Priority Algorithm for Near-data Scheduling: Throughput and Heavy-Traffic Optimality

Qiaomin Xie, Yi Lu
{qxie3, yilu4}@illinois.edu
University of Illinois at Urbana-Champaign

***Abstract*—The prevalence of data-parallel applications has made near-data scheduling an important problem. An example is the map task scheduling in the map-reduce framework. Wang et. al. [13] was the first to identify its capacity region and proposed a throughput-optimal algorithm based on MaxWeight. However, the study of the algorithm's delay performance revealed that it is only heavy-traffic optimal for a very special traffic scenario, where all traffic concentrates on a subset of servers. We propose a simple "local-tasks first" priority algorithm and show that it is throughput-optimal and heavy-traffic optimal for *all* traffic scenarios, i.e., it asymptotically minimizes the average delay as the arrival rate vector approaches the boundary of the capacity region. So far, it is the only known heavy-traffic optimal algorithm for this setting. As the algorithm is based on pre-determined priority, a direct application of the Lyapunov drift technique does not work. The main proof ideas are the construction of an *ideal load decomposition* and the separate treatment of two subsystems based on their ideal load. To the best of our knowledge, this is the only setup of affinity scheduling where a simple priority algorithm is shown to be heavy-traffic optimal. Simulation shows that our algorithm also significantly outperforms existing algorithms at loads away from the boundary of the capacity region.**

I. Introduction

The collection of large data sets by online social networks, search engines, scientific research and the health care industry has made traditional methods of data processing inadequate. The timely extraction of useful patterns and information from the large data set has motivated data-parallel processing. A popular example of data-parallel processing is the map-reduce framework, first used by Google [5], and widely adopted in various open-source and proprietary versions of Hadoop [3]. The *map* stage of the framework assigns the tasks to process the input data set, hence requires a near-data scheduler.

In order to facilitate parallel computation, a large data set is divided into small data chunks. Each data chunk is replicated on a few servers to increase availability. For each task, we call a server a *local server* for the task if the data chunk associated with the task is stored locally, and we call this task a *local task* for the server; Otherwise, the server is called a *remote server* for the task and this task is called a *remote task* for the server. While assigning tasks, it is a critical consideration to schedule a task on a local server [15], [1], [2], [10]. Scheduling in this setting is called the *near-data* scheduling problem, or scheduling with *data locality*.

A. The Problem

The near-data scheduling problem is a special case of affinity scheduling [4], [11], [12], [8], [9] where each type of tasks has different processing rates on different subsets of servers. It has two unique features:

1. It is impractical to have one queue for each type of tasks.

The type of a particular task is determined by the location of its data chunks. To achieve high availability and yet avoid excessive amount of storage, each data chunk is typically replicated on a small number of servers. For instance, the number is 3 for the map-reduce cluster.

When a server breaks down and its disks are replaced, all its local data chunks are restored by copying from their respective replicas. To avoid excessive amount of traffic from any single server, which can disrupt its service, it is desirable to distribute the replicas over a large number of servers. Hence, the current practice in a map-reduce cluster is for each data chunk to uniformly sample 3 servers. This makes the number of types *cubic* in the number of servers in a cluster, which itself can be as large as tens of thousands. As a result, it is impractical to have one queue for each type of tasks.

2. Local servers are faster (on average) than remote servers.

At a remote server, the data chunk for a particular task needs to be retrieved over the network before processing. It is measured in [15] that a remote server on average takes twice as much time as a local server. From the perspective of each task, the cluster is divided into two subsets, one with a faster rate than the other, although the subsets vary for different tasks. This regularity in service rates calls for a simple optimal algorithm. There can also be an intermediate rate in map-reduce clusters called rack-local. We do not include it in our model, although it is possible to extend our algorithm to this setting. We do not consider pre-fetching in this paper.

In addition, the algorithm should not assume any knowledge of arrival rates in order to be robust with load variations.

B. Previous Work

The existing work on affinity scheduling [4], [11], [12], [8], [9] requires a queue for each type of tasks, hence does not apply in this setting.

Wang et. al. [13] were the first to formulate this problem from a stochastic network perspective and identified its capacity region. They proposed a new queueing structure and a scheduling algorithm consisting of the Join the Shortest

Queue (JSQ) together with the MaxWeight policy. The JSQ step distributes the load into local and remote queues: a task is pre-assigned as remote if its local queues are longer than the remote queue. The MaxWeight step stabilizes the queues with a threshold-based priority policy.

The JSQ-MaxWeight algorithm was shown to be throughput-optimal. However, it was shown in [13] that it is heavy-traffic optimal only for a very special traffic scenario, where all traffic concentrates on a subset of servers. In particular, some servers receive *zero* local tasks and only provide remote service; and any server with non-zero local tasks is overloaded (with load exceeding 1) and *requires* remote service as a result.

There are a number of heuristics [3], [15], [10] proposed for near-data scheduling, but their fundamental throughput and delay properties are not known.

C. Our Approach

We use the same formulation as [13]. The cluster is modeled as a time-slotted system, in which tasks arrive at the beginning of each time slot according to some stochastic process. Each task processes one data chunk. Within each time slot, a task is completed with probability α at a local server, or with probability γ ($\gamma < \alpha$) at a remote server. We consider two traffic scenarios that require distinct proof techniques (although our algorithm does not distinguish between them as we assume no knowledge of arrival rates):

Evenly loaded. This is the case where with appropriate load balancing, each server can accommodate its load locally. No remote service is necessary in this scenario.

Locally overloaded (hotspots). More often, the data requested by the incoming traffic are skewed towards a subset of servers [2] and exceed their capacity. We call these servers *beneficiaries* as they require remote service to remain stable, and call the servers with spare capacity *helpers*. This includes the special scenario in [13] for which the JSQ-MaxWeight algorithm is shown to be heavy-traffic optimal, and is more general as it allows non-zero local traffic at helpers, as well as traffic that is local to both a helper and a beneficiary.

We use a simple queueing structure where each server has a queue storing its local tasks. We propose an algorithm where a newly arrived task is routed to a local server with the shortest queue; each server processes tasks from its local queue as long as it is non-empty, and when its local queue is empty, the server processes a remote task from the longest queue in the system. We establish the following results:

- We prove that our algorithm is throughput optimal, i.e., it can stabilize any arrival rate vector strictly within the capacity region identified in [13]. Since the algorithm has a predetermined priority of "local-tasks first", existing techniques using the L_2 norm Lyapunov drift, such as in [13], do not apply: There exist states with arbitrarily large L_2 norm where the drift remains positive. The main idea is the construction of the *ideal load decomposition* for each arrival vector, which separates the servers into helpers and beneficiaries. The stability of the helper subsystem (which by itself is not Markovian) is established first, and the spare capacity helps stabilize the beneficiary subsystem.
- In addition, we prove that our algorithm is heavy-traffic optimal for both the evenly loaded and locally overloaded scenarios, i.e., it asymptotically minimizes the average delay as the arrival rate vector approaches the boundary of the capacity region. Since [13] shows heavy-traffic optimality only for a special traffic scenario, our algorithm is so far the only known heavy-traffic optimal algorithm. Further, to the best of our knowledge, this is the only setting of affinity scheduling where a "local-tasks first" algorithm is shown to be heavy-traffic optimal, which can be of separate interest.
  The locally overloaded case is the more challenging of the two. The proof first establishes *state-space collapse*, where we show that the helper subsystem has uniformly bounded moments independent of the heavy-traffic parameter, and the beneficiary subsystem reduces to a single dimension where all queue lengths are equal. We remark that this result depends on our "local-tasks first" policy as the helper queues are drained first, *independent* of the beneficiaries. In contrast, JSQ-MaxWeight results in helper queues growing proportionally with the beneficiaries. The proof uses construction of *ideal* processes to bound the dependence between helpers and beneficiaries through shared local arrivals and remote services.

Finally, simulation shows that our algorithm significantly outperforms the JSQ-MaxWeight algorithm at loads away from the boundary of the capacity region. Delay improvement up to a factor of 4 is observed.

II. System Model

We consider a discrete-time model for a computing cluster with M parallel servers, indexed by $m \in \mathcal{M}$, where $\mathcal{M} = \{1, 2, \cdots, M\}$. Each data chunk is replicated on a set $\bar{L}$ of servers. As each task processes one data chunk, it has $|\bar{L}|$ local servers. Define the *type* of a task as the set $\bar{L}$ of its local servers. For instance, with $|\bar{L}| = 3$ the task type $\bar{L}$ is defined as:

$$\bar{L} \in \{(m_1, m_2, m_3) \in \mathcal{M}^3, m_1 < m_2 < m_3\},$$

where m_1, m_2, m_3 are the indices of the three local servers. We use $m \in \bar{L}$ to denote that server m is a local server for type $\bar{L}$ tasks. We denote by $\mathcal{L}$ the set of task types.

Arrivals. Let $A_{\bar{L}}(t)$ denote the number of type $\bar{L}$ tasks that arrive at the beginning of time slot t. We assume that the arrival process of type $\bar{L}$ tasks is i.i.d. with rate-$\lambda_{\bar{L}}$. We denote the arrival rate vector by $\lambda = (\lambda_{\bar{L}} : \bar{L} \in \mathcal{L})$. The number of total arrivals in one time slot is assumed to be bounded.

Services. For each task, we assume that its service time follows a geometric distribution with mean $1/\alpha$ if processed at a local server, and with mean $1/\gamma$ at a remote server. On average, a task is processed faster at a local server. So we assume $\alpha > \gamma$. At most one task is being processed at each server at any time and all services are non-preemptive.

A. Algorithm Description

Our proposed algorithm is illustrated in Fig. 1. The central scheduler maintains a set of M, where the m-th queue, denoted by Q_m, only receives tasks local to server m. We call it a local queue for tasks of type $\bar{L}$ if $m \in \bar{L}$. Note that there can be tasks local to server m but buffered at Q_n $(n \neq m)$, where server n is another local server for the tasks. Let the vector $Q(t) = (Q_1(t), Q_2(t), \cdots, Q_M(t))$ denote the queue lengths at time t. At the beginning of each time slot t, the central scheduler routes new arrivals to one of the queues and schedules a new task for an idle server as follows:

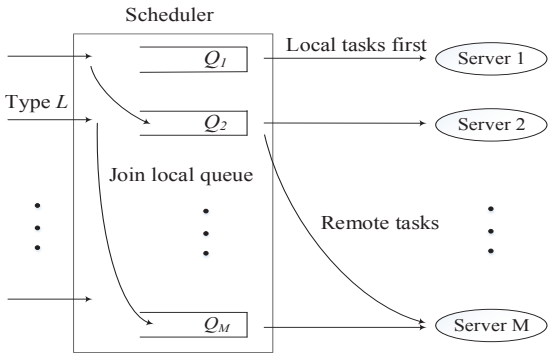

Fig. 1. The proposed algorithm

Load balancing: When a task of type $\bar{L}$ arrives, the scheduler compares the lengths of the task's local queues, $\{Q_m | m \in \bar{L}\}$, and inserts the task into the shortest queue. Ties are broken randomly. Let $A_{\bar{L},m}(t)$ denote the number of type $\bar{L}$ tasks that are routed to Q_m. The total number of tasks that join queue Q_m, denoted by $A_m(t)$, is given by

$$A_m(t) = \sum_{\bar{L}: m \in \bar{L}} A_{\bar{L},m}(t).$$

Prioritized scheduling: Let $f_m(t)$ denote the working status of server m at time slot t,

$$f_m(t) = \begin{cases} -1 & \text{if server } m \text{ is idle} \\ n & \text{if server } m \text{ serves a task from queue } n \end{cases}$$

When server m completes a task at the end of time slot $t-1$, i.e., $f_m(t^-) = -1$, it is available for a new task at time slot t. $f_m(t) = m$ indicates that server m is working on a local task, and $f_m(t) = n$, where $n \neq m$, implies that server m is working on a remote task. The scheduling decision is based on the working status vector $f(t) = (f_1(t), f_2(t), \cdots, f_M(t))$ and the queue length vector $Q(t)$.

- **Local tasks first.** When server m becomes idle, the scheduler sends the head-of-line task from Q_m.
- **Remote tasks.** When server m becomes idle and Q_m is empty, the scheduler sends a remote task to server m from the longest queue in the system, if the length of the longest queue, denoted by Q^{max}, exceeds the threshold $T_s = \alpha/\gamma$. The threshold is to ensure that the remote task will experience a smaller completion time in expectation, since the mean processing time at a remote server is $1/\gamma$, and the mean waiting time plus processing time at a local server is Q^{max}/α.

Let $\eta_m(t)$ denote the scheduling decision for server m at time slot t, which is the index of the queue server m is scheduled to serve. Note that $\eta_m(t) = f_m(t)$ for all busy servers, and when $f_m(t^-) = -1$, i.e., server m is idle, $\eta_m(t)$ is determined by the scheduler according to the algorithm.

B. Queue Dynamics

Let $S_m^l(t)$ and $R_m(t)$ denote the local and remote service provided by server m respectively, where $S_m^l(t) \sim \text{Bern}(\alpha I_{\{\eta_m(t)=m\}})$ and $R_m(t) \sim \text{Bern}(\gamma I_{\{\eta_m(t)\neq m\}})$ are two Bernoulli random variables with varying probability: $S_m^l(t) \sim \text{Bern}(\alpha)$ when server m is scheduled to the local queue, and Bern(0) otherwise; $R_m(t) \sim \text{Bern}(\gamma)$ when server m is scheduled to a remote queue, and Bern(0) otherwise.

Note that the local service *received* by queue Q_m is also $S_m^l(t)$, whereas the remote service *received* by queue Q_m is $S_m^r(t) \equiv \sum_{n:n\neq m} R_n(t) I_{\{\eta_n(t)=m\}}$, which is the sum of all remote service provided by other servers to queue Q_m. Let $S_m(t) \equiv S_m^l(t) + S_m^r(t)$ denote the departure process for queue m. Hence the queue lengths satisfy the following equation:

$$Q_m(t+1) \quad = \quad Q_m(t) + A_m(t) - S_m(t) + U_m(t),$$

where $U_m(t) = \max\{0, S_m(t) - A_m(t) - Q_m(t)\}$ is the unused service. As the service times follow geometric distributions, $Q(t)$ together with the working status vector $f(t)$ form an irreducible and aperiodic Markov chain $\{Z(t) = (Q(t), f(t)), t \geq 0\}$.

III. Ideal Load Decomposition

A key component of the proof of both throughput and heavy-traffic optimality is a construction we call the ideal load decomposition. It is ideal in the sense that it *minimizes* the work in the system by locally serving as many tasks as possible. The construction serves two purposes: 1) The ideal load obtained for each server is used as an intermediary in the proofs of stability and state-space collapse; 2) The construction uniquely identifies two subsystems, helpers and beneficiaries, which have very different behaviors and require distinct treatment in the proofs.

Helpers and Beneficiaries

A server is a helper if it is *not overloaded*, *provides* remote service and its local queue does *not receive* remote service under the ideal load decomposition. In contrast, a server is a beneficiary if it is *overloaded*, does *not provide* remote service, and its local queue *receives* remote service from the helpers. We will define an overloaded server in a more precise manner in III-B. While pure helpers and beneficiaries do not exist in a real system, the ideal load decomposition approximately depicts the load distribution in the heavy-traffic regime.

In the rest of the section, we construct the ideal load decomposition. We start from a new definition of the capacity region, which is equivalent to that identified in [13], but uses

a more refined decomposition appropriate for our algorithm. The ideal load decomposition is constructed from this refined decomposition in two steps: 1) Identify the overloaded servers; 2) Construct the decomposition that produces helpers and beneficiaries.

A. An Equivalent Capacity Region

Let Λ be the set of arrival rates such that each element has a decomposition satisfying the following condition:

$$\begin{aligned}\Lambda \;=\; & \big\{\lambda = \big(\lambda_{\bar{L}} : \bar{L} \in \mathcal{L}\big) \,| \\ & \exists\ \lambda_{\bar{L},n,m} \geq 0,\ \forall \bar{L} \in \mathcal{L}, \forall n \in \bar{L}, \forall m \in \mathcal{M},\ s.t. \\ & \lambda_{\bar{L}} = \sum_{n:n\in\bar{L}} \sum_{m=1}^{M} \lambda_{\bar{L},n,m}, \forall \bar{L} \in \mathcal{L}, \\ & \sum_{\bar{L}:m\in\bar{L}} \sum_{n:n\in\bar{L}} \frac{\lambda_{\bar{L},n,m}}{\alpha} + \sum_{\bar{L}:m\notin\bar{L}} \sum_{n:n\in\bar{L}} \frac{\lambda_{\bar{L},n,m}}{\gamma} < 1 \quad (1) \\ & \forall m \in \mathcal{M}\},\end{aligned}$$

where inequality (1) states that the sum of the local and remote load at each server is less than 1.

The rate $\lambda_{\bar{L}}$ is decomposed in [13] into $\lambda_{\bar{L},m}$, which is the rate of type-$\bar{L}$ arrival allocated to server m. We further refine the decomposition by simply writing $\lambda_{\bar{L},m} \equiv \sum_n \lambda_{\bar{L},n,m}$, where n is the index of the queue at which a task is *queued* till processed at server m. Observe that $\lambda_{\bar{L},n,m} = 0$ if $n \notin \bar{L}$, since tasks only join their local queues with the proposed algorithm. We have the following lemma.

Lemma 1: The capacity region Λ is equivalent to the capacity region in [13].

B. Overloaded Servers

Let $\nu_{n,m}$ denote the total rate of arrivals routed to Q_n, and eventually processed by server m, i.e., $\nu_{n,m} \equiv \sum_{\bar{L}:n\in\bar{L}} \lambda_{\bar{L},n,m}$.

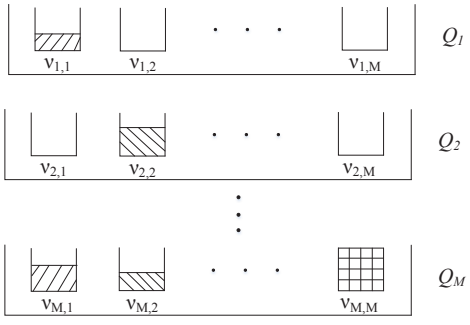

Fig. 2. Ideal load decomposition.

Figure 2 illustrates $(\nu_{n,m})$ where the m-th sub-queue at Q_n denotes the arrivals routed to Q_n but processed by server m. Note that the sub-queues are only a part of the construction, and do not exist in the actual data structure.

Let $\mathcal{S} \subseteq \mathcal{M}$ be a subset of servers. We denote by $\mathcal{L}_{\mathcal{S}}$ the set of task types *only local* to servers in $\mathcal{S}$.

Lemma 2: For any arrival rate vector $\lambda \in \Lambda$, there exists a decomposition $\{\tilde{\lambda}_{\bar{L},n,m}\}$ which satisfies Eqn. (1) and $\forall n \in \mathcal{D} = \{n \in \mathcal{M} : \sum_{\bar{L}:n\in\bar{L}} \sum_m \tilde{\lambda}_{\bar{L},n,m} \geq \alpha\}$, where $\mathcal{D}$ denotes the overloaded set with arrival rate greater than α,

$$\tilde{\lambda}_{\bar{L},n,m} = 0, \quad \forall \bar{L} \notin \mathcal{L}_{\mathcal{D}}, m \in \mathcal{M}. \quad (2)$$

Note that the decomposition $\{\tilde{\lambda}_{\bar{L},n,m}\}$ is such that for the overloaded set $\mathcal{D}$, it only receives non-zero arrivals from task types that are only local to $\mathcal{D}$. In other words, any task type that is also local to some server m not in the overloaded set, will be directed to m. This ensures that the set $\mathcal{D}$ is truly overloaded as no load balancing with the rest of the system will reduce its load. The decomposition also minimizes the total load in the system as the amount of local load is maximized. Note that $\mathcal{D}$ is unique for a given arrival vector λ, although the decomposition $\{\tilde{\lambda}_{\bar{L},n,m}\}$ is not unique. When $\mathcal{D}$ is non-empty, we call the system *locally overloaded*.

The proof takes a decomposition $\left\{\lambda_{\bar{L},n,m}\right\}$ satisfying Eqn. (1), and iteratively moves an appropriate amount of load from overloaded queues ($\sum_{\bar{L}:n\in\bar{L}} \sum_m \lambda_{\bar{L},n,m} \geq \alpha$) to underloaded queues ($\sum_{\bar{L}:n\in\bar{L}} \sum_m \lambda_{\bar{L},n,m} < \alpha$). This is possible whenever an overloaded queue receives local arrivals that are also local to some underloaded queue. At the end of each step, either there is no more shared local load between the two queues, or they have both become underloaded or overloaded. It can be shown that at each step, the decomposition continues to satisfy Eqn. (1) and reduces the total load in the system. The full proof can be found in [14].

C. Ideal Load Decomposition

Lemma 3: For any arrival rate vector $\lambda \in \Lambda$, there exists a decomposition $\left\{\lambda^*_{\bar{L},n,m}\right\}$ satisfying Eqn. (1) and for $\forall m \in \mathcal{M}$, either $m \in \mathcal{H}$ or $m \in \mathcal{B}$, where

$$\begin{aligned}\mathcal{H} \;=\; & \{n \in \mathcal{M} | \sum_{\bar{L}:n\in\bar{L}} \sum_m \lambda^*_{\bar{L},n,m} < \alpha, \\ & \text{and } \forall \bar{L} \in \mathcal{L}, \forall m \neq n,\ \lambda^*_{\bar{L},n,m} = 0\}, \\ \mathcal{B} \;=\; & \{n \in \mathcal{M} | \sum_{\bar{L}:n\in\bar{L}} \sum_m \lambda^*_{\bar{L},n,m} \geq \alpha, \\ & \text{and } \forall \bar{L} \notin \mathcal{L}_{\mathcal{B}}, \forall m \in \mathcal{M},\ \lambda^*_{\bar{L},n,m} = 0, \\ & \text{and } \forall \bar{L} \in \mathcal{L}, \forall m \neq n,\ \lambda^*_{\bar{L},m,n} = 0\}.\end{aligned}$$

Lemma 3 states that for any arrival vector, there exists an ideal load decomposition, under which a server is either a helper or a beneficiary. A helper $n \in \mathcal{H}$ receives no remote service, hence $\nu_{n,m} = 0$ for all $m \neq n$. The Q_1 and Q_2 in Fig. 2 belong to such servers. Only the local sub-queue has non-zero rate, denoted by $\nu_{n,n}$. A beneficiary server $m \in \mathcal{B}$, provides no remote service, but receives remote service from helpers. The Q_M in Fig. 2 illustrates such a situation. Note that Q_M receives remote service from server 1 and 2.

The proof constructs the ideal load decomposition iteratively from $\{\tilde{\lambda}_{\bar{L},n,m}\}$ given in Lemma 2. The main idea is that if an underloaded server receives remote service, it can process this work locally while reducing the remote service it provides, until it becomes a helper; if an overloaded server provides remote service, it can instead use this service towards its local load while reducing the remote service it receives, until it becomes a beneficiary. The full proof can be found in [14].

IV. Throughput Optimality

We devote this section to the proof of the following theorem:

Theorem 1: **(Throughput Optimality)** The proposed algorithm is throughput optimal. That is, it stabilizes any arrival rate vector strictly within the capacity region.

By Lemma 1, it is equivalent to prove that the proposed algorithm stabilizes any arrival rate vector within Λ, defined in III-A. The standard approach using a quadratic Lyapunov function does not apply in our setting, as a remote queue, despite its large queue length, can continue to grow, while a shorter queue receives local service, hence increasing the quadratic drift. Although the remote queue will be served after the local queue is empty, the time taken to obtain a negative drift will depend on the system state.

To address the challenge, we treat the helper and beneficiary subsystems, as defined in Lemma 3, separately. The proof has three main steps. First, we show that the *helper* subsystem is stable using an extension of Lemma 1 in [6]. If the beneficiary subsystem is empty, this alone proves Theorem 1. In the case where the *beneficiary* subsystem is non-empty, we show that the beneficiary queues are either all stable or none of them is stable. This allows us to show the stability of the *beneficiary* subsystem by contradiction.

Let M_h and M_b denote the number of helpers and beneficiaries, respectively. For simplicity, assume that $\mathcal{H} = \{1, 2, \cdots, M_h\}$, and $\mathcal{B} = \{M_h + 1, \cdots, M\}$. Let $Q^H(t)$ and $Q^B(t)$ denote the vector of helper queues and beneficiary queues, respectively, i.e.,

$$Q^H(t) = (Q_1(t), \cdots, Q_{M_h}(t)),$$
$$Q^B(t) = (Q_{M_h+1}(t), \cdots, Q_M(t)).$$

A. Stability of Helper Subsystem

Since the arrivals and services for helpers depend on the state of beneficiaries, $\{Z^H(t) = (Q^H(t), f^H(t)), t \geq 0\}$ itself is not a Markov chain. We use an extension of Lemma 1 in [6], which can be derived from [7]. It states that the subsystem is stable if there exists a positive integer T, and a Lyapunov function V defined on the subsystem only whose T time slot drift satisfies the following two conditions: (i) Finite drift with probability 1; (ii) Negative drift for sufficiently large V. We consider the Lyapunov function

$$V_h(Z(t)) = ||Q^H(t)||.$$

Lemma 4: For any arrival rate vector $\lambda \in \Lambda$, the helper queues defined by its ideal load decomposition will be stabilized with the proposed algorithm.

We defer the full proof to our technical report [14] due to space constraint. The key step is to use the ideal load decomposition from Lemma 3 as an intermediary to obtain the bound on the Lyapunov drift.

Consider the helper subsystem in steady state. Observe that the total arrival rate for this subsystem is at most $\sum_{\bar{L} \in \mathcal{L}^*_{\mathcal{H}}} \lambda_{\bar{L}}$, where $\mathcal{L}^*_{\mathcal{H}}$ is the set of task types that have at least one local server in $\mathcal{H}$. Since arrivals at helper subsystem can be processed remotely, the total amount of local service provided by helper servers is no greater than $\sum_{\bar{L} \in \mathcal{L}^*_{\mathcal{H}}} \lambda_{\bar{L}}$. Hence the total amount of remote service provided by all helpers in steady state, denoted by $R_{\mathcal{H}}$, can be lower bounded as

$$R_{\mathcal{H}} \geq \gamma \left(M_h - \frac{1}{\alpha} \sum_{\bar{L} \in \mathcal{L}^*_{\mathcal{H}}} \lambda_{\bar{L}} \right). \tag{3}$$

B. Stability of Beneficiary Subsystem

Assume that the beneficiaries subsystem is non-empty. We will prove the following important property of the set $\mathcal{B}$.

Lemma 5: For any arrival rate vector $\lambda \in \Lambda$, either all queues in $\mathcal{B}$ are stable or none of them is stable.

Proof. We prove this lemma by contradiction. Let $\mathcal{F}$ and $\mathcal{F}^c$ denote the set of stable and unstable beneficiaries, respectively. Assume that $\mathcal{F} \neq \emptyset$ and $\mathcal{F}^c \neq \emptyset$. By Lemma 4, helper queues $\mathcal{H}$ are stable. Consider the system with queues of $\mathcal{F} \cup \mathcal{H}$ in steady state. Since queues in $\mathcal{F}^c$ grow with time, the probability that the maximum queue is among $\mathcal{F} \cup \mathcal{H}$ is arbitrarily small. Hence the amount of remote service offered by helpers and devoted to queues $\mathcal{F} \cup \mathcal{H}$ can be arbitrarily small, denoted by $\delta > 0$.

Consider two arrival scenarios for $\mathcal{F}$:

(1) $\mathbb{E}\left[\sum_{m \in \mathcal{F}} A_m(t)\right] > |\mathcal{F}|\alpha$. Then $\exists k \in \mathcal{F}$ such that $\mathbb{E}\left[A_k(t)\right] \geq \mathbb{E}\left[\sum_{m \in \mathcal{F}} A_m(t)\right] / |\mathcal{F}| > \alpha$. Thus there exits a constant $\theta > 0$ such that $\mathbb{E}\left[A_k(t)\right] \geq \alpha + \theta$. Note that $\forall m \in \mathcal{F}$, the amount of service it receives satisfies $\mathbb{E}\left[S_m(t)\right] \leq \alpha + \delta$. Choosing sufficiently small $\delta < \theta$, we can have $\mathbb{E}\left[S_k(t)\right] < \mathbb{E}\left[A_k(t)\right]$, which contradicts with the assumption that beneficiary k is stable.

(2) $\mathbb{E}\left[\sum_{m \in \mathcal{F}} A_m(t)\right] = |\mathcal{F}|\alpha$. The total arrival rate for $\mathcal{F}^c$ is given by $\mathbb{E}[\sum_{m \in \mathcal{F}^c} A_m(t)] = \sum_{\bar{L} \in \mathcal{L}_{\mathcal{B}}} \lambda_{\bar{L}} - |\mathcal{F}|\alpha + \sigma$, where $\sigma \geq 0$ is the amount of arrivals local to some server in $\mathcal{H}$ but join $\mathcal{F}^c$. It can be made arbitrarily small as queues $\mathcal{F}^c$ become sufficiently large.

Consider service for $\mathcal{F}^c$. For any $m \in \mathcal{F}^c$, its instability implies $\mathbb{P}(Q_m(t) = 0) = 0$, hence $\mathbb{E}\left[S^l_m(t)\right] = \alpha$. We have $\mathbb{E}\left[\sum_{m \in \mathcal{F}^c} S_m(t)\right] \geq \alpha|\mathcal{F}^c| + R_{\mathcal{H}} - \delta$. Define $\epsilon = \alpha M_b + \gamma(M_h - \frac{1}{\alpha}\sum_{\bar{L} \in \mathcal{L}^*_{\mathcal{H}}} \lambda_{\bar{L}}) - \sum_{\bar{L} \in \mathcal{L}_{\mathcal{B}}} \lambda_{\bar{L}}$. Since the ideal load decomposition satisfies Eqn. (1), ϵ is a positive constant. Select small δ and σ such that $\delta < \frac{\epsilon}{4}$, and $\sigma < \frac{\epsilon}{4}$. Then $\exists T > 0$ such that for $\forall t > T$,

$$\mathbb{E}\left[\sum_{m \in \mathcal{F}^c} S_m(t)\right] > \mathbb{E}\left[\sum_{m \in \mathcal{F}^c} A_m(t)\right] + \frac{\epsilon}{2}.$$

This contradicts with the assumption that all queues in $\mathcal{F}^c$ are unstable. This completes the proof. ∎

Next we show the stability of the beneficiary subsystem.

Lemma 6: For any arrival rate vector $\lambda \in \Lambda$, all queues in $\mathcal{B}$ will be stabilized under the proposed algorithm.

Proof. Again, we prove the statement by contradiction. By Lemma 5, we can assume that all beneficiaries are unstable. With a similar argument as the second case in the proof of Lemma 5, we can show that this assumption does not hold. Therefore, all beneficiaries are stable. ∎

V. Heavy Traffic Optimality

In this section, we show that the proposed algorithm achieves queue length optimality in the heavy-traffic limit. We consider the two cases separately: locally overloaded and evenly loaded. The proof follows the Lyapunov drift-based approach recently developed in [6]. However, as the "local-tasks first" policy excludes the use of a quadratic Lyapunov function, the main challenge is to prove the state-space collapse and derive a matching upper bound for the locally overloaded case. The main idea is to first show uniform boundedness for the helper queues, and analyze the Lyapunov drift for the beneficiary subsystem with the helper subsystem in steady-state, and bound the amount of remote service received by helpers and the amount of helper traffic routed to beneficiaries. The proof for the evenly loaded case is considerably simpler, and we will only briefly state the results.

A. Locally Overloaded Traffic

With locally overloaded traffic, the beneficiary set is non-empty. Using the same notation as in IV, we assume that there are M_h helpers and M_b beneficiaries. Under the ideal load decomposition, all task types in $\mathcal{L}^*_{\mathcal{H}}$, i.e., types that have at least one local server in helpers, are all routed to helper queues.

We consider the heavy-traffic regime where the total local load on helpers is

$$\sum_{\bar{L}\in\mathcal{L}^*_{\mathcal{H}}} \lambda_{\bar{L}} \equiv \Phi\alpha. \tag{4}$$

For any $\lambda \in \Lambda$, it satisfies $\sum_{\bar{L}\in\mathcal{L}} \lambda_{\bar{L}} < M_b\alpha + \Phi\alpha + \gamma(M_h - \Phi)$. We assume that

$$\sum_{\bar{L}\in\mathcal{L}} \lambda_{\bar{L}} = M_b\alpha + \Phi\alpha + \gamma(M_h - \Phi) - \epsilon, \tag{5}$$

where $\epsilon > 0$ characterizes the distance of the arrival rate vector from the capacity boundary. We will make a further assumption that the $\sum_{\bar{L}\in\mathcal{L}^*_{\mathcal{H}}} \lambda_{\bar{L}}$ is independent of ϵ. That is, the total local load for helpers is fixed. This assumption can be removed with more care.

Consider the arrival processes $\{A^{(\epsilon)}_{\bar{L}}(t)\}_{\bar{L}\in\mathcal{L}}$ with arrival rate vector $\lambda^{(\epsilon)}$ satisfying $\mathcal{B} \neq \emptyset$ and Eqs. (4)-(5). Note that the variance of $\{A^{(\epsilon)}_{\bar{L}}(t)\}_{\bar{L}\in\mathcal{L}^*_{\mathcal{H}}}$ is independent of ϵ. We denote by $(\sigma^{(\epsilon)}_b)^2$ the variance of the number of arrivals that are only local to beneficiaries, i.e., $Var(\sum_{\bar{L}\in\mathcal{L}_{\mathcal{B}}} A^{(\epsilon)}_{\bar{L}}(t)) = (\sigma^{(\epsilon)}_b)^2$, which converges to σ_b^2 as $\epsilon \to 0$.

The corresponding Markov chain $\{Z^{(\epsilon)}(t) = (Q^{(\epsilon)}(t), f^{(\epsilon)}(t)), t \geq 0\}$ has been shown to be positive recurrent. All theorems in this section will concern the *steady-state* queue lengths with $0 < \epsilon < \Phi\alpha + \gamma(M_h - \Phi)$. Due to space constraint, we defer all proofs to [14] and only highlight the important steps.

Theorem 2: (**Helper queues**)

$$\lim_{\epsilon\to 0^+} \epsilon\mathbb{E}\left[\sum_{m\in\mathcal{H}} Q^{(\epsilon)}_m(t)\right] = 0,$$

Theorem 2 states that the expected helper-queue length is bounded and independent of ϵ. The theorem follows from the same Lyapunov function for Lemma 4, which, with the positive recurrence of the entire system, implies that all moments of $Q^H(t)$ are bounded according to Lemma 1 in [6]. Therefore, we only need to consider the beneficiary queue lengths in the rest of this section.

1) Lower Bound: Consider a hypothetical single server system with the arrival process $\{a^{(\epsilon)}(t), t \geq 0\}$ and the service process $\{\beta^{(\epsilon)}(t), t \geq 0\}$, where

$$a^{(\epsilon)}(t) = \sum_{\bar{L}\in\mathcal{L}_{\mathcal{B}}} A^{(\epsilon)}_{\bar{L}}(t), \quad \beta^{(\epsilon)}(t) = \sum_{i\in\mathcal{B}} X_i(t) + \sum_{j\in\mathcal{H}} Y_j(t).$$

Here $\{X_i(t)\}_{i\in\mathcal{B}}$ and $\{Y_j(t)\}_{j\in\mathcal{H}}$ are independent and each process is i.i.d. For any $i \in \mathcal{B}$, $X_i(t) \sim$ Bern(α). And $\forall j \in \mathcal{H}$, $Y_j(t) \sim$ Bern($\gamma(1-\rho_j)$), where ρ_j is the proportion of time helper j spends on local tasks in steady state. Hence $\mathbb{E}\left[\sum_{j\in\mathcal{H}} Y_j(t)\right]$ represents the total amount of remote service provided by helpers. We denote $Var(\beta^{(\epsilon)}(t))$ by $(\nu^{(\epsilon)}_b)^2$, which converges to a constant ν_b^2 as $\epsilon \to 0$. Let $\{\Psi(t)\}$ denote the corresponding queue-length process. Then in steady state, $\{\Psi(t)\}$ is stochastically smaller than the total beneficiary queue-length process $\{\sum_{m\in\mathcal{B}} Q^{(\epsilon)}_m(t)\}$ of the original system. Using Lemma 4 in [6], we can obtain a lower bound on $\mathbb{E}[\Psi(t)]$, which gives the following theorem.

Theorem 3: (**Lower Bound**)

$$\mathbb{E}\left[\sum_{m\in\mathcal{B}} Q^{(\epsilon)}_m(t)\right] \geq \frac{(\sigma^{(\epsilon)}_b)^2 + (\nu^{(\epsilon)}_b)^2 + \epsilon^2}{2\epsilon} - \frac{M}{2}.$$

Therefore, in the heavy traffic limit as $\epsilon \downarrow 0$,

$$\liminf_{\epsilon\to 0^+} \epsilon\mathbb{E}\left[\sum_{m\in\mathcal{B}} Q^{(\epsilon)}_m(t)\right] \geq \frac{\sigma_b^2 + \nu_b^2}{2}. \tag{6}$$

2) State Space Collapse: Throughout this section, we use notations with superscript ^{B} to represent the corresponding vectors for beneficiaries. For ease of exposition, the superscript $^{(\epsilon)}$ is omitted temporarily. Denote the queueing and working status process for beneficiaries as $\{Z^B(t) = (Q^B(t), f^B(t)), t \geq 0\}$. Define

$$c_b = \frac{1}{\sqrt{M_b}}(\underbrace{1, 1, \cdots, 1}_{M_b}). \tag{7}$$

Then the parallel and perpendicular components of the queue length vector Q^B with respect to the direction c_b are given by:

$$Q^B_{||} = \langle c_b, Q^B\rangle c_b, \quad Q^B_{\perp} = Q^B - Q^B_{||}.$$

We will establish state-space collapse of Q^B along the direction c_b, by showing that $Q^B_{\perp}$ is bounded and independent of the heavy-traffic parameter ϵ.

Remark. With the bounded moments of the helper queue lengths, the queue length vector Q collapses to the following direction c:

$$c = \frac{1}{\sqrt{M_b}}(\underbrace{0, 0, \cdots, 0}_{M_h}, \underbrace{1, 1, \cdots, 1}_{M_b}). \tag{8}$$

We consider the Lyapunov function

$$V(Z^B) = ||Q^B_\perp||.$$

From the extended version of Lemma 1 in [6], it is sufficient to show that the T-period drift of $V(Z^B)$ is always finite and is negative for sufficiently large V. It is easy to verify that the drift is finite. Here we provide an outline for the latter condition and defer the full analysis to [14].

By the boundedness of arrivals and service, we can bound the drift of $V(Z^B)$ as

$$\mathbb{E}\left[\Delta V(Z^B)|Z^B(t_0)\right] \leq \frac{\mathbb{E}\left[\sum_{t=t_0}^{t_0+T-1} G(t)|Z^B(t_0)\right] + C}{||Q^B_\perp(t_0)||}$$

where C is a constant and $G(t) = \langle Q^B(t), A^B(t) - S^B(t)\rangle - \langle c_b, Q^B(t)\rangle\langle c_b, A^B(t) - S^B(t)\rangle$.

To bound $\mathbb{E}\left[\sum_{t=t_0}^{t_0+T-1} G(t)|Z^B(t_0)\right]$, we decompose the probability space by conditioning on t^*, which is defined as the first time slot where all servers have made scheduling decision at least once after t_0. Let $T = JK$. Define $X_1 = \{t^* > t_0 + K|Z^B(t_0)\}$ and $X_2 = \{t^* \leq t_0 + K|Z^B(t_0)\}$. We denote by Y_i the expectation term conditioned on X_i, $i = 1, 2$, i.e., $Y_i = \mathbb{E}\left[\sum_t G(t)|Z^B(t_0), X_i\right]$. Thus the expectation term is broken down into two terms: $Y_1\mathbb{P}[X_1]$ and $Y_2\mathbb{P}[X_2]$. It is easy to obtain bounds on Y_1 and $\mathbb{E}\left[\sum_{t=t_0}^{t^*} G(t)|Z^B(t_0), X_2\right]$ by the boundedness of arrivals and service. The key step is to bound $\mathbb{E}\left[\sum_{t=t^*+1}^{t_0+T-1} G(t)|Z^B(t_0), X_2\right]$.

For each $m \in \mathcal{B}$, define $\hat{A}_m(t) = \sum_{\bar{L}\in\mathcal{L}_\mathcal{B}} A_{\bar{L},m}$, i.e., $\hat{A}$ excludes arrivals that are also local to helpers. Define $\lambda^{*l}_m = \sum_{\bar{L}:m\in\mathcal{L}} \lambda^*_{\bar{L},m,m}$ and $\lambda^{*r}_m = \sum_{\bar{L}:m\in\mathcal{L}}\sum_{n:n\neq m} \lambda^*_{\bar{L},m,n}$ from the ideal load decomposition $\{\lambda^*_{\bar{L},m,n}\}$. Let $Q^{max}(t)$ denote the maximum queue length at time t. We then break $G(t)$ into four terms and obtain bound for each term. We temporarily omit the superscript ^{B} in the following argument.

$$\begin{aligned}
G(t) &= \langle Q(t), A(t) - S(t)\rangle - \langle c_b, Q(t)\rangle\langle c_b, A(t) - S(t)\rangle \\
&= \langle Q(t), \hat{A}(t)\rangle - \langle Q(t), \lambda^{*l}\rangle - \langle Q^{max}(t)\sqrt{M_b}c_b, \lambda^{*r}\rangle \quad (9)\\
&+ \langle Q(t), \lambda^{*l}\rangle - \langle Q(t), S^l(t)\rangle \quad (10)\\
&+ \langle Q^{max}(t)\sqrt{M_b}c_b, \lambda^{*r}\rangle - \langle Q(t), S^r(t)\rangle \\
&- \langle c_b, Q(t)\rangle\langle c_b, \hat{A}(t) - S(t)\rangle \quad (11)\\
&+ \langle Q(t), A(t) - \hat{A}(t)\rangle - \langle c_b, Q(t)\rangle\langle c_b, A(t) - \hat{A}(t)\rangle. \quad (12)
\end{aligned}$$

Bounds for the four terms (9)-(12) are established by Lemmas 10-13 in our technical report [14]. The bound on term (9) allows us to distribute the remote service among Q^{max} and the remaining beneficiary queues. This ensures that the remote service provided by helpers will suffice even if a small proportion of it is devoted to helpers. The key observation is that the helper subsystem becomes decoupled from the beneficiaries as Q^B becomes large, which allows us to bound the amount of remote service received by helpers and the arrivals local to a helper but joining a beneficiary queue, as in Lemma 12-13 for terms (11) and (12). Combining the bounds, we can show that the drift of $V(Z^B)$ is negative for sufficiently large V, which gives the following theorem.

Theorem 4: (**State space collapse**) There exists a sequence of finite numbers $\{C_r : r \in \mathbb{N}\}$ such that for each positive integer r,

$$\mathbb{E}\left[||Q_\perp||^r\right] \leq C_r,$$

where $Q_\perp$ is the component of the queue length vector Q perpendicular to the direction c.

3) Upper Bound: We use the Lyapunov drift-based moment bounding technique developed in [6]. The main difficulty arises from the fact that the total amount of service received at beneficiary queues, $\sum_{m\in\mathcal{B}} S_m(t)$, depends on the queueing process $Q(t)$: for any $m \in \mathcal{B}$, the local service provided by server m, $\{S^l_m(t)\}$ is neither i.i.d. nor independent of $Q_m(t)$; The amount of remote service $\mathcal{B}$ received, $\sum_{m\in\mathcal{B}} S^r_m(t)$, relies on whether the maximum queue is among $\mathcal{B}$. In addition, the existence of tasks types shared among $\mathcal{H}$ and $\mathcal{B}$ makes total arrivals for $\mathcal{B}$, $\sum_{m\in\mathcal{B}} A_m(t)$, depend on $Q(t)$ as well. We define the following ideal processes to deal with the dependence.

Ideal scheduling decision process $\hat{\eta}(t)$: For any $m \in \mathcal{B}$, $\hat{\eta}_m(t) = m$. For any $m \in \mathcal{H}$, $\hat{\eta}_m(t) = \eta_m(t)$ if $\eta_m(t) = m$; when $f_m(t^-) = -1$ and $Q_m(t) = 0$, $\hat{\eta}_m(t) = argmax_{n\in\mathcal{B}}\{Q_n(t)\}$. That is, each beneficiary is scheduled to serve its local queue only under the *ideal scheduling*, and an idle helper with an empty local queue is scheduled to serve the maximum beneficiary queue.

Ideal local service process $\hat{S}^l(t)$:

$$\hat{S}^l_m(t) = \begin{cases} X^l_m(t) & \text{if } m \in \mathcal{B} \\ S^l_m(t) & \text{if } m \in \mathcal{H} \end{cases}$$

where each process $\{X^l_m(t), t \geq 0\}_{m\in\mathcal{B}}$ is i.i.d. with $X^l_m(t) \sim$ Bern(α), and is coupled with $\{S^l_m(t), t \geq 0\}_{m\in\mathcal{B}}$ in the following way: If $\eta_m(t) = m$, $X^l_m(t) = S^l_m(t)$; if $\eta_m(t) \neq m$, $X^l_m(t) = 1$ when $R_m(t) = 1$, and $X^l_m(t) \sim$ Bern$(\frac{\alpha-\gamma}{1-\gamma})$ when $R_m(t) = 0$.

Ideal remote service process $\hat{R}(t)$:

$$\hat{R}_m(t) = \begin{cases} 0 & \text{if } m \in \mathcal{B} \\ X^r_m(t) & \text{if } m \in \mathcal{H} \end{cases}$$

where each process $\{X^r_m(t), t \geq 0\}_{m\in\mathcal{H}}$ is i.i.d. with $X^r_m(t) \sim$ Bern(γ), and is coupled with $\{R_m(t), t \geq 0\}_{m\in\mathcal{H}}$ in the following way: If $\eta_m(t) \neq m$, $X^r_m(t) = R_m(t)$; if $\eta_m(t) = m$, $X^r_m(t) \sim$ Bern(γ).

Ideal remote service received $\hat{S}^r(t)$:

$$\hat{S}^r_n(t) = \begin{cases} \sum\limits_{m\in\mathcal{H}} \hat{R}_m(t)\cdot I_{\{\hat{\eta}_m(t)=n\}} & \text{if } n \in \mathcal{B} \\ 0 & \text{if } n \in \mathcal{H} \end{cases}$$

The *ideal departure* for queue m is given by $\hat{S}_m(t) = \hat{S}^l_m(t) + \hat{S}^r_m(t)$.

Ideal arrival process $\hat{A}(t)$: all shared task types are routed to helpers and distributed evenly among the local helper servers.

For $m \in \mathcal{B}$, let

$$\hat{A}_m(t) = \sum_{\bar{L}\in\mathcal{L}_\mathcal{B}:m\in\bar{L}} A_{\bar{L},m} = A_m(t) - \sum_{\bar{L}\notin\mathcal{L}_\mathcal{B}:m\in\bar{L}} A_{\bar{L},m}.$$

For $m \in \mathcal{H}$, let

$$\hat{A}_m(t) = A_m(t) + \sum_{\bar{L} \notin \mathcal{L}_{\mathcal{B}}: m \in \bar{L}} \frac{\sum_{n \in \bar{L} \cap \mathcal{B}} A_{\bar{L},n}}{|\{k : k \in \bar{L} \cap \mathcal{H}\}|}.$$

Then we can rewrite the queue dynamics as

$$Q(t+1) = Q(t) + \hat{A}(t) - \hat{S}(t) + \hat{U}(t),$$

where $\hat{U}(t) = \hat{S}(t) - S(t) + A(t) - \hat{A}(t) + U(t)$. Consider Lyapunov function $W_{||}(Z) = \left|\left|Q_{||}\right|\right|^2$, where $Q_{||}$ is the parallel component of the queue length vector Q with respect to the direction c defined in (8). By setting the drift of $W_{||}(Z)$ to zero, we have

$$\begin{aligned} & 2\mathbb{E}\left[\langle c, Q(t)\rangle \langle c, \hat{S}(t) - \hat{A}(t)\rangle\right] \\ = \; & \mathbb{E}\left[\langle c, \hat{A}(t) - \hat{S}(t)\rangle^2\right] + \mathbb{E}\left[\langle c, \hat{U}(t)\rangle^2\right] \\ & + 2\mathbb{E}\left[\langle c, Q(t) + \hat{A}(t) - \hat{S}(t)\rangle \langle c, \hat{U}(t)\rangle\right]. \end{aligned}$$

We can obtain an upper bound on $\mathbb{E}\left[\langle c, Q(t)\rangle\right]$ by bounding each of the above terms.

Theorem 5: (**Upper Bound**) For the map-scheduling system, consider the arrival process $\{A_{\bar{L}}^{(\epsilon)}(t), t \geq 0\}_{\bar{L} \in \mathcal{L}}$ parameterized by $\epsilon > 0$, with rate $\{\lambda_{\bar{L}}^{(\epsilon)}\}_{\bar{L} \in \mathcal{L}}$ satisfying $\mathcal{B} \neq \emptyset$ and Eqs. (4)- (5). Assume that the Markov chain $\{(Q^{(\epsilon)}(t), f^{(\epsilon)}(t))\}$ is in steady sate under the proposed algorithm. Then for any t and any ϵ with $0 < \epsilon < \Phi\alpha + \gamma(M_h - \Phi)$, the expected beneficiary queue lengths in steady-state can be upper bounded as

$$\mathbb{E}\left[\sum_{m \in \mathcal{B}} Q_m^{(\epsilon)}(t)\right] \leq \frac{(\sigma_b^{(\epsilon)})^2 + (\nu_b^{(\epsilon)})^2}{2\epsilon} + D^{(\epsilon)}, \tag{13}$$

where $D^{(\epsilon)} = o(\frac{1}{\epsilon})$, i.e., $\lim_{\epsilon \to 0^+} \epsilon D^{(\epsilon)} = 0$.

Therefore, in the heavy-traffic limit, the upper bound becomes,

$$\limsup_{\epsilon \to 0^+} \epsilon \mathbb{E}\left[\sum_{m \in \mathcal{B}} Q_m^{(\epsilon)}(t)\right] \leq \frac{\sigma_b^2 + \nu_b^2}{2}. \tag{14}$$

This upper bound under heavy-traffic limit coincides with the lower bound (6), which establishes the first moment heavy-traffic optimality of the proposed algorithm.

B. Evenly Loaded Traffic

We consider the heavy-traffic regime where the arrival rate vector $\lambda \in \Lambda$ satisfies $\mathcal{H} = \mathcal{M}$. There exists $\epsilon > 0$ such that

$$\sum_{\bar{L} \in \mathcal{L}} \lambda_{\bar{L}} = M\alpha - \epsilon. \tag{15}$$

First we consider a special case that any two servers $(\hat{m}, m)$ in the system are connected, in the sense that there exists a sequence of servers $(\hat{m}, m_1, \cdots, m_k, m)$ such that for any two consecutive servers in the sequence, there exists a task type $\bar{L} \in \mathcal{L}$ with $\lambda_{\bar{L}} > 0$ local to both servers. That is, all servers are connected by the arrivals.

Consider the arrival processes $\{A_{\bar{L}}^{(\epsilon)}(t), t \geq 0\}_{\bar{L} \in \mathcal{L}}$, parameterized by $\epsilon > 0$, with arrival rate vector $\lambda^{(\epsilon)}$ connecting all servers and satisfying $\mathcal{H} = \mathcal{M}$ and Eqn. (15). Denote the variance of the arrival process as $(\sigma^{(\epsilon)})^2$. For all queue lengths in steady state, and $0 < \epsilon < M\alpha$, we obtain the three theorems analogous to the locally overloaded case.

Theorem 6: (**Lower bound**)

$$\mathbb{E}\left[\sum_{m=1}^{M} Q_m^{(\epsilon)}(t)\right] \geq \frac{(\sigma^{(\epsilon)})^2 + \nu^2 + \epsilon^2}{2\epsilon} - \frac{M}{2},$$

where ν^2 is the variance of the ideal service process. Therefore, in the heavy traffic limit, we have

$$\liminf_{\epsilon \to 0^+} \epsilon \mathbb{E}\left[\sum_{m=1}^{M} Q_m^{(\epsilon)}(t)\right] \geq \frac{\sigma^2 + \nu^2}{2}.$$

Theorem 7: (**State space collapse**) Let

$$c_u = \frac{1}{\sqrt{M}}(\underbrace{1, 1, \cdots, 1}_{M}).$$

There exists a sequence of finite numbers $\{C_r' : r \in \mathbb{N}\}$ such that for each positive integer r,

$$\mathbb{E}\left[\left|\left|Q_\perp\right|\right|^r\right] \leq C_r',$$

where $Q_\perp$ is the component of Q perpendicular to c_u.

Theorem 8: (**Upper bound**)

$$\mathbb{E}\left[\sum_{m=1}^{M} Q_m^{(\epsilon)}(t)\right] \leq \frac{(\sigma^{(\epsilon)})^2 + \nu^2}{2\epsilon} + D_u^{(\epsilon)},$$

where $D_u^{(\epsilon)} = o(\frac{1}{\epsilon})$, i.e., $\lim_{\epsilon \to 0^+} \epsilon D_u^{(\epsilon)} = 0$. Therefore, in the heavy-traffic limit, we have

$$\limsup_{\epsilon \to 0^+} \epsilon \mathbb{E}\left[\sum_{m=1}^{M} Q_m^{(\epsilon)}(t)\right] \leq \frac{\sigma^2 + \nu^2}{2}.$$

The heavy-traffic optimality of the proposed algorithm follows by the coincidence of lower and upper bounds.

Remark: If the arrival rate vector λ makes some server pairs $(\hat{m}, m)$ isolated from each other, we can always decompose servers into disjoint groups, such that servers within each group are connected, while isolated from servers outside. For each connected group $\mathcal{H}_i$, we can obtain the corresponding upper bound on $\mathbb{E}\left[\sum_{m \in \mathcal{H}_i} Q_m^{(\epsilon)}(t)\right]$ as Theorem 8. Together they give an upper bound on $\mathbb{E}\left[\sum_m Q_m^{(\epsilon)}(t)\right]$, which coincides with the lower bound in the heavy traffic limit.

VI. Evaluation

We evaluate the proposed algorithm against existing algorithms via simulation. We simulate a continuous-time system of $M = 500$ servers with local service rate $\alpha = 1$ and remote service rate $\gamma = 0.5$, which corresponds to a mean slowdown of 2, consistent to the measurements in [15]. We consider both task-level and job-level simulations.

1. Task-level. We model the task arrival as a Poisson process.

2. Job-level. The job arrival is modelled as a Poisson process. We use a truncated Pareto distribution with shape parameter 1.9 to generate the number of tasks for each job, consistent with the workload analysis in [15].

Due to space constraint, we focus on exponential service time distribution for each task, and include the results on constant, Erlang2 and heavy-tail distributions like Bimodal and Weibull in [14]. Similar amount of improvement by the proposed algorithm is observed in all cases.

At each task arrival, a set of three servers are chosen to be local for the task according to the distribution of the requested data. We consider two cases:

1. Evenly loaded traffic. The set of three local servers are sampled uniformly randomly from all M servers. This simulates the scenario when the data requested by the incoming traffic are evenly distributed on all servers.

2. Locally overloaded traffic. At each task arrival, with probability σ, three local servers are sampled uniformly randomly from a subset of N servers, and with probability $1-\sigma$, they are sampled from the remaining $M-N$ servers. When σ is large enough, the N servers form a hot-spot in the system. We report the results for $\sigma = 0.8$ and $N/M = 0.5$.

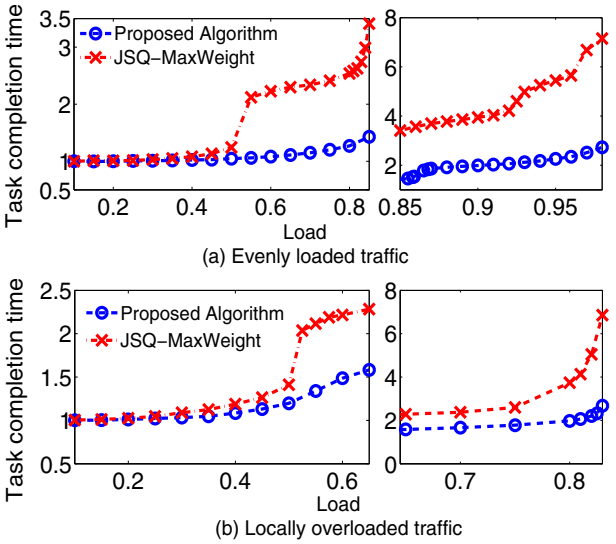

Fig. 3. Average task completion time.

Figure 3 compares the performance of task-level JSQ-MaxWeight and the proposed algorithm. We separate the figures into two load regions and use different vertical scales to make the comparison more visible. With evenly loaded traffic, JSQ-MaxWeight experiences a sharp increase in completion time around the medium load 0.5. This is due to queues building up in the system, which results in the pre-assignment of a significant number of remote tasks. Similar behavior of JSQ-MaxWeight is observed with locally overloaded traffic. For both evenly loaded and locally overloaded traffic, the proposed algorithm has similar performance as JSQ-MaxWeight at low load, and achieves up to 4-fold improvement over medium to high load.

Figure 4 shows the average job completion time of the Hadoop Fair Scheduler (HFS), the fair JSQ-MaxWeight, and our fair scheduler. We choose the wait-time for HFS to achieve 95% data-locality. HFS suffers a loss of throughput as shown

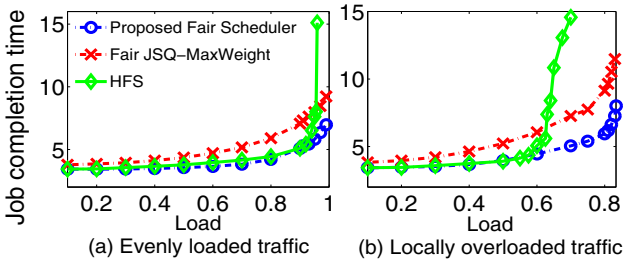

Fig. 4. Average job completion time.

in the drastic increase of completion time at load 0.9 under evenly loaded traffic in Fig. 4(a). The loss of throughput is exacerbated with locally overloaded traffic in Fig. 4(b). Our algorithm achieves the fastest average job completion, which shows that many *small* jobs have benefited from a high-throughput task-level algorithm and relaxed job priority.

VII. Conclusion

We proposed a near-data scheduling algorithm and proved its throughput and heavy-traffic optimality. The algorithm is also shown to have superior performance in simulation.

References

[1] C. Abad, Y. Lu, and R. Campbell. Dare: Adaptive data replication for efficient cluster scheduling. In *Proc. IEEE Intl Conf. Cluster Comp. (CLUSTER)*, 2011.

[2] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris. Scarlett: Coping with skewed popularity content in MapReduce clusters. In *Proc. Eur. Conf. Comput. Syst. (EuroSys)*, 2011.

[3] Apache Hadoop, June 2011.

[4] S. L. Bell and R. J. Williams. Dynamic scheduling of a system with two parallel servers in heavy traffic with resource pooling: asymptotic optimality of a threshold policy. *Annals of Applied Probability*, 11(3):608–649, 2001.

[5] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proc. of OSDI*, 2004.

[6] A. Eryilmaz and R. Srikant. Asymptotically tight steady-state queue length bounds implied by drift conditions. *Queueing Syst. Theory Appl.*, 72(3-4):311–359, 2012.

[7] B. Hajek. Hitting-time and occupation-time bounds implied by drift analysis with applications. *Advances in Applied Probability*, 14(3):502–525, 1982.

[8] J. M. Harrison. Heavy traffic analysis of a system with parallel servers: Asymptotic optimality of discrete review policies. *Annals of Applied Probability*, 8(3):822–848, 1998.

[9] J. M. Harrison and M. J. López. Heavy traffic resource pooling in parallel-server systems. *Queueing Syst. Theory Appl.*, 33(4):339–368, 1999.

[10] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: Fair scheduling for distributed computing clusters. In *Proc. of SOSP*, 2009.

[11] A. Mandelbaum and A. Stolyar. Scheduling flexible servers with convex delay costs: Heavy-traffic optimality of the generalized $c\mu$-rule. *Operations Research*, 52(6):836–855, 2004.

[12] M. Squillante, C. Xia, D. Yao, and L. Zhang. Threshold-based priority policies for parallel-server systems with affinity scheduling. In *Proc. of the IEEE American Control Conference*, 2001.

[13] W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang. A throughput optimal algorithm for map task scheduling in mapreduce with data locality. *SIGMETRICS Perform. Eval. Rev.*, 40(4):33–42, 2013.

[14] Q. Xie and Y. Lu. Priority algorithm for near-data scheduling: Throughput and heavy-traffic optimality. *Techical Report*, 2014.

[15] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *Proc. Eur. Conf. Comput. Syst. (EuroSys)*, 2010.