Data Transfer Scheduling for Maximizing Throughput of Big-Data Computing in Cloud Systems

Ruitao Xie and Xiaohua Jia, *Fellow, IEEE*

Abstract—Many big-data computing applications have been deployed in cloud platforms. These applications normally demand concurrent data transfers among computing nodes for parallel processing. It is important to find the best transfer scheduling leading to the least data retrieval time—the maximum throughput in other words. However, the existing methods cannot achieve this, because they ignore link bandwidths and the diversity of data replicas and paths. In this paper, we aim to develop a max-throughput data transfer scheduling to minimize the data retrieval time of applications. Specifically, the problem is formulated into mixed integer programming, and an approximation algorithm is proposed, with its approximation ratio analyzed. The extensive simulations demonstrate that our algorithm can obtain near optimal solutions.

Index Terms—Data transfer scheduling, big-data computing, throughput maximization, data center

1 Introduction

Many big-data computing applications have been deployed in cloud platforms, e.g., Amazon's Elastic Compute Cloud (EC2), Windows Azure, IBM Cloud etc. In big-data computing under MapReduce framework [1], tasks run on computing nodes in parallel. But, the data may not be stored in the same nodes as they are processed for a variety of reasons. For instance, when those nodes have insufficient computing capacity, or when they are not preferred by other objectives (e.g., load balancing and energy saving).

In Data Center Networks (DCN), data are usually replicated for redundancy and robustness, e.g., in HDFS, every data block has two replicas in addition to the original one [2]. Furthermore, from each data node, multiple paths are available for data transfer, sometimes, all of which are shortest paths, due to path redundancy in DCN [3], [4]. It is important to select the best node and the best path to retrieve a non-local data. This is the data retrieval problem.

Different selections of nodes and paths may result in different data retrieval time. It is important to find the selection leading to the least data retrieval time, because long data retrieval time of a computing task may result in long completion time for the application to whom this task belongs.

However, the existing method to retrieve data, which is used in current HDFS systems and DCN, cannot obtain the least data retrieval time. In the existing method, when a non-local data is required, a request is sent to any one of the closest replicas [2]. Then, the data is transferred from the selected node through any one of the shortest paths, determined by routing protocols like Equal-Cost Multipath Routing (ECMP) [5] or per-flow Valiant Load Balancing (VLB) [4]. It is noted that many tasks are retrieving data concurrently. This method may result in heavy congestions on some links, leading to long data retrieval time, because it ignores link bandwidths and the overlaps of selected nodes and paths. Some researchers proposed flow scheduling systems to avoid path collisions [6], [7]. However, they only exploited path diversity but not data replica diversity.

To minimize the data retrieval time (i.e., to maximize the throughput) of an application consisting of concurrent tasks, we propose a max-throughput data transfer scheduling, utilizing both replica and path diversities. In our method, the problem is formulated into mixed integer programming, and an approximation algorithm is proposed, with its approximation ratio analyzed. We also solve the data retrieval problem for the case of multiple applications. Our simulations demonstrate that the approximation results are almost as good as the optimal solutions. We also show that the availability of a small number of additional data replicas can be greatly beneficial in many cases, regardless of path diversity.

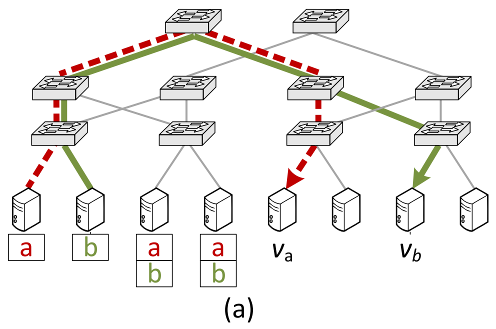
The rest of this paper is organized as follows. Section 2 presents the overview and the motivation of the data retrieval problem. Section 3 presents problem formulations, for the scenarios of a single application and multiple applications separately. Section 4 presents an approximation algorithm and the analyses on its approximation ratio. The simulations and performance evaluations are presented in Section 5. Section 6 presents the related works on the data retrieval in cloud. Section 7 concludes the paper.

2 Problem Overview and Motivation

In cloud, although data is distributed among computing nodes, not all data can be obtained locally, so some nodes

• *The authors are with the Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong.*
E-mail: {ruitao.xie, csjia}@cityu.edu.hk.

Manuscript received 16 Apr. 2015; revised 5 July 2015; accepted 29 July 2015. Date of publication 5 Aug. 2015; date of current version 7 Mar. 2018.
Recommended for acceptance by M. Qiu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCC.2015.2464808



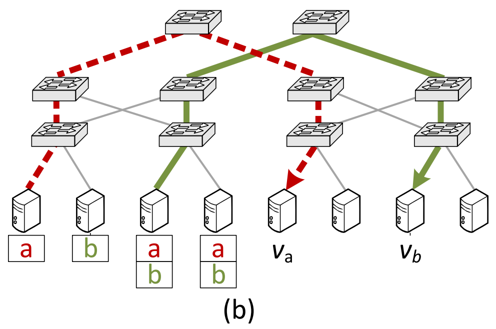


Fig. 1. An example to motivate the optimal data retrieval method. Node v_a is retrieving data a (red dash lines) and v_b is retrieving data b (green solid lines). In (a) both data transfers share common links, which has more traffic and may lead to longer transmission time, while in (b) they passes through disjoint sets of links, resulting in shorter data retrieval time.

may need to retrieve data from distant nodes. A requested data can be retrieved from one of the nodes where its replica is stored. When a node is chosen for data retrieval, a path from it to the requesting node needs to be specified for data transfer. A reasonable choice would be the shortest path (in terms of the number of hops). However, there may exist multiple shortest paths, so one of them must be selected. It is noted that we select only one node and one path for each requested data, because otherwise it would result in multipath TCP, which suffers from high jitter and is not widely deployed in DCN yet. A naive method is to select nodes and paths randomly, but it may result in heavy congestions on some links, leading to long data retrieval time, because it ignores link bandwidths and the overlaps of selected paths and nodes.

For example, considering the case of a single application, in a topology as shown in Fig. 1, two data objects (a and b) are stored with replication factor of 3, and each link has the bandwidth of 1 data per second. Note that it takes at least 1 second to transfer a data between any two nodes. Suppose at the same time, node v_a is about to retrieve data a and v_b is about to retrieve data b, both belonging to the same application, then the naive method may result in the data retrieval time of 2 seconds; while the optimal solution only takes 1 second. The naive method has a worse performance, because both data transfers pass through some common links, becoming bottlenecks (as shown in Fig. 1a). The optimal solution takes the least time, because by selecting nodes and paths both data transfers pass through disjoint sets of links (as shown in Fig. 1b). This motivates us to investigate the optimal data retrieval method, where nodes and paths are selected carefully. Our objective is to select nodes and paths such that the data retrieval time of an application is minimized.

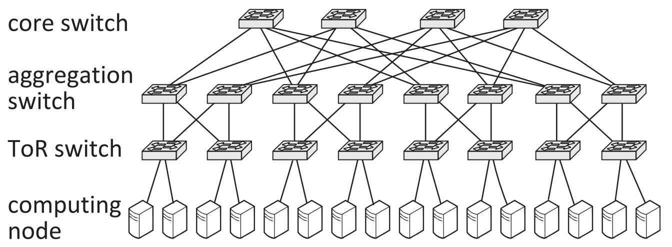


Fig. 2. A typical topology of data center network: a fat-tree topology composed of 4-port switches.

The naive method also falls short when multiple applications are running in the cloud, where different applications may have different requirements, i.e., the upper bound of data retrieval time. As it may be impossible to satisfy all requirements, we minimize the penalties of applications. Thus our problem is to select nodes and paths for each required data such that the penalties of applications are minimized.

3 Problem Formulation

In this section, we formulate the data retrieval problem. We start with the case of a single application, and deal with multiple applications later.

3.1 Single Application

A Data Center Network consists of switches and computing nodes, as illustrated in Fig. 2. The DCN is represented as a graph with nodes V and edges E, i.e., $G = \langle V, E\rangle$, where each edge represents a bi-directional link. This is a typical network configuration in DCN. V consists of both computing nodes V_c and switch nodes V_x, that is $V = V_c \cup V_x$. Let B_e denote the bandwidth on link e.

Suppose an application processes a set of data objects $D = \{d_1, d_2, \ldots, d_k, \ldots, d_m\}$, which have been stored in computing nodes. For simplicity, we assume that all data objects have the same size S. A data object is replicated in multiple nodes for redundancy. Let $V_{ck} \subseteq V_c$ denote the set of nodes that have data object d_k. Let A_{jk} denote whether v_j needs data object d_k to run the tasks assigned on it. If v_j requires d_k (i.e., $A_{jk} = 1$), we have to select a node $v_i \in V_{ck}$ from which d_k can be retrieved. Each selection forms a flow f^k_{ij}, which represents the data transfer of d_k from v_i to v_j. The set of possible flows of transferring data d_k to node v_j is denoted by F_{jk}. Hence, the set of all flows is

$$F = \bigcup\nolimits_{\forall (j,k) \text{ where } A_{jk}=1} F_{jk}. \tag{1}$$

Because one and only one node is selected for each retrieval, exactly one flow in F_{jk} can be used. Let binary variable x_f denote whether or not flow f is used, then

$$\sum_{f \in F_{jk}} x_f = 1 \qquad \forall\, (j,k) \text{ where } A_{jk} = 1. \tag{2}$$

After selecting a flow, we have to find a path for it. To shorten data retrieval time, we use shortest paths. Let $P(f^k_{ij})$ denote the set of paths that can be used to actuate f^k_{ij}, which are all shortest paths from v_i to v_j. Let binary variable y_{fp} denote whether path p is selected to actuate flow f, where $p \in P(f)$, then,

$$\sum_{p \in P(f)} y_{fp} = x_f \qquad \forall\, f \in F. \tag{3}$$

Only one path will be selected if flow f is used (i.e., $x_f = 1$), and none otherwise (i.e., $x_f = 0$).

Since all flows are transferred concurrently, the data retrieval time of an application (which is the total time to complete all data transfers) is dominated by the longest data transfer time among all flows. Let t_f denote the data transfer time of flow f, then data retrieval time t is computed as

$$t = \max\{t_f, \quad \forall\, f \in F\}. \tag{4}$$

Let r_f denote the sending rate of flow f, then the data transfer time of flow f equals to transmission delay as

$$t_f = \frac{S}{r_f}. \tag{5}$$

The other delays, such as processing delay, propagation delay and queueing delay, are all negligible, because the flows of transferring data objects are large in size, e.g., 64 MB in HDFS.

Ideally, the least data retrieval time can be simply obtained by setting the sending rates of all flows to the maximum values possible. However, as multiple flows may pass through the same link, which has limited bandwidth, the sending rates of these flows may not be able to reach the maximum values simultaneously. Suppose F_e is the set of all flows passing through link e, then the aggregate data rate on link e (i.e., $\sum_{f \in F_e} r_f$) is bounded by its bandwidth as

$$\sum_{f \in F_e} r_f \le B_e. \tag{6}$$

Let P_e denote the set of all paths passing through link e. A flow f passes through link e if its selected path passes through link e, i.e., $\exists\, p \in P(f) \cap P_e$ such that $y_{fp} = 1$. Thus,

$$\sum_{f \in F_e} r_f = \sum_{f \in F} \sum_{p \in P(f) \cap P_e} r_f \cdot y_{fp} \le B_e. \tag{7}$$

Replacing r_f in (7) with S/t_f, we get

$$\sum_{f \in F} \sum_{p \in P(f) \cap P_e} \frac{S}{t_f} \cdot y_{fp} \le B_e. \tag{8}$$

Note that this constraint is not linear, thus together with $t_f \le t$ we transform it to a linear one,

$$\sum_{f \in F} \sum_{p \in P(f) \cap P_e} \frac{S}{t} \cdot y_{fp} \le B_e. \tag{9}$$

This new constraint means the amount of traffic passing through each link (i.e., $\sum_{f \in F} \sum_{p \in P(f) \cap P_e} S \cdot y_{fp}$) is bounded by the maximum amount of data that can be transmitted within data retrieval time (i.e., $B_e t$). In other words, the data retrieval time is the maximum data transmission time over all links.

Theorem 1. *Constraints (8) and (9) are equivalent.*

Proof. To prove the equivalence, we have to prove that any feasible t_f to (8) and t to (4) are also feasible to (9), and vice versa. First, since $t \ge t_f, \forall f \in F$, if (8) is satisfied, then (9) is also satisfied. Second, for any feasible t to (9), we can easily set all t_f to t, then both (4) and (8) are satisfied. □

To sum up, the data retrieval problem is to select nodes and paths for all requests, such that the data retrieval time is minimized. The data retrieval time is affected by those selections through the resulted amount of traffic on each link. It is formulated into an MIP as follows,

$$\min \quad t \tag{10a}$$

$$\text{s.t.} \sum_{f \in F_{jk}} x_f = 1 \qquad \forall\,(j,k) \text{ where } A_{jk} = 1 \tag{10b}$$

$$\sum_{p \in P(f)} y_{fp} = x_f \qquad \forall\, f \in F \tag{10c}$$

$$\sum_{f \in F} \sum_{p \in P(f) \cap P_e} S \cdot y_{fp} \le B_e \cdot t \qquad \forall\, e \in E \tag{10d}$$

$$x_f \in \{0,1\} \qquad \forall\, f \in F \tag{10e}$$

$$y_{fp} \in \{0,1\} \qquad \forall\, f \in F,\ p \in P(f) \tag{10f}$$

$$t \ge 0. \tag{10g}$$

3.2 Multiple Applications

In cloud, multiple applications run on shared resources simultaneously to utilize resources more efficiently. To select nodes and paths for the requests of each application, a simple approach is to treat all applications as a single one and to solve the problem using the above model. This is equivalent to minimize the maximum data retrieval time among all applications. However, different applications may have different requirements on their data retrieval time, the naive approach ignores the difference of requirements. Thus, instead of minimizing data retrieval time, we minimize a penalty.

Given a set of applications U, suppose application $u \in U$ has an upper bound $\bar{t}_u$ on its data retrieval time t_u, penalty c_u is defined as

$$c_u = \max\left\{\frac{t_u - \bar{t}_u}{\bar{t}_u},\ 0\right\}. \tag{11}$$

A penalty is induced if t_u exceeds a threshold, no penalty otherwise. The same to the single-application model, t_u of application u is dominated by the longest data transfer time among all its flows, that is

$$t_u = \max\{t_f, \quad \forall\, f \in F_u\}, \tag{12}$$

where F_u is the set of possible flows of application u.

To maintain fairness among applications, we minimize the maximum penalty denoted by c,

$$c = \max\{c_u, \quad \forall\, u \in U\}. \tag{13}$$

Thus, our problem is to select nodes and paths for the requests of each application, such that c is minimized. The same to the single-application model, the selections affect which flows pass through each link and the resulted aggregate data rate, restricted by the link bandwidth. But here aggregate data rate is aggregated over the flows in all applications rather than a single one. Let r_e^u denote the aggregate data rate of application u on link e, then

$$\sum_{u\in U} r_e^u \leq B_e. \tag{14}$$

Following the same rule of (7) r_e^u is computed as

$$r_e^u = \sum_{f\in F_u} \sum_{p\in P(f)\cap P_e} r_f \cdot y_{fp}. \tag{15}$$

Recall that $r_f = S/t_f$, combining above two constraints we obtain

$$\sum_{u\in U} \sum_{f\in F_u} \sum_{p\in P(f)\cap P_e} \frac{S}{t_f} \cdot y_{fp} \leq B_e. \tag{16}$$

Note that this constraint is not linear, thus together with (11), (13) and (12), we transform it to a linear one,

$$\sum_{u\in U} \sum_{f\in F_u} \sum_{p\in P(f)\cap P_e} \frac{S}{(c+1)\cdot \bar{t}_u} \cdot y_{fp} \leq B_e. \tag{17}$$

Theorem 2. *Constraints (16) and (17) are equivalent.*

Proof. To prove the equivalence, we have to prove that any feasible t_f and c to the set of constraints (11), (12), (13) and (16) are also feasible to (17), and vice versa. First, for a flow f of application u, i.e., $f \in F_u$, its data transfer time t_f must satisfy the following,

$$t_f \leq t_u \leq (c_u+1)\cdot \bar{t}_u \leq (c+1)\cdot \bar{t}_u. \tag{18}$$

In above deduction, the first inequality is obtained from (12), the second inequality is obtained from (11), and the last one is obtained from (13). Thus, if all t_f satisfy constraint (16), then c satisfies constraint (17).

Second, for any maximum penalty c which satisfies (17), we can build a set of t_f satisfying (16), setting t_f to the maximum possible value $(c+1)\cdot \bar{t}_u$ where $f \in F_u$ as follows,

$$t_f = t_u = (c_u+1)\cdot \bar{t}_u = (c+1)\cdot \bar{t}_u. \tag{19}$$

That is, all flows of an application have the same data transfer time, being proportional to the upper bound of the application, and all applications have the same penalty. All these results satisfy (11), (13), (12) and (16). □

Due to the equivalence proved above, (17) possesses the same meaning as (14), that is the aggregated data rate of all applications on each link is limited by its bandwidth. The difference is that in (17) we set the data transfer time of each flow to the maximum possible value, i.e., $(c+1)\cdot \bar{t}_u$.

Besides bandwidth constraints, the same as in the case of a single application, exactly one flow can be used for each request, and exactly one path can be selected for each used flow. To sum up, for the case of multiple applications the data retrieval problem is to select nodes and paths for the requests of each application, such that the maximum penalty among all applications is minimized. It is formulated into an MIP, as follows,

$$\min \quad c \tag{20a}$$

$$\text{s.t.} \sum_{f\in F_{jk}} x_f = 1 \qquad \forall\, (j,k) \text{ where } A_{jk}=1 \tag{20b}$$

$$\sum_{p\in P(f)} y_{fp} = x_f \qquad \forall\, f \in F \tag{20c}$$

$$\sum_{u\in U} \sum_{f\in F_u} \sum_{p\in P(f)\cap P_e} \frac{S}{\bar{t}_u} \cdot y_{fp} \leq B_e \cdot (c+1) \quad \forall\, e \in E \tag{20d}$$

$$x_f \in \{0,1\} \qquad \forall\, f \in F \tag{20e}$$

$$y_{fp} \in \{0,1\} \qquad \forall\, f \in F, p \in P(f) \tag{20f}$$

$$c \geq 0. \tag{20g}$$

3.3 Discussion on Complexity

The data retrieval problems for both cases are NP-hard, because even when source nodes are determined, the remaining problem to select paths is NP-hard. When a source node is determined for each request, a set of commodities are formed. Here we call a triple consisting of a source, a destination, and a demand (i.e., the amount of data to be routed) a commodity. For the case of a single application, given the set of commodities and a network, then our problem is to compute the maximum value $1/t$ for which there is a feasible multicommodity flow in the network with all demands multiplied by $1/t$, which is a *concurrent flow problem* [8]. Since we have the additional restriction that each commodity must be routed on one path, the problem is an *unsplittable concurrent flow problem*, which is NP-hard [8]. It is the same for the case of multiple applications.

4 Approximation Algorithm

We propose an approximation algorithm to solve the data retrieval problem.

4.1 Max-Throughput Algorithm

Given a data retrieval problem and its MIP formulation, our algorithm has three major steps: 1) solve its LP relaxation; 2) construct an integral solution from the relaxation solution using a rounding procedure; 3) analytically compute the data sending rate of each flow for scheduling.

The LP relaxation can be obtained by relaxing binary variables x_f and y_{fp}, whose optimal solution is denoted by x_f^* and y_{fp}^*. The rounding procedure constructs an integral solution x_f^A and y_{fp}^A from fractional solution x_f^* and y_{fp}^*, by keeping objective value changing as less as possible and two sets of constraints still satisfied: 1) *one-flow constraints*, that is exactly one flow is used for each request, i.e., (10b) or (20b); 2) *one-path constraints*, that is exactly one path is used for each flow, i.e., (10c) or (20c). We first select the flow that has the largest x_f^* for each request to construct x_f^A satisfying one-flow constraints. Then we select the path that has the largest y_{fp}^* for each flow in use to construct y_{fp}^A satisfying one-path constraints.

With nodes and paths determined, the objective value of a data retrieval problem can be derived analytically, so is the data sending rate of each flow. For the case of a single application, with variables x_f and y_{fp} having been determined, its MIP becomes

$$\min \quad t \tag{21a}$$

$$\text{s.t.} \quad \sum_{f\in F}\sum_{p\in P(f)\cap P_e} S\cdot y_{fp} \le B_e\cdot t \qquad \forall\, e\in E \tag{21b}$$

$$t\ge 0. \tag{21c}$$

Let Y denote the set of y_{fp}, then we have

$$t(Y) = \max\left\{\frac{\sum_{f\in F}\sum_{p\in P(f)\cap P_e} S\cdot y_{fp}}{B_e},\quad \forall\, e\in E\right\}, \tag{22}$$

that is the largest data transmission time among all links. As $t_f \le t(Y)$, we have $r_f = \frac{S}{t_f} \ge \frac{S}{t(Y)}$. To ensure bandwidth constraints, the data sending rate of each flow can be set to the same value as the slowest flow, that is,

$$r_f = \frac{S}{t(Y)}. \tag{23}$$

For the case of multiple applications, with variables x_f and y_{fp} having been determined, the amount of the traffic of application u on link e (denoted by $\beta_e^u(Y)$) is also determined as

$$\beta_e^u(Y) = \sum_{f\in F_u}\sum_{p\in P(f)\cap P_e} S\cdot y_{fp}. \tag{24}$$

The MIP becomes

$$\min \quad c \tag{25a}$$

$$\text{s.t.} \quad \sum_{u\in U}\frac{\beta_e^u(Y)}{\bar{t}_u} \le B_e\cdot (c+1) \qquad \forall\, e\in E \tag{25b}$$

$$c\ge 0. \tag{25c}$$

Then we have

$$c(Y) = \max\left\{0,\ \max\left\{\frac{\sum_{u\in U}\beta_e^u(Y)/\bar{t}_u}{B_e}, \forall\, e\in E\right\} - 1\right\}. \tag{26}$$

To ensure bandwidth constraints, the data sending rate of each flow can be set as $t_f = (c(Y)+1)\cdot \bar{t}_u$, where $f\in F_u$, following the analyses in Theorem 2.

We analyze that our algorithm has polynomial complexity. First, it takes polynomial time to solve a LP relaxation in the first step. Since given a data retrieval problem, there are polynomial number of variables and constraints in its LP formulation, and a LP can be solved in polynomial time [9]. Second, the rounding procedure in the second step takes polynomial time. Since in this procedure each x is compared once, so is each y. Finally, the last analytical step takes $O(|E|)$ time to compute (22) or (26).

4.2 Analysis on Approximation Ratio

Next, we analyze the approximation ratio of the above algorithm.

4.2.1 Single Application

The above approximation algorithm has an approximation ratio of RL, where R is the replication factor (the number of replicas) of data, and L is the largest number of candidate paths that can be used by a flow. Let t^A denote the objective value of an approximation solution, and $\textsc{opt}(MIP)$ denote the optimal value for an MIP instance, then we have

$$t^A \le RL\cdot \textsc{opt}(MIP). \tag{27}$$

In other words,

$$\textsc{opt}(MIP) \ge \frac{1}{RL}\cdot t^A. \tag{28}$$

The derivation of the approximation ratio is based on two facts: 1) the optimal value of the LP relaxation (denoted by t^*) provides a lower bound for the optimal value of an MIP, that is

$$\textsc{opt}(MIP) \ge t^*, \tag{29}$$

since the solution space of the LP relaxation becomes larger due to relaxation; 2) the optimal fractional solution y_{fp}^* and an approximation solution y_{fp}^A satisfy

$$y_{fp}^* \ge \frac{1}{RL}\cdot y_{fp}^A. \tag{30}$$

This can be derived from the process of building approximation solutions. In constructing y_{fp}^A for flow f, we have

$$y_{fp}^* \ge \frac{1}{|P(f)|}\cdot x_f^*\cdot y_{fp}^A, \quad \forall\, p\in P(f). \tag{31}$$

For $y_{fp}^A = 0$, $y_{fp}^*\ge 0$ is always valid; for $y_{fp}^A = 1$, $y_{fp}^* \ge \frac{1}{|P(f)|}\cdot x_f^*$ should be valid; otherwise, one-path constraints would be contradicted. This is because for flow f, path p whose y_{fp}^* is the largest is selected, so if y_{fp}^* where p is selected (i.e., $y_{fp}^A = 1$) is less than $\frac{1}{|P(f)|}\cdot x_f^*$, then any other y_{fp}^* where p is not selected is also less than $\frac{1}{|P(f)|}\cdot x_f^*$, and finally $\sum_{p\in P(f)} y_{fp}^* < \sum_{p\in P(f)}\frac{1}{|P(f)|}\cdot x_f^* = x_f^*$, which contradicts the one-path constraint. Furthermore, in constructing x_f^A we have that for flow f being selected (i.e., $x_f^A = 1$),

$$x_f^* \ge \frac{1}{R}. \tag{32}$$

Otherwise, one-flow constraints would be contradicted. This is because for a request, flow f whose x_f^* is the largest is selected, so if x_f^* whose flow is selected is less than $\frac{1}{R}$, then any other x_f^* whose flow is not selected is also less than $\frac{1}{R}$, and finally $\sum_{f\in F(q)} x_f^* < \sum_{f\in F(q)} \frac{1}{R} = 1$, which contradicts with the one-flow constraint. Recall that if y_{fp}^A is 1, then x_f^A must be 1, so when y_{fp}^A is 1, (32) must be satisfied. Thus, we have

$$y_{fp}^* \geq \frac{1}{|P(f)|} \cdot x_f^* \cdot y_{fp}^A \geq \frac{1}{|P(f)|} \cdot \frac{1}{R} \cdot y_{fp}^A \geq \frac{1}{RL} \cdot y_{fp}^A. \tag{33}$$

Based on the above two facts, we can derive the approximation ratio. Let e^* denote the bottleneck link having the maximum data transmission time in the optimal factional solution, and e^A denote the counterpart in an approximation solution, then we have

$$\textsc{opt}(MIP) \geq t^* \tag{34a}$$

$$= \frac{\sum_{f\in F}\sum_{p\in P(f)\cap P_{e^*}} S \cdot y_{fp}^*}{B_{e^*}} \tag{34b}$$

$$\geq \frac{\sum_{f\in F}\sum_{p\in P(f)\cap P_{e^A}} S \cdot y_{fp}^*}{B_{e^A}} \tag{34c}$$

$$\geq \frac{\sum_{f\in F}\sum_{p\in P(f)\cap P_{e^A}} S \cdot \frac{1}{RL} \cdot y_{fp}^A}{B_{e^A}} \tag{34d}$$

$$= \frac{1}{RL} \cdot t^A. \tag{34e}$$

In the above derivation, (34b) and (34e) are obtained from (22), (34c) is because e^* rather than e^A is the bottleneck link in a fractional solution, and (34d) is from the second fact (30). Therefore, the data retrieval time obtained by the approximation algorithm is at most RL times greater than the optimal value.

4.2.2 *Multiple Applications*

In the same manner, we can obtain that the approximation algorithm has the same approximation ratio of RL for the case of multiple applications, but now the approximation ratio affects penalty plus 1, i.e., $c+1$, rather than c, as follows,

$$c^A + 1 \leq RL \cdot (\textsc{opt}(MIP) + 1), \tag{35}$$

where c^A is the objective value of an approximation solution, and $\textsc{opt}(MIP)$ is the optimal value for an MIP instance. (35) means that the worst time ratio obtained by the approximation algorithm is at most RL times greater than the optimal value.

As $\textsc{opt}(MIP) \geq 0$, RHS of (35) $\geq RL \geq 1$, thus (35) is valid if $c^A = 0$. Next we prove it is valid if $c^A > 0$. Let e^* denote the bottleneck link having the largest $\sum_{u\in U} \beta_e^u(Y^*)/\bar{t}_u$ in the optimal fractional solution, and e^A denote the counterpart in an approximation solution, and let c^* denote the optimal value of the LP relaxation, then we have

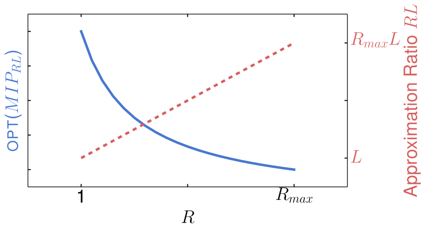


Fig. 3. $\textsc{opt}(MIP_{RL})$ and approximation ratio RL may change as R varies, with L fixed, where MIP_{RL} is an MIP instance formulated after preprocessing with parameters R and L.

$$\textsc{opt}(MIP) \geq c^* \tag{36a}$$

$$\geq \frac{\sum_{u\in U} \frac{\beta_{e^*}^u(Y^*)}{\bar{t}_u}}{B_{e^*}} - 1 \tag{36b}$$

$$\geq \frac{\sum_{u\in U} \frac{\beta_{e^A}^u(Y^*)}{\bar{t}_u}}{B_{e^A}} - 1 \tag{36c}$$

$$\geq \frac{\sum_{u\in U} \frac{\frac{1}{RL}\cdot\beta_{e^A}^u(Y^A)}{\bar{t}_u}}{B_{e^A}} - 1 \tag{36d}$$

$$= \frac{1}{RL} \cdot (c^A + 1) - 1. \tag{36e}$$

(36a) is from the fact that the optimal value of the LP relaxation provides a lower bound for the optimal value of an MIP. (36b) is obtained from (26). (36c) is because e^* rather than e^A is the bottleneck link in the optimal fractional solution. (36d) is because $\beta_e^u(Y^*) \geq \frac{1}{RL} \cdot \beta_e^u(Y^A)$, obtained from (24) and the fact $y_{fp}^* \geq \frac{1}{RL} \cdot y_{fp}^A$. (36e) is from the definition when $c^A > 0$.

4.3 Best Approximation Result

As analyzed previously, approximation results are upper bounded by approximation ratio RL and the optimal value of an MIP instance. The lower the upper bound is, the better approximation results may be. Thus, we may obtain better approximation results by reducing the upper bound. We propose a *preprocessing procedure* to reduce approximation ratio RL. That is, for each request we randomly select a subset of nodes as source candidates (R in total), and for each flow we randomly select a subset of paths as routing candidates (L in total), as the inputs of an MIP formulation. As such, RL is less than the value in the case without preprocessing. However, since data replicas and routing paths are selected from a subset of nodes and paths rather than the whole set, the optimal value of the current MIP instance may be worse than that in the case without preprocessing. If only one node is selected for each request as candidates, and only one path is selected for each flow as candidates, then the solution to the resulted MIP instance is obvious, which is comparable to the result of naive random selection method. Thus, with the preprocessing, as R or L decreases, the approximation ratio decreases, but the optimal value of the resulted MIP instance may increase, as illustrated in Fig. 3 where R varies and L is fixed (changes are similar when L varies and R is fixed). So there exists a pair of R

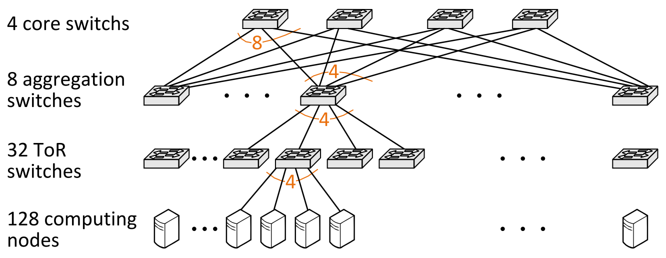


Fig. 4. A three-tier topology of data center network.

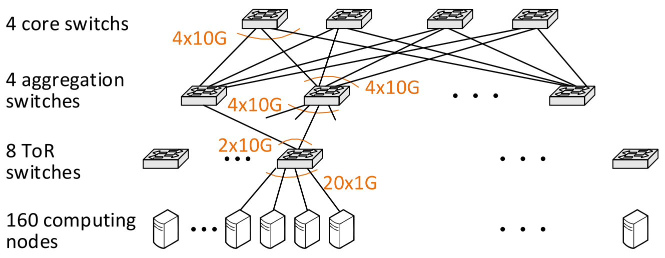


Fig. 5. A VL2 topology of data center network.

and L, such that the upper bound is lowest. With this motivation, we take the preprocessing procedure with various pairs of R and L, and train the approximation algorithm on the resulted MIP instances. Then, the best pair of R and L is selected, which leads to the best approximation result. It is practical, since it takes polynomial time to run the algorithm.

It is noted that some flows may be unable to have more than one candidate paths, which are all shortest paths, e.g., any two nodes in the same rack have only one shortest path in between. In this case, the preprocessing is unnecessary for those flows, and the value of L is not affected, since it is determined by the largest set of candidate paths.

4.4 Discussion on Deployment

When deploying our scheduling algorithms in real systems, the data retrieval problem can be solved once an application has been distributed in computing nodes, under the bandwidth conditions at that time. Once the application starts running, its data retrieval is scheduled according to precomputed results. During runtime, bandwidth conditions may change due to finishing or starting of other applications. To adapt to dynamic bandwidth conditions, the data retrieval problem can be resolved periodically. The length of the period depends on the tradeoff between computation overhead and how quickly the scheduling responses to changing bandwidth.

5 Performance Evaluation

In this section, we evaluate the performance extensively and demonstrate that our algorithm can obtain near optimal solutions, with the availability of additional data replicas and abundant paths. We also show that in many cases even one additional replica can improve performance greatly.

5.1 Methods for Comparison

We take two methods in comparison: 1) the optimal algorithm by solving the MIP (OPT for short); 2) the existing method by randomly selecting nodes and paths (RND for short), where a node is chosen from all available ones. In our simulations, both the MIP and the LP relaxation are solved by Gurobi [10] with a default setting.

5.2 Simulation Setups

We first introduce network topologies and parameters, and then discuss data retrieval setups.

Our simulation testbed has three types of Data Center topologies: 1) a *fat-tree topology* built from 8-port switches; 2) a *three-tier topology* (as shown in Fig. 4) in which every four neighboring hosts are connected to a ToR switch, and every four neighboring ToR switches are connected to an aggregation switch, and all eight aggregation switches are connected to each core switch (4 in total); 3) a *VL2 topology* [4] with 8-port aggregation and 4-port core switches, as shown in Fig. 5. Both the fat-tree topology and the three-tier topology have 128 computing nodes, while the VL2 topology has 160 instead.

We set link bandwidths as follows: in the fat-tree and the three-tier topologies each link is 1 Gbps; in the VL2 topology each server link is 1 Gbps, while each switch link is 10 Gbps, the same as in [4]. Since many different applications may share the network fabric in data centers, we simulate under two settings: *full bandwidth* and *partial bandwidth* in the presence of background traffic. We generate background traffic by injecting flows between random pairs of computing nodes, each of which passes along a random path in between and consumes 100 Mbps. In order to ensure that each link has at least 100 Mbps remained, we accept a background flow only if the currently available bandwidth of every link along its path is at least 200 Mbps. Otherwise we reject. The amount of remaining bandwidth depends on how many background flows are accepted. In our simulations, we inject 400 flows.

The inputs of a data retrieval problem include the locations of tasks and data objects, as well as access relationships between them. We generate these inputs synthetically. We place data first, and each data object is stored with replication factor of 3, as follows: 1) it is first placed in a randomly chosen node; 2) its first replica is stored in a distinctive node in the same rack; 3) its second replica is stored in a remote rack. This rule is the same to that in HDFS [2]. In real applications, most of the data is accessed locally, not affecting data retrieval time. Thus in simulations we only consider the non-local data requiring transfers. We assign tasks randomly on the nodes not having their required data. There are two types of access relationships between tasks and data objects: 1) *one-to-one*, where each task accesses a unique data object; 2) *many-to-one*, where multiple tasks access a common data object. Note that in both cases, each task only requires one data object to work with, and this is because other relationships (one-to-many and many-to-many) are special cases of our consideration. In many-to-one setting, when a node has several tasks which access the same data, we assume that the node initiates only one data request and the retrieved data can be used by all the tasks.

Both the amount of data and the amount of tasks are varied in simulations. For one-to-one setting, the number of tasks equals to the number of data objects, both varying

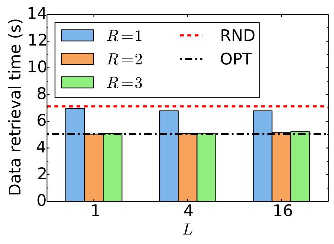


Fig. 6. The performance of APX with different R and L, where the dashed lines represent the results of RND (red) and OPT (black). The APX with R being 2 or 3 perform close to OPT.

from 100 to 1000. For many-to-one setting, the number of data objects is set to 100, and the number of tasks is varied from 100 to 1,000. All simulation results are averaged over 20 runs.

5.3 Single Application

We first evaluate the data retrieval time of a single application. It is obtained from objective value for OPT, while for other algorithms it can be computed as in (22). We use the reduction ratio of data retrieval time over RND to evaluate the performance improvement of our algorithm (i.e., $(t_{RND} - t)/t_{RND}$, where t is the data retrieval time of the algorithm being evaluated).

5.3.1 Fat-Tree Topology

We first simulate for a fat-tree topology. We start with running simulations to find optimal R and L where APX performs best. As now $R_{\max}$ is 3 and $L_{\max}$ is 16, we try nine pairs, where R is chosen from {1, 2, 3} and L is chosen from {1, 4, 16}, both adjusted by the preprocessing of random selections discussed in Section 4.3. We simulate a scenario of having 500 tasks and 100 data objects with full bandwidth and many-to-one access relationship. The performance of the approximation algorithm (APX for short) in nine settings are shown in Fig. 6, where the results of RND and OPT are represented by red line and black line respectively. It is observed that the APX algorithms with R being 2 or 3 perform close to OPT. Thus we choose to evaluate APX with $R = 2$ and $L = 1$ (APX-R2-L1 for short), besides the one without the preprocessing (APX-Rmax-Lmax for short).

The results are shown in Fig. 7. It is observed that APX-R2-L1 performs almost as good as OPT, much better than RND, and a bit better than APX-Rmax-Lmax. In the topology with full bandwidth, the reduction ratio is around 3%∼13% for one-to-one setting; while for many-to-one setting it increases significantly (20%∼40%) as more tasks

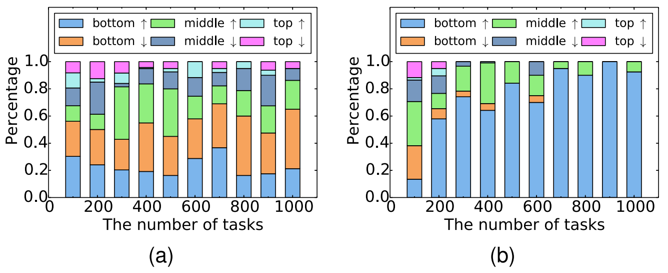


Fig. 8. The percentages of each link-type in bottleneck links of RND. (a) is for one-to-one access relationship; (b) is for many-to-one access relationship. ↑ represents uplink, and ↓ represents downlink.

access each data at the same time. This is because, in RND, when many tasks access the same set of data objects, some tasks are possible to choose the same node to retrieve a common data, resulting in heavy congestions near the selected node. The more the tasks, the worse the congestions, and the longer the retrieval time. To verify the locations of the congestions in RND, we calculate the percentages of each link-type in bottleneck links (that is where data transmission time equals to the data retrieval time). There are six link-types belonging to three layers and two directions. The results are shown in Fig. 8. Fig. 8b demonstrates that for many-to-one access relationship as the number of tasks increases, the bottlenecks mainly consist of the uplinks in the bottom layer. In comparison, for one-to-one access relationship, congestions probably happen at any layer of links, as demonstrated in Fig. 8a.

For the topology with partial bandwidth, the reduction ratios are significant, as shown in Fig. 7c and Fig. 7d, i.e., roughly around 40 percent for one-to-one setting, and 50 percent for many-to-one setting. Both ratios are higher than that in the case of full bandwidth, because RND ignores different bandwidths, in addition to path and replica diversities, which should have been considered in selecting nodes and paths, as our algorithm does.

We demonstrate that data retrieval time cannot be minimized when we ignore replica diversity. We simulate two methods. One fully utilizes replica diversity but randomly chooses a path for each possible flow formed by each replica (OPT-Rmax-L1 for short), and the other randomly chooses a replica for each request (ignoring replica diversity) but fully utilizes path diversity (OPT-R1-Lmax for short). They are formulated into MIPs and solved optimally. We compare them to RND and OPT, and show the results in Fig. 9. It is observed that OPT-Rmax-L1 performs as good as OPT, but OPT-R1-Lmax performs much worse than OPT, even close

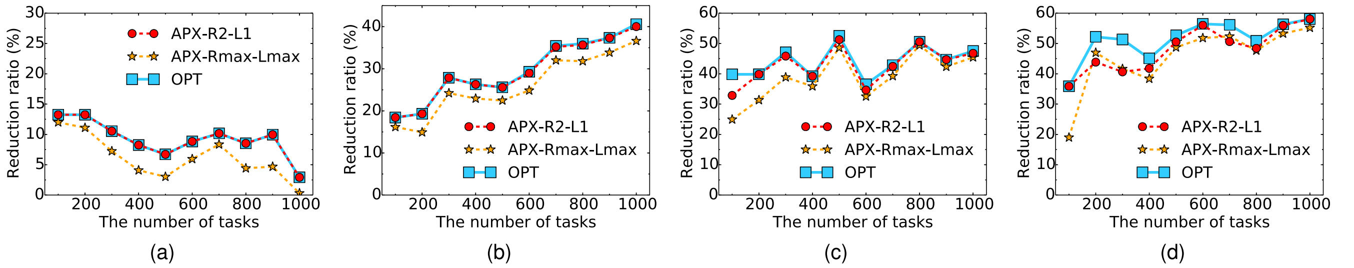


Fig. 7. The reduction ratio of data retrieval time over RND for a fat-tree topology. (a) is for full bandwidth and one-to-one access relationship; (b) is for full bandwidth and many-to-one access relationship; (c) is for partial bandwidth and one-to-one access relationship; (d) is for partial bandwidth and many-to-one access relationship.

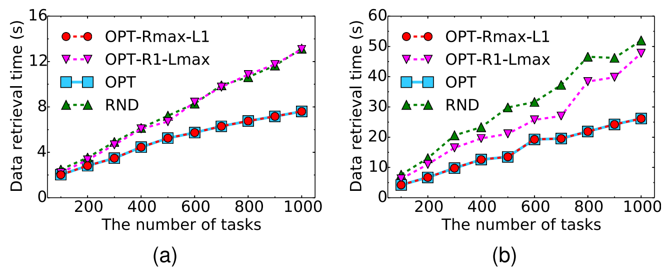


Fig. 9. The data retrieval time of an application for a fat-tree topology, using the methods discussed in Section 5.3.1. (a) is for full bandwidth and many-to-one access relationship, (b) is for partial bandwidth and one-to-one access relationship.

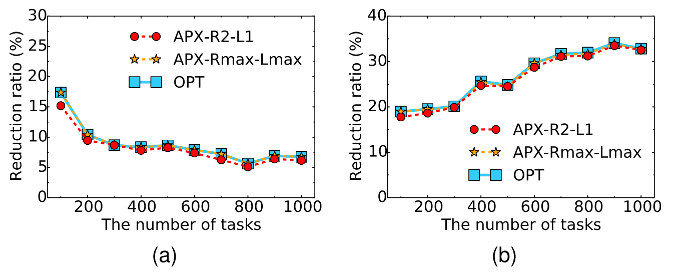


Fig. 10. The reduction ratio of data retrieval time over RND, for a three-tier topology with partial bandwidth. (a) is for one-to-one access relationship; (b) is for many-to-one access relationship.

to RND, in various simulation settings. Thus, it validates the motivation to exploit replica diversity. In addition, Fig. 9a illustrates that in the topology with full bandwidth, path diversity may not improve performance; Fig. 9b illustrates that in a topology with partial bandwidth, path diversity may improve performance but not as much as replica diversity does.

5.3.2 *Three-Tier Topology*

We also simulate for a three-tier topology. Now $R_{\max}$ is 3 and $L_{\max}$ is 4. We simulate APX-Rmax-Lmax and the APX with two replicas and one path randomly selected in the preprocessing (i.e., APX-R2-L1) as above. We show reduction ratios in Fig. 10. The results are for a topology with partial bandwidth, and the results for the topology with full bandwidth are similar, which are omitted due to limited space. It is demonstrated that both APX algorithms perform almost as good as OPT, around 6%$\sim$17% better than RND in one-to-one setting, and around 20%$\sim$32% better than RND in many-to-one setting. The reduction ratios in both cases are lower than that for fat-tree topology, because in three-tier topology the difference made by replica selections or path selections is not as significant as that in fat-tree topology. Specifically, in three-tier topology, congestions always happen on the links between ToR layer and aggregation layer due to oversubscribed bandwidths, since each of those links is shared by four nodes connected to its ToR switch. To mitigate congestions, we can select replicas such that the traffic on those congestion links can be balanced. However, such mitigation is less than that for fat-tree topology, because two replicas in the same rack share a common link between ToR and aggregation layers, leading to the same traffic. In addition, we cannot mitigate congestions by selecting paths, because the data transfers below aggregation layer have only one path to use.

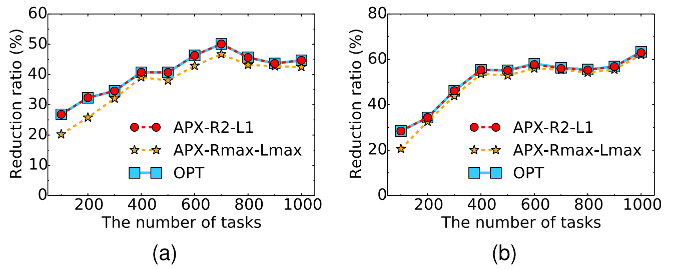


Fig. 11. The reduction ratio of data retrieval time over RND, for a VL2 topology with partial bandwidth. (a) is for one-to-one access relationship; (b) is for many-to-one access relationship.

5.3.3 *VL2 Topology*

For the VL2 topology as shown in Fig. 5, we simulate APX-R2-L1 and APX-Rmax-Lmax where $R_{\max}$ is 3 and $L_{\max}$ is 16. We show reduction ratios in Fig. 11. The results are for a topology with partial bandwidth, and the results for the topology with full bandwidth are similar, which are omitted. It is demonstrated that both APX algorithms perform almost as good as OPT, around 25%$\sim$50% better than RND in one-to-one setting, and around 30%$\sim$60% better than RND in many-to-one setting. The results are similar to that in the fat-tree topology, since both topologies have richly connected links and full-bisection bandwidths.

5.4 Multiple Applications

In this section, we evaluate the performance of multiple applications. The same to the case of a single application, we take OPT and RND for comparison. The worst penalty among all applications is used for evaluation. It is directly

TABLE 1
The Settings of the Upper Bounds on Data Retrieval Time in the Simulation of Two Applications (S)

			the number of tasks									
topology	bandwidth	access	100	200	300	400	500	600	700	800	900	1,000
fat-tree	partial	many-to-one	(6, 9)	(7, 11)	(9, 13)	(13, 20)	(16, 24)	(17, 25)	(19, 29)	(22, 32)	(23, 35)	(26, 38)
fat-tree	partial	one-to-one	(5, 8)	(8, 12)	(12, 18)	(15, 23)	(16, 24)	(23, 35)	(23, 35)	(26, 39)	(29, 44)	(31, 47)
fat-tree	full	many-to-one	(2, 4)	(3, 5)	(4, 6)	(5, 8)	(6, 9)	(7, 10)	(8, 11)	(8, 12)	(9, 13)	(9, 14)
fat-tree	full	one-to-one	(2, 4)	(4, 5)	(4, 6)	(5, 8)	(6, 9)	(7, 10)	(8, 12)	(9, 13)	(9, 13)	(11, 16)
three-tier	partial	many-to-one	(37, 56)	(64, 96)	(90, 135)	(112, 169)	(137, 206)	(155, 233)	(171, 256)	(194, 291)	(214, 321)	(239, 359)
three-tier	partial	one-to-one	(38, 57)	(62, 94)	(92, 138)	(112, 168)	(136, 204)	(157, 235)	(181, 271)	(207, 311)	(228, 342)	(241, 361)
VL2	partial	many-to-one	(4, 6)	(6, 10)	(8, 12)	(8, 13)	(10, 15)	(11, 17)	(13, 19)	(15, 23)	(16, 23)	(15, 23)
VL2	partial	one-to-one	(4, 6)	(7, 10)	(8, 12)	(10, 15)	(10, 15)	(14, 21)	(15, 23)	(19, 28)	(20, 29)	(22, 33)

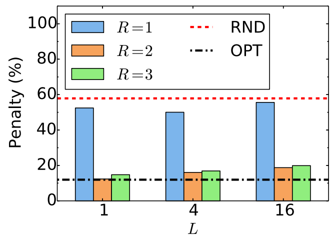


Fig. 12. The performance of APX with different R and L, where the dashed lines represent the results of RND (red) and OPT (black). The APX with $R=2$ and $L=1$ performs closest to OPT.

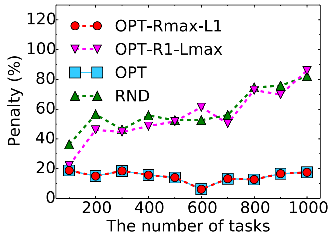


Fig. 14. The worst penalty of two applications for the fat-tree topology with full bandwidth and many-to-one access relationship, using the methods discussed in Section 5.3.1.

obtained from objective value for OPT, while for APX and RND it can be computed as in (26).

5.4.1 Fat-Tree Topology

We simulate two applications having separate data retrieval setups, each of which is generated as introduced in Section 5.2. We set different upper bounds on their data retrieval time. The optimal data retrieval time in the case of a single application (obtained from OPT) is used as a baseline. One upper bound is set to 1.2 times, and the other is set to 1.8 times as much as the baseline. The settings are listed in Table 1, averaged over 20 runs.

We also start with running simulations to find optimal R and L where APX performs best. Nine pairs of R and L are tried, where R is chosen from {1, 2, 3} and L is chosen from {1, 4, 16}. We simulate a scenario of two applications each having 500 tasks and 100 data objects with many-to-one access relationship under full bandwidth. The performance of APX in nine settings are shown in Fig. 12, where the results of RND and OPT are represented by red line and black line respectively. It is observed that the APX with $R=2$ and $L=1$ performs closest to OPT. Thus we choose to evaluate APX with this setting (i.e., APX-R2-L1), besides the one without the preprocessing (i.e., APX-Rmax-Lmax).

Next we evaluate the performance for two-application scenario. We show results in Fig. 13. For full bandwidth as shown in Figs. 13a and 13b, it is observed that APX-R2-L1 performs almost as good as OPT, and RND is much worse than OPT, with APX-Rmax-Lmax being in between. In one-to-one setting, APX-R2-L1 reduces penalty by 15 percentage points roughly, and in many-to-one setting, it reduces penalty by 20$\sim$60 percentage points. APX-Rmax-Lmax is usually better than RND, and is comparable with RND in some settings of few tasks. For partial bandwidth as shown in Figs. 13c and 15b, it is observed that APX algorithms with both parameters perform close to OPT, and significantly better than RND. In one-to-one setting, both APXs reduce penalty by 25$\sim$90 percentage points, and in many-to-one setting, they reduce penalty by 70$\sim$200 percentage points.

We also demonstrate that for the case of multiple applications, penalty cannot be minimized if we ignore replica diversity. We simulate OPT-Rmax-L1 and OPT-R1-Lmax (discussed in Section 5.3.1), and compare them to OPT and RND. Remind that in OPT-R1-Lmax, replica diversity is ignored, as data replica is randomly selected. The results for the fat-tree topology with full bandwidth and many-to-one access setting are shown in Fig. 14, and the results for other settings are similar, which are omitted due to limited space. It is observed that OPT-Rmax-L1 performs as good as OPT, but OPT-R1-Lmax performs much worse than OPT, similar to RND. Thus, it validates our motivation to exploit replica diversity.

5.4.2 Three-Tier Topology and VL2 Topology

We also simulate the case of two applications in a three-tier topology and a VL2 topology. The upper bounds on their data retrieval time are set as discussed previously, listed in Table 1. The simulation results for the three-tier topology are shown in Fig. 15, and that for the VL2 topology are shown in Fig. 16. We only show the results for partial bandwidth, and the results for full bandwidth are similar, which are omitted due to limited space. It is observed that both APX algorithms perform close to OPT, much better than RND. For the three-tier topology, APX algorithms reduce penalty by 5$\sim$20 percentage points in one-to-one setting, and reduce penalty by 20$\sim$50 percentage points in many-to-one setting. For the VL2 topology, they reduce penalty by 37$\sim$65 percentage points in one-to-one setting, and reduce penalty by 87$\sim$260 percentage points. All of these results demonstrate that our algorithm is effective.

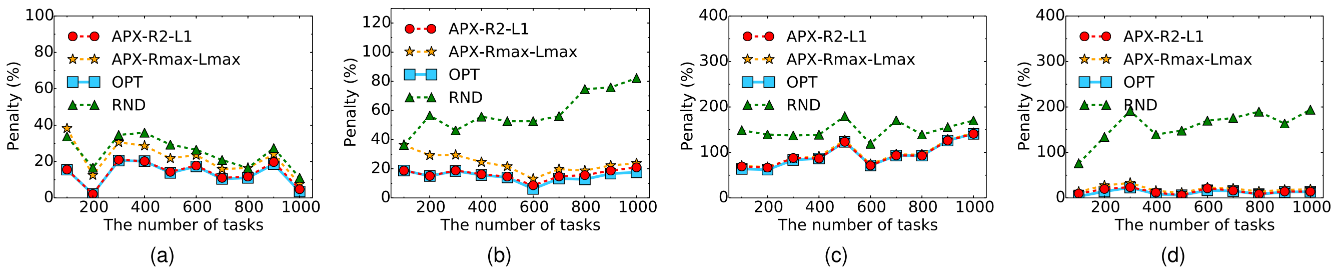


Fig. 13. The worst penalty of two applications for a fat-tree topology. (a) is for full bandwidth and one-to-one access relationship; (b) is for full bandwidth and many-to-one access relationship; (c) is for partial bandwidth and one-to-one access relationship; (d) is for partial bandwidth and many-to-one access relationship.

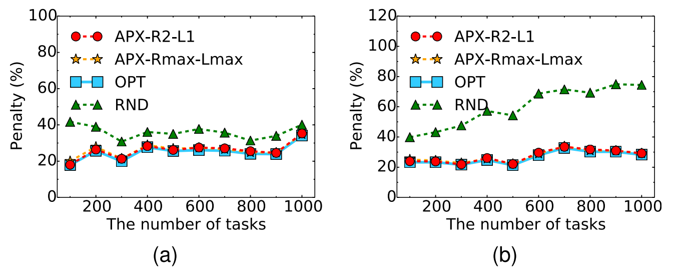


Fig. 15. The worst penalty of two applications for a three-tier topology with partial bandwidth. (a) is for one-to-one access relationship; (b) is for many-to-one access relationship.

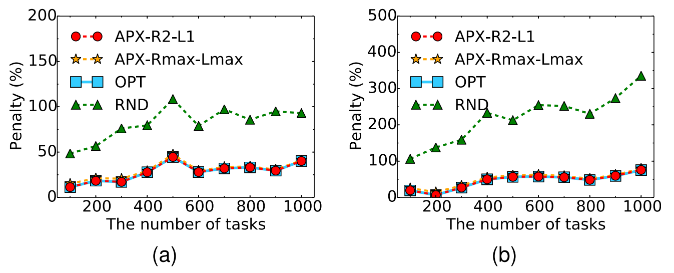


Fig. 16. The worst penalty of two applications for a VL2 topology with partial bandwidth. (a) is for one-to-one access relationship; (b) is for many-to-one access relationship.

6 Related Works

We review the existing works related to the data retrieval problem, and categorize them into three groups: traffic scheduling (in three levels), replica selection, and joint replica selection and routing.

6.1 Packet-Level Traffic Scheduling

Dixit et al. in [11] argued that packet-level traffic splitting, where packets of a flow are sprayed through all available paths, would lead to a better load-balanced network and much higher throughput compared to ECMP. Tso and Pezaros in [12] proposed to improve link utilization by implementing penalizing exponential flow-spliting algorithm in data center.

6.2 Flow-Level Traffic Scheduling

Greenberg et al. in [4] proposed using per-flow Valiant Load Balancing to spread traffic uniformly across network paths. Benson et al. in [13] developed a technique, MicroTE, that leverages the existence of short-term predictable traffic to mitigate the impact of congestion due to the unpredictable traffic. Al-Fares et al. in [6] proposed a flow scheduling system, exploiting path diversity in data center, to avoid path collisions. Cui and Qian in [14] proposed Distributed Flow Scheduling (DiFS) to balance flows among different links and improves bandwidth utilization for data center networks. Alizadeh et al. in [15] proposed a very simple design that decouples flow scheduling from rate control, to provide near-optimal performance.

6.3 Job-Level Traffic Scheduling

Chowdhury et al. in [7] proposed a global network management architecture and algorithms to improve data transfer time in cluster computing. They focused on the massive data transfer between successive processing stages, such as shuffle between the map and reduce stages in MapReduce. Dogar et al. in [16] designed a decentralized task-aware scheduling system to reduce task completion time for data center applications, by grouping flows of a task and scheduling them together.

Although these traffic scheduling methods can be used to schedule flows in data retrievals, but they do not optimize replica selections for flows, which we focus on.

6.4 Replica Selection in Data Grids

AL-Mistarihi and Yong in [17] researched on the replica selection problem in a Grid environment that decides which replica location is the best for Grid users. Their aim is to establish fairness among users in selections. Rahman et al. in [18] proposed replica selection strategies to minimize access latency by selecting the best replica. These works do not consider the impact of route selections on data transfer time.

6.5 Joint Replica Selection and Routing

Valancius et al. in [19] designed a system that performs joint content and network routing for dynamic online services. The system controls both the selection of replicas and the routes between the clients and their associated replicas. They demonstrated the benefits of joint optimization. With similar goals, Narayana et al. in [20] proposed to coordinate modular mapping and routing systems, already owned by OSP, to achieve global performance and cost goals of a joint system. Xu and Li in [21] distributed the joint optimization for scale. These works for wide area networks are inapplicable to data center networks, because data centers have much different traffic and data access patterns.

7 Conclusion

In this paper, we investigate the data retrieval problem in the DCN, that is to jointly select data replicas and paths for concurrent data transfers such that data retrieval time is minimized (i.e., throughput is maximized). We propose an approximation algorithm to solve the problem, with an approximation ratio of RL, where R is the replication factor of data and L is the largest number of candidate paths. We also solve the data retrieval problem for the case of multiple applications, keeping fairness among them. The simulations demonstrate that our algorithm can obtain near-optimal performance with the best R and L.

Acknowledgments

This work was supported by grant from Research Grants Council of Hong Kong [Project No. CityU 11205014].

References

[1] J. Dean and S. Ghemawat, “Mapreduce: Simplified data processing on large clusters,” *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[2] D. Borthakur. (2008). “HDFS architecture guide,” [Online]. Available: http://hadoop.apache.org/docs/r1.2.1/hdfs design.pdf

[3] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, “BCube: A high performance, Server-centric network architecture for modular data centers,” in *Proc. SIGCOMM*, 2009, pp. 63–74.

[4] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A scalable and flexible data center network," in *Proc. SIGCOMM*, 2009, pp. 51–62.

[5] C. Hopps. (2000). "Analysis of an equal-cost multi-path algorithm," RFC 2992, IETF, [Online]. Available: https://tools.ietf.org/html/rfc2992

[6] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. 7th USENIX Conf. Netw. Syst. Des. Implementation*, 2010, pp. 19–19.

[7] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," in *Proc. SIGCOMM*, 2011, pp. 98–109.

[8] J. L. Gross, J. Yellen, and P. Zhang, *Handbook of Graph Theory*, 2nd ed. London, U.K.: Chapman & Hall, 2013.

[9] R. J. Vanderbei, *Linear Programming: Foundations and Extensions*. New York, NY, USA: Springer, 1996.

[10] I. G. Optimization. (2015). Gurobi optimizer reference manual [Online]. Available: http://www.gurobi.com

[11] A. Dixit, P. Prakash, and R. R. Kompella, "On the efficacy of fine-grained traffic splitting protocolsin data center networks," in *Proc. SIGCOMM*, 2011, pp. 430–431.

[12] F. P. Tso and D. Pezaros, "Improving data center network utilization using near-optimal traffic engineering," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1139–1148, Jun. 2013.

[13] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine grained traffic engineering for data centers," in *Proc. 7th Conf. Emerging Netw. Exp. Technol.*, 2011, pp. 8:1–8:12.

[14] W. Cui and C. Qian, "DiFS: Distributed flow scheduling for adaptive routing in hierarchical data center networks," in *Proc. 10th ACM/IEEE Symp. Archit. Netw. Commun. Syst.*, 2014, pp. 53–64.

[15] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pFabric: Minimal near-optimal datacenter transport," in *Proc. SIGCOMM*, 2013, pp. 435–446.

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

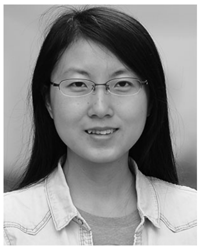
[17] H. H. E. AL-Mistarihi and C. H. Yong, "On fairness, optimizing replica selection in data grids," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 8, pp. 1102–1111, Aug. 2009.

[18] R. M. Rahman, R. Alhajj, and K. Barker, "Replica selection strategies in data grid," *J. Parallel Distrib. Comput.*, vol. 68, no. 12, pp. 1561–1574, 2008.

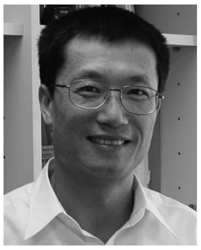
[19] V. Valancius, B. Ravi, N. Feamster, and A. C. Snoeren, "Quantifying the benefits of joint content and network routing," in *Proc. ACM SIGMETRICS/Int. Conf. Meas. Modeling Comput. Syst.*, 2013, pp. 243–254.

[20] S. Narayana, W. Jiang, J. Rexford, and M. Chiang, "Joint server selection and routing for Geo-replicated services," in *Proc. IEEE/ACM 6th Int. Conf. Utility Cloud Comput.*, 2013, pp. 423–428.

[21] H. Xu and B. Li, "Joint request mapping and response routing for Geo-distributed cloud services," in *Proc. INFOCOM*, 2013, pp. 854–862.



Ruitao Xie received the BEng degree from the Beijing University of Posts and Telecommunications in 2008 and the PhD degree in computer science from the City University of Hong Kong in 2014. She is currently a senior research associate in the Department of Computer Science at the City University of Hong Kong. Her research interests include cloud computing, distributed systems, and wireless sensor networks.



Xiaohua Jia received the BSc and MEng degrees from the University of Science and Technology of China, in 1984 and 1987, respectively, and the DSc degree in information science from the University of Tokyo, in 1991. He is currently a chair professor with the Department of Computer Science at the City University of Hong Kong. His research interests include cloud computing and distributed systems, computer networks, and mobile computing. He is an editor of the *IEEE Internet of Things*, *IEEE Transaction on Parallel and Distributed Systems* in 2006 to 2009, *Wireless Networks*, *Journal of World Wide Web*, *Journal of Combinatorial Optimization*, etc. He is the general chair of ACM MobiHoc 2008, TPC co-chair of IEEE GlobeCom 2010 Ad Hoc and Sensor Networking Symp, area-chair of IEEE INFOCOM 2010 and 2015. He is a fellow of the IEEE.