# Distributed and Optimal RDMA Resource Scheduling in Shared Data Center Networks

Dian Shen[1], Junzhou Luo[1], Fang Dong[1], Xiaolin Guo[1], Kai Wang[1], John C.S. Lui[2]
[1]*School of Computer Science and Engineering, Southeast University, Nanjing, China*
*Email: {dshen, jluo, fdong, xlguo, kwang}@seu.edu.cn*
[2]*Department of Computer Science and Engineering, The Chinese University of Hong Kong, China*
*Email: cslui@cse.cuhk.edu.hk*

***Abstract*—Remote Direct Memory Access (RDMA) suffers from unfairness issues and performance degradation when multiple applications share RDMA network resources. Hence, an efficient resource scheduling mechanism is urged to optimally allocates RDMA resources among applications. However, traditional Network Utility Maximization (NUM) based solutions are inadequate for RDMA due to three challenges: 1) The standard NUM-oriented algorithm cannot deal with coupling variables introduced by multiple dependent RDMA operations; 2) The stringent constraint of RDMA on-board resources complicates the standard NUM by bringing extra optimization dimensions; 3) Naively applying traditional algorithms for NUM suffers from scalability and convergence issues in solving a large-scale RDMA resource scheduling problem.**

**In this paper, we present distributed and optimal resource scheduling for RDMA networks to tackle the aforementioned challenges. First, we propose Distributed RDMA NUM (DRUM) to model the RDMA resource scheduling problem as a new variation of the NUM problem. Second, we present a distributed algorithm based on the alternating directional method of multipliers (ADMM), which has the property of convergence guarantee. Third, we implement our proposed algorithm in the real-world RDMA environment, and extensively evaluate it through large scale simulations and testbed experiments. Experimental results show that our method significantly improves applications' performance under resource contention, achieving $1.4-1.7\times$ higher throughput even under heavy background traffic, and $69.3\%$ improvement in terms of network utility.**

## I. Introduction

Remote Direct Memory Access (RDMA) is a technology of high speed data transfer among applications across networks [1, 2]. Based on kernel bypass, RDMA allows applications to perform data transfers from user-space directly to RDMA Network Interface Card (RNIC) without the involvement of the operation system kernel. RDMA can achieve significantly higher throughput, lower latency, and lower CPU utilization than traditional TCP/IP based protocols, thus becoming a promising networking technology for data center applications.

Although the kernel bypassing design of RDMA is efficient for data transfer, it does not work well in a shared data center environment as reported in [3–5]. In particular, native RDMA does not provide efficient resource management across applications [4]. When multiple applications share the RDMA-enabled network, one greedy application could monopolize the resources by issuing a large batch of requests. Thus, RDMA fails to deliver quality of service (QoS), performance isolation, or fairness guarantee in the shared environment.

To this end, one urging question is how to design an efficient resource scheduling mechanism which optimally allocates the RDMA resources among applications. Existing proposals [3–6] addressed this issue at the system level, by implementing RDMA resource scheduling solutions with performance isolation and rate allocation. However, the scheduling policies were based primarily on engineering heuristics, and they are far away from the optimality of resource allocation efficiency. Theoretically, network resource scheduling optimization can be formulated as a Network Utility Maximization (NUM) problem [7–10], where the utility provides a metric to measure the optimality of resource allocation efficiency, such as QoS or fairness. While previous efforts [11–13] in TCP/IP networks have been successful in deriving optimal resource scheduling solutions based on solving some specific NUM, none has considered RDMA. In this paper, we propose a novel adoption of NUM to address the optimal RDMA resource scheduling.

However, the aforementioned NUM-based solutions require a strong assumption on the optimization variables and utility functions. Specifically, a common assumption used in previous methods is that there is only one set of optimization variables (e.g., a rate allocation vector) and the utility function is dependent on this set of optimization variables only [8, 11–13]. Because such an assumption contradicts the inherent complexities in RDMA networks, it is challenging to adopt the NUM for RDMA resource scheduling. The challenges include the following three aspects.

**Multiple dependent RDMA operations introduce many coupled variables.** RDMA exposes to higher layer applications multiple low-level hardware primitives such as operations on multiple queues. These operations, with different functionalities, can be described by different utility functions. Since these operations are dependent (Section II-C), the RDMA resource scheduling problem can be formulated as a multi-block optimization problem with coupled variables. Consequently, the assumption of NUM is invalid and that the standard algorithms are unable to deal with.

**Stringent constraint of RDMA on-board resources brings one extra dimension in the utility function.** RDMA caches the connection information on the RNIC to achieve low latency communications. As the on-board RNIC cache is limited, when the number of connections grows, the total size of connection states will exceed the RNIC cache size and cause

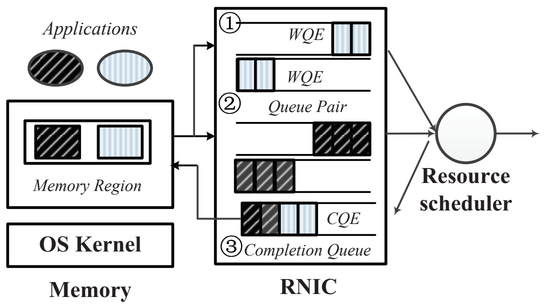


Fig. 1: RDMA abstracts resources in the semantics of queues and allows kernel bypass data transfer for applications.

cache thrashing (Section II-D), impairing the performance of hosting applications. Standard NUM [7, 8] treats the number of connections as unconstrained flows. With such a stringent cache constraint, the active connections in RDMA should be carefully selected or prioritized. Thus, this is another optimization dimension which invalidates the assumption of NUM, making the standard NUM more complicated to solve.

**The inherently large RDMA network scale causes the scalability issue.** In a production RDMA network, the number of hosts is on the order of $O(10^4)$ to $O(10^5)$, and the number of applications on each host is of $O(10^3)$. Thus the number of variables can be up to $O(10^8)$. Naively applying traditional algorithms for NUM suffers from scalability issues in solving such a large-scale optimization. Besides, some standard NUM algorithms such as Dual decomposition are difficult to tune, resulting in slow convergence. Thus, we are motivated to consider a more robust distributed optimization, and an implementation that can provide a fast and scalable solution.

In this paper, we present how to optimally schedule applications under a finite RDMA resource environment with a distributed algorithm. Specifically, we first model the RDMA resource scheduling problem as a new variation of the NUM problem to characterize the complexities of RDMA networks, called Distributed RDMA NUM (DRUM). DRUM is inherently a multi-block constrained optimization problem, which jointly optimize the sharing on multiple RDMA resources. Second, we modify the objective function of DRUM by applying a top-$k$ applications selector. We then analyze the tractability of the proposed model by convexity analysis. Third, we present a distributed solution which splits the large-scale global optimization problem into many small local subproblems. We propose a novel distributed algorithm based on the alternating directional method of multipliers (ADMM) and prove the convergence guarantee and parallelism of our ADMM-based algorithms through theoretical analysis.

To demonstrate the applicability of our proposed algorithms, we implement our proposed algorithm in the real-world RDMA environment and conduct extensive testbed experiments with a comparison to the state-of-the-art methods. Experimental results show that DRUM can significantly improve applications' performance in the shared RDMA network, achieving $1.4-1.7\times$ higher throughput under heavy background traffic and $69.3\%$ improvement in terms of total network utility.

To the best of our knowledge, this paper is the first work to discuss the theory, algorithms and implementation of the RDMA resource scheduling problem. Our major contributions are summarized as follows:

- We propose a new variation of the NUM model called DRUM, addressing the inherent complexities within RDMA networks.
- We present a distributed and modular algorithm to solve the problem, and prove the convergence guarantee and parallelism through theoretical analysis.
- We implement and evaluate our method through large scale simulations and testbed experiments. The experimental results show that the RDMA network achieves higher throughput and improved network utility, compared with the state-of-the-art methods.

## II. Background and Motivation

### A. RDMA and RDMA resource management

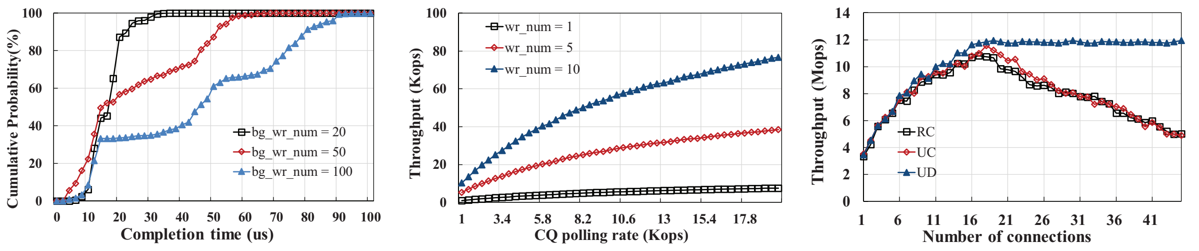
RDMA is a technology of high speed data transfer among applications across a network [1, 2]. Different from traditional TCP/IP networks, RDMA abstracts the resources as various queues, and applications implement the communication using these queues. As Fig. 1 shows, there are primarily three queues. The "send queue" and "receive queue" are always created in pairs and are referred to as a Queue Pair (QP). To perform data transfer, an application creates a QP and places instructions on the QP. These instructions are small data structures called Work Queue Elements (WQEs), which contain the memory location where the data resides and where it wants to send. The third one is the Completion Queue (CQ), which is used to notify the applications when the WQEs have been completed. The completion notifications are called Completion Queue Entries (CQEs). The applications actively poll the CQEs to determine the completion of messages sent.

When multiple applications share the network, native RDMA does not provide efficient management of resources across applications [4]. As a consequence, one application can simply post a large batch of WQEs to monopolize the resources, thereby degrading the performance of other applications. As a simple illustration in Fig. 2(a), the $80\%$ completion time of one foreground traffic degrades by $3.9\times$ from 20 $\mu$s to 78 $\mu$s when the co-located application increases the WQE requests batch size from 20 to 100.

### B. Network Utility Maximization

NUM is a modeling tool widely used in network resource management. In the NUM formulation, it captures network resource management objectives through utility functions and models various types of resource constraints as the constraint set. The basic NUM problem has the following formulation

$$\begin{aligned} \text{maximize} \quad & \sum_s U_s(x_s), \\ s.t. \quad & \forall s : x_s \in \mathcal{X}_s, \end{aligned}$$



(a) The 80% completion time of an 1KB RDMA message degrades from 19.8 $\mu$s to 74.3 $\mu$s when the WQE requests batch size of background application increases from 20 to 100.

(b) The throughput of applications grows with the increase of CQ polling rates by different amounts, when the batch size of WQEs are set as 1, 5, and 10, respectively.

(c) Average throughput of RC and UC connections drop drastically when the number of concurrent connections grows larger than 20.

Fig. 2: Characteristics of RDMA communication: (a) The unfairness issue in the shared network; (b) dependent RDMA operations; (c) cache thrashing issue under large amount of connections.

where $s$ denotes any source in the network, $x_s$ denotes the source rate such as the bandwidth, $U_s(x_s)$ denotes the utility functions, and $\{\mathcal{X}_s\}$ is the constraint set, such as bandwidth constraints. Previous researches modeled TCP/IP networks and derived resource scheduling solutions based on solving some specific NUM with particular utility functions [12, 13].

**Assumption of NUM**. In a standard NUM model, the rate allocation vector $\mathbf{x} = \{x_s\}$ is usually assumed to be the only set of optimization variables and utility functions $U_s(x_s)$ are often assumed to be dependent on $\mathbf{x}$ only [8, 12, 13].

However, such an assumption is not appropriate due to the inherent complexities in RDMA networks. In the following, we demonstrate the complexities inherent within RDMA networks through experimental results.

### C. The dependency of multiple RDMA operations

RDMA resource scheduling involves the management on multiple queues, e.g., allocating the WQE requesting rates on QP and deciding the CQE polling rates on CQ. At the application level, the operations on these queues are inherently dependent. Specifically, the applications usually implement the communications through a "WQE requesting-CQE polling-WQE requesting" loop that an application can only issue a batch of WQE requests after polling one CQE. The batch size is controlled by the parameter `wr_num`. For example, TensorFlow-RDMA [14] sets `wr_num` to be 1 when processing tensors and Spark [15] sets it dynamically according to the shuffle data in the buffer. In the experiment, we manually adjust the CQE polling rate of one application and measure its throughput. From Fig. 2(b), we observe that the throughput grows as the CQE polling rate increases, because the application can issue more WQE requests in one loop. Therefore, controlling one communication operation in RDMA affects the other in turn, yielding multiple dependent variables in the resource scheduling optimization.

### D. RNIC cache thrashing under large concurrent connections

RDMA caches the connection information on the RNIC to achieve low latency communications. However, since the RNIC cache is limited, when the number of connections grows, the total size of QP states will exceed the RNIC cache size and cause cache thrashing. Cache thrashing significantly impairs the performance of hosting applications. To better understand this issue, we vary the number of concurrent connections in one host and measure the average throughput. In the experiment, RDMA connections are set with different types: Reliable Connection (RC), Unreliable Connection (UC), and Unreliable Datagram (UD). As Fig. 2(c) shows, when the number of connections using RC and UC exceeds 20, the RNIC is unable to cache all the connection information and the average throughput drops significantly. Therefore, the number of QPs should be constrained to avoid cache thrashing and thus the active connections should be carefully selected or prioritized. This one more optimization dimension on the application selection makes the standard NUM rather complicated to solve.

## III. RDMA Resource Scheduling Optimization

In this section, we present a new mathematical model for RMDA resource scheduling with theoretical analysis.

### A. Problem formulation

We consider an RDMA-enabled data center network as illustrated in Fig. 3. It consists of a finite set of connected hosts and a finite set of applications consuming network resources. Let $h_i$ be the $i^{th}$ physical host. There are in total $n$ hosts in the network, i.e., $i \in \mathbb{Z}_n$, where $\mathbb{Z}_n = \{1, 2, ..., n\}$. Any $h_i$ runs up to $m$ RDMA-enabled applications. The scheduler allocates the WQE requesting rates $s_j$ to application $j \in \mathbb{Z}_m$, where $\mathbb{Z}_m = \{1, 2, ..., m\}$. Define $x_i \in \mathbb{R}_+^m$ as the rate allocation vector for $h_i$, we have $x_{i,j} = s_j$ and $x_i = \{s_1, s_2, ..., s_m\}, i \in \mathbb{Z}_n$. The problem is how to determine $x_i$ for each $h_i$. For a straight forward adoption of NUM, the model becomes

$$\begin{aligned} &\text{maximize} \sum_{i=1}^{n} f_i(x_i), \quad x_i \in \mathbb{R}_+^m, \\ &s.t. \quad h_i(x_i) \le q_i, \quad \forall i \in \mathbb{Z}_n, \end{aligned} \tag{1}$$

where $f_i(\cdot) : \mathbb{R}^m \to \mathbb{R}$ is the utility function. In general, $f_i(\cdot)$ could be an arbitrary concave function. For example, we can use a feasible function that satisfies the definition of proportional fairness [16], i.e., the summation of $\alpha$-fair utilities, $f_i(x_i) = \sum_{j=1}^{m}(1-\alpha)^{-1} s_j^{1-\alpha}, \forall s_j \in x_i$, where

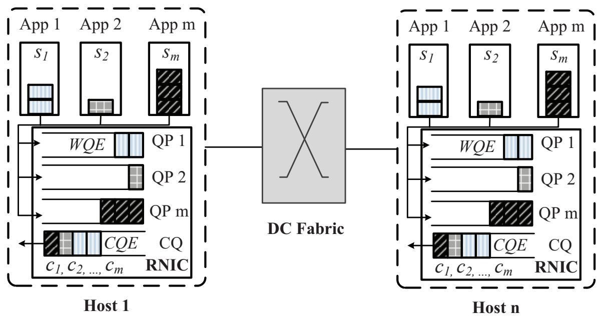


Fig. 3: RDMA resource scheduling model with $n$ connecting hosts and $m$ co-located applications

$0 < \alpha < 1$. The utility function is parameterized by $\alpha$. If $\alpha$ is set homogeneously for all applications, the solution will yield a fair resource allocation. The cost function $h_i(\cdot) : \mathbb{R}^m \to \mathbb{R}$ is an arbitrary convex function, such as physical bandwidth limitations or resource prices. We refer to $\mathbf{x}^* = \{x_1^*, x_2^*, ..., x_n^*\}$ as an optimal rate allocation if it solves problem (1), i.e., $\sum_i f_i(x_i^*) \geq \sum_i f_i(x_i), \forall x_i$. Traditionally, this problem can be solved via a convex programming solver or Dual Decomposition method.

However, as described in Section II, the complexities inherent in RDMA make the direct transformation from NUM impractical. First, RDMA abstracts the resources by multiple queues. In addition to QP and associated WQE requesting rates, scheduling CQ and associated CQE polling rates should also not be overlooked. We define another rate vector $z_i = \{r_1, r_2, ..., r_m\}, i \in \mathbb{Z}_n$, where $r_j$ is the allocated CQE polling rate of each application $j$ and $z_{i,j} = r_j, j \in \mathbb{Z}_m$. In RDMA, the CQE polling rate is closely related to the application-perceived latency, because the faster one application polls the CQE, the earlier it is notified the completion of message transferring. Therefore, the optimization of QP and CQ sharing should be conducted with different objectives. We hereby follow the existing work [17, 18] and define a different utility function for CQ to quantify the latency related utility.

**Definition 1. CQ Utility.** The function is characterized by a multiplier $g_i(z_i) = -\beta \cdot \sum_{j=1}^{m}(1/z_{i,j})^2$, where $z_i \in \mathbb{R}_+^m$.

In this definition, $\beta$ is the weighting factor that captures the relative importance of latency-related utility. The application-perceived latency for application $j$ on $h_i$ is captured by $(1/z_{i,j})^2$, which is inversely proportional with its CQE polling rate $z_{i,j}$. Then the overall utility function $g_i(\cdot)$ depends on the application-perceived latency. Clearly, $g_i(\cdot)$ is a concave function and achieves its maximum value when $z_i \to +\infty$.

To capture the dependency among operations on multiple queues, extra constraints should be imposed on $x_i$ and $z_i$. That is, $F(x_i) = z_i$ where $F(\cdot) : \mathbb{R}^m \to \mathbb{R}^m$. Essentially, the dependency among $x_i$ and $z_i$ is complicated, because the CQEs are generated by the completion of WQEs for both data sending and receiving. In this section, we first assume that $z_i$ only depends on the local WQEs sending rate, and we will discuss how to extend this model later. As described in Section II-C, the CQ polling rate and QP requesting rate have an approximately linear relationship. Then we have $F(x_i) = A \cdot x_i = z_i, A \in \mathbb{R}^{m \times m}$. We further define that $A = diag\{a_1, a_2, ..., a_m\}$, where $a_1$ varies for applications with different WQE batch size.

In the RDMA design, due to the on-board cache size limitation, the number $k$ of active connections using QPs on each $h_i$ should be bounded, i.e. $k < m$. In practice, $k$ is set at $k = 20$ and homogeneous across the data center. Therefore, the selection of $k$ active applications should be considered. Therefore, we rewrite the original $f_i(x_i)$ as $\widetilde{f}_i(x_i)$ using the following definition.

**Definition 2. QP Utility.** We define the utility as a multiplier $\widetilde{f}_i(x_i)$, where $x_i \in \mathbb{R}_+^m$. The multiplier prioritizes $k$ applications with largest utilities.

$$\widetilde{f}_i(x_i) = \sum_{j=1}^{k}(1-\alpha)^{-1}s_{[j]}^{1-\alpha}, \quad \forall s_{[j]} \in x_i, \tag{2}$$

where $[j]$ is the application with $j$-th largest utility value. Note that $\widetilde{f}_i(x_i)$ is not restrictive to a top-$k$ utility selection. It can be formulated to describe any prioritizing mechanism such as FIFO or smallest job first. Therefore, the utility function of QP is more complicated by applying the functionality of application selection. We theoretically analyze the complexity through convex analysis and derive the following lemma.

**Lemma 1.** *The function $\widetilde{f}_i(x_i)$ is strictly concave.*

**Proof.** First, order the element $j$ in vector $x_i$ by the value $(1-\alpha)^{-1}s_j^{1-\alpha}$ in the descending order. Then equation (2) is mathematically equal to sum the top $k$ largest ones up from $m$ elements, which is

$$\begin{aligned}\widetilde{f}_i(x_i) = max\{&(1-\alpha)^{-1}s_{[i_1]}^{1-\alpha} + ... + (1-\alpha)^{-1}s_{[i_k]}^{1-\alpha} \\ &|i_1, i_2, ..., i_k \in \mathbb{N}, 1 \leq i_1 \leq ... \leq i_k \leq m\}.\end{aligned} \tag{3}$$

Thus, there are $C_m^k$ combinations of such selection. We can define a binary vector $t \in \mathbb{R}^m$, where the corresponding element of an accepted application is set to 1, while the others be 0. Transform equation (3) to $\sup_t\{f_i(t \otimes x_{[i]})|\forall t\}$. This means that we conduct a supremum operation on $C_m^k$ combinations of $f_i(\cdot)$, which is $\widetilde{f}_i(\cdot) = \mathrm{g}(f_i(x_i))$, where $\mathrm{g}(\cdot) = \sup_t\{\cdot\}$. Because $f_i(x_i)$ is concave and $\mathrm{g}(\cdot)$ is the operation that maintains concavity, then $\widetilde{f}_i(\cdot)$ is still a concave function. Therefore, the lemma is proved. □

Other than the objective function, we consider the cost function $h_i(\cdot)$, which should be an arbitrary convex function that restricts the physical RNIC capabilities. We can use normalization functions such as the logarithm function, or a more general piecewise linear function with increasing slopes. For simplicity, we define $h_i^q(x_i) = \sum_j log(x_{i,j}) \leq q_i$ as the physical constraints for $x_i$, and $h_i^c(z_i) = \sum_j log(z_{i,j}) \leq c_i$ for $z_i$, where $q_i$ and $c_i$ are set according to the hardware specifications.

With the above definitions, we formulate a variant of the basic NUM from Equation (1) to characterize the resource scheduling problem in RDMA, denoted as DRUM, the objective of which is to jointly optimize the utility of $x_i$ and $z_i$. We rewrite problem (1) as

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} \widetilde{f}_i(x_i) + \sum_{i=1}^{n} g_i(z_i), \quad x_i, z_i \in \mathbb{R}_+^m, \\ s.t. \quad & h_i^q(x_i) \le q_i, \quad \forall i \in \mathbb{Z}_n, \\ & h_i^c(z_i) \le c_i, \quad \forall i \in \mathbb{Z}_n, \\ & F(x_i) = z_i, \quad \forall i \in \mathbb{Z}_n. \end{aligned} \tag{4}$$

Problem (4) is inherently a large scale multi-block optimization problem with large scale equality and inequality constraints and dependent optimization variables, which is hard to solve in a timely manner.

## *B. Problem analysis*

Traditionally, to solve problem (4) is based on the method of multipliers, in which there is an augmented Lagrangian parameter $\rho$, and Lagrangian dual variable $u$. The optimal value $x^*$, $z^*$ and dual optimal value $u^*$ are computed iteratively by solving the following equation:

$$\begin{aligned} (x^{k+1}, z^{k+1}) &:= arg \max_{x,z} \mathcal{L}_\rho(x, z, u^k), \\ (u^{k+1}) &= u^k + \rho(Ax^{k+1} - z^{k+1}). \end{aligned}$$

Here, the augmented Lagrangian $\mathcal{L}_\rho(x, z, u^k)$ is maximized jointly with respect to the two jointly dependent primal variables, which is inefficient to compute in practice. While traditional dual decomposition with (sub)gradient methods solving NUM suffer from the difficulty in parameter tuning and computational inefficiency, we leverage the Alternating Direction Method of Multipliers (ADMM) [19] method to solve problem (4), with $z_i$ as an auxiliary variable that assists iteration. In ADMM, $x_i$ and $z_i$ are updated in an alternating fashion; that is, $x_i^k$ is computed as an intermediate result from the previous state $(z_i^{k-1}, u_i^{k-1})$. The auxiliary variable $z_i^{k+1}$ is a function of $(x_i^{k+1}, u_i^k)$, and the Dual variable $u^{k+1}$ is updated from $u^k$ by collecting all the $x_i^{k+1}$ and $z_i^{k+1}$. Following the ADMM framework, we obtain an iteration on the order of the $x$-minimization step, $z$-minimization step and $u$-update step, which we illustrate as:

**Step 1**, $x$-minimization step:

$$\begin{aligned} \min_{x_i} -\widetilde{f}_i(x_i) + u^k(F(x_i) - z_i^k) + (\rho/2)||F(x_i) - z_i^k||_2^2 \\ s.t. \quad h_i^q(x_i) \le q_i. \end{aligned} \tag{5}$$

**Step 2**, $z$-minimization step:

$$\begin{aligned} \min_{z_i} -g_i(z_i) + u^k(F(x_i^{k+1}) - z_i) + (\rho/2)||F(x_i^{k+1}) - z_i||_2^2 \\ s.t. \quad h_i^c(z_i) \le c_i. \end{aligned} \tag{6}$$

**Step 3**, $u$-update step:

$$u^{k+1} = u^k + \rho \cdot (\sum_{i=1}^{n} F(x_i^{k+1}) + \sum_{i=1}^{n} z_i^{k+1}). \tag{7}$$

By design, the alternating update in ADMM reduces the computing complexity, and the extra regulation term $(\rho/2)||F(x_i) - z_i||_2^2$ stabilizes the iterations. One important property of the ADMM-based solution is its theoretical guarantee of convergence. The convergence is formally guaranteed by the following theorem:

**Theorem 1.** *When a feasible solution to the DRUM exists, the ADMM-based solution converges to the optimal value $p^*$. In particular, the following are true:*
*1. The consistency is achieved: $x_i^k \to F^{-1}(z_i^k)$ as $k \to +\infty$.*
*2. The solution is optimal: $\sum_{i=1}^{n} \widetilde{f}_i(x_i^k) + \sum_{i=1}^{n} g_i(z_i^k) \to p^*$ as $k \to +\infty$.*
*3. An optimal dual value is found: $\rho u^k \to \rho u^*$ as $k \to +\infty$.*

**Proof.** Let $(x_i^*, z_i^*)$ be the primal optimal solution to problem (4). In particular, $A \cdot x_i^* = z_i^*$. Let $u^*$ be a dual optimal solution. Because of the condition that a feasible solution to the DRUM exists, along with the strong duality theorem, $u^*$ exists. Then we have $(x_i^k, z_i^k, u^k)$ as an arbitrary set of results generated in iteration $k$. Let $v^k = u^k/\rho$ and $z_i^k = Ax_i^k + v^{k-1} - v^k$, so we have

$$V^k = \sum_{i=1}^{n} (||z_i^k - z_i^*||_2^2 + ||v^k - v^*||_2^2). \tag{8}$$

This equation leads to convergence if $\exists \varepsilon, D^k = V^{k+1} - V^k < \varepsilon$. [19] has proved that $D^k$ is Lipschitz continuous and $\varepsilon$ exists $\forall k$ with any start point $\{x_i^0, z_i^0, u^0\}$, so $\varepsilon$ is

$$\varepsilon = \rho||r^{k+1}||_2^2 + \rho||A^{-1}(z^{k+1} - z^k)||_2^2, \tag{9}$$

where $r^k$ is the primal residual after $k$-th iteration. Equation (9) holds with the saddle point theorem when the objective functions $\widetilde{f}_i(\cdot)$ and $g_i(\cdot)$ are closed, proper, and convex. Note that it does not require the combination $\widetilde{f}_i(\cdot) + g_i(\cdot)$ to be convex. With Lemma 1, $\widetilde{f}_i(x_i) = \sum_{j=1}^{k} (1 - \alpha)^{-1} s_{[j]}^{1-\alpha}$ is strictly concave, so $-\widetilde{f}_i(x_i)$ is a convex function. By definition, $g_i(z_i) = -q \cdot \sum_j (1/z_{ij})^2$. $g_i(\cdot)$ is concave when $dom(g_i) \in \mathbb{R}_+^m$. Since $z_i \in \mathbb{R}_+^m$, $-g_i(z_i)$ is a convex function. When the above conditions hold, we can derive the convergence of $V_k$. Along with the fact that $A$ is a diagonal matrix and its inverse $A^{-1}$ exists, we can derive that $A^{-1}(z^{k+1} - z^*)$ is bounded and both $r^{k+1}$ and $A^{-1}(z^{k+1} - z^k)$ go to 0, when $k \to +\infty$. Thus $F^{-1}(z_i^*) = x_i^*$, and Theorem 1.1 holds.

Since the inequality holds that $p^{k+1} - p^* \le u^{*T} r^{k+1}$, we have $lim_{k \to +\infty} p^k - p^* = 0$, i.e., the objective convergence. Thus Theorem 1.2 is proved.

Finally, as the Slater's condition holds, the dual gap is 0, so $u^k \to u^*$ when $p^k \to p^*$. Thus Theorem 1.3 is proved. □

Theorem 1 indicates that the ADMM-based method will converge to the optimality with enough $k$. We then analyze how to solve the local subproblem on each host $h_i$ efficiently. We computed $x_i$ guided by the following theorem.

**Theorem 2.** *$\forall h_i$, the solution for x-minimization step is*

$$x_i^{k+1} = \begin{cases} x_{i,j} = -\frac{1}{2}c_j + (c_j^2 - 4b_jd_j)^{1/2} b_j^{-1}, \forall x_{i,j} \notin I \\ \{x_{i,j}^{1/2} | b_j x_{i,j}^4 + c_j x_{i,j}^2 + x_{i,j} + d_j = 0\}, \\ \qquad \forall x_{i,j} \in I, \alpha = \frac{1}{2} \\ \{x_{i,j} | x_{i,j} = (b_j x_{i,j}^2 - c_j x_{i,j} - d_j)^{1/3}\}, \\ \qquad \forall x_{i,j} \in I, \alpha \neq \frac{1}{2} \end{cases}, \tag{10}$$

where $b_j = -\rho a_j^2$, $c_j = \rho a_j z_{i,j}^k + a_j u_j^k$ and $d_j = \lambda_i^q$.

**Proof.** For the $x$-minimization subproblem, the Lagrangian for the subproblem is

$$\begin{aligned}\mathcal{L}_{x_i,\lambda_i^q} = -\widetilde{f}_i(x_i) + u^k(F(x_i) - z_i^k) +\\ (\rho/2)||F(x_i) - z_i||_2^2 + \lambda_i^q(h_i(x_i) - qp_i).\end{aligned} \tag{11}$$

Based on this, we can derive the dual problem as

$$\begin{aligned}\text{maximize} \quad & D_{x_i}(\lambda_i^q) = \inf_{x_i \in D}\{\mathcal{L}_{x_i,\lambda_i^q}\},\\ & \lambda_i \geq 0, \quad \forall i.\end{aligned} \tag{12}$$

As $D_{x_i}(\lambda_i^q)$ is always convex, the solution of optimization problem (12) is based on the Karush-Kuhn-Tucker (KKT) conditions. Define $p_i^*$ as the primal optimal value, $x_i^*$ and $\lambda_i^{q*}$ as the primal and dual optimal solution, respectively, then we have the following set of equations.

$$\begin{cases}\lambda_i^{q*} \cdot (h_i(x_i^*) - q_i) = 0\\ \dfrac{\partial \mathcal{L}_{x_i,\lambda_i^q}}{\partial x_i}|_{x_i = x_i^*} = 0\\ \lambda_i > 0\end{cases} . \tag{13}$$

For the second term of equation(13), it can be rewritten as

$$\begin{aligned}&\frac{\partial \mathcal{L}_{x_i,\lambda_i^q}}{\partial x}|_{x_i = x_i^*} = 0\\ \Leftrightarrow\ & -\nabla \widetilde{f}_i(x_i^*) + Au^k + \rho(A^T A x_i^* - A z_i^k) + \lambda_i^q \nabla h_i(x_i^*) = 0\\ \Leftrightarrow\ & -x_i^{*-\alpha}|_I + Au^k + \rho(A^T A x_i^* - A z_i^k) + \lambda_i^q x_i^{*-1} = 0,\end{aligned} \tag{14}$$

where $I \in \mathbb{R}^m$ and $I$ contains element $I_j \in \{0, 1\}$. For example, $I = \{1, 1, 0, 1, ..., 0\}$. $x_i^{*-\alpha}|_I$ works as a filter that the vector $x_i^{*-\alpha}$ is partially selected by $I$, i.e., $x_i^{*-\alpha}|_I = diag\{I\} \cdot x_i^{*-\alpha}$. $I$ is derived by $\widetilde{f}_i(\cdot)$ that the corresponding item $I_j$ of selected applications are set to be 1. Arranging the equation (14), we have

$$\begin{aligned}-x_i^{*-\alpha}|_I + \rho A^T A \cdot x_i^* + \lambda_i x_i^{*-1} - \rho A \cdot z_i - Au_i^k = 0\\ \Leftrightarrow x_i^{*1-\alpha}|_I + B \cdot x_i^{*2} + C \cdot x_i^* + D = 0,\end{aligned} \tag{15}$$

where $B \in \mathbb{R}^{m \times m}$,$C, D \in \mathbb{R}^m$. $B = -\rho A^T A$, $C = \rho A z_i^k + Au^k$, and $D = -\lambda_i^q \mathbf{1}$. To solve this equation, we conduct element-wise computation to calculate the $x_{i,j}$. Then the calculation of $x_{i,j}$ becomes a scalar computation, which is divided into the following three cases:

**Case 1**: If $x_{i,j} \notin I$. In this case, $x_i^{1-\alpha}|_I$ is eliminated from equation (15), and we have

$$-\rho a_j^2 x_{i,j}^2 + (\rho a_j z_{i,j}^k + a_j u_j^k) x_{i,j} + \lambda_i^q = 0. \tag{16}$$

Equation (16) is a quadratic function of $x_{i,j}$ and its root can be calculated directly by $x_{i,j} = \frac{1}{2} - c_j + (c_j^2 - 4b_j d_j)^{1/2} b_j^{-1}$, where $b_j = -\rho a_j^2$, $c_j = \rho a_j z_{i,j}^k + a_j u_j^k$ and $d_j = \lambda_i^q$.

**Case 2**: If $x_{i,j} \in I$ and $\alpha = 1/2$. In this case, rewrite equation (15) and substitute $x_{i,j}$ by $x_{i,j}^{1/2}$, we have

$$\begin{aligned}x_{i,j}^{1/2} - \rho a_j^2 x_{i,j}^2 + (\rho a_j z_{i,j}^k + a_j u_j^k) x_{i,j} + \lambda_i^q = 0\\ \Leftrightarrow \rho a_j^2 x_{i,j}^4 + (\rho a_j z_{i,j}^k + a_j u_j^k) x_{i,j}^2 + x_{i,j} + \lambda_i^q = 0.\end{aligned} \tag{17}$$

Equation (17) is a biquadratic formula, and its root can be calculated in a close form.

**Case 3**: If $x_{i,j} \in I$ and $\alpha \neq 1/2$. In this case, $x_{i,j}$ can not be calculated directly from any formula, so we rewrite $x_{i,j}$ as

$$\begin{aligned}x_{i,j}^{1-\alpha} - \rho a_j^2 x_{i,j}^2 + (\rho a_j z_{i,j}^k + a_j u_j^k) x_{i,j} + \lambda_i^q = 0\\ \Leftrightarrow x_{i,j} = (\rho a_j^2 x_{i,j}^2 - (\rho a_j z_{i,j}^k - a_j u_j^k) x_{i,j} - \lambda_i^q)^{1/3}.\end{aligned} \tag{18}$$

After converting the transcendental equation algebraically into the form $x = g(x)$, $x_{i,j}$ can be solved using the iterative scheme with the Fix Point Method of iteration.

Finally, testing all the solutions using the Complementary Slackness condition $\lambda_i \cdot (h_i(x_i) - q_i) = 0$ to derive $x_i$. Therefore, the theorem is proved. □

Guided by Theorem 1, we have calculated $x_i^{k+1}$, and now we consider the CQ polling strategy part $g_i(z_i)$, with the following theorem to guide the update of vector $z_i^{k+1}$.

**Theorem 3.** *$\forall h_i$ and $\forall z_{i,j} \in z_i$, the solution for z-minimization is*

$$z_i^{k+1} = \{z_{i,j} | z_{i,j} = \lambda_i^c((\rho + u_j^k) z_{i,j} - 2z_{i,j}^{-3} - \rho a_j x_{i,j}^{k+1})^{-1}\}. \tag{19}$$

**Proof.** For the $z$-minimization step, the Lagrangian and dual problem are

$$\begin{aligned}\mathcal{L}_{z_i,\lambda_i^c} = -g_i(x_i^{k+1}) + u^k(F(x_i^{k+1}) - z_i) +\\ (\rho/2)||F(x_i^{k+1}) - z_i||_2^2 + \lambda_i^c(h_i(z_i) - cq_i)\end{aligned} \tag{20}$$

and

$$\begin{aligned}\text{maximize} \quad & D_{z_i}(\lambda_i^c) = \inf_{z_i \in D}\{\mathcal{L}_{z_i,\lambda_i^c}\},\\ & \lambda_i^c \geq 0, \quad \forall i.\end{aligned} \tag{21}$$

The derivation for CQ polling rate is similar to the proof of Theorem 2 with a different utility function $g_i(\cdot)$. Therefore, for the stationarity of KKT conditions, the optimal solution for $z_i^*$ should satisfy

$$\begin{aligned}&\frac{\partial \mathcal{L}_{z_i,\lambda_i^c}}{\partial z_i}|_{z_i = z_i^*} = 0\\ \Leftrightarrow\ & -\nabla g_i(z_i^*) - u^k z_i^* + \rho(A \cdot x_i^{k+1} - z_i^*) + \lambda_i^c \nabla h_i^c(z_i^*) = 0\\ \Leftrightarrow\ & 2z_i^{*^{-3}} - u^k z_i^* + \rho(A \cdot x_i^{k+1} - z_i^*) + \lambda_i^c z_i^{*^{-1}} = 0.\end{aligned} \tag{22}$$

It is difficult to find the exact roots of the equation, we rewrite the equation (22) in the form

$$z_i^* = \lambda_i^c \cdot ((\rho \mathbf{1} + u^k) z_i^* - 2z_i^{*^{-3}} - \rho A \cdot x_i^{k+1})^{-1}. \tag{23}$$

Therefore, we use the Fixed Point Method to find an approximate solution to $z_i^*$ with element-wise computation on $z_{i,j}$. Thus, the theorem is proved. □

### *C. Distributed algorithm for DRUM*

We discuss in this section how to implement a distributed and modular algorithm to efficiently solve the DRUM problem. The proposed algorithm performs the iterations of the decomposition-coordination procedure. As described in Algorithm 1, at each iteration $k$, the DRUM-decomposition algorithm solves small local subproblems via the $x$-minimization and $z$-minimization steps. It maintains two important internal variables $x_i^k$ and $z_i^k$ and calculate $x_i^{k+1}$ and $z_i^{k+1}$ based on Theorems 2 and 3.

**Algorithm 1** DRUM-decomposition

```
 1: procedure DRUM-DECOMPOSITION($z_i^k, u^k$)
 2:     input: $z_i^k, u^k$
 3:     output:$x_i^{k+1}, z_i^{k+1}$
 4:     for $j$ in $\{1, 2, ..., m\}$ do
 5:         Calculate $x_{i,j}$ using Theorem 2
 6:         Test $x_{i,j}$ with the condition $\lambda_i^q \cdot (h_i(x_{i,j}) - qp_i) = 0$
 7:     end for
 8:     Update all $x_{i,j}$ into $x_i^{k+1}$
 9:     for $j$ in $\{1, 2, ..., m\}$ do
10:         Calculate $z_{i,j}$ using Theorem 3
11:         Test $z_{i,j}$ with Complementary Slackness condition
12:     end for
13:     return $x_i^{k+1}$, $z_i^{k+1}$
14: end procedure
```

For steps 4-7 and 9-12 in Algorithm 1, because the computations of $x_{i,j}$ and $z_{i,j}$ ($m$ variables of $x_i$) are independent with each other, this iteration can be implemented using multi-thread parallel processing, improving computation efficiency. Algorithm 1 works independently to calculate $x_i^{k+1}$, $z_i^{k+1}$ in each iteration $k$. Between iterations, it reports the updated values to a coordinator, which collects all the $x_i^{k+1}$, $z_i^{k+1}$, conducts a simple summation and then broadcasts a message with updated $u^{k+1}$. The DRUM-coordination algorithm is described as in Algorithm 2.

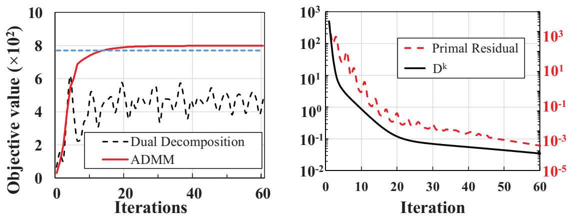
**Algorithm 2** DRUM-coordination

```
1: procedure DRUM-COORDINATION($x_i^{k+1}$, $z_i^{k+1}$, $u^k$)
2:     input: $x_i^{k+1}$, $z_i^{k+1}$, $u^k$
3:     output: $u^{k+1}$
4:     for $j$ in $[1 - m]$ do
5:         compute $u_j^{k+1}$ by $u_j^{k+1} = u_j^k + \rho \cdot (\sum_{i=1}^n a_j x_{i,j}^{k+1} + \sum_{i=1}^n z_{i,j}^{k+1})$
6:     end for
7:     Update and return $u^{k+1}$
8: end procedure
```

Algorithm 1 and 2 runs iteratively to find the global utility optimality of the network, thus the large scale DRUM problem can be solved efficiently in a distributed manner.

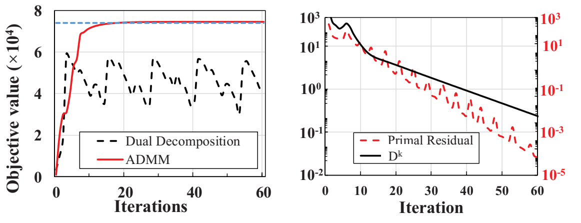
### *D. Simulations*

To illustrate the basic behavior of the proposed algorithms, we report the simulation results for some large scale instances of the DRUM problem. In the simulation, the parameters are set as $k = 20$, $c_i = 15$, $q_i = 25$, and $a_j$ in $A$ are randomly generated in the normal distribution in $[1, 10]$. We empirically set the penalty parameter $\rho = 10^{-3}$. The variables $u^0$ and $z^0$ are initialized to be a zero vector. The problem scale is set as $n \in \{10^2, 10^3, 10^4\}$, $m \in \{50, 5 \times 10^2, 5 \times 10^3\}$. Algorithms 1 and 2 are implemented in Matlab and embedded in Python to execute in parallel threads. The evaluation metric is the number of iterations, which is platform-independent.



(a) Comparison of objective value. (b) Comparison of the primal residual and $D^k$.

Fig. 4: Convergence analysis when $n = 100$, $m = 50$.



(a) Comparison of objective value. (b) Comparison of the primal residual and $D^k$.

Fig. 5: Convergence analysis when $n = 10^3$, $m = 500$.

For comparison, we also implement the conventional Dual Decomposition approach with gradient methods to solve the standard NUM described in problem (1). The step size $\rho_k$ is chosen following the commonly accepted diminishing step size rule [18], with $\rho_k = 10^{-3}/\sqrt{k}$.

**Convergence** We first evaluate the convergence of the DRUM algorithm under varying $n$ and $m$. From Figs. 4(a) and 5(a), we observe that our algorithm converges very fast in less than 20 iterations for all cases. Figs. 4(b) and 5(b) depict the trajectory of the primal residual defined as $\sum_i^n ||x_i - z_i||_2^2$ and $D^k$ used in the proof of Theorem 1. From the figures, we observe that the primal residual becomes less than 1 when $n = 100$, $m = 50$, and less than 10 when $n = 10^3$, $m = 500$ after 10 iterations. We also observe that $D^k$ is indeed non-increasing and Lipschitz continuous. We also find out that the convergence rate is independent of the problem size by the fact that the convergence behaviors of the proposed algorithm are the same in all cases when $n \in \{10^2, 10^3, 10^4\}$, $m \in \{50, 5 \times 10^2, 5 \times 10^3\}$. This means that our algorithm is scalable for solving large-scale problems. The blue dotted line in Fig. 4(a) and Fig. 5(a) is an 99.5% approximated optimal value for the objective. The algorithm stops at 17-th iteration when $n = 10^3$, $m = 500$. These results confirm that our algorithm generates a solution with high accuracy even faster.

Compared to the Dual Decomposition method, we observe from Figs. 4(a) and 5(a) that Dual Decomposition yields fluctuating results. The objective values are still not close to its convergence after 60 iterations. Furthermore, the observed objective values generated by Dual Decomposition are on average 69.3% less than that of DRUM. The reason is that DRUM considers multiple queues and their dependencies in RDMA, yielding better utilities for the RDMA network.

**Discussion** Though the above description, we have demonstrated that how our proposed ADMM-based method is able to solve a large-scale DRUM problem optimally and in a distributed manner. It is worth noticing our proposed method can also adapt to a number of variations of DRUM problem. First, the utility functions for CQ and QP are not restricted to Definition 1 and 2. The functions can be any convex utility functions and they are not required to be strongly convex. Although the effectiveness of current convex functions are evaluated and the optimization results significantly outperform the state-of-the-art methods, we are still exploring novel functions to better describe the utility of CQ.

Another important variation of the DRUM problem is the function to describe the dependency among CQ and QP operations. We have demonstrated that our ADMM-based method can solve the problem when $x_i$ and $z_i$ are dependent and derive close form solutions. In a more generalized case, $z_i$ is dependent on WQEs requesting rates for sending to and receiving data. In this case, $z_i$ is dependent on $x_i$ on all other hosts, and we can rewrite the last constraint in equation (4) as

$$\sum_{i=1}^{n} F(x_i) = z_{i'}, \quad \forall i' \in \mathbb{Z}_n, \tag{24}$$

where $F(\cdot)$ can be more complex than an affine function. For example, $F(\cdot) = T_i \cdot x_i$, where $T_i$ is a traffic matrix to denote the traffic distribution of all applications on $h_i$. With this variation, our ADMM-based method is still able to solve it. Moreover, the variation of DRUM has close form solutions in most cases. We will introduce the extension of our method to solve the variations of DRUM in the future work.

## IV. Implementation and Experiment

### A. Implementation

We implement the proposed DRUM algorithm in the DRUM agent, which is a software module inserted between applications and the RDMA driver. One DRUM agent decides the rates of fetching WQEs for different applications according to the DRUM algorithm. Meanwhile, it maintains one ingress completion queue of CQEs with one dedicated thread actively polling CQEs. Distributed DRUM agents are deployed on each host in the RDMA network. They first work independently to schedule the local applications and then coordinates iteratively to update the network information, which is similar with the most network protocols such as TCP/IP congestion control.

### B. Experiment settings

We evaluate DRUM's performance in real-world RDMA network environment through extensive testbed experiments.

**Testbed**. The testbed is built with 7 nodes, each of which is equipped with two 2.2GHz Intel Xeon E5-2650 v4 processors and 64 GB of memory. All these servers are installed with Ubuntu 16.04 and are equipped with a Mellanox ConnectX Series RNIC (40 Gbps over InfiniBand) to enable RDMA.

**Application**. We implement an RDMA-enabled Key-Value (KV) store service, and then run several clients concurrently requesting the server. In client nodes, each client creates multiple connections requesting data in the KV store through RDMA network. Each connection requests for data with random data size ranging between $[32B, 1MB]$.

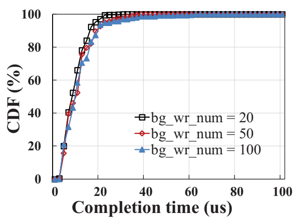


Fig. 6: WQE completion time of a 32B message.

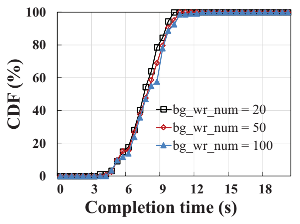
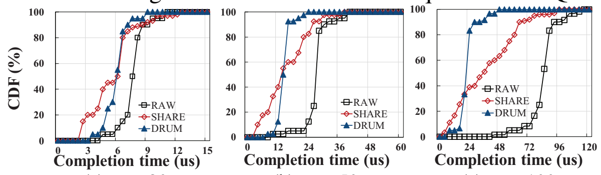


Fig. 7: 1GB flow completion time of a sequence of WQEs.



(a) $cc = 20$ (b) $cc = 50$ (c) $cc = 100$

Fig. 8: Comparison on 32B packet completion time.

**Benchmark**. For comparison, we have implemented the following mechanisms:

(1) *RAW*: Represents the raw RDMA-based primitives and the standard operation in the RDMA (same as in FaRM [20]).

(2) *SHARE*: Denotes the connection grouping mechanism proposed in [3, 4], where the connections designated for the same host share one QP for resource multiplexing. As the authors suggest, the multiplexing ratio of QP is set as 5.

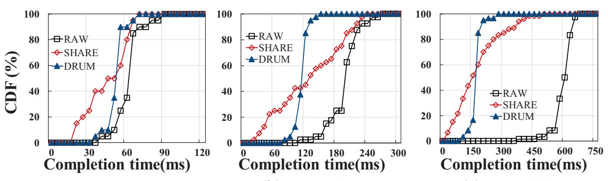
(3) *DRUM*: The implementation of our method.

### C. Experimental Results

**Test Case 1** (**Packet level performance**). To evaluate whether our model could provide fine-grained resource scheduling, we test the completion time of a WQE to continuously request for a 32B data from the KV store server, while the background connection varies its WQE batch size from 20 to 100. Fig. 6 demonstrates the distribution of the WQE completion time. We observe that the single WQE completion time is unaffected by the background traffic.

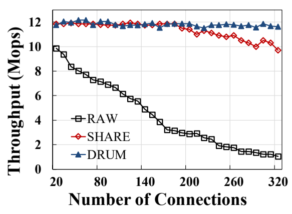
We then compare *DRUM* with the benchmarking mechanisms. In this case, we adjust the number of concurrent connections $cc$ in the host. $cc$ is set as $\{20, 50, 100\}$. The evaluation results are depicted in Fig. 8. When $cc = 20$, the $80\%$ completion time of the single WQE are 7.1 $\mu$s, 7.2 $\mu$s, 8.8 $\mu$s under *RAW*, *SHARE* and *DRUM*, respectively. When $cc$ increases, *RAW* does not work properly. For example, the $80\%$ completion time under *RAW* degrades from 26.5 $\mu$s to 90.4 $\mu$s, when $cc$ increases from 50 to 100. Although *SHARE* performs better than *RAW*, it suffers from the long tail distribution of the completion time. From our analysis, this is because of the lock contention on the shared QP, causing performance fluctuation. On the contrary, *DRUM* accesses QP by the single DRUM agent in a lock-free manner. It completes the WQE in 14.5 $\mu$s and 24.0 $\mu$s, when $cc = 50$ and $cc = 100$. It is worth noting that when $cc = 100$, *DRUM* completes the WQE 3.1$\times$ faster than *RAW* and 1.7$\times$ than *SHARE*.

**Test Case 2** (**Flow level performance**). In this test case, we evaluate the performance of *DRUM* when transferring a large data file. The foreground traffic transmits 1MB message



(a) $cc = 20$ (b) $cc = 50$ (c) $cc = 100$

Fig. 9: Completion time of 1MB flows (32B/packet).



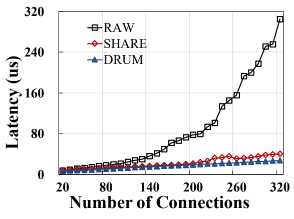


Fig. 10: Average throughput. Fig. 11: Average latency.

through one WQE and continuously to transfer 1GB data in total. The background connections request for the data at random sizes between $[1KB, 10MB]$ and the batch size parameter is set between $[20, 100]$. From Fig. 7, we see that the completion time of foreground traffic is unaffected by the background traffic. The reason is that DRUM jointly controls the resource scheduling on CQ and QP, providing an enhanced resource scheduling.

We then evaluate *DRUM*'s performance by transferring a 1MB flow that is composed by a sequence of 32B WQEs. The evaluation results are depicted in Fig. 9. When $cc = 20$, the $80\%$ completion time of the WQEs to send 1MB data are 53.1 ms, 82.8 ms, 62.9 ms under *RAW*, *SHARE* and *DRUM*, respectively. When the number of concurrent connections increase, where the contention on CQ polling is more severe, *DRUM* outperforms all other mechanisms. When $cc = 100$, *DRUM* completes the WQE $3.75\times$ faster than *RAW* and $1.4\times$ than *SHARE*.

**Test Case 3** (**Overall system performance**). We evaluate *DRUM* by collecting both the average throughput and latency of all connections in the network. In this case, we vary the concurrent number of clients requesting data from the KV store from 20 to 400. The clients request for data at random sizes between $[32B, 1MB]$. As shown in Fig. 10, *DRUM* significantly outperforms all other mechanisms in terms of the average throughput. *SHARE* performs better than *RAW*, but when connections increase, the lock contention on shared QP becomes the bottleneck, thus hurting the performance of applications. Note that, when the total number of connections is larger than 300, the average throughput under *DRUM* is $1.8\times$ greater than *SHARE*.

Fig. 11 shows the average latency of messages. We record the latency of each batch of WQEs. From the results, we observe that *DRUM* is capable of keeping most of the requests within extremely low latency less than 18 $\mu$s. When the connections increase, the network is fully saturated and the performance of *SHARE* degrades, the average latency is $67.3\%$ higher than *DRUM* when the total number of connections is larger than 300.

## V. Related work

RDMA suffers from the unfairness issue and performance degradation in the shared data center network as reported in [3–5, 21]. Addressing this issue, various resource management mechanisms for RDMA are proposed. Freeflow [22] proposed a virtualization based software solution to isolate the performance of RDMA-enabled applications. Youmin Chen et al. [3] designed the connection grouping mechanism, allowing multiple applications to share one QP. LITE [4] allowed applications to share resources by establishing a shared memory pool in the kernel. Haonan Qiu [5] el al. proposed a pooling mechanism between applications and RNIC driver to manage RDMA resources. FaRM [20] applied QP sharing in the kernel. Existing proposals lack sufficient theoretical analysis, and they are far from the optimality of resource scheduling.

Theoretically, the network resource scheduling optimization was modeled as a NUM problem [9, 11, 12]. Previous efforts [11–13] in resource management for TCP/IP networks have been successful in deriving NUM-based resource scheduling solutions. For example, Jian Guo et al. [23] proposed an instance of NUM to optimize the bandwidth allocation. Also, [8] provided a relaxation-based approach to solve the NUM with non-convex utility functions. However, none of prior methods considered RDMA. We have proposed a new variation of NUM model and designed a distributed and optimal solution based on ADMM [11, 19].

## VI. Conclusion

In this paper, we have introduced a distributed solution to optimal RDMA resource scheduling. Characterizing the complexities inherent in RDMA networks, we have abstracted the problem as a multi-block utility maximization problem with coupling variables. To efficiently solve it, we have presented a distributed and modular algorithm, and demonstrated the parallelism and convergence guarantee through theoretical analysis. Through large scale simulations and testbed experiments, we have shown that, compared with the state-of-the-art methods, our method has a number of unique advantages, such as achieving higher network utility and higher overall throughput when multiple applications share the RDMA network. We believe such advantages are highly desirable for the resource management in large-scale shared RDMA networks.

## Acknowledgements

This work is supported by National Key R&D Program of China 2017YFB1003000, National Natural Science Foundation of China under Grants No. 61632008, 61902065, 61872079, 61702096, 61702097, 61972085, 61906040, the Natural Science Foundation of Jiangsu Province under grant BK20190345, BK20190335, Jiangsu Provincial Key Laboratory of Network and Information Security under Grants No.BM2003201. We also thank the Big Data Computing Center of Southeast University for providing the experiment environment and computing facility. The work by John C.S. Lui was supported in part by the GRF 14201819.

## References

[1] A. Kalia, M. Kaminsky, and D. G. Andersen, "Design guidelines for high performance rdma systems," in *USENIX Annual Technical Conference (USENIX ATC' 16)*, 2016, pp. 437–450.

[2] D. Y. Yoon, M. Chowdhury, and B. Mozafari, "Distributed lock management with rdma: Decentralization without starvation," in *Proceedings of the 2018 International Conference on Management of Data*. ACM, 2018, pp. 1571–1586.

[3] Y. Chen, Y. Lu, and J. Shu, "Scalable rdma rpc on reliable connection with efficient resource sharing," in *Proceedings of the Fourteenth EuroSys Conference 2019 (EuroSys '19 )*. ACM, 2019, p. 19.

[4] S.-Y. Tsai and Y. Zhang, "Lite kernel rdma support for datacenter applications," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 306–324.

[5] H. Qiu, X. Wang, T. Jin, Z. Qian, B. Ye, B. Tang, W. Li, and S. Lu, "Toward effective and fair rdma resource sharing," in *Proceedings of the 2nd Asia-Pacific Workshop on Networking*. ACM, 2018, pp. 8–14.

[6] "Mellanox OFED for Linux User Manual," http://www.mellanox.com/related-docs/prod_software/Mellanox_OFED_Linux_User_Manual_v4_3.pdf.

[7] M. Zhang and J. Huang, "Mechanism design for network utility maximization with private constraint information," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 919–927.

[8] M. Ashour, J. Wang, C. Lagoa, N. Aybat, and H. Che, "Non-concave network utility maximization: A distributed optimization approach," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[9] S. Ramakrishnan and V. Ramaiyan, "Completely uncoupled algorithms for network utility maximization," *IEEE/ACM Transactions on Networking*, vol. 27, no. 2, pp. 607–620, 2019.

[10] L. Vigneri, G. Paschos, and P. Mertikopoulos, "Large-scale network utility maximization: Countering exponential growth with exponentiated gradients," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1630–1638.

[11] D. Palomar and M. Chiang, "A tutorial on decomposition methods and distributed network resource allocation," *IEEE J. Sel. Areas Commun*, vol. 24, no. 8, pp. 1439–1451, 2006.

[12] M. Chiang, S. H. Low, A. R. Calderbank, and J. C. Doyle, "Layering as optimization decomposition: A mathematical theory of network architectures," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 255–312, 2007.

[13] W.-C. Liao, M. Hong, H. Farmanbar, X. Li, Z.-Q. Luo, and H. Zhang, "Min flow rate maximization for software defined radio access networks," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 6, pp. 1282–1294, 2014.

[14] "TensorFlow-RDMA," https://github.com/tensorflow.

[15] "Spark-RDMA," https://github.com/Mellanox/SparkRDMA.

[16] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Transactions on networking*, no. 5, pp. 556–567, 2000.

[17] H. Xu, C. Feng, and B. Li, "Temperature aware workload managementin geo-distributed data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 6, pp. 1743–1753, 2014.

[18] C. Feng, H. Xu, and B. Li, "An alternating direction method approach to cloud traffic management," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 8, pp. 2145–2158, 2017.

[19] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[20] A. Dragojevi, D. Narayanan, M. Castro, and O. Hodson, "Farm: Fast remote memory," in *11th USENIXSymposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 401–414.

[21] A. Kalia, M. Kaminsky, and D. G. Andersen, "Fasst: Fast, scalable and simple distributed transactions with two-sided rdma datagram rpcs," in *12th USENIX Symposium on Operating Systems Design and Implementation OSDI 16)*, 2016, pp. 185–201.

[22] D. Kim, T. Yu, H. H. Liu, Y. Zhu, J. Padhye, S. Raindel, C. Guo, V. Sekar, and S. Seshan, "Freeflow: Software-based virtual rdma networking for containerized clouds." in *NSDI*, 2019, pp. 113–126.

[23] J. Guo, F. Liu, J. C. Lui, and H. Jin, "Fair network bandwidth allocation in iaas datacenters via a cooperative game approach," *IEEE/ACM Transactions on Networking*, vol. 24, no. 2, pp. 873–886, 2015.