Broomfield, CO, USA, 2014, pp. 599–613.

[24] S. Zander, T. Nguyen, and G. Armitage, "Automated traffic classification and application identification using machine learning," in *Proc. LCN*, Sydney, NSW, Australia, Nov. 2005, pp. 250–257.

[25] A. Ghodsi *et al.*, "Dominant resource fairness: Fair allocation of multiple resource types," in *Proc. NSDI*, Boston, MA, USA, 2011, pp. 323–336.

[26] R. Grandl, M. Chowdhury, A. Akella, and G. Ananthanarayanan, "Altruistic scheduling in multi-resource clusters," in *Proc. OSDI*, Savannah, GA, USA, 2016, pp. 65–80.

[27] W. Wang, B. Li, B. Liang, and J. Li, "Multi-resource fair sharing for datacenter jobs with placement constraints," in *Proc. SC*, Salt Lake City, UT, USA, Nov. 2016, p. 86.

[28] S. Wang *et al.*, "A survey of coflow scheduling schemes for data center networks," *IEEE Commun. Mag.*, vol. 56, no. 6, pp. 179–185, Jun. 2018.

[29] S. Bassoy, H. Farooq, M. A. Imran, and A. Imran, "Coordinated multi-point clustering schemes: A survey," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 2, pp. 743–764, 2nd Quart., 2017.

[30] S. Yu, M. Liu, W. Dou, X. Liu, and S. Zhou, "Networking for big data: A survey," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 531–549, 1st Quart., 2017.

[31] A. Riekstin *et al.*, "A survey of policy refinement methods as a support for sustainable networks," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 222–235, 1st Quart., 2016.

[32] K. Bilal *et al.*, "A taxonomy and survey on green data center networks," *Future Gener. Comput. Syst.*, vol. 36, no. 7, pp. 189–208, 2014.

[33] A. Greenberg *et al.*, "VL2: A scalable and flexible data center network," *Commun. ACM*, vol. 54, no. 3, pp. 95–104, 2011.

[34] J. Rajahalme, A. Conta, B. Carpenter, and S. Deering, "IPv6 flow label specification," Internet Eng. Task Force, Fremont, CA, USA, RFC 3697, 2004.

[35] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 59–62, 2006.

[36] S. Dutta, V. R. Cadambe, and P. Grover, "Coded convolution for parallel and distributed computing within a deadline," *CoRR*, vol. abs/1705.03875, pp. 2403–2407, Jun. 2017, doi: 10.1109/ISIT.2017.8006960.

[37] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proc. HotNets*, Redmond, WA, USA, Oct. 2012, pp. 31–36.

[38] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with Varys," in *Proc. SIGCOMM*, Chicago, IL, USA, Aug. 2014, pp. 443–454.

[39] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," in *Proc. SIGCOMM*, London, U.K., Aug. 2015, pp. 393–406.

[40] H. Zhang *et al.*, "CODA: Toward automatically identifying and scheduling coflows in the dark," in *Proc. SIGCOMM*, Florianópolis, Brazil, Aug. 2016, pp. 160–173.

[41] W. Wang, S. Ma, B. Li, and B. Li, "Coflex: Navigating the fairness-efficiency tradeoff for coflow scheduling," in *Proc. INFOCOM*, Atlanta, GA, USA, May 2017, pp. 1–9.

[42] J. Jiang, S. Ma, B. Li, and B. Li, "Adia: Achieving high link utilization with coflow-aware scheduling in data center networks," *IEEE Trans. Cloud Comput.*, to be published.

[43] L. Chen, W. Cui, B. Li, and B. Li, "Optimizing coflow completion times with utility max–min fairness," in *Proc. INFOCOM*, San Francisco, CA, USA, Apr. 2016, pp. 1–9.

[44] W. Borjigin, K. Ota, and M. Dong, "Time-saving first: Coflow scheduling for datacenter networks," in *Proc. VTC*, Toronto, ON, Canada, Sep. 2017, pp. 1–5.

[45] R. Appuswamy, C. Gkantsidis, D. Narayanan, O. Hodson, and A. Rowstron, "Scale-up vs scale-out for Hadoop: Time to rethink?" in *Proc. SoCC*, Santa Clara, CA, USA, Oct. 2013, Art. no. 20.

[46] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements & analysis," in *Proc. IMC*, Chicago, IL, USA, Nov. 2009, pp. 202–208.

[47] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 92–99, 2010.

[48] M. Nuyens, A. Wierman, and B. Zwart, "Preventing large sojourn times using SMART scheduling," *Oper. Res.*, vol. 56, no. 1, pp. 88–101, 2008.

[49] J. Nair, A. Wierman, and B. Zwart, "Tail-robust scheduling via limited processor sharing," *Perform. Eval.*, vol. 67, no. 11, pp. 978–995, 2010.

[50] M. Zaharia *et al.*, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. NSDI*, San Jose, CA, USA, Apr. 2012, p. 2.

[51] T. Condie *et al.*, "MapReduce online," in *Proc. NSDI*, San Jose, CA, USA, Apr. 2010, p. 21.

[52] C. J. Rossbach, Y. Yu, J. Currey, J.-P. Martin, and D. Fetterly, "Dandelion: A compiler and runtime for heterogeneous systems," in *Proc. SOSP*, Farmington, CT, USA, Nov. 2013, pp. 49–68.

[53] G. Ananthanarayanan *et al.*, "PACMan: Coordinated memory caching for parallel jobs," in *Proc. NSDI*, San Jose, CA, USA, Apr. 2012, p. 20.

[54] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. NSDI*, San Jose, CA, USA, Apr. 2010, p. 19.

[55] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with Orchestra," in *Proc. SIGCOMM*, Toronto, ON, Canada, Aug. 2011, pp. 98–109.

[56] R. Kapoor, G. Porter, M. Tewari, G. M. Voelker, and A. Vahdat, "Chronos: Predictable low latency for data center applications," in *Proc. SoCC*, San Jose, CA, USA, Oct. 2012, Art. no. 9.

[57] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. MSST*, Incline Village, NV, USA, May 2010, pp. 1–10.

[58] *Aparch Hadoop*. Accessed: Jul. 2008. [Online]. Available: http://hadoop.apache.org

[59] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: The Condor experience: Research articles," *Concurrency Comput. Pract. Exp. Grid Perform.*, vol. 17, nos. 2–4, pp. 323–356, Feb. 2005.

[60] W. R. Mark, R. S. Glanville, K. Akeley, and M. J. Kilgard, "Cg: A system for programming graphics hardware in a C-like language," in *Proc. SIGGRAPH*, San Diego, CA, USA, 2003, pp. 896–907.

[61] D. Tarditi, S. Puri, and J. Oglesby, "Accelerator: Using data parallelism to program GPUs for general-purpose uses," in *Proc. ASPLOS*, San Jose, CA, USA, Oct. 2006, pp. 984–988.

[62] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proc. SOSP*, Oct. 2003, pp. 29–43.

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

[64] M. Li, J. Tan, Y. Wang, L. Zhang, and V. Salapura, "SparkBench: A comprehensive benchmarking suite for in memory data analytic platform spark," in *Proc. CF*, 2015, Art. no. 15.

[65] Y. Yu *et al.*, "DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language," in *Proc. OSDI*, San Diego, CA, USA, Dec. 2008, pp. 1–14.

[66] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "HaLoop: Efficient iterative data processing on large clusters," *Proc. VLDB Endowment*, vol. 3, nos. 1–2, pp. 285–296, 2010.

[67] G. Malewicz *et al.*, "Pregel: A system for large-scale graph processing," in *Proc. SIGMOD*, Indianapolis, IN, USA, Jun. 2010, pp. 135–146.

[68] G. DeCandia *et al.*, "Dynamo: Amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 205–220, Oct. 2007.

[69] S. Rao, "Distributed systems: An algorithmic approach," *IEEE Distrib. Syst. Online*, vol. 9, no. 11, p. 3, Nov. 2008.

[70] D. Sun *et al.*, "Re-stream: Real-time and energy-efficient resource scheduling in big data stream computing environments," *Inf. Sci.*, vol. 319, pp. 92–112, Oct. 2015.

[71] V. Chakaravarthy, S. Kenkre, S. A. Mondal, V. Pandit, and Y. Sabharwal, "Reusable resource scheduling via colored interval covering," in *Proc. IPDPS*, Chicago, IL, USA, May 2016, pp. 1003–1012.

[72] Y. Li *et al.*, "Efficient online coflow routing and scheduling," in *Proc. MobiHoc*, Paderborn, Germany, Jul. 2016, pp. 161–170.

[73] C. Guo *et al.*, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *Proc. SIGCOMM*, London, U.K., Aug. 2015, pp. 139–152.

[74] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling," in *Proc. SOSP*, Farmington, UT, USA, Nov. 2013, pp. 1–16.

[75] M. Alizadeh *et al.*, "pFabric: Minimal near-optimal datacenter transport," in *Proc. SIGCOMM*, Hong Kong, Aug. 2013, pp. 435–446.

[76] Z. Li, Y. Zhang, D. Li, K. Chen, and Y. Peng, "OPTAS: Decentralized flow monitoring and scheduling for tiny tasks," in *Proc. INFOCOM*, San Francisco, CA, USA, Apr. 2016, pp. 1–9.

[77] A. Dragojević, D. Narayanan, O. Hodson, and M. Castro, "FaRM: Fast remote memory," in *Proc. NSDI*, Seattle, WA, USA, Apr. 2014, pp. 401–414.

[78] Y. Peng *et al.*, "HadoopWatch: A first step towards comprehensive traffic forecasting in cloud computing," in *Proc. INFOCOM*, Toronto, ON, Canada, 2014, pp. 19–27.

[79] Y. Zhao *et al.*, "Rapier: Integrating routing and scheduling for coflow-aware data center networks," in *Proc. INFOCOM*, Hong Kong, 2015, pp. 424–432.

[80] I. Cho, K. Jang, and D. Han, "Credit-scheduled delay-bounded congestion control for datacenters," in *Proc. SIGCOMM*, Los Angeles, CA, USA, Aug. 2017, pp. 239–252.

[81] Z. Hu, B. Li, and J. Luo, "Flutter: Scheduling tasks closer to data across geo-distributed datacenters," in *Proc. INFOCOM*, San Francisco, CA, USA, Apr. 2016, pp. 1–9.

[82] G. Ananthanarayanan *et al.*, "GRASS: Trimming stragglers in approximation analytics," in *Proc. NSDI*, Seattle, WA, USA, Apr. 2014, pp. 289–302.

[83] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones," in *Proc. NSDI*, Lombard, IL, USA, Apr. 2013, pp. 185–198.

[84] X. Ouyang, P. Garraghan, R. Yang, P. Townend, and J. Xu, "Reducing late-timing failure at scale: Straggler root-cause analysis in cloud datacenters," in *Proc. DSN-FAST-ABSTRACT*, Toulouse, France, Jun. 2016, p. 2.

[85] A. Harlap *et al.*, "Addressing the straggler problem for iterative convergent parallel ML," in *Proc. SoCC*, Santa Clara, CA, USA, Oct. 2016, pp. 98–111.

[86] C. Guo *et al.*, "BCube: A high performance, server-centric network architecture for modular data centers," in *Proc. SIGCOMM*, Barcelona, Spain, Aug. 2009, pp. 63–74.

[87] C. Guo *et al.*, "DCell: A scalable and fault-tolerant network structure for data centers," in *Proc. SIGCOMM*, Seattle, WA, USA, Aug. 2008, pp. 75–86.

[88] R. N. Mysore *et al.*, "PortLand: A scalable fault-tolerant layer 2 data center network fabric," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 39–50, 2009.

[89] G. Ananthanarayanan *et al.*, "Reining in the outliers in map-reduce clusters using Mantri," in *Proc. OSDI*, Vancouver, BC, Canada, Oct. 2010, pp. 265–278.

[90] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine grained traffic engineering for data centers," in *Proc. CoNEXT*, Tokyo, Japan, Dec. 2011, Art. no. 8.

[91] W. Bai *et al.*, "Information-agnostic flow scheduling for commodity data centers," in *Proc. NSDI*, Oakland, CA, USA, May 2015, pp. 455–468.

[92] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proc. SIGCOMM*, Toronto, ON, Canada, Aug. 2011, pp. 242–253.

[93] M. Chowdhury, S. Kandula, and I. Stoica, "Leveraging endpoint flexibility in data-intensive clusters," in *Proc. SIGCOMM*, Hong Kong, Aug. 2013, pp. 231–242.

[94] P. Costa, A. Donnelly, A. Rowstron, and G. O'Shea, "Camdoop: Exploiting in-network aggregation for big data applications," in *Proc. NSDI*, San Jose, CA, USA, Apr. 2012, p. 3.

[95] Z. Guo *et al.*, "Spotting code optimizations in data-parallel pipelines through PeriSCOPE," in *Proc. OSDI*, Oct. 2012, pp. 121–133.

[96] V. Jeyakumar *et al.*, "EyeQ: Practical network performance isolation at the edge," in *Proc. NSDI*, Lombard, IL, USA, Apr. 2013, pp. 297–312.

[97] L. Popa *et al.*, "FairCloud: Sharing the network in cloud computing," in *Proc. SIGCOMM*, Helsinki, Finland, Aug. 2012, pp. 187–198.

[98] P. Prakash, A. Dixit, Y. C. Hu, and R. Kompella, "The TCP outcast problem: Exposing unfairness in data center networks," in *Proc. NSDI*, San Jose, CA, USA, Apr. 2012, p. 30.

[99] D. Xie, N. Ding, Y. Hu, and R. Kompella, "The only constant is change: Incorporating time-varying network reservations in data centers," in *Proc. SIGCOMM*, Helsinki, Finland, Aug. 2012, pp. 199–210.

[100] J. Zhang *et al.*, "Optimizing data shuffling in data-parallel computation by understanding user-defined functions," in *Proc. NSDI*, San Jose, CA, USA, Apr. 2012, p. 22.

[101] H. Susanto, H. Jin, and K. Chen, "Stream: Decentralized opportunistic inter-coflow scheduling for datacenter networks," in *Proc. ICNP*, Singapore, Nov. 2016, pp. 1–10.

[102] Y. Gao, H. Yu, S. Luo, and S. Yu, "Information-agnostic coflow scheduling with optimal demotion thresholds," in *Proc. ICC*, May 2016, pp. 1–6.

[103] Z. Li, Y. Zhang, Y. Zhao, and D. Li, "Efficient semantic-aware coflow scheduling for data-parallel jobs," in *Proc. CLUSTER*, Sep. 2016, pp. 154–155.

[104] M. Chowdhury, Z. Liu, A. Ghodsi, and I. Stoica, "HUG: Multi-resource fairness for correlated and elastic demands," in *Proc. NSDI*, Santa Clara, CA, USA, 2016, pp. 407–424.

[105] J. Zhang *et al.*, "Data rate guarantee for coflow scheduling in network function virtualization," in *Proc. IEEE/ACM 24th Int. Symp. Qual. Service (IWQoS)*, 2016, pp. 1–6.

[106] A. Sivaraman *et al.*, "Programmable packet scheduling at line rate," in *Proc. SIGCOMM*, Florianópolis, Brazil, Aug. 2016, pp. 44–57.

[107] M. Gao, K. Wang, and L. He, "Probabilistic model checking and scheduling implementation of an energy router system in energy Internet for green cities," *IEEE Trans. Ind. Informat.*, vol. 14, no. 4, pp. 1501–1510, Apr. 2018.

[108] C. Xu, K. Wang, and M. Guo, "Intelligent resource management in blockchain-based cloud datacenters," *IEEE Cloud Comput.*, vol. 4, no. 6, pp. 50–59, Nov./Dec. 2017.

[109] A. Rasmussen *et al.*, "Themis: An I/O-efficient MapReduce," in *Proc. SoCC*, San Jose, CA, USA, Oct. 2012, Art. no. 13.

[110] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Tachyon: Reliable, memory speed storage for cluster computing frameworks," in *Proc. SoCC*, Seattle, WA, USA, Nov. 2014, pp. 1–15.

[111] L. Wei, W. Lian, K. Liu, and Y. Wang, "Hippo: An enhancement of pipeline-aware in-memory caching for HDFS," in *Proc. ICCCN*, Shanghai, China, Aug. 2014, pp. 1–5.

[112] M. Zaharia, A. Konwinski, A. Joseph, R. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in *Proc. OSDI*, San Diego, CA, USA, Dec. 2008, pp. 29–42.

[113] X. Ouyang *et al.*, "Adaptive speculation for efficient Internetware application execution in clouds," *ACM Trans. Internet Technol.*, vol. 18, no. 2, Art. no. 15, 2017.

[114] R. O. Suminto *et al.*, "PBSE: A robust path-based speculative execution for degraded-network tail tolerance in data-parallel frameworks," in *Proc. SoCC*, Santa Clara, CA, USA, 2017, pp. 295–308.

[115] C. Chen, W. Wang, and B. Li, "Speculative slot reservation: Enforcing service isolation for dependent data-parallel computations," in *Proc. ICDCS*, Atlanta, GA, USA, Jun. 2017, pp. 549–559.

[116] Y. Guo, J. Rao, C. Jiang, and X. Zhou, "Moving Hadoop into the cloud with flexible slot management and speculative execution," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 798–812, Mar. 2017.

[117] P. Garraghan, X. Ouyang, P. Townend, and J. Xu, "Timely long tail identification through agent based monitoring and analytics," in *Proc. Int. Symp. Real Time Distrib. Comput.*, Apr. 2015, pp. 19–26.

[118] S. Lu *et al.*, "Log-based abnormal task detection and root cause analysis for spark," in *Proc. ICWS*, Honolulu, HI, USA, Jun. 2017, pp. 389–396.

[119] E. B. Khunayn, S. Karunasekera, H. Xie, and K. Ramamohanarao, "Straggler mitigation for distributed behavioral simulation," in *Proc. ICDCS*, Atlanta, GA, USA, Jun. 2017, pp. 2638–2641.

[120] K. V. Rashmi, M. Chowdhury, J. Kosaian, I. Stoica, and K. Ramchandran, "EC-cache: Load-balanced, low-latency cluster caching with online erasure coding," in *Proc. OSDI*, Savannah, GA, USA, Nov. 2016, pp. 401–417.

[121] K. Ousterhout *et al.*, "The case for tiny tasks in compute clusters," in *Proc. HotOS*, Santa Ana Pueblo, NM, USA, May 2013, p. 14.

[122] V. Jalaparti *et al.*, "Network-aware scheduling for data-parallel jobs: Plan when you can," in *Proc. SIGCOMM*, London, U.K., Aug. 2015, pp. 407–420.

[123] M. Isard *et al.*, "Quincy: Fair scheduling for distributed computing clusters," in *Proc. SOSP*, Big Sky, MT, USA, Oct. 2009, pp. 261–276.

[124] M. Zaharia *et al.*, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *Proc. EuroSys*, Paris, France, Apr. 2010, pp. 265–278.

[125] Q. Zhou *et al.* "Swallow: Joint online scheduling and coflow compression in datacenter networks," in *Proc. IPDPS*, Vancouver, BC, Canada, May 2018, pp. 1–10.

[126] A. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca, "Jockey: Guaranteed job latency in data parallel clusters," in *Proc. EuroSys*, Bern, Switzerland, Apr. 2012, pp. 99–112.

[127] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *Proc. SIGCOMM*, Toronto, ON, Canada, Aug. 2011, pp. 50–61.

[128] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," in *Proc. SIGCOMM*, New Delhi, India, Sep. 2010, pp. 63–74.

[129] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[130] N. Dukkipati, N. McKeown, and A. G. Fraser, "RCP-AC: Congestion control to make flows complete quickly in any environment," in *Proc. IEEE 25th IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Barcelona, Spain, Apr. 2006, pp. 1–5.

[131] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proc. SIGCOMM*, Pittsburgh, PA, USA, 2002, pp. 89–102.

[132] B. Vamanan, J. Hasan, and T. N. Vijaykumar, "Deadline-aware datacenter TCP (D2TCP)," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 115–126, 2012.

[133] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast congestion control for TCP in data center networks," in *Proc. Co-NEXT*, Philadelphia, PA, USA, Nov. 2010, p. 13.

[134] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload analysis of a large-scale key-value store," in *Proc. SIGMETRICS*, London, U.K., 2012, pp. 53–64.

[135] H. Garcia-Molina and K. Salem, "Main memory database systems: An overview," *IEEE Trans. Knowl. Data Eng.*, vol. 4, no. 6, pp. 509–516, Dec. 1992.

[136] J. Ousterhout *et al.*, "The case for RAMClouds: Scalable high-performance storage entirely in DRAM," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 4, pp. 92–105, 2010.

[137] W. Dai, I. Ibrahim, and M. Bassiouni, "An improved straggler identification scheme for data-intensive computing on cloud platforms," in *Proc. CSCloud*, New York, NY, USA, Jun. 2017, pp. 211–216.

[138] B. Memishi, M. S. Pérez, and G. Antoniu, "Failure detector abstractions for MapReduce-based systems," *Inf. Sci.*, vol. 379, pp. 112–127, Feb. 2017.

[139] C. Li, H. Shen, and T. Huang, "Learning to diagnose stragglers in distributed computing," in *Proc. MTAGS*, Salt Lake City, UT, USA, Nov. 2016, pp. 1–6.

[140] X. Ren, G. Ananthanarayanan, A. Wierman, and M. Yu, "Hopper: Decentralized speculation-aware cluster scheduling at scale," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 379–392, Aug. 2015.

[141] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "SkewTune: Mitigating skew in MapReduce applications," in *Proc. SIGMOD*, 2012, pp. 25–36.

[142] *Oracle*. [Online]. Available: http://www.oracle.com/

[143] K. Nguyen *et al.*, "Yak: A high-performance big-data-friendly garbage collector," in *Proc. OSDI*, Savannah, GA, USA, 2016, pp. 349–365.

[144] M. Xu, S. Alamro, T. Lan, and S. Subramaniam, "Optimizing speculative execution of deadline-sensitive jobs in cloud," in *Proc. SIGMETRICS*, 2017, pp. 17–18.

[145] X. Ouyang *et al.*, "ML-NA: A machine learning based node performance analyzer utilizing straggler statistics," in *Proc. ICPADS*, 2017, pp. 73–80.

[146] X. Ouyang, H. Zhou, S. Clement, P. Townend, and J. Xu, "Mitigate data skew caused stragglers through ImKP partition in MapReduce," in *Proc. IEEE Int. Perform. Comput. Commun. Conf.*, San Diego, CA, USA, 2017, pp. 1–8.

[147] M. Xu, S. Alamro, T. Lan, and S. Subramaniam, "LASER: A deep learning approach for speculative execution and replication of deadline-critical jobs in cloud," in *Proc. ICCCN*, Vancouver, BC, Canada, Jul. 2017, pp. 1–8.

[148] H. Xu and W. C. Lau, "Optimization for speculative execution in big data processing clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 2, pp. 530–545, Feb. 2017.

[149] J. Leung, L. Kelly, and J. H. Anderson, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Boca Raton, FL, USA: CRC Press, 2004.

[150] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding TCP incast throughput collapse in datacenter networks," in *Proc. WREN*, New York, NY, USA, 2009, pp. 73–82.

[151] L. Jose *et al.*, "High speed networks need proactive congestion control," in *Proc. HotNets-XIV*, New York, NY, USA, 2015, pp. 1–7.

[152] M. Alizadeh *et al.*, "Less is more: Trading a little bandwidth for ultra-low latency in the data center," in *Proc. NSDI*, San Jose, CA, USA, Apr. 2012, pp. 253–266.

[153] C. Lee, C. Park, K. Jang, S. Moon, and D. Han, "Accurate latency-based congestion feedback for datacenters," in *Proc. USENIX ATC*, Santa Clara, CA, USA, 2015, pp. 403–415.

[154] R. Mittal *et al.*, "Timely: RTT-based congestion control for the datacenter," in *Proc. SIGCOMM*, London, U.K., Aug. 2015, pp. 537–550.

[155] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized 'zero-queue' datacenter network," in *Proc. SIGCOMM*, Chicago, IL, USA, Aug. 2014, pp. 307–318.

[156] H. Zhang *et al.*, "Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters," *CoRR*, vol. abs/1706.03292, pp. 181–193, Jul. 2017.

[157] X. Zhao, K. Rodrigues, Y. Luo, D. Yuan, and M. Stumm, "Non-intrusive performance profiling for entire software stacks based on the flow reconstruction principle," in *Proc. OSDI*, Savannah, GA, USA, Nov. 2016, pp. 603–618.

[158] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. OSDI*, Savannah, GA, USA, Nov. 2016, pp. 265–283.

[159] Y. He *et al.*, "MR-DBSCAN: An efficient parallel density-based clustering algorithm using MapReduce," in *Proc. ICPADS*, Dec. 2011, pp. 473–480.

[160] X. He *et al.*, "Green resource allocation based on deep reinforcement learning in content-centric IoT," *IEEE Trans. Emerg. Topics Comput.*, to be published.

[161] P. Viswanath and V. S. Babu, "Rough-DBSCAN: A fast hybrid density based clustering method for large data sets," *Pattern Recogn. Lett.*, vol. 30, no. 16, pp. 1477–1488, 2009.

[162] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic classification on the fly," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 2, pp. 23–26, 2006.

[163] P. Cheeseman and J. Stutz, *Bayesian Classification (Autoclass): Theory and Results*. Menlo Park, CA, USA: Amer. Assoc. Artif. Intell., 1997, pp. 153–180.

[164] G. Murray, "Comments on 'maximum likelihood from incomplete data via the EM algorithm' by Dempster, Laird, and Rubin," *J. Roy. Stat. Soc. B*, vol. 39, pp. 27–28, Apr. 1977.

[165] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "BLINC: Multilevel traffic classification in the dark," in *Proc. SIGCOMM*, Philadelphia, PA, USA, Aug. 2005, pp. 229–240.

[166] A. W. Moore and D. Zuev, "Internet traffic classification using Bayesian analysis techniques," in *Proc. SIGMETRICS*, Banff, AB, Canada, Jun. 2005, pp. 50–60.

[167] A. McGregor, M. Hall, P. Lorier, and J. Brunskill, "Flow clustering using machine learning techniques," in *Passive and Active Network Measurement*. Heidelberg, Germany: Springer, 2004, pp. 205–214.

[168] T. T. T. Nguyen and G. Armitage, "Training on multiple sub-flows to optimise the use of machine learning classifiers in real-world IP networks," in *Proc. LCN*, Tampa, FL, USA, Nov. 2006, pp. 369–376.

[169] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield, "Class-of-service mapping for QoS: A statistical signature-based approach to IP traffic classification," in *Proc. IMC*, Taormina, Italy, Oct. 2004, pp. 135–148.

[170] T. T. T. Nguyen and G. Armitage, "A survey of techniques for Internet traffic classification using machine learning," *IEEE Commun. Surveys Tuts.*, vol. 10, no. 4, pp. 56–76, 4th Quart., 2008.

[171] K. Ousterhout, C. Canel, S. Ratnasamy, and S. Shenker, "Monotasks: Architecting for performance clarity in data analytics frameworks," in *Proc. SOSP*, Shanghai, China, Oct. 2017, pp. 184–200.

[172] *Openblas*. [Online]. Available: http://www.openblas.net/

[173] H. Zhang *et al.*, "Live video analytics at scale with approximation and delay-tolerance," in *Proc. 14th USENIX Conf. Netw. Syst. Design Implement. (NSDI)*, 2017, pp. 377–392.

[174] C. J. Van Rijsbergen, "A theoretical basis for the use of co-occurrence data in information retrieval," *J. Doc.*, vol. 33, no. 2, pp. 106–119, 1977.

[175] T. Akidau *et al.*, "MillWheel: Fault-tolerant stream processing at Internet scale," *Proc. VLDB Endowment*, vol. 6, no. 11, pp. 1033–1044, 2013.

[176] M. Zaharia *et al.*, "Discretized streams: Fault-tolerant streaming computation at scale," in *Proc. SOSP*, Farmington, PA, USA, 2013, pp. 423–438.

[177] M. Li *et al.*, "Scaling distributed machine learning with the parameter server," in *Proc. OSDI*, Broomfield, CO, USA, Oct. 2014, pp. 583–598.

[178] Y. Huang *et al.*, "FlexPS: Flexible parallelism control in parameter server architecture," *Proc. VLDB Endowment*, vol. 11, no. 5, pp. 566–579, 2018.

[179] *Apache Giraph*. [Online]. Available: http://incubator.apache.org/giraph

[180] *Apache Hama*. [Online]. Available: http://hama.apache.org

[181] L. G. Valiant, "A bridging model for parallel computation," *Commun. ACM*, vol. 33, no. 8, pp. 103–111, 1990.

[182] A. J. Smola and S. M. Narayanamurthy, "An architecture for parallel topic models," *Proc. VLDB Endowment*, vol. 3, no. 1, pp. 703–710, 2010.

[183] Q. Ho *et al.*, "More effective distributed ML via a stale synchronous parallel parameter server," in *Proc. NIPS*, Dec. 2013, pp. 1223–1231.

[184] *Tencent Angel*. [Online]. Available: https://github.com/Tencent/angel

[185] J. Jiang, B. Cui, C. Zhang, and L. Yu, "Heterogeneity-aware distributed parameter servers," in *Proc. SIGMOD*, Chicago, IL, USA, May 2017, pp. 463–478.

[186] X. Zhu, W. Chen, W. Zheng, and X. Ma, "Gemini: A computation-centric distributed graph processing system," in *Proc. OSDI*, Savannah, GA, USA, 2016, pp. 301–316.

[187] B. Gufler, N. Augsten, A. Reiser, and A. Kemper, "Load balancing in MapReduce based on scalable cardinality estimates," in *Proc. ICDE*, Washington, DC, USA, 2012, pp. 522–533.

[188] V. Jeyakumar, M. Alizadeh, C. Kim, and D. Mazières, "Tiny packet programs for low-latency network control and monitoring," in *Proc. HotNets*, College Park, MD, USA, Nov. 2013, Art. no. 8.

[189] C. Ge, Z. Sun, N. Wang, K. Xu, and J. Wu, "Energy management in cross-domain content delivery networks: A theoretical perspective," *IEEE Trans. Netw. Service Manag.*, vol. 11, no. 3, pp. 264–277, Sep. 2014.
[190] X. Chen, J. Wu, Y. Cai, H. Zhang, and T. Chen, "Energy-efficiency oriented traffic offloading in wireless networks: A brief survey and a learning approach for heterogeneous cellular networks," *IEEE J. Sel. Areas Commun.*, vol. 33, no. 4, pp. 627–640, Apr. 2015.
[191] H. Li, M. Dong, X. Liao, and H. Jin, "Deduplication-based energy efficient storage system in cloud environment," *Comput. J.*, vol. 58, no. 6, pp. 1373–1383, Jun. 2015.
[192] K. Wang, Y. Wang, Y. Sun, S. Guo, and J. Wu, "Green industrial Internet of Things architecture: An energy-efficient perspective," *IEEE Commun. Mag.*, vol. 54, no. 12, pp. 48–54, Dec. 2016.
[193] H. Huang, S. Guo, J. Wu, and J. Li, "Green datapath for TCAM-based software-defined networks," *IEEE Commun. Mag.*, vol. 54, no. 11, pp. 194–201, Nov. 2016.
[194] J. An *et al.*, "Achieving sustainable ultra-dense heterogeneous networks for 5G," *IEEE Commun. Mag.*, vol. 55, no. 12, pp. 84–90, Dec. 2017.
[195] R. Atat *et al.*, "Enabling cyber-physical communication in 5G cellular networks: Challenges, spatial spectrum sensing, and cyber-security," *IET Cyber Phys. Syst. Theory Appl.*, vol. 2, no. 1, pp. 49–54, Apr. 2017.
[196] K. Wang, Y. Wang, D. Zeng, and S. Guo, "An SDN-based architecture for next-generation wireless networks," *IEEE Wireless Commun.*, vol. 24, no. 1, pp. 25–31, Feb. 2017.

Kun Wang (M'13–SM'17) received the first Ph.D. degree from the Nanjing University of Posts and Telecommunications, Nanjing, China, in 2009 and the second Ph.D. degree from the University of Aizu, Japan, in 2018, both in computer science. From 2013 to 2015, he was a Post-Doctoral Fellow with the Electrical Engineering Department, University of California, Los Angeles, CA, USA. He is currently a Research Fellow with the Department of Computing, Hong Kong Polytechnic University, Hong Kong, and also a Full Professor with the School of Internet of Things, Nanjing University of Posts and Telecommunications. He has published over 100 papers in referred international conferences and journals. He was a recipient of the Best Paper Award at IEEE GLOBECOM16. He serves as an Associate Editor for IEEE ACCESS, an Editor for the *Journal of Network and Computer Applications*, the *Journal of Communications and Information Networks*, and *EAI Transactions on Industrial Networks and Intelligent Systems* and a Guest Editor for IEEE ACCESS, *Future Generation Computer Systems*, *Peer-to-Peer Networking and Applications*, and the *Journal of Internet Technology*. He was the Symposium Chair/Co-Chair of IEEE IECON16, IEEE EEEIC16, IEEE WCSP16, and IEEE CNCC17. He is a member of ACM.

Qihua Zhou is currently pursuing the Ph.D. degree with the School of Computer Science, Nanjing University of Posts and Telecommunications, China. His current research interests include operating system, distributed processing, parallel computing, and machine learning.

Song Guo (SM'11) received the Ph.D. degree in computer science from the University of Ottawa. He was a Full Professor with the University of Aizu, Japan. He is currently a Full Professor with the Department of Computing, Hong Kong Polytechnic University. His research has been sponsored by JSPS, JST, MIC, NSF, NSFC, and industrial companies. His research interests are mainly in the areas of cloud and green computing, big data, wireless networks, and cyber-physical systems. He has published over 300 conferences and journal papers in the above areas. He was a recipient of multiple best paper awards from IEEE/ACM conferences. He has served as an Editor of several journals, including IEEE TPDS, IEEE TETC, IEEE TGCN, *IEEE Communications Magazine*, and *Wireless Networks*. He has been actively participating in international conferences as the general chair and TPC chair. He is a Senior Member of ACM and an IEEE Communications Society Distinguished Lecturer.

Jiangtao Luo (M'11–SM'15) received the B.S. degree from Nankai University in 1993 and the Ph.D. degree from the Chinese Academy of Science in 1998. He is currently a Full Professor, a Ph.D. Supervisor, and the Deputy Dean of the Electronic Information and Networking Research Institute, Chongqing University of Posts and Telecommunications. He has been a Visiting Scholar with the University of Hamburg, Germany, and the University of Southern California, USA, in 2015 and 2016. His major research interests are network protocol analysis, network data mining, urban computing, and future Internet architecture. He has published over 100 papers and owned 21 patents in the above areas. He was a recipient of the Chinese State Award of Scientific and Technological Progress in 2011, the Chongqing Provincial Award of Scientific and Technological Progress twice in 2010 and 2007, respectively, and the Chongqing Science and Technology Award for Youth in 2010. He is an ACM Member.