JMS: Joint Bandwidth Allocation and Flow Assignment for Data Transfers with Multiple Sources

Paper ID: 1570386532

Abstract—**The increasing prevalence of data-intensive applications has made large-scale data transfers more important in datacenter networks. Oversubscribed networks have excessive traffic demand causing serious performance bottlenecks. Data replicas, with the advantage of source diversity, can potentially improve the transmission performance, but current works focus heavily on best replica selection rather than multi-source transmission. In this paper, we present JMS, a novel traffic management system that optimizes bulk multi-source transfers in software-defined datacenter networks. With a global network view and consistent data access, JMS conveys data in parallel from multiple distributed sources and dynamically adjusts the flow volumes to maximize network utilization. The joint bandwidth allocation and flow assignment optimization problem poses a major challenge with respect to nonlinearity and multiple objectives. To cope with this, we design an online fair allocation algorithm that derives a novel transformation with simple equivalent canonical linear programming to achieve global optimality efficiently. Simulation results demonstrate that JMS outperforms other single-source and multi-source transmission approaches with substantial gains, where JMS improves the network throughput by up to 52% and reduces the transfer completion time by up to 44%, meanwhile it continues to provide good performance for large files as well as high traffic loads.**

I. Introduction

During the last decade, datacenters have been continuing to thrive with the emergence of cloud-related services. Companies like Google, Microsoft and Amazon utilize datacenters to accomplish various application functions, including web search, storage, e-commerce, streaming media and large-scale computations [1], [2]. Datacenters nowadays contain up to hundreds of thousands of servers, running multiple data-intensive distributed services. Many of them that rely on low-latency and high-throughput data transmission, require network operators to carefully orchestrate the large-scale transfers among servers. Sub-optimal flow routing and transfer scheduling will cause network congestion and slow transmission time.

A number of traffic engineering (TE) solutions have been developed to improve the transfer efficiency in datacenter networks. Traditional TCP-based transfer protocols reactively adjust the flow rate, which is far from optimal for satisfying transfer requirements [3]–[5]. Centralized TE solutions, aiming at maximizing network utilization or minimizing flow completion time, are well investigated [6]–[12]. By dynamically changing routing and rate allocation with a global network view, centralized TEs achieve better optimality. Recently, the concept of software-defined networking (SDN) that separates the control and data planes, has been increasingly exploited in datacenter networks/WANs [1], [2], [13]–[15]. SDN allows operators to directly program on the open hardware under central control, thereby making routing and engineering protocols more customized for a variety of requirements.

Data replication is emerging increasingly in datacenter networks, to improve data availability and access efficiency [16], [17]. However, current solutions focus heavily on best replica selection and data replication placement, instead of multi-source transmission [18]–[20]. The single-source transmission with multiple data replicas results in two major shortcomings. i) When several sources have almost the same transmission performance to the destination, choosing a slightly better one and discarding all the others fails to fully utilize the network resources. ii) Selecting only one source for the whole transfer does not adapt to the dynamics because, if there are flows entering/exiting or network state changing during the transmission, the pre-selected best replica may no longer remain as the best.

In this paper, we introduce JMS, a novel traffic management system for bulk multi-source transfers in datacenter networks. Leveraging SDN principles, JMS orchestrates bulk transfers in a centralized manner. JMS's key insight is to concurrently convey data from multiple sources and to dynamically adjust the flow volumes for maximizing network utilization. The joint bandwidth allocation and flow assignment optimization problem poses a major challenge, because its direct formulation is nonlinear and multi-objective. To cope with this, JMS runs an online fair allocation algorithm, which derives a novel transformation with simple equivalent canonical linear programming (LP) to achieve global optimality efficiently. The allocation decisions are enforced through SDN controller to reconfigure the network and transfer sessions.

This paper presents the first approach that optimizes bulk transfers with multiple sources in software-defined datacenter networks. The **major contributions** of this paper are summarized as follows:

1) Leveraging concepts from Software Defined Networking, we build a new traffic management system for bulk multi-source transfers. In particular, each transfer is accomplished by retrieving data from all its available replica sources to make the best of network utilization.
2) We propose a unified data access mechanism to realize dynamic flow assignments from different sources. With the data being consistently divided into partitions, JMS

can re-distribute the data volume and rate across multiple flows in response to the dynamic network.

3) We design an optimized algorithm to jointly determine bandwidth allocations and flow assignments for all transfers in a max-min fair manner. This algorithm solves the nonlinear and multi-objective optimization problem by a novel transformation with simple equivalent canonical LP.

We perform extensive simulations, which show that JMS leads to better performance on network throughput with a gain of up to 52% and reduces transfer completion time by up to 44%, compared to other single-source and multi-source transmission approaches. Furthermore, the results validate that JMS can optimize large-scale bulk transfers by continuing to provide good performance for large files and high traffic loads.

The following section briefly introduces the background and the motivation of JMS. Sec. III presents the detailed design of JMS with its constituent components. The optimal MultiSource Fair Allocation algorithm is discussed in Sec. IV, including a preliminary instance with one multi-source transfer and the generic solution that optimizes arbitrary bulk transfers. We evaluate JMS in Sec. V and introduce related works in Sec. VI. Finally, we conclude this paper in Sec. VII.

II. Background and Motivation

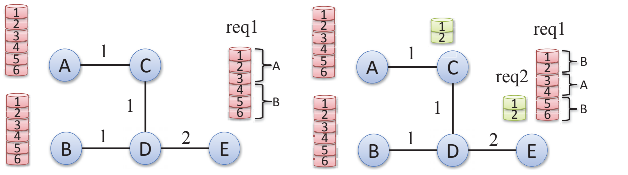
In this section, we outline some background, and give two motivating examples to show the benefits of leveraging multiple sources for transmission.

A. Background

Large files and bulk data transfers. There are many big-data applications generating large files and bulk data in datacenter networks, e.g., scientific experiments and streaming media. These applications heavily rely on frequent data transfers for storing and retrieving large datasets. Bulk transfers have large size (terabytes) and account for a big proportion of traffic, e.g., 85-95% for some datacenter networks [1], [2], [13]. High throughput and low transfer completion time is essential for their service qualities.

Centralized TE and SDN. The inefficiency of traditional TCP-based protocols motivates the works on proactive rate allocation in datacenter networks [3]–[5]. Centralized TE solutions schedule packet-level flows with a global network view. Some of them aim at maximizing the aggregate network utilization with minimal scheduler overhead [6]–[8]. Priority-based flow scheduling considers not only network utilization, but also some fine-grained transfer metrics, such as minimizing flow completion time and meeting deadlines [9]–[12]. Further more, SDN that leverages the network programmability, can help datacenters make the best routing decision and achieve customized scheduling [1], [2], [13]–[15].

Multiple data replicas. Data replication, a technique that amends both data availability and access efficiency, are frequently used in datacenter networks. Distributed filesystems including Google File System (GFS) [18], Hadoop Distributed File System (HDFS) [19] are typically deployed with a replication factor of three [18]–[20]. Media companies supply video replicas in different areas to allow clients have closer data access. Data-intensive scientific applications require large amounts of data in a distributed computing environment, so the experimental data is usually stored in different servers [17]. A number of approaches have been proposed for selecting the best data replica based on various criteria [16], [17]. However, those approaches only allow users to specify one replica in each selection, while discarding the others.



(a) (b)

Fig. 1. Motivation examples of multi-source transfers, where bandwidth capacity is labeled on each link. (a) The example with 1 request to demonstrate that leveraging multi-source transmission outperforms the single-source one. (b) The example with 2 requests to demonstrate that using dynamic flow assignment outperforms the fixed one.

B. Motivating Examples

Multiple data replicas open a new opportunity for optimizing bulk transfers in datacenter networks. Existing approaches assume a given and fixed data source for every transfer, and schedule bulk transfers by controlling the routing and the flow rate of each one. We provide two simple motivating examples to demonstrate that leveraging multi-source transmission with dynamic flow assignment outperforms single-source approaches in terms of network utilization and transfer completion time.

As the example shown in Fig. 1(a), a client sends a request req1 to download a certain file with size of 6 data units to the destination node E, and both node A and B have the source file. If we conduct the transportation by the traditional single-source approach, only B is chosen as the data source with the minimum cost and $B-D-E$ as the best shortest path. Here source A is unused, and it takes 6 time units for the whole transfer task. But if we control A to send 50% of the source file and B to send the other 50%, the transfer is thus split into two flows $A-C-D-E$ and $B-D-E$, which are transmitted simultaneously. By making complete use of the available sources, the network is fully utilized and it takes only 3 time units for the whole task, which is much faster than single-source transmission.

Fig. 1(b) shows another example of dynamic flow assignments from multiple sources. We assume that there is another request req2 to the destination node E, and only C has the corresponding file with size 2. At the beginning, transfer 1 comes as a single flow $B-D-E$ and transfer 2 comes as the flow $C-D-E$ for transfer-level fairness. Then 2 time units later when transfer 2 is finished, we can re-adjust the flow assignment of transfer 1 and start to use both A and B to complete the last 4 units of file (3-6). In total, it takes 4 time units to finish the two transfer tasks by dynamic flow assignment from multiple sources. Whereas by the fixed flow assignment approach for multi-source transfer (1/2 from source A and 1/2 from source B), the total completion time is 2+3=5 units, and by the traditional single-source approach, the completion time is 6. *As we show, better transmission performance can be implemented by dynamically adjusting the flow volumes from feasible sources.*

III. JMS Design

JMS is a centralized system with a series of components which orchestrate bulk transfers and enforce bandwidth allocations in the datacenter network. The primary design goal is to provide high transmission performance for large files by leveraging all the available replica sources. The basic system architecture of JMS consists of four main functional components as shown in Fig. 2. *Transfer Submission* monitors clients' transfer requests and searches the candidate data sources; *Flow Mapping* calculates the shortest path then maps the flows and links according to the network state; *Scheduling* decides the data rate and flow assignment of each transfer; *Transfer Enforcer*, running within the SDN controller, reconfigures the network to start/continue transfer sessions.

JMS is a synchronous system, where time is divided into time slots with equal length. A time slot (minutes) is much longer than the time (seconds) of reconfiguring the network and adjusting transmission rates. There is a stream of new transfers arriving at the system. So in each time slot, Flow Mapping and Scheduling will recompute the bandwidth allocation in response to the dynamic network. Transfer Enforcer will update the network configuration to orchestrate bulk transfers. Additionally, any new transfer request received in Transfer Submission can also trigger the allocation and update.

Each file in JMS is partitioned into large data blocks, where the block size is an adjustable system parameter for different files. To offer data consistency, the block number and identifiers for the same file must remain consistent in different source servers. JMS system adopts dynamic flow assignment for each multi-source transfer. By "dynamic flow assignment", we mean that the flow rate proportions from different sources are dynamically changed in every scheduling slot. JMS informs source servers about the calculated assignment in terms of data block numbers. This way, the multiple flows of the same transfer task can be finished at the same time. For instance, in the time slot with a flow assignment of (1/3, 2/3), JMS retrieves data block 1 from source A and data block 2 and 3 from source B; in the next slot with an updated flow assignment of (2/3, 1/3), JMS retrieves data block 4 and 5 from source A and block 6 from source B.

The details of each component are described as below.

1) Transfer Submission: Transfer Submission provides an interface to clients, and is responsible for real-time monitoring clients' bulk transfer requests. A request req$i$ submitted by clients is a tuple ($file_i$, dst_i) that denotes the file ID and destination address of transfer i. Transfer Submission then queries a persistent key-value database (DB) according to the file ID $file_i$. Note that there might be multiple servers storing the file replicas, so the DB can return a set of source addresses which will then be attached in req$i$ as ($file_i$, dst_i, $\{src_{ik}\}$).

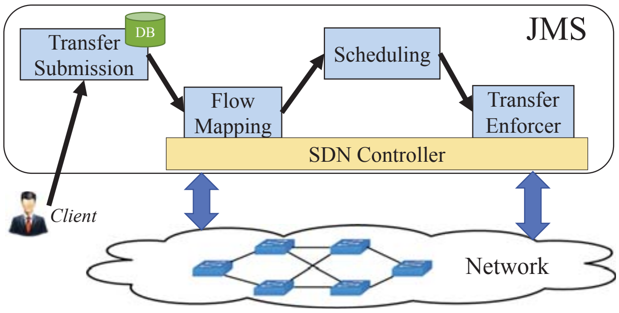


Fig. 2. JMS architecture.

Transfer Submission collects all the requests and hands them to the next component.

2) Flow Mapping: Flow Mapping has a global view of the physical topology and on-going transfers with the help of SDN controller. It periodically queries for the bandwidth utilization of links and keeps track of the existing flows in each time slot. The measured bandwidth information is used as an instantaneous snapshot of the network state to compute the best shortest paths of new flows. Here a flow is defined as a (src, dst) pair, so the transfer with multiple available sources is decomposed into multiple parallel data flows. Each flow is restricted with a single shortest path, and the path is assumed to be unchanged in different time slots. Flow Mapping integrates the routing paths of new flows, together with those of existing flows, into a flow-link mapping matrix (described in Section IV-B) as an output.

3) Scheduling: Taking the calculated flow paths and mapping results from Flow Mapping as the inputs, Scheduling executes the optimized MultiSource Fair Allocation algorithm in each time slot. The algorithm that computes the joint bandwidth allocation and flow assignment for each transfer, will be introduced in Section IV.

4) Transfer Enforcer: Transfer Enforcer is the direct manipulator for conducting data transmission in the datacenter network. Through SDN controller, it can install/update the flow rules on corresponding switches and set up transfer sessions on related servers. Specifically, Transfer Enforcer periodically queries for the unfinished data blocks from sources, and keeps recording the transfer state. In the mean time, it can control and manage the detailed assemblage of data blocks to be conveyed. According to the rate allocation and flow assignment results from component Scheduling, Transfer Enforcer carefully redistributes the pending data volumes among the source servers in each time slot. In brief, Transfer Enforcer instructs the servers more precisely about which data blocks to transfer, and at what data rate.

IV. Joint Bandwidth Allocation and Flow Assignment for Multi-source Transfers

One of the biggest challenges for taking best advantage of the data sources is to optimally assign flows among the sources and to allocate each flow's bandwidth. First, we must change the optimized object from the individual 5-tuple flow into the group of flows that belong to the same transfer. Hence, multiple potential bottlenecks might be considered simultaneously. Second, we need a joint optimization algorithm that computes bandwidth allocation and flow assignment at once.

Here we present an adaptation of the traditional max-min fairness objective which is transfer-level rather than flow-level. We design a centralized MultiSource Fair Allocation (MSFA) algorithm that maximizes network utilization while providing global max-min fairness. The key to MSFA algorithm is a novel transformation with simple equivalent canonical LP. For clear illustration, we first present the MSFA with one multi-source transfer (Section IV-C). Then we provide the generic solution (Section IV-D) that optimizes arbitrary bulk transfers with limited computational complexity.

A. Network Model and Problem Formulation

Consider a telecommunications network composed of a set of nodes and a set of links $\mathcal{L}$. The capacity of link L_j ($j \subseteq [1, M]$) is defined as C_j. Suppose there are a number of data transfer requests, each of which may come from multiple sources, with the paths pre-calculated and given. Thus each transfer i ($i \subseteq [1, N]$) is assigned with a set of flow paths $\{\mathcal{P}_{i1}, ..., \mathcal{P}_{ik}\}$, and each such path is identified with the set of links that it traverses, i.e., $\mathcal{P}_i \in \mathcal{L}$. Now let r_i denote the data rate of transfer i, which is the sum bandwidth of the constituent flows from all its sources. We use a variable set $\mathcal{X}_i = \{x_{i1}, ..., x_{ik}\}$ to express the flow assignment proportions from different sources for transfer i, and $\sum_{k=1}^{K_i} x_{ik} = 1$, where K_i is the total source number of transfer i.

We are interested in solving the joint bandwidth allocation and flow assignment problem in each time slot, i.e., finding the transmission rate r_i of each transfer, together with the assignment proportions $\mathcal{X}_i$ from all its sources. The solution is required to provide a fair and efficient allocation result, and its precise objective and constraints are described as below.

Objective. When computing allocated bandwidth, our goal is to maximize network utilization while in a max-min fair manner. A vector of transfer rate allocations $\{r_i\}$ in a slot is said to be max-min fair if, for any other feasible allocation $\{r'_i\}$, the following has to be true: if $\exists r'_p > r_p$ for the data transfer p, then there exists another transfer q such that $p, q \in [1, N]$, $r'_q < r_q$, $r_q \leq r_p$. In other words, increasing some components must be at the expense of decreasing some other existing smaller or equal components.

Constraints. The constraints of this problem are given as below. Constraint (1) is called the capacity constraint, which assures that for any link L_j, its load does not exceed its capacity C_j. Constraint (2) promises the sum fractions of a transfer from all available sources equal to 1.

$$s.t. \quad \sum_{L_j \subseteq \mathcal{P}_{ik}} r_i \cdot x_{ik} \leq C_j \qquad \forall j, \tag{1}$$

$$\sum_{k=1}^{K_i} x_{ik} = 1 \qquad \forall i, \tag{2}$$

$$0 \leq x_{ik} \leq 1 \qquad \forall i, k. \tag{3}$$

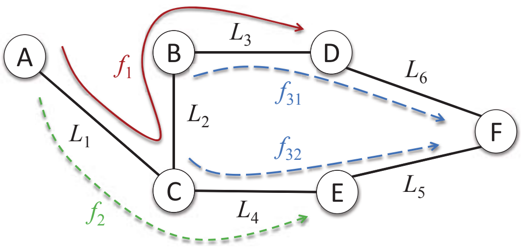


Fig. 3. An example with 3 transfer requests. Transfer 3 comes from two feasible sources B and C, corresponding to two parallel flows f_{31} and f_{32}.

Fig. 3 shows an example of the network consisting of six links $\{L_1, ..., L_6\}$, with capacities $\{8, 5, 4, 5, 7, 6\}$ respectively. Assume all bandwidth numbers are in *Gbps* here. Suppose there are three data transfer requests in the current time slot, among which transfer 1 and 2 have a single data source while transfer 3 has two available sources. In total, there are four potential flows in the network with given paths, including $f_1 : A \to C \to B \to D$, $f_2 : A \to C \to E$, $f_{31} : B \to D \to F$ and $f_{32} : C \to E \to F$. Given the topology and values of link capacities, we aim at finding the max-min fair solution of the rate vector $\{r_1, r_2, r_3\}$ and the flow assignment $\{x_{31}, x_{32}\}$ for transfer 3.

B. Flow-link Mapping Matrix and Single Source Case

The proposed multi-source algorithm for joint bandwidth allocation and flow assignment is fundamentally based on a flow-link mapping matrix (FL matrix) with solvable variables, which is the output of component Flow Mapping and the input of Scheduling in system JMS. In this section, we define the FL matrix, and illustrate the traditional water-filling algorithm for single source case with this matrix.

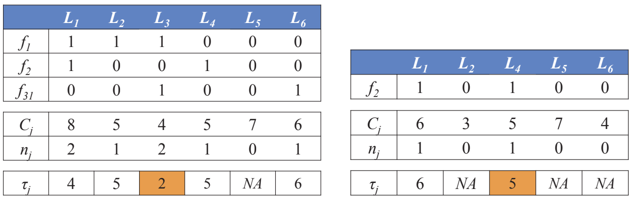
Definition 1 (Flow-link Mapping Matrix)**.** The flow-link mapping matrix *(FL matrix)* $\{fl_{ij}\}$ expresses the flow paths and the traffic shares of each transfer in matrix form. The matrix element fl_{ij} is defined as the proportion of flow i in its belonging transfer that traverses link L_j.

For single source case, since every transfer comes as an individual flow, the matrix elements are either 0 or 1. In this simple case, the bandwidth allocation can be plainly obtained by the traditional water-filling algorithm (**Algorithm 1**). Using the FL matrix as an input, we first denote the saturated average bandwidth allocation as $\tau_j = C_j/n_j$, where n_j is the total number of flows that use link L_j. The algorithm iteratively finds the minimum τ^* and the corresponding bottleneck link L_{j^*}. Set the bandwidth of the flows that use link L_{j^*} to τ^*. Then update the FL matrix by subtracting those flows and the bottleneck link to calculate a new set of $\{\tau_j\}$. Such process iterates until all transfers obtain their allocated rates, i.e., all flows have bottleneck links.

Consider the single-source version of Fig. 3, where we assume transfer 3 only uses source B, so there are 3 flows over the network. Fig. 4 illustrates the allocation algorithm for this single-source example. In the first iteration, L_3 is first

Algorithm 1 Traditional Water-Filling Algorithm

```
Input:
  Flow-link mapping matrix: $FL = \{fl_{ij}\}$;
  Capacity of each link: $\{C_j\}, 1 \leqslant j \leqslant M$;
Output:
  Transmission rate of each transfer: $\{r_i\}, 1 \leqslant i \leqslant N$;
while num of rows in $FL \neq 0$ do
    $n_j \leftarrow \sum_i fl_{ij}, \quad \forall 1 \leqslant j \leqslant M$;
    $\tau_j \leftarrow C_j / n_j, \quad \forall 1 \leqslant j \leqslant M$;
    Find $\tau^* \leftarrow min\{\tau_j\}$, $j^* \leftarrow j | \tau_j = \tau^*$;
    Set $r_i \leftarrow \tau^*$, for $i | fl_{ij^*} = 1$;
    Update $FL$;
end while
return $\{r_i\}$.
```



(a)

	L_1	L_2	L_3	L_4	L_5	L_6
f_1	1	1	1	0	0	0
f_2	1	0	0	1	0	0
f_{31}	0	0	1	0	0	1
C_j	8	5	4	5	7	6
n_j	2	1	2	1	0	1
τ_j	4	5	2	5	NA	6

(b)

	L_1	L_2	L_4	L_5	L_6
f_2	1	0	1	0	0
C_j	6	3	5	7	4
n_j	1	0	1	0	0
τ_j	6	NA	5	NA	NA

Fig. 4. An example of the FL matrix in single-source case, where transfer 3 only uses one source. C_j is the bandwidth capacity, $n_j = \sum_i fl_{ij}$ is the total number of flows that use link L_j, and $\tau_j = C_j / n_j$ is the saturated average bandwidth share. (a) Illustration of the first iteration, where L_3 is found as the bottleneck link. (b) Illustration of the second iteration, where L_4 is found as the bottleneck link.

saturated because τ_3 is of the minimum value $\tau^* = 2$. We allocate the bandwidth of f_1 and f_{31} that traverse L_3 to 2, and then remove the rows of f_1, f_{31} and column L_3. The FL matrix is updated accordingly for the next iteration, where link L_4 is then found as the bottleneck and bandwidth of f_2 is set to 5. By this point, the ultimate solution to the rate allocation for the 3 transfers is (2, 5, 2).

The traditional water-filling algorithm succeeds in obtaining the max-min fair allocation for single-source case, yet it is incapable of dealing with the multi-source transfers. This principally stems from the fact that the algorithm is based on flows, rather than on transfers. For the example in Fig. 3, transfer 3 will have double weights by using water-filling algorithm, which is unfair to the others. A naïvely improved solution is to normalize the weight of each transfer, and to assign an equal share to the flows from different sources. With regard to the example, the elements for f_{31} and f_{32} become 1/2 and 1/2 in the FL matrix, such that each transfer has the same sum weight of 1. The allocation result turns into (8/3, 10/3, 3), which is slightly better than the single-source cases, which are (2, 5, 2) for using only source B and (2.5, 4, 2.5) for using only source C. However, as we will soon learn, equally sharing the flow weight is still not the optimal solution. *The transfer-level max-min fair allocation is conditioned by the optimal flow assignment.*

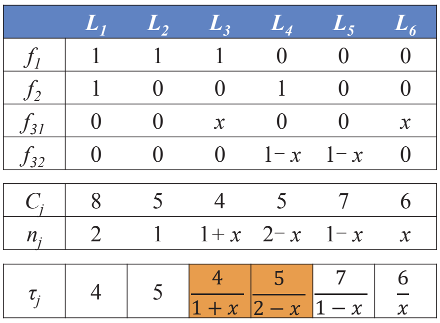
Algorithm 2 MSFA Algorithm with one multi-source transfer

```
Input:
  Flow-link mapping matrix: $FL = \{fl_{ij}\}$;
  Capacity of each link: $\{C_j\}, 1 \leqslant j \leqslant M$;
Output:
  Transmission rate of each transfer: $\{r_i\}, 1 \leqslant i \leqslant N$;
  Flow assignment of transfer $m$: $\mathcal{X}_m = \{x_{m1}, ..., x_{mk}\}$;
• Setp 1:                                         ▷ Initiation
$r_i \leftarrow 0, \quad \forall i = 1, ..., N$;
$\mathcal{L} \leftarrow \{L_1, L_2, ..., L_M\}$;
• Setp 2:             ▷ Calculate the saturated average bandwidth
$n_j(\mathcal{X}_m) \leftarrow \sum_i fl_{ij}, \quad \forall j \in \mathcal{L}$;
$\tau_j(\mathcal{X}_m) \leftarrow C_j / n_j(\mathcal{X}_m), \quad \forall j \in \mathcal{L}$;
• Setp 3:                      ▷ Find the bottleneck fair share $\tau^*$
if $min\{\tau_j(\mathcal{X}_m)\}$ is a constant then
    $\mathcal{X}^* \leftarrow \varnothing$;
else
    $\mathcal{X}^* \leftarrow \mathcal{X}_m | max\ min\{\tau_j(\mathcal{X}_m)\}$;
end if
$\tau^* \leftarrow min\{\tau_j(\mathcal{X}_m)\}$;
• Setp 4:                 ▷ Set data rate and update the FL matrix
$\mathcal{L}_{j^*} \leftarrow \{L_j | \tau_j(\mathcal{X}_m) = \tau^*\}$;
$\mathcal{L} \leftarrow \complement_{\mathcal{L}_{j^*}} \mathcal{L}$;
for $i | P_i \cap \mathcal{L}_{j^*} \neq \varnothing$ do
    $r_i \leftarrow r_i + \tau^*$;
    Remove $f_i$ from $FL$;
end for
Update $FL$;
• Step 5:                                          ▷ Iteration
if No transfers left then
    return $\{r_i\}$ and $\mathcal{X}_m$;
else
    goto Step 2.
end if
```

C. MSFA with One Multi-source Transfer

Given the FL matrix and its application, now let's consider a more complex case by adding just one multi-source transfer into the network. We assume there are K_m available sources for the particular transfer m, though the flow assignment $\mathcal{X}_m$ is unknown and needs to be solved. The MSFA algorithm continues to use the FL matrix as an input, but the elements are no longer 0 and 1 as in the single-source case. Instead, the matrix elements related to transfer m are replaced by the unknown variables in $\mathcal{X}_m$. The MSFA algorithm with one multi-source transfer (**Algorithm 2**) can be described in the following high-level steps:

1) Start from zero allocation with the whole link set, and build the FL matrix with variables $\mathcal{X}_m$.
2) Calculate the saturated average bandwidth $\tau_j(\mathcal{X}_m)$ on each link L_j.
3) Find the bottleneck fair share $\tau^* = min\{\tau_j(\mathcal{X}_m)\}$ by solving $\mathcal{X}^* = \mathcal{X}_m | max\, min\{\tau_j(\mathcal{X}_m)\}$.
4) Set the data rate to τ^* for the flows that traverse the



	L_1	L_2	L_3	L_4	L_5	L_6
f_1	1	1	1	0	0	0
f_2	1	0	0	1	0	0
f_{31}	0	0	x	0	0	x
f_{32}	0	0	0	$1-x$	$1-x$	0
C_j	8	5	4	5	7	6
n_j	2	1	$1+x$	$2-x$	$1-x$	x
τ_j	4	5	$\frac{4}{1+x}$	$\frac{5}{2-x}$	$\frac{7}{1-x}$	$\frac{6}{x}$

Fig. 5. An example of MSFA, where transfer 3 comes as two flows. x is the flow assignment variable to be calculated. τ_j in orange cell is found as the minimum bottleneck share.

bottleneck links, and update the FL matrix by removing those flows and links.

5) If there are no transfers left then stop, otherwise return to Step 2.

In Step 2, similar to the traditional water-filling algorithm, we first calculate $n_j(\mathcal{X}_m)$ by summarizing the elements of column j in the FL matrix. Here $n_j(\mathcal{X}_m)$ denotes the number of transfers that use link L_j. Due to the fact that transfer m comes from multiple parallel flows, there might be only a fraction of the transfer using link L_j. As a result, n_j becomes a function of $\mathcal{X}_m$ instead of an integer value as in the single-source case. Next, we compute the average bandwidth $\tau_j(\mathcal{X}_m) = C_j/n_j(\mathcal{X}_m)$, which is also a function of $\mathcal{X}_m$.

In Step 3, given $\{\tau_j(\mathcal{X}_m)\}$ on the current link set, one or several bottleneck links are found. Specifically, since all the variables $\mathcal{X}_m$ are within a certain range $[0,1]$, $min\{\tau_j(\mathcal{X}_m)\}$ is sometimes a constant value. In that case, no flow assignment variable is calculated, i.e., $\mathcal{X}^* = \varnothing$. Otherwise, the flow assignment is determined by $\mathcal{X}^* = \mathcal{X}_m | max\, min\{\tau_j(\mathcal{X}_m)\}$. We use τ^* to denote the minimum bandwidth share, and the set of $\{L_{j^*}\}$ is found as the bottleneck links.

Back to the example in Fig. 3, transfer 3 has two available sources to access, resulting in two parallel flows f_{31} and f_{32}, respectively. For simplicity, we use only one variable x to denote the proportion of f_{31}, and that of f_{32} will thus be $1-x$. Fig. 5 illustrates the FL matrix, followed by the saturated average bandwidths $\{\tau_j(\mathcal{X}_m)\}$.

From the example, we can tell that *Step 3, finding $\mathcal{X}^*$ that maximizes $min\{\tau_j(\mathcal{X}_m)\}$, is the major challenge in MSFA.* Accordingly, it is to find $x^* = x | max\, min(4, 5, 4/(1+x), 5/(2-x), 7/(1-x), 6/x)$ in Fig. 5. First, as formulated in **Problem 1**, it is a nonlinear programming problem, which can not be solved directly. Second, even if we can find a linear expression, we still need long sequences of LPs for the max-min objective (multi-objective), which is computationally intense in real implementation.

Problem 1 (The optimization problem in Step 3)**.**

$$max \quad min\{\tau_j(\mathcal{X}_m)\}, \tag{4}$$

$$s.t. \quad \sum_{k=1}^{K_m} x_{ik} = 1 \qquad x_{ik} \in \mathcal{X}_m, \tag{5}$$

$$0 \le x_{ik} \le 1 \qquad \forall i, k. \tag{6}$$

To cope with this, we transform this nonlinear optimization problem into a canonical form of LP problem based on **Theorem 1**. The equivalent canonical LP problem expressed as **Problem 2** can be solved efficiently.

Problem 2 (The equivalent LP problem in Step 3)**.**

$$min \quad t \tag{7}$$

$$s.t. \quad t \ge \tau'_j(\mathcal{X}_m) \qquad \forall j, \tag{8}$$

$$\sum_{k=1}^{K_m} x_{ik} = 1 \qquad x_{ik} \in \mathcal{X}_m, \tag{9}$$

$$0 \le x_{ik} \le 1 \qquad \forall i, k. \tag{10}$$

Theorem 1. *Problem 1 is equivalent to Problem 2 as a canonical LP problem, where $\tau'_j(\mathcal{X}_m) = 1/\tau_j(\mathcal{X}_m)$.*

Proof: Given an arbitrary instance of the FL matrix, $\tau_j(\mathcal{X}_m)$ satisfies two conditions: i) $\tau_j(\mathcal{X}_m) \ge 0$, ii) the inverse of $\tau_j(\mathcal{X}_m)$ is a linear function of $\mathcal{X}_m$. So we let $\tau'_j(\mathcal{X}_m) = 1/\tau_j(\mathcal{X}_m)$, and the objective of $max\, min\{\tau_j(\mathcal{X}_m)\}$ is then equivalent to $min\, max\{\tau'_j(\mathcal{X}_m)\}$, which becomes linear accordingly. Next, we introduce a temporary variable $t = max\{\tau'_j(\mathcal{X}_m)\}$, and use a sequence of constraints $t \ge \tau'_j(\mathcal{X}_m)$ for all j to express t. Since $\tau'_j(\mathcal{X}_m)$ is a linear function of $\mathcal{X}_m$, the constraint (8) as a set of inequalities is also linear. In the end, the optimization problem in Step 3 (**Problem 1**) turns into an equivalent canonical LP problem (**Problem 2**), whereby the decision variables to be solved are the flow assignment set $\mathcal{X}_m$ and t. ■

As a result, the optimization problem of the example in Fig. 5 is well transformed into a simple equivalent LP problem, which is expressed as below.

$$min \quad t \tag{11}$$

$$s.t. \quad t \ge \frac{1}{4}, \tag{12}$$

$$t \ge \frac{1}{5}, \tag{13}$$

$$4t - x \ge 1, \tag{14}$$

$$5t + x \ge 2, \tag{15}$$

$$7t + x \ge 1, \tag{16}$$

$$6t - x \ge 0, \tag{17}$$

$$0 \le x \le 1. \tag{18}$$

The results of the above LP come out as $x = 1/3$ and $t = 1/3$. Then $\tau^* = 1/t = 3$ is the minimum fair share, and L_3 and L_4 are the bottleneck links that are saturated in this iteration. We set $r_1 = r_2 = 3$ as the bandwidth of f_1 and f_2, and $r_3 = 3$ as the sum bandwidth of f_{31} and f_{32}. Meanwhile, the data volume assignment of transfer 3 concludes with $1/3$ from source B and $2/3$ from source C. The final rate allocation to the 3 transfers are (3, 3, 3), which is more max-min fair than the allocation (2, 5, 2) where transfer 3 uses only source B (as in Fig. 4), as well as the allocation (2.5, 4, 2.5) where transfer 3 uses only source C. In addition, if we don't consider the impact of date volume and assume each transfer has the equal volume of $3Gbits$, then MSFA outperforms the single source approaches in terms of the average completion time (MSFA: (3/3+3/3+3/3)/3=1, source B: (3/2+3/5+3/2)/3=1.2) and source C: (3/2.5+3/4+3/2.5)/3=1.05), as well as total completion time (MSFA: 3/3=1, source B: 3/2=1.5 and source C: 3/2.5=1.2).

D. Generic MSFA

Having solved the preliminary instance with one multi-source transfer, now we consider the generic MSFA with arbitrary transfer combinations. In each time slot, as existing transfers getting finished, new transfers coming, or the network state getting changed, the flow assignments $\{\mathcal{X}_i\}$ $(i \subseteq [1, N])$ for all transfers need updated to maintain global optimality. Except for the transfers with single source, whose $\mathcal{X}_i = \{1\}$ for all the time, the rest of $\{\mathcal{X}_i\}$ are to be computed by MSFA. The main challenge is that the flow assignments of different transfers correlate to each other and can not be calculated independently. One transfer's assignment plan affects another's optimal decision. *Therefore, the max-min fair allocation requires the joint calculation for all flow assignments.*

The generic MSFA mainly follows the procedures in **Algorithm 2**. Exceptionally, we put all sets of the variables $\{\mathcal{X}_i\}$ into the FL matrix, such that τ_j turns into a function of $\{\mathcal{X}_1, ..., \mathcal{X}_N\}$. By the same token, we transform the nonlinear optimization problem in Step 3 into an equivalent canonical LP problem with the help of one additional decision variable t. In each iteration, parts of the flow assignment sets are solved by LP (**Theorem 2**). Then we plug the values into the FL matrix and remove them from $\{\mathcal{X}\}$. Continue iterating until all flow assignment variables $\mathcal{X}_i$ are determined. Finally, we sum up the constituent flow rates as the multi-source transfer rate, i.e., $r_i = \sum_{k=1}^{K_i} r_{ik}$, where K_i is the total source number of transfer i.

Theorem 2. *Multiple sets of variables $\mathcal{X}_i$ can be jointly calculated by MSFA.*

We simplify this theorem by a small scale instance that involves only two multi-source transfers, and each of them has only one variable. Specifically, we denote x_1 as the assignment for one transfer, and x_2 for the other. Formally, we prove the following lemma.

Lemma 1. *The optimal x_1^* and x_2^* can be jointly calculated by the LP in MSFA.*

Proof: Each iteration is divided into 5 (types of) cases.

Case 1: One bottleneck is found as $t = \tau'_{j1}$. Since it's a constant, neither x_1^* or x_2^* is determined in this iteration.

Case 2: One bottleneck is found as $t = \tau'_{j1}(x_1)$. Then we use the constraint $0 \leq x_1 \leq 1$ to find an intersection point x_1^* that minimizes t. Substitute x_1^* and solve x_2^* accordingly.

Case 3: Two bottlenecks are found as $t = \tau'_{j1}(x_1)$ and $t = \tau'_{j2}(x_1)$. Since they are not related to x_2, we can simply make $\tau'_{j1}(x_1) = \tau'_{j2}(x_1)$ to solve the intersection point x_1^*. Substitute x_1^* and solve x_2^* accordingly.

Case 4: Two bottlenecks are found as $t = \tau'_{j1}(x_1, x_2)$ and $t = \tau'_{j2}(x_1)$. We first decompose $\tau'_{j1}(x_1, x_2)$ as $\tau'_{j1}(x_1, x_2) = \tau'_{j1}(x_1) + \tau'_{j1}(x_2)$ with respect to linearity. Solve x_2^* first from $\tau'_{j1}(x_2)$ via $0 \leq x_2 \leq 1$ as in Case 2. Substitute $\tau'_{j1}(x_2^*)$ and solve x_1^* as in Case 3.

Case 5: Two bottlenecks are found as $t = \tau'_{j1}(x_1, x_2)$ and $t = \tau'_{j2}(x_1, x_2)$. Make $\tau'_{j1}(x_1, x_2) = \tau'_{j2}(x_1, x_2)$ to solve the relation function $x_1^* = f(x_2^*)$. Ignore the current two bottlenecks and find the next one(s), until Case 2, 3, or 4 happens. Solve either x_1^* or x_2^* and substitute the result into $x_1^* = f(x_2^*)$ to solve the other.

To summarize, in any cases, multiple sets of variables are solvable by the canonical LP. ■

The MSFA algorithm is scalable and extensible for more complex use cases. For instance, differentiated qualities of service lead to transfers with variations in priority or other requirements, such that MSFA is capable of supporting weighted fair allocation by taking priority factors into account. The max-min fair principle can also be applied to different optimization objectives (e.g., transfer completion time), and the adaption of MSFA with consideration of data size can likewise yield the optimal results for multi-source transfers. The computation complexity is significantly reduced in MSFA by transforming a nonlinear multi-objective problem into a single LP, and it can be further reduced by removing the redundant constraints in implementation. Thus MSFA with limited complexity is scalable to handle a datacenter network with 100s of switches.

V. Performance Evaluation

To evaluate how JMS works on a large-scale network, we implement a flow-level datacenter network simulator.

A. Simulation Methodology

Topologies: We conduct our experiments by emulating a 3-tier datacenter network topology with 8:1 oversubscription. The topology contains 64 servers, whereby each edge link is of $1Gbps$ capacity, and aggregated link is of $10Gbps$ capacity.

Workloads: We synthesize a stream of transfer requests with a total number of 1000. The request arrival is modeled as a Poisson process, where the arrival rate λ is defined as the average number of new transfers per time slot. We set the slot length as one second for fast simulation. A transfer has multiple sources with probability ρ. A multi-source transfer is assumed to have a random number of replicas between [2,5], and the replicas are randomly placed in servers. In simulations, we do not consider the fluctuation of transfer size, and assume all transfers have an uniform size of V.

Performance metrics: We use *network throughput* and *average transfer completion time* to show the improvements of JMS over other approaches.

Traffic engineering: We compare the following TE approaches, each of which computes per-flow bandwidth allocation with max-min fairness.

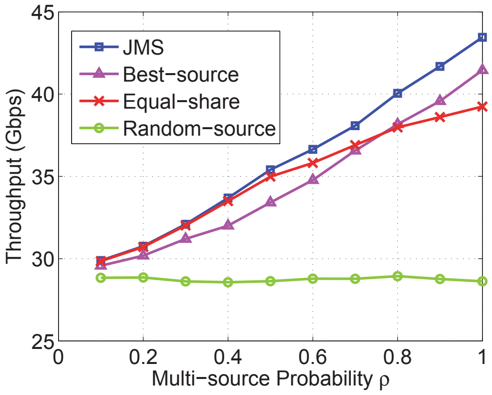


Fig. 6. Network throughput vs. multi-source probability ρ, where the arrival rate $\lambda = 2$ and the data size $V = 10Gbits$.

- **JMS:** The approach in this paper, runs MSFA algorithm to dynamically assigns flows from different sources.
- **Best-source:** This approach selects a best replica source based on the algorithm in [17] for multi-source transfers.
- **Equal-share:** This approach equally splits the transfer across different sources. For example, if a transfer has 3 replicas, each replica will send 1/3 of the data.
- **Random-source:** This approach randomly selects an available source to transmit data.

B. *Simulation Results*

Fig. 6 shows the simulation results of the network throughput for various TE approaches. Here we set the arrival rate $\lambda = 2$ and the data size $V = 10Gbits$ for all of the 1000 transfers. As the results show, Random-source approach disregards the source dissimilarity, therefore performs the worst with a constant throughput value. Equal-share approach takes advantage of source diversity simplistically. When the diversity is limited to a small number of multi-source transfers (at low multi-source probabilities), equal flow sharing approximates the optimal assignment, and thus obtains the similar performance as JMS. But as the multi-source proportion increases, there are more flows entering the network. The effect of "bad flows" enlarges, and hence drags down the overall throughput improvement. Accordingly, Best-source approach begins to outperform Equal-share. By jointly optimizing the bandwidth allocation and flow assignment, JMS achieves a much higher throughput than the others, resulting in higher utilization of the network. When all the transfers have multiple sources ($\rho = 1$), JMS obtains a substantial throughput gain of up to 52% compared to the single-source transmission.

Fig. 7 compares the transfer completion time versus three factors respectively: the multi-source probability ρ, the transfer size V and the transfer arrival rate λ. It is shown that, JMS achieves the smallest transfer completion time across all parameter configurations. This improvement is derived from two aspects: first is the optimized transmission rate which stems from a more max-min fair allocation by MSFA, second is the dynamic flow assignment to remain optimal as new transfers arrive and existing transfers complete.

Fig. 7(a) focuses on the impact of multi-source probability ρ. Approximately, ρ equals to the proportion of multi-source transfers in all the 1000. The gap between Random-source and the other approaches verifies that, leveraging multiple sources is more efficient for data transmission. Moreover, the sustained decline in completion time with the multi-source probability implies that, the the multi-source transmission obtains more performance gains by placing more replicas in datacenters. Compared to single-source transmission, JMS reduces the average completion time by up to 44%.

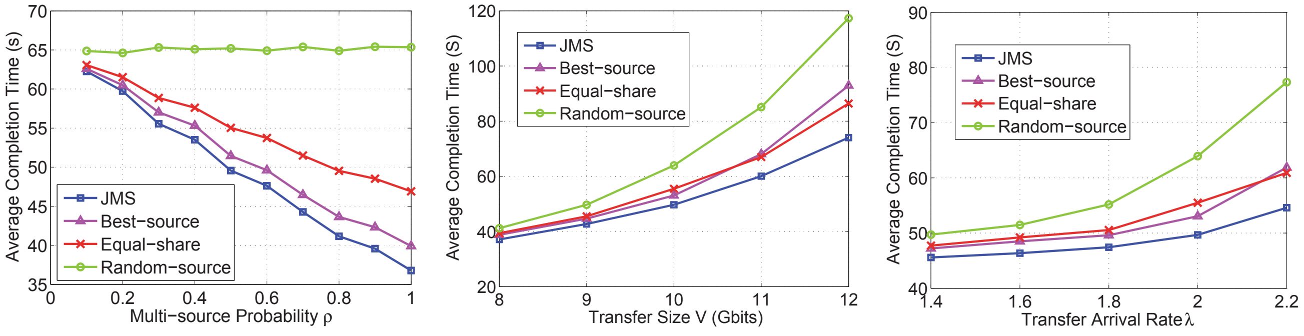
Fig. 7(b) shows the relationship between completion time and the transfer size V. As the transfer size increases, the transfer backlog starts to cause more bottleneck links, which leads to the degradation of transmission rates. Therefore the completion time increases superlinearly along with the transfer size for all the approaches. But the almost linear completion time growth of JMS suggests that, by completing transfers as quick as possible, JMS is capable of optimizing bulk transfers for small files as well as large files.

Fig. 7(c) illustrates the impact of transfer arrival rate λ. As expected, at higher rates, the number of transfers over the network potentially increases, and links are more likely to become congested. Accordingly, the performance degrades quickly for all the approaches except JMS. The smaller growth in completion time demonstrates that, by effectively avoiding the congestion point, JMS manages to handle relatively a larger amount of traffic without degrading performance.

VI. Related Work

Datacenter traffic management: Most of the recent works in datacenter networks aim at scheduling flows with centralized TE to maximize aggregate network utilization [6]–[8], [15]. MicroTE [6] leverages historical traffic to predict and mitigate the impact of congestion at the granularity of seconds. DevoFlow [15], based on SDN, is a modification of the OpenFlow model to reduce the number of switch-controller interactions and TCAM entries. Some works focus on the optimization of some fine-grained transfer metrics, such as minimizing flow completion time and meeting deadlines [9]–[12]. D^3 [9] is the first work aiming at meeting deadline based on RCP. PDQ [10] achieves better deadline-meeting rate by allocating different priorities. However, none of them takes into consideration that data replication provides more sources to improve transmission performance.

Distributed filesystems: Several high-performance distributed filesystems with sufficient data replicas have been developed, including GFS [18], HDFS [19] and Quantacast File System [20]. Sinbad [16] is the first distributed filesystem that utilizes replica placement flexibility to avoid congested links for write operations. Leveraging SDN, Mayflower [17] performs global optimizations to make intelligent replica selection and flow scheduling decisions based on both filesystem and network information. Nevertheless, all the systems completely rely on single-source transmission, instead of conveying data



(a)

(b)

(c)

Fig. 7. Impact of (a) the multi-source probability ρ, (b) the data size V and (c) the transfer arrival rate λ on average transfer completion time. In each subfigure, we adjust one factor and fixed the other two, with the default values of $\rho = 0.5$, $V = 10$ and $\lambda = 2$.

in parallel from multiple distributed sources to adapt to the variability of network bandwidths as in JMS.

Task-based flow scheduling: The works that schedule parallel flows have been developed to optimize transfers at the level of coflow rather than individual ones. Coflow [21], Varys [22] and Barrat [23] improve application-level performance by minimizing coflow completion times and guaranteeing predictable completions. However, their basic assumption is that the flows are streamed for different data and the volume of each flow is designated in advance, so they can easily predict the completion time and allocate the rate to meet their deadlines. JMS's improvement over them is the dynamic flow volume assignment, which enables re-distribution of the pending data among available sources for the same data. Moreover, this assignment in JMS is jointly optimized with bandwidth allocation to achieve global optimality.

VII. Conclusion

We present JMS, a novel traffic management system that orchestrates bulk transfers with multiple sources in software-defined datacenter networks. JMS conveys data in parallel from multiple sources and dynamically adjusts the flow volumes to maximize the network utilization. The core of JMS is an online fair allocation algorithm that jointly computes the bandwidth allocation and flow assignment with simple equivalent canonical LP to achieve global optimality. Extensive simulations validate that, compared to other single-source and multi-source transmission approaches, JMS achieves a better throughput gain of up to 52% and decreases transfer completion time by up to 44% for large-scale bulk transfers.

References

[1] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, “B4: Experience with a globally-deployed software defined wan,” *SIGCOMM CCR*, 2013.

[2] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, “Achieving high utilization with software-driven wan,” in *SIGCOMM CCR*, 2013.

[3] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar *et al.*, “Data center tcp (dctcp),” in *ACM SIGCOMM CCR*, 2010.

[4] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, and B. Khan, “Minimizing flow completion times in data centers,” in *INFOCOM*, 2013.

[5] B. Vamanan, J. Hasan, and T. Vijaykumar, “Deadline-aware datacenter tcp (d2tcp),” *ACM SIGCOMM CCR*, 2012.

[6] T. Benson, A. Anand, A. Akella, and M. Zhang, “Microte: Fine grained traffic engineering for data centers,” in *Proceedings of Conference on emerging Networking EXperiments and Technologies*, 2011.

[7] X. Wu and X. Yang, “Dard: Distributed adaptive routing for datacenter networks,” in *ICDCS*, 2012.

[8] S. Radhakrishnan, M. Tewari, R. Kapoor, G. Porter, and A. Vahdat, “Dahu: Commodity switches for direct connect data center networks,” in *ANCS*, 2013.

[9] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, “Better never than late: Meeting deadlines in datacenter networks,” in *ACM SIGCOMM CCR*, 2011.

[10] C.-Y. Hong, M. Caesar, and P. Godfrey, “Finishing flows quickly with preemptive scheduling,” *SIGCOMM CCR*, 2012.

[11] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, “pfabric: Minimal near-optimal datacenter transport,” in *SIGCOMM CCR*, 2013.

[12] E. Danna, S. Mandal, and A. Singh, “A practical algorithm for balancing the max-min fairness and throughput objectives in traffic engineering,” in *INFOCOM, 2012 Proceedings IEEE*.

[13] X. Jin, Y. Li, D. Wei, S. Li, J. Gao, L. Xu, G. Li, W. Xu, and J. Rexford, “Optimizing bulk transfers with software-defined optical wan,” in *Proceedings of SIGCOMM 2016 Conference*.

[14] A. Kumar, S. Jain, U. Naik, A. Raghuraman, N. Kasinadhuni, E. C. Zermeno, C. S. Gunn, J. Ai, B. Carlin, M. Amarandei-Stavila *et al.*, “Bwe: Flexible, hierarchical bandwidth allocation for wan distributed computing,” in *SIGCOMM CCR*, 2015.

[15] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, “Devoflow: Scaling flow management for high-performance networks,” *SIGCOMM CCR*, 2011.

[16] M. Chowdhury, S. Kandula, and I. Stoica, “Leveraging endpoint flexibility in data-intensive clusters,” in *ACM SIGCOMM CCR*, vol. 43, 2013.

[17] S. Rizvi, X. Li, B. Wong, F. Kazhamiaka, and B. Cassell, “Mayflower: Improving distributed filesystem performance through sdn/filesystem co-design,” in *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*, 2016.

[18] S. Ghemawat, H. Gobioff, and S.-T. Leung, “The google file system,” in *ACM SIGOPS operating systems review*, 2003.

[19] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The hadoop distributed file system,” in *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*, 2010.

[20] M. Ovsiannikov, S. Rus, D. Reeves, P. Sutter, S. Rao, and J. Kelly, “The quantcast file system,” *Proceedings of the VLDB Endowment*, vol. 6, no. 11, 2013.

[21] M. Chowdhury and I. Stoica, “Coflow: A networking abstraction for cluster applications,” in *Proceedings of HotNet*, 2012.

[22] M. Chowdhury, Y. Zhong, and I. Stoica, “Efficient coflow scheduling with varys,” in *ACM SIGCOMM CCR*, 2014.

[23] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, “Decentralized task-aware scheduling for data center networks,” in *ACM SIGCOMM CCR*, 2014.