Non-Work-Conserving Effects in MapReduce: Diffusion Limit and Criticality

†Jian Tan, Yandong Wang, Weikuan Yu, †Li Zhang
†IBM T. J. Watson Research Center, NY 10598, USA
Auburn University, AL 36849, USA
tanji@us.ibm.com, {yzw0025,wkyu}@auburn.edu, zhangli@us.ibm.com

ABSTRACT

Sequentially arriving jobs share a MapReduce cluster, each desiring a fair allocation of computing resources to serve its associated map and reduce tasks. The model of such a system consists of a processor sharing queue for the MapTasks and a multi-server queue for the ReduceTasks. These two queues are dependent through a constraint that the input data of each ReduceTask are fetched from the intermediate data generated by the MapTasks belonging to the same job. A more generalized form of MapReduce queueing model can capture the essence of other distributed data processing systems that contain interdependent processor sharing queues and multi-server queues.

Through theoretical modeling and extensive experiments, we show that, this dependence, if not carefully dealt with, can cause non-work-conserving effects that negatively impact system performance and scalability. First, we characterize the heavy-traffic approximation. Depending on how tasks are scheduled, the number of jobs in the system can even exhibit jumps in diffusion limits, resulting in prolonged job execution times. This problem can be mitigated through carefully applying a tie-breaking rule for ReduceTasks, which as a theoretical finding has direct engineering implications. Second, we empirically validate a criticality phenomenon using experiments. MapReduce systems experience an undesirable performance degradation when they have reached certain critical points, another finding that offers fundamental guidance on managing MapReduce systems.

Categories and Subject Descriptors

G.3 [**Probability and Statistics**]: *Queuing Theory, Stochastic Processes*; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

1. INTRODUCTION

A MapReduce cluster supports multiple users and processes a large number of jobs with distinctive service level objectives, such as job response time and throughput. To fulfill the service commitment, efficient scheduling is critical, as evidenced by the popularity of Hadoop Fair Scheduler [2] and Capacity Scheduler [1].

Each MapReduce job is composed of a certain number of map and reduce tasks. The MapReduce model for serving multiple jobs consists of a processor sharing queue for the MapTasks and a multi-server queue for the ReduceTasks [28]. The underlying rationale is based on the well-known pattern of map and reduce task execution durations. Usually MapTasks are short and ReduceTasks are long. For example, empirical studies show that MapTasks take 19 seconds and ReduceTasks take 231 seconds in median numbers [32]. The map queue model is attributed to the fact that round-robin task scheduling with a small time quantum at an appropriate scale can be well approximated by processor sharing. However, once the execution times become significant for long ReduceTasks, a different treatment at the same time scale, e.g., using a multi-server queue, can better capture the essence of the queueing behavior. Intriguingly, these two queues are dependent through a constraint that the intermediate data fetched by the ReduceTasks cannot exceed those already generated by the MapTasks at any time. Since data movement and processing can be conducted in a fine-grained unit (e.g., a few bytes) [22], this model is different from the traditional tandem queue model (e.g., [9]), for which a job in its entirety moves between several stations.

It is common for data to move through processing units governed by different scheduling disciplines. Thus, the previous MapReduce model can be generalized to capture the essential component of other distributed data processing systems that contain dependent processor sharing queues and multi-server queues. Some of the typical examples include graph computation framework [23], distributed dataflow computing [24], Dryad [16] (a computation framework structured as a directed graph), online streaming jobs [11], and distributed programs with partitioned tables [25].

As one of the dominant paradigms for processing large unstructured data sets in a massively parallel manner, MapReduce has attracted increasing interest in practice. However, there is still a lack of theoretical understanding on its fundamental performance trends at scale. Most of the previous studies seem to focus on either the execution of MapTasks or that of ReduceTasks, abiding by the distinct difference between them but ignoring the dependence therein [1, 2, 7, 17, 32]. This approach for analysis can provide reasonable approximations when the experienced workload is mild. However, under heavy traffic scenarios, the oblivion of this internal dependence misses some intriguing non-work-conserving

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGMETRICS'14, June 16–20, 2014, Austin, Texas, USA.
Copyright 2014 ACM 978-1-4503-2789-3/14/06 ...$15.00.
http://dx.doi.org/10.1145/2591971.2592007 .

effects that, if not carefully dealt with, can greatly degrade system performance.

MapReduce background

MapReduce has a number of implementations. Hadoop [3] is a popular open source realization. Usually, a typical job includes both MapTasks and ReduceTasks. Each MapTask generates intermediate data in key-value pairs after taking a block of input data. Then, ReduceTasks fetch these intermediate data through the copy/shuffle phase according to keys, and proceed to the reduce phase after receiving all the intermediate results. The management of computing resources is through the allocation of task slots; each slot only permits a single task to be launched. As already mentioned, MapTasks are small and independent, executed in multiple parallel waves. Once a MapTask finishes, it immediately releases the occupied slot. Therefore, the available slots can be evenly allocated to multiple competing jobs in a fine-grained time scale. On the contrary, ReduceTasks tend to be long-running and are non-preemptive [17, 7, 32] in most of the MapReduce implementations. They lack the flexibility that MapTasks exhibit, and their exeuction times are usually on a much larger time scale than those of MapTasks [32].

Upon the completion of a small percentage (by default 5% for Hadoop [3]) of the MapTasks, ReduceTasks of the same job are launched greedily to a maximum that is determined by max-min fairness criteria constrained by the number of available slots. However, only using max-min fairness cannot fully describe the scheduling. A tie-breaking rule is needed when there are more competing jobs than the number of available slots. Interestingly, this tie-breaking rule, if not carefully chosen, can result in jumps to the number of jobs when the system is heavily loaded.

For many of the existing implementations [2, 16, 7, 17], ReduceTasks do not support preemptions; they only release the occupied slots after completion, once launched. That is also why their executions are modeled by a multi-server queue. Recently some implementations have introduced preemptions for ReduceTasks [29, 5, 27]. Though this mechanism is interesting, we do not study it in this paper, due to the following reasons. Apart from the fact that many existing MapReduce implementations do not support it yet, there are scenarios that data processing frameworks with components captured by the model studied in this paper are not preferable to execute preemptions due to possible heavy overhead.

Summary of contributions

We study two non-work-conserving effects for the scheduling model of MapReduce that are caused by the dependence constraint between map and reduce tasks. First, we use diffusion limit to describe the typical sample paths that happen with a high probability in a normalized time and space scale (heavy traffic regime). Second, we empirically characterize a criticality phenomenon for the impact on the tail of the job delay distribution, which occurs with a small probability (large deviation regime).

First, we investigate the diffusion approximation. In a heavy traffic regime, we derive the diffusion limits ($Q_m(t)$, $Q_r(t)$) for the number of jobs in the map queue and the reduce queue under a proper scaling. Specifically, the number of jobs in the system can even exhibit jumps in the diffusion limit. A similar result has been shown for servers with vacations [19]. In the case of MapReduce, the jumps are caused by the dependence constraint between the processor sharing queue and the multi-server queue. When the processor sharing queue of the MapTasks is heavily loaded, the intermediate data generating speed can be significantly lower than the fetching speed of the ReduceTasks. Because ReduceTasks are non-preemptive, the computing resources of the occupied slots will not be fully utilized while being taken by idle ReduceTasks. During these periods, the number of jobs in the system can built up quickly, which contributes to the jumps in the diffusion limit. Interestingly, this problem can be avoided through careful designing a tie-breaking rule. These underutilized periods can be minimized through choosing a job with the shortest remaining service time for its map phase. In some existing implementations, this tie-breaking rule is based on the arrival order. This rule can cause jumps in its diffusion limit if the map service time distribution has an increasing hazard function. However, if the distribution has a decreasing hazard function or a finite support, then it is almost as efficient as the one based on the shortest remaining service time for the map phase. This insight has direct engineering implications.

Second, we empirically validate a criticality phenomenon for MapReduce that was theoretically discovered in [28]. Specifically, this phenomenon is on configuring the number of ReduceTasks. It depicts an undesirable performance degradation when the system reaches certain critical points. The existing MapReduce systems often rely on users to specify the number of ReduceTasks. Most users apply large numbers of ReduceTasks to achieve high parallelism, expecting to accelerate the job execution without realizing the dynamics of the number of available reduce slots in a shared cluster. In general, a selfish optimal strategy taken by individuals does not necessarily lead to a globally optimal one. In this regard, an analytic result on how the number of ReduceTasks can impact the delay distribution is established in [28]. It reveals that when the number of ReduceTasks, configured by independent users, reaches a critical value that depends on all the jobs running in the cluster, then the job processing time distribution tail can even change by one order for typical heavy-tailed workload. It implies that, if not carefully managed, much longer job execution times can occur when multiple users configure their jobs independently in a shared cluster. This undesirable consequence is due to the non-work-conserving effect that some large jobs can occupy underutilized reduce slots beyond a critical number, even when they do not have enough work for their ReduceTasks to process. However, this phenomenon remains to be verified against real experiments. We empirically validate it with workloads that are similar to Facebook traffic [6].

Paper organization

Section 2 presents the MapReduce model, with a detailed description of the tie-breaking rules. In Section 3, we first prove a lower bound for the number of jobs in the system, independent of the tie-breaking rules. Then, we investigate conditions under which the previous lower bound is attainable. Specifically, we show that the diffusion limit can even exhibit jumps to the number of jobs in the system, depending on the tie-breaking rules. After that, we demonstrate in Section 4 the criticality phenomenon for typical MapReduce workloads using real experiments. Section 5 contains the details of the testbed and the conducted experiments, which is followed by the conclusion in Section 6.

2. MODEL DESCRIPTION

We use a schematic diagram in Fig. 1 to describe the MapReduce model with arriving jobs. The map phase is modeled as an $M/G/1$ processor sharing queue, and the reduce phase is modeled as a multi-server queue. This model was first proposed in [28].

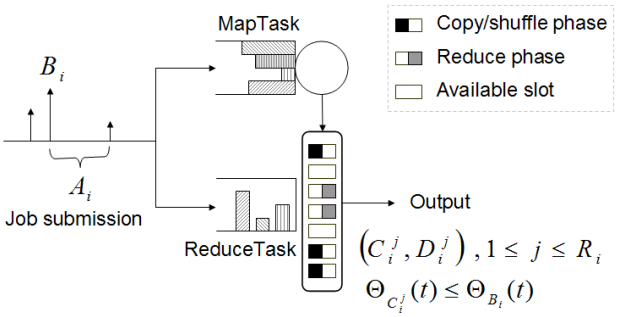

Figure 1: MapReduce model for job scheduling

Job arrivals are modeled by a Poisson random process with rate λ, with the inter-arrival times $\{A_i\}_{i>-\infty}$. Let B_i be the total map service time of job i. The i.i.d. sequence $\{B_i\}_{i>-\infty}$ is independent of other random variables with $B_i \stackrel{d}{=} B$ ("$\stackrel{d}{=}$" means equal in distribution).

The reduce phase is modeled as a multi-server queue. Let $r > 1$ be the total number of reduce slots (servers) in the cluster and R_i the number of ReduceTasks of Job i. ReduceTask scheduling follows the max-min fairness criteria. Denote by $R_i(t)$ the number of running ReduceTasks of job i at time t. When available reduce slots exist, all of them will be allocated to the maximum number of existing jobs such that the assignments $R_i(t)$ for all i in service are as close as possible, constrained by the condition that $R_i(t)$ cannot be larger than the number of unfinished ReduceTasks of job i. In addition, for any job i, the set of time points when at least one of its ReduceTasks are in service forms a time interval $[s_i, e_i]$. In other words, between the starting point s_i and the ending point e_i, there is always at least one ReduceTask running for job i. It means that intermittent services are avoided. We refer the previous scheduling rules as the *primary requirement*. Still, it is possible when multiple jobs compete for a single slot the primary requirement is not sufficient to fully describe the scheduling decisions. In this case, a tie-breaking rule is needed, which will be elaborated in the next section.

A reduce server permits a single ReduceTask to run, which needs to sequentially process the copy/shuffle phase and the reduce phase. We use a tuple (C_i^j, D_i^j) to characterize the jth reduce task of job i, which has a workload C_i^j for the copy/shuffle phase, and D_i^j for the reduce phase. The copy/shuffle phase of a job can overlap in time with its map phase but needs to wait for the map phase to generate the intermediate data. In order to capture this dependence constraint, we introduce function $\Theta_J(t)$, which is defined to be the finished percentage of the workload for job J at time t. For example, $\Theta_{C_i^j}(t)$ represents the attained fraction of service for copy/shuffle phase C_i^j, and $\Theta_{B_i}(t)$ the finished fraction of work for map phase B_i at time t. For MapReduce, we have $\Theta_{C_i^j}(t) \le \Theta_{B_i}(t)$ for all $t, i, 1 \le j \le R_i$, because intermediate data can be fetched by ReduceTasks only after MapTasks have generated them. We term it the *dependence constraint* for MapReduce. This dependence, seemingly simple, tremendously complicates the system and results in interesting analytical characterization.

For illustration purposes, we use measurements collected from a real experiment that is conducted using Fair Scheduler [2] on Hadoop [3]. We plot the number of map and reduce slots taken by each job at each time in the left part of Fig. 2, which are represented by different shaded areas. In addition, we also plot in the right part of Fig. 2 at each time the amount of intermediate data generated by the MapTasks for each job as well as for all of the jobs and the amount of data already fetched by the ReduceTasks.

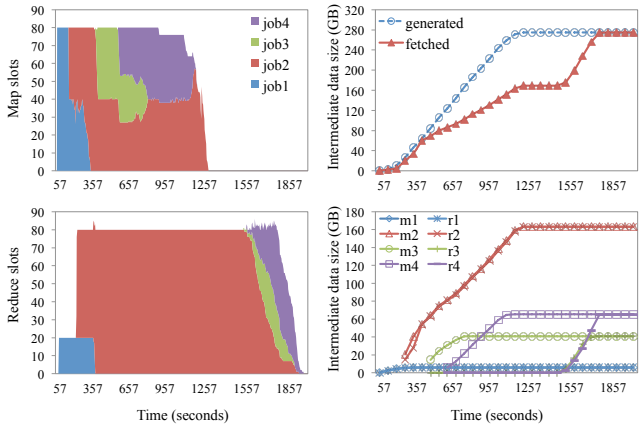

Figure 2: Runtime profile of slot allocation and intermediate data shuffling

As shown in the top-left of Fig. 2, when the second job arrives, the map slots are evenly shared between the first two jobs. However, as soon as the first job completes, all reduce slots are taken by job 2, as shown in the bottom-left of Fig. 2. When jobs 3 and 4 arrive, they can share the map slots evenly with job 2. But only until job 2 releases some reduce slots, can jobs 3 and 4 be able to share these available reduce slots. The curves on the *generated* and *fetched* intermediate data for all of the jobs demonstrate that the progress of copy/shuffle phase cannot exceed that of the map phase. In the bottom-right of Fig. 2, for $1 \le i \le 4$, m_i represents the generated data from the MapTasks of job i and r_i the fetched data by the ReduceTasks of job i.

ASSUMPTION 1. *The three mutually independent random sequences $\{C, C_i^j\}$, $\{D, D_i^j\}$ and $\{R, R_i\}$ are i.i.d. for each sequence. They are also independent of $\{A_i\}$.*

Because of the i.i.d. assumptions, we can simply represent the workload by (B, C, D, R) with arrival rate λ.

In this paper, we use the following notation. The function space $\mathcal{D}[0,1]$, endowed with the Skorohod metric, contains all right continuous functions on $[0,1]$ with left limits; many results in this space can be easily extended to $\mathcal{D}[0,\infty)$ [8]. A subspace of continuous functions is denoted by $\mathcal{C}[0,1]$; $W^*(t) \in \mathcal{C}[0,1]$ is the standard Brownian motion. Our analysis relies on the weak convergence of probability measures; see [8] for a complete treatment. If probability measures P_n and P satisfy $\lim_{n\to\infty} \int_{\mathcal{D}} f dP_n = \int_{\mathcal{D}} f dP$ for every bounded, continuous and real-valued functions f on $\mathcal{D}$, then P_n is said

to converge weakly to P as $n \to \infty$, denoted by $P_n \Rightarrow P$. Let $\rho(x,y) \triangleq \sup_{0\le t\le 1}|x(t)-y(t)|$ be the uniform metric for $x, y \in \mathcal{D}$. When the limit has unmatched jumps, M_1 topology [19] is needed. Denote by $Y_n \xrightarrow{\mathcal{P}} Y$ that a sequence of random variables Y_n converges in distribution to Y. Recall that $a_n \to a$ represents a sequence of real numbers a_n converging to a. Let $\mathbf{1}_{\mathcal{S}}(t)$ be the indicator function of the set $\mathcal{S}$.

3. DIFFUSION LIMIT

Consider a sequence of MapReduce systems indexed by n. The workload $\left(B^{(n)}, C^{(n)}, D^{(n)}, R^{(n)}\right)$ is fed into the nth system with a Poisson arrival rate $\lambda^{(n)}$, $\lambda^{(n)} \to \lambda > 0$.

Let $Q_m^{(n)}(t)$ be the number of submitted jobs that still have not completed their map phases at time t. Regarding the total number of jobs in the reduce queue, we use a restricted definition as follows. Denote by $Q_r^{(n)}(t)$ the number of the jobs in the reduce queue at time t that either 1) have finished their map phases but still have not started any ReduceTasks before time t or 2) successfully occupy at least one reduce server at time t. In addition to the previous two kinds of jobs, the others are the jobs that are still running in the map queue but have not started any ReduceTasks yet. Here we intentionally exclude these jobs from being counted in the reduce queue, since this definition can ease the analysis and presentation. Denote by $Q^{(n)}(t)$ the total number of running jobs in the system at time t. Then, due to the r servers of the reduce queue, we have

$$Q_m^{(n)}(t) + Q_r^{(n)}(t) - r \le Q^{(n)}(t) \le Q_m^{(n)}(t) + Q_r^{(n)}(t). \quad (1)$$

In heavy traffic, we are interested in the scaled processes

$$\hat{Q}_m^{(n)}(t) = \frac{Q_m^{(n)}(nt)}{\sqrt{n}}, \hat{Q}_r^{(n)}(t) = \frac{Q_r^{(n)}(nt)}{\sqrt{n}}, \hat{Q}^{(n)}(t) = \frac{Q^{(n)}(nt)}{\sqrt{n}}.$$

Assume that the map queue is in stationarity at time 0. Thus, $Q_m^{(n)}(0)$ follows a geometric distribution with parameter $\lambda^{(n)}\mathbb{E}\left[B^{(n)}\right]$. Conditional on $Q_m^{(n)}(0) = n$, the first n job sizes are i.i.d. and follow the residual distribution of $B^{(n)}$. We make the following heavy traffic assumptions. For two positive constants η, ξ,

$$\sqrt{n}\left(1 - \lambda^{(n)}\mathbb{E}\left[B^{(n)}\right]\right) \to \eta, \quad (2)$$

$$\sqrt{n}\left(1 - \lambda^{(n)}\frac{1}{r}\mathbb{E}\left[\left(C^{(n)} + D^{(n)}\right)R^{(n)}\right]\right) \to \xi. \quad (3)$$

Under conditions (2) and (3), both the map queue and the reduce queue are in heavy traffic. Furthermore, we need the following technical assumption for the processor sharing queue [21], for $\theta > 0$, $\mu_b \triangleq \mathbb{E}[B] < \infty$ and $\sigma_b \triangleq (\mathbb{V}ar[B])^{1/2} < \infty$,

$$\left(\mathbb{E}\left[B^{(n)}\right], \mathbb{V}ar\left[B^{(n)}\right]\right) \to \left(\mu_b, \sigma_b^2\right). \quad (4)$$

For the reduce queue, assume that it is empty before time 0 ($Q_r^{(n)}(0-) = 0$). Let $Z_i^{(n)} \triangleq \sum_{j=1}^{R_i^{(n)}}\left(C_i^{(n),j} + D_i^{(n),j}\right)$ be an i.i.d. sequence with $Z_i^{(n)} \overset{d}{=} Z^{(n)}$. Specify $\nu_r^{(n)} \triangleq \mathbb{E}\left[R^{(n)}\right]$, $\mu_r^{(n)} \triangleq \mathbb{E}\left[Z^{(n)}\right] < \infty$, $\sigma_r^{(n)} \triangleq \left(\mathbb{V}ar\left[Z^{(n)}\right]\right)^{1/2} < \infty$. For $0 < \nu_r, \mu_r, \sigma_r < \infty$ and every $k > 0$,

$$\left(\nu_r^{(n)}, \mu_r^{(n)}, \sigma_r^{(n)}\right) \to \left(\nu_r, \mu_r, \sigma_r\right), \quad (5)$$

$$\lim_{n\to\infty}\int_{|y|>k\sqrt{n}} y^2 d\mathbb{P}\left[Z^{(n)} - \mu_r^{(n)} < y\right] < \infty. \quad (6)$$

Equation (6) is equivalent to the condition of Lindeberg [26]. This condition, by Theorem 3.1 in [26], ensures the weak convergence to a standard Brownian motion $W^*(t)$

$$\frac{\sum_{i=1}^{\lceil nt\rceil}\left(Z_i^{(n)} - \mu_r^{(n)}\right)}{\sigma_r^{(n)}\sqrt{n}} \Rightarrow W^*(t). \quad (7)$$

For $f \in D$, define a continuous mapping function $\Phi(f(t)) = f(t) - \inf_{0\le s\le t} f(s)$.

Map queue: Under the aforementioned conditions, using Theorem 5.1 and Proposition 5.6 in [21] (see also Theorem 2.3 and Corollary 2.4 in [12]), we know, as $n \to \infty$,

$$\hat{Q}_m^{(n)}(t) \Rightarrow \Phi\left(\hat{Q}_m(0) + W_m^*(t)\right) \triangleq \hat{Q}_m(t), \quad (8)$$

where $W_m^*(t)$ is a Brownian motion with $W_m^*(0) = 0$, drift $-2\eta\mu_b/(\mu_b^2+\sigma_b^2)$, variance $4(\mu_b/\lambda + \sigma_b^2)\mu_b/(\mu_b^2+\sigma_b^2)^2$ and $\hat{Q}_m(0)$ independent of $W_m^*(t)$. The distribution of $\hat{Q}_m(0)$ follows from the stationarity assumption, for $x \ge 0$,

$$\begin{aligned}\mathbb{P}[\hat{Q}_m(0) > x] &= \lim_{n\to\infty}\mathbb{P}\left[\frac{Q_m^{(n)}(0)}{\sqrt{n}} > x\right] \\ &= \lim_{n\to\infty}\left(1 - \frac{\eta}{\sqrt{n}}\right)^{\sqrt{n}x} = e^{-\eta x}.\end{aligned}$$

The map queue, using processor sharing, is symmetric with Poisson arrivals. Thus, the departure process is also Poisson. Due to the reversibility (Theorem 3.11 of [20]), the departure process prior to t is independent of $Q_m^{(n)}(t)$.

Reduce queue: Because of the dependence constraint, the reduce queue can idle when the MapTasks do not generate enough intermediate data. We prove a lower bound to $\hat{Q}_r^{(n)}(t)$ in Section 3.2. Whether this lower bound is attainable depends on the tie-breaking rules described below.

In Section 2 we have specified the *primary requirement* for scheduling ReduceTasks. However, it is not sufficient to fully determine the scheduling. If there are more than r jobs running in the system, say k, then at least $k-r$ jobs are not in service in the reduce queue. In this case when a server becomes available, a tie-breaking rule is needed. In order to show that such a rule can greatly impact $\hat{Q}_r(t)$, we introduce three policies, all satisfying the *primary requirement*. When a tie-breaking decision is needed, we first consider jobs in the reduce queue that have already finished their map services. If there are multiple such jobs, an arbitrary rule that does not depend on the reduce service times (e.g., using the arrival order) can be used. Then, the remaining tie-breaking decision is according to one of the following policies.

- I: Jobs with smaller remaining map service times have higher priorities.
- II: Tie breaking is through uniform random selection.
- III: Jobs arrived earlier have higher priorities.

Interestingly, Policy I attains the lower bound. However, Policy II can incur jumps to the number of jobs in the reduce queue $\hat{Q}_r(t) \notin \mathcal{C}[0,\infty)$. Thus it cannot attain the lower

bound, implying longer delays for the job executions. Policy III may or may not cause jumps, depending on the map service time distribution. These theoretical results have direct engineering implications.

3.1 Intuition

Before conducting the rigorous analysis, we first describe the underlying intuition on when the limit can exhibit jumps. In heavy traffic, $Q_m(t)$ can be approximated by a reflected Brownian motion with drift [12]. When heavily loaded, $Q_r(t)$ is larger than the number r of reduce slots most of the time. Because of the max-min fairness requirement for launching ReduceTasks, the reduce queue will be very close, but different, to a FIFO multi-server queue with a service time $\sum_{j=1}^{R_i}\left(C_j^i + D_j^i\right)$ for job i. In the following, we explain how the tie-breaking rules impact performance.

Policy I chooses the job with the smallest remaining map service. This job completes its map workload earlier than other jobs that are already in the system. The only problem is that the newly arrived jobs can have an even smaller map service. However, this event happens with a small probability. Thus, the arrival process of the reduce queue is close (yet not exactly due to $\Theta_{C_i^j}(t) \le \Theta_{B_i}(t)$) to the departure process of the map queue. Since the map queue is symmetric, its departure process prior to t is independent of $Q_m(t)$. Because of this independence, $\hat{Q}_m^{(n)}(t)$ and $\hat{Q}_r^{(n)}(t)$ converge weakly to $\hat{Q}_m(t)$ and $\hat{Q}_r(t)$, two independent reflected Brownian motions with drifts.

Next, we explain the jumps in the diffusion limit under Policy II. Recall that time t is on the scale of n and the queue size on the scale of $\sqrt{n}$; let $Q_m(nt_0) = q_m\sqrt{n} + o(\sqrt{n})$ and later we will omit $o(\sqrt{n})$. Immediately when $Q_r(nt_0)$ is below r at time nt_0, a random job J^* from the $Q_m(nt_0)$ number of jobs running in the map queue will be selected to take the available reduce slot. Using the state space collapse result for processor sharing queue in heavy traffic (Theorem 4.7 in [12]), we know that approximately job J^* will have a remaining service time B^e that follows the residual distribution of B, with $\mathbb{P}[B^e > x] = \int_x^\infty \mathbb{P}[B > u]/\mathbb{E}[B]du, x \ge 0$. Let $B^e = b^e$. Job J^* shares the map queue equally with $q_m\sqrt{n}$ number of jobs. Therefore, job J^* will work on the reduce slot during the time interval $[nt_0, nt_0 + b^e q_m\sqrt{n}]$. However, during this period of time the reduce queue can only use at most $r-1$ slots to serve other ReduceTasks. Due to condition (3), the service rate of the reduce queue in this time interval is smaller than the input rate $\lambda\mu_r \approx r$ (μ_r is defined in (5)). By law of large numbers, when n is large, during $[nt_0, nt_0 + b^e q_m\sqrt{n}]$, there will be $\lambda b^e q_m\sqrt{n}$ number of jobs joining the reduce queue. Thus, job J^* will cause at least $\lambda b^e q_m\sqrt{n}/r$ number of jobs queueing in the reduce queue at time $nt_0 + b^e q_m\sqrt{n}$, i.e., $Q_r(nt_0 + b^e q_m\sqrt{n}) \gtrsim \lambda b^e q_m\sqrt{n}/r$. The time interval $[nt_0, nt_0 + b^e q_m\sqrt{n}]$ will degenerate to a single point t_0 since the time is on a scale of n. However, there will be a jump that is larger than $\lambda b^e q_m/r$ at time t_0, i.e., $\hat{Q}_r(t_0+) - \hat{Q}_r(t_0-) \ge \lambda b^e q_m/r$.

The analysis for Policy III is based on the arguments for Policy I and II. First, consider the case when B has an increasing hazard function. If a job has processed its map phase for a long time, then its remaining service time is expected to be also long. In this case, the chosen job will have a remaining service time that is stochastically larger than the one under Policy II (uniform random selection). Thus, it can incur jumps. The second case is when B has a decreasing hazard function or a finite support. The chosen job will likely have a short remaining service time. To better understand this point, consider a special case when $B \equiv c$ for a constant $c > 0$. In this case, Policy III is equivalent to Policy I. Thus, the diffusion limit does not have jumps.

3.2 Lower bound

We first prove a lower bound that holds independent of the tie-breaking policies. The proof relies on the construction of a work-conserving queue.

THEOREM 1. *There exists a sequence $\hat{Q}_r^{l,(n)}(t) \le \hat{Q}_r^{(n)}(t)$ such that $\hat{Q}_r^{l,(n)}(t) \Rightarrow \Phi\left(W_r^*(t)\right)$ where $W_r^*(t)$ is a Brownian motion with $W_r^*(0) = 0$, drift $-r\xi/\mu_r$, variance $\lambda + \sigma_r^2/\mu_r^3$ and independent of $W_m^*(t)$.*

Remark 1. When $\{C_i^{(n),j}, D_i^{(n),j}, R_i^{(n),j}\}$ is dependent of $\{B_i^{(n)}\}$, the two Brownian motions $W_r^*(t)$ and $W_m^*(t)$ can be also dependent, but we still have $\hat{Q}_r^{l,(n)}(t) \Rightarrow \Phi\left(W_r^*(t)\right)$.

PROOF. The reduce queue is not work-conserving since $\Theta_{C_i^{(n),j}}(t) \le \Theta_{B_i^{(n)}}(t)$. We construct a work-conserving reduce queue by coupling arguments. The new reduce queue has the same input as the original one.

In the new reduce queue, we specify the following three requirements. First, we abandon the constraint $\Theta_{C_i^{(n),j}}(t) \le \Theta_{B_i^{(n)}}(t)$ so that a task can keep running until finish using the assigned server without being idle. Second, after a job arrives to the new queue it is always given at least an equal number of servers as the same job that is running in the original queue at any time unless it does not have any remaining workload in the new queue. Clearly, under these two requirements, a running job in the new queue completes no later than the same job running in the original queue. Third, only using the preceding two requirements, the new queue may idle even when there are enough workloads. In this case, it arbitrarily allocates the available servers to one of the jobs that are already in service by splitting the workload of its running ReduceTasks. Denote by $Q_r^{l,(n)}(t)$ the number of jobs and by $W_r^{l,(n)}(t)$ the total workload in the new reduce queue at time t. It is clear that $Q_r^{l,(n)}(t) \le Q_r^{(n)}(t)$ for all t based on the construction. Furthermore, the new queue is work-conserving and can process at most r jobs simultaneously.

The only technical issue is that the arrival time points of the reduce queue are not equal to the departure points of the map queue; recall the definition of $Q_r^{(n)}(t)$ at the beginning of Section 3. There are two different cases for a departure from the map queue: 1) if this job still has not started any of its ReduceTasks, then this departure is immediately an arrival to the reduce queue; 2) if this job has taken at least one reduce slot earlier, then the departure does not correspond to an arrival of the reduce queue. The second case is illustrated in Fig. 3. When job i finishes its map phase at time t_i, it has already occupied at least one reduce slot before t_i. The arrival point of this job is the earliest time when a slot is taken, i.e., time ν_i in Fig. 3.

Denote by $L_r^{(n)}(t)$ the number of jobs that have arrived to the reduce queue and by $X^{(n)}(t)$ the number of departures from the map queue on $[0,t]$ for the nth system. It is easy to observe that, due to $\Theta_{C_i^{(n),j}}(t) \le \Theta_{B_i^{(n)}}(t)$, r reduce slots,

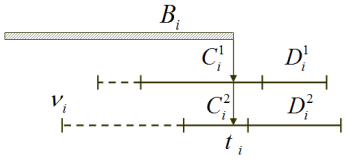

Figure 3: Arrivals of the reduce queue

and the fact that ReduceTasks are not preemptive,

$$X^{(n)}(t) \le L_r^{(n)}(t) \le X^{(n)}(t) + r, \tag{9}$$

which implies that

$$\rho\left(\frac{L_r^{(n)}(nt) - \lambda^{(n)}nt}{\sqrt{n}}, \frac{X^{(n)}(nt) - \lambda^{(n)}nt}{\sqrt{n}}\right) \le \frac{r}{\sqrt{n}} \to 0. \tag{10}$$

Let $U_i^{(n)}$ be the inter-arrival time of the ith job into the reduce queue. Note that $U_i^{(n)}$ are not i.i.d. because some jobs arrive to the reduce queue before their map services finish; see Fig. 3. Therefore, we cannot directly apply Donsker's Theorem (pg.137 of [8]) to derive the diffusion limit. Instead, we use the relationship between the partial sum and the associated counting process.

Applying Theorem 3.1 in [26], we obtain $n^{-1/2}\big(X^{(n)}(nt) - \lambda^{(n)}nt\big) \Rightarrow \lambda^{1/2}W^*(t)$, which, by (10) and the Convergence Together Theorem (Theorem 4.1 in [8]), implies

$$\frac{\left(L_r^{(n)}(nt) - \lambda^{(n)}nt\right)}{\sqrt{n}} \Rightarrow \lambda^{1/2}W^*(t). \tag{11}$$

Then, using the connection between the counting process $L_r^{(n)}(nt)$ and the partial sums $\sum_{i=1}^n \left(U_i^{(n)} - 1/\lambda^{(n)}\right)$ (Theorem 1 of [15]), we obtain

$$\frac{\sum_{i=1}^{\lceil nt \rceil} U_i^{(n)} - nt/\lambda^{(n)}}{\sqrt{n}} \Rightarrow -\lambda^{-1}W^*(t). \tag{12}$$

Based on the construction, the workload $W_r^{l,(n)}(t)$ in the new queue is equal to the workload in a single server FIFO queue. Combining (7), (12) and (3) and using the well-known result on single server FIFO queues (e.g., see Section 6 in [9]) yield

$$\frac{W_r^{l,(n)}(nt)}{\sqrt{n}} \Rightarrow \mu_r \Phi\left(W_r^*(t)\right). \tag{13}$$

Because the new queue can only serve at most r jobs simultaneously and the remaining service of each job being served at time t is upper bounded by $\max_{1\le k\le L^{(n)}(t)} Z_k^{(n)}$, we obtain

$$\left|\sum_{i=L_r^{(n)}(t)-Q_r^{(n)}(t)}^{L_r^{(n)}(t)} Z_i^{(n)} - W_r^{l,(n)}(t)\right| \le r \max_{1\le k\le L_r^{(n)}(t)} Z_k^{(n)}.$$

This result, using

$$\sum_{i=L_r^{(n)}(t)-Q_r^{(n)}(t)}^{L_r^{(n)}(t)} Z_i^{(n)} = Q_r^{(n)}(t)\mu_r^{(n)} + \left(\sum_{i=L_r^{(n)}(t)-Q_r^{(n)}(t)}^{L_r^{(n)}(t)} \left(Z_i^{(n)} - \mu_r^{(n)}\right)\right),$$

implies, for $t \in [0,1]$,

$$\begin{aligned}\rho\left(\frac{W_r^{l,(n)}(nt)}{\sqrt{n}}, \frac{\mu_r^{(n)}Q_r^{(n)}(nt)}{\sqrt{n}}\right) &\le \frac{r\max_{1\le k\le L_r^{(n)}(n)} Z_k^{(n)}}{\sqrt{n}} \\ &+ \sup_{0\le t\le 1}\left|\sum_{i=L_r^{(n)}(nt)-Q_r^{(n)}(nt)}^{L_r^{(n)}(nt)} \left(Z_i^{(n)} - \mu_r^{(n)}\right)\right| \\ &\triangleq I_1^{(n)} + I_2^{(n)}. \end{aligned} \tag{14}$$

Using $L_r^{(n)}(n)/n \to 1/\mu_r$, $\sigma_r^{(n)} \to \sigma_r < \infty$ and Lemma 3.3 in [13], we obtain $I_1^{(n)} \to 0$ almost surely. Observe that (7) validates the C-tightness (Lemma 3.2 in [13]) of the sequence $\left(\sum_{i=1}^{\lceil nt \rceil}\left(Z_i^{(n)} - \mu_r^{(n)}\right)\right)/\sqrt{n}$, we can repeat the same arguments in the proof of Lemma 5.1 (pg. 165) in [13] to show $I_2^{(n)} \xrightarrow{\mathcal{P}} 0$. These two facts yield

$$\rho\left(\frac{W_r^{l,(n)}(nt)}{\sqrt{n}}, \frac{\mu_r^{(n)}Q_r^{(n)}(nt)}{\sqrt{n}}\right) \xrightarrow{\mathcal{P}} 0, \tag{15}$$

which, by (13), the condition $\mu_r^{(n)} \to \mu_r$ and the Convergence Together Theorem (Theorem 4.1 in [8]), implies $\hat{Q}_r^{l,(n)}(t) \Rightarrow \Phi\left(W_r^*(t)\right)$ on $\mathcal{D}[0,1]$. This result can be easily extended to $\mathcal{D}[0,\infty)$. The reason why $W_r^*(t)$ is independent of $W_m^*(t)$ is from the fact that $\{X^{(n)}(s), s < t\}$ is independent of $Q_m^{(n)}(t)$ (Theorem 3.11 of [20]) and that $\{Z_i^{(n)}\}$ is also independent of $Q_m^{(n)}(t)$. □

3.3 Is the lower bound attainable?

Next, we investigate whether the previous lower bound is attainable or not. If the limit is continuous, we can use $\mathcal{D}([0,\infty), J_1)$ with the Skorohod J_1 topology [8]. If the limit has unmatched jumps, M_1 topology is needed [19]. Later our result shows that $Q_r^{(n)}(0) = 0$ but its limit can be equal to a nonzero jump size. Since the pointwise convergence must hold at 0 even using M_1 topology, we exclude the origin and use the space $\mathcal{D}((0,\infty), M_1)$. After excluding the origin, we do not have a problem; this approach to overcome this minor technical issue has been exploited in [19].

THEOREM 2. *1) Under Policy I, if $\mathbb{P}[B_n \le \kappa] = 1, \kappa > 0$, then,*

$$\hat{Q}_r^{(n)}(t) \Rightarrow \Phi\left(W_r^*(t)\right), \tag{16}$$

and $\hat{Q}^{(n)}(t) \Rightarrow \hat{Q}_m(t) + \Phi\left(W_r^*(t)\right)$ in $\mathcal{D}([0,\infty), J_1)$.

2) Under Policy II, if $\mathbb{P}[B_n \le \kappa] = 1, \kappa > 0$, then,

$$\hat{Q}_r^{(n)}(t) \Rightarrow \Phi^*\left(W_r^*(t)\right), \text{in } \mathcal{D}((0,\infty), \mathrm{M}_1), \tag{17}$$

where $\Phi^*\left(W_r^*(t)\right)$ is modified from $\Phi\left(W_r^*(t)\right)$ by having jumps up of random size J_k when $\Phi^*\left(W_r^*(t)\right)$ hits zero at time $\tau_k, k = 0,1,2,\cdots,\tau_0 = 0$, with $J_k \ge \lambda B_k^e \hat{Q}_m(\tau_k)/r$ for i.i.d. $B_k^e \stackrel{d}{=} B^e$. In addition, we have $\hat{Q}^{(n)}(t) \Rightarrow \hat{Q}_m(t) + \Phi^*\left(W_r^*(t)\right)$ in $\mathcal{D}((0,\infty), M_1)$.

3) Under Policy III, if B has an increasing hazard function, the result is the same as in (17) with $\Phi^*\left(W_r^*(t)\right)$ having jumps. If B has a decreasing hazard function or a finite support, the result is the same as in (16).

Remark 2. This result has a direct implication on throughput and delay. Even though these policies result in the same throughput when the system is stable, the experienced job delay will be different. By Little's law [20], the average delay is equal to the average number of jobs in the system divided by the throughput. These possible jumps increase the number of running jobs in the system, implying a prolonged average job execution time. It has been shown that servers with vacations can incur jumps to the number of jobs in heavy traffic [19]. Under Policy II and Policy III with some conditions, jumps also occur. This is an indication that the computing resources are not fully utilized, as if the servers took vacations from time to time. The result on Policy I shows that these jumps can be eliminated through careful scheduling.

Remark 3. The condition $\mathbb{P}[B_n \leq \kappa] = 1$ simplifies the proof, which can be relaxed. In addition, for Policy II, we can prove $J_k \leq \lambda \max\left\{B^e_{k,1}, B^e_{k,2}, \cdots, B^e_{k,r}\right\} \hat{Q}_m(\tau_k)$ for i.i.d. $B^e_{k,l} \stackrel{d}{=} B^e$. Note that $\{J_k\}$ are not i.i.d. since they depend on the map queue size observed at τ_k. It can be shown that $\{J_k\}$ is a Harris-recurrent Markov chain.

We first establish a lemma that shows the lower bound is indeed attainable if the MapReduce system does not have the dependence constraint between the map queue and the reduce queue. When the constraint $\Theta_{C^j_i}(t) \leq \Theta_{B_i}(t)$ is ignored, we term the model a *relaxed MapReduce*. Tie-breaking can use any rule that does not depend on the reduce service times. Though a relaxed MapReduce is not realistic, we will prove that under Policy I, typical sample paths in heavy traffic are close to the ones in the relaxed MapReduce. Therefore, even with the dependence constraint, the lower bound is still attainable with careful scheduling.

LEMMA 1. *For the relaxed MapReduce, as $n \to \infty$,*

$$\hat{Q}^{(n)}_r(t) \Rightarrow \Phi\left(W^*_r(t)\right).$$

PROOF. By Theorem 1, we only need to prove the upper bound. An upper bound for a multi-server queue in heavy traffic is provided in [13], relying the construction of a modified queuing system. The proof in [13] does not directly apply for the reduce queue, since a job have multiple ReduceTasks that can run simultaneously on several servers of the reduce queue. We take a different approach by first characterizing the number $\check{Q}^{(n)}_r(t)$ of ReduceTasks running in the reduce queue at time t. Then, we use the Convergence Together Theorem to connect $\check{Q}^{(n)}_r(t)$ and $Q^{(n)}_r(t)$.

Let $A^{(n)}(t)$ be the total number of ReduceTasks submitted to the reduce queue on $[0,t]$. Recall that $L^{(n)}_r(t)$ denote the number of jobs arriving in the reduce queue on $[0,t]$. Clearly, $A^{(n)}(t) = \sum_{i=1}^{L^{(n)}_r(t)} R^{(n)}_i$. Repeating the same arguments for showing $I^{(n)}_2 \stackrel{\mathcal{P}}{\to} 0$ in (14), we can prove that, on $\mathcal{D}[0,1]$,

$$\rho\left(\frac{A^{(n)}(nt) - \lambda^{(n)}\mathbb{E}[R^{(n)}]nt}{\sqrt{n}}, \frac{\left(L^{(n)}_r(nt) - \lambda^{(n)}nt\right)\mathbb{E}[R^{(n)}]}{\sqrt{n}}\right) \leq \sup_{1\leq t\leq 1} \frac{\sum_{i=1}^{L^{(n)}_r(nt)}\left(R^{(n)}_i - \mathbb{E}[R^{(n)}]\right)}{\sqrt{n}} \stackrel{\mathcal{P}}{\to} 0,$$

which, recalling $\mathbb{E}[R^{(n)}] \to \nu_r$ in (5), using (11) and applying Theorem 4.1 in [8], yields $\left(A^{(n)}(nt) - \lambda^{(n)}\mathbb{E}[R^{(n)}]nt\right)/\sqrt{n} \Rightarrow \lambda^{1/2}\nu_r W^*(t)$. The preceding result, in conjunction with (3) and using Theorem 1 in [14], implies

$$\frac{\check{Q}^{(n)}_r(nt)}{\sqrt{n}} \Rightarrow \nu_r \Phi\left(W^*_r(t)\right). \tag{18}$$

Next, we study $Q^{(n)}_r(t)$ using (18). Note that

$$\left|\sum_{i=L^{(n)}_r(t)-Q^{(n)}_r(t)}^{L^{(n)}_r(t)} R^{(n)}_i - \check{Q}^{(n)}_r(t)\right| \leq r \max_{1\leq k\leq L^{(n)}_r(t)} R^{(n)}_k,$$

which, using the same argument for (14) and (15), implies that $Q^{(n)}_r(nt)/\sqrt{n} \Rightarrow \Phi\left(W^*_r(t)\right)$. □

In the next two subsections, we provide the proofs for Policy I and II. We skip the proof for Policy III, since the result is expected using the intuition explained in Section 3.1 and the proving methods for Policy I and II.

3.3.1 Proof for Policy I

Let $R^{(n)}(t)$ be the smallest remaining service time among all the jobs that are running in the processor sharing map queue at time t; if there are no tasks in the queue then $R^{(n)}(t) = 0$. In addition, let $M^{(n)}(t)$ be the maximum number of jobs in the map queue on the time interval $[0,t]$. Denote by $W^{(n)}_m(t)$ and $W^{(n)}_r(t)$ the total workload in the map queue and the reduce queue at time t, respectively. For $\epsilon_q, \epsilon_r > 0$, define

$$\mathcal{A}^{(n)}_{\epsilon_q,\epsilon_r} = \left\{R^{(n)}(t) < \epsilon_r, \forall t \in \left\{s : s \in [0,n], W^{(n)}_m(s) > \epsilon_q\sqrt{n}\right\}\right\}.$$

This event means that for each time $t \in [0,n]$ with $W^{(n)}_m(t) > \epsilon_q\sqrt{n}$, the processor sharing map queue always contains at least one task with a remaining service time less than ϵ_r.

LEMMA 2. *For any $\epsilon_q, \epsilon_r > 0$ and $0 < \eta < 1$, there exists n_0 such that $n > n_0$ implies*

$$\mathbb{P}\left[\mathcal{A}^{(n)}_{\epsilon_q,\epsilon_r}\right] \geq 1 - \eta. \tag{19}$$

PROOF. The proof relies on the state space collapse property of processor sharing queue in heavy traffic [12, 21]. The state descriptor $\{\mu^{(n)}_t : t \geq 0\}$ is such that for each $t \geq 0$, $\mu^{(n)}_t$ is a random measure on $[0,\infty)$ that has a unit of mass at the residual service time of each task in the nth processor sharing map queue. For $0 < \epsilon < \epsilon^\circ$ and a continuous function

$$g(x) = \mathbf{1}_{x\in[0,\epsilon]}(x) + \left(1 - \frac{x-\epsilon}{\epsilon^\circ - \epsilon}\right)\mathbf{1}_{x\in(\epsilon,\epsilon^\circ]}(x),$$

define a random function

$$H(t) = \left|\int_0^{\epsilon^\circ} g(x)d\mu^{(n)}_t - W^{(n)}_m(t)\int_0^{\epsilon^\circ} g(x)\frac{\mathbb{P}[B^{(n)} > x]}{\mathbb{E}[B^{(n)}]}dx\right|.$$

The state space collapse is characterized by Theorem 4.7 in [12] for GI/GI/1 queue under the condition of a finite forth moment. For M/GI/1 queue it has been relaxed to a finite second moment condition [21]. It implies that, for any $0 < \eta < 1, \xi > 0$, there exists n_0 such that for all $n > n_0$,

$$\mathbb{P}\left[\sup_{t\in[0,n]} H(t) \leq \xi\sqrt{n}\right] \geq 1 - \eta. \tag{20}$$

Recall (4) with $\mathbb{E}[B^{(n)}] = \int_0^\infty \mathbb{P}[B^{(n)} > x]dx \to \mu_b > 0$. Since $\mathbb{P}[B^{(n)} > 0-] = 1$ (0− to avoid the mass at the origin) and $\mathbb{P}[B^{(n)} > x]$ is non-increasing in x, there exists $\zeta, \delta > 0$ and n_0 such that $\inf_{n>n_1} \mathbb{P}[B^{(n)} > \delta]/\mathbb{E}[B^{(n)}] > \zeta$.

The condition $W_m^{(n)}(t) > \epsilon_q\sqrt{n}$ implies, for $\epsilon^\circ < \delta$,

$$W_m^{(n)}(t) \int_0^{\epsilon^\circ} g(x) \frac{\mathbb{P}[B^{(n)} > x]}{\mathbb{E}[B^{(n)}]} dx \geq \epsilon\zeta\epsilon_q\sqrt{n}.$$

In addition, we have $\mu_t^{(n)}([0, \epsilon^\circ]) \geq \int_0^{\epsilon^\circ} g(x)d\mu_t$. Thus, for $0 < \epsilon < \epsilon^\circ < \delta$ and $0 < \xi < \epsilon\zeta\epsilon_q$, the event $H(t) \leq \xi\sqrt{n}$ implies that, if $W_m^{(n)}(t) > \epsilon_q\sqrt{n}$, then $\mu_t^{(n)}([0, \epsilon^\circ]) \geq (\epsilon\zeta\epsilon_q - \xi)\sqrt{n}$. Because $\epsilon\zeta\epsilon_q - \xi > 0$, choosing $\epsilon^\circ < \epsilon_r$, we know $R^{(n)}(t) < \epsilon_r$. Therefore, $\left\{\sup_{t\in[0,n]} H(t) \leq \xi\sqrt{n}\right\} \subseteq \mathcal{A}_{\epsilon_q,\epsilon_r}^{(n)}$, which finishes the proof. □

LEMMA 3. *For any $0 < \eta < 1$, there exists m_η and n_0 such that $n > n_0$ implies*

$$\mathbb{P}[M^{(n)}(n) \leq m_\eta\sqrt{n}] \geq 1 - \eta. \tag{21}$$

PROOF. This is a direct consequence of $\hat{Q}_m^{(n)}(t) \Rightarrow \hat{Q}_m(t)$ in $D([0, 1], J_1)$, as shown in (8). Since taking the maximum of a function is a continuous mapping, we obtain $M^{(n)}(n)/\sqrt{n} \Rightarrow \sup_{t\in[0,1]} \hat{Q}_m(t) \triangleq M_Q$. By the well known result on the maximum of a reflected Brownian motion with negative drift on $[0, 1]$, M_Q is almost sure finite, which proves the lemma. □

For $\epsilon_q, \epsilon_m > 0$, define

$$\mathcal{B}_{\epsilon_q,\epsilon_m}^{(n)} = \left\{ \sup_{0\leq s,t\leq n, |s-t|<\epsilon_m n} |W_m^{(n)}(s) - W_m^{(n)}(t)| < \epsilon_q\sqrt{n} \right\}.$$

It characterizes the continuity of $W_m^{(n)}(t)$ on $[0, n]$. Using Lemma 3.2 of [13], we obtain the following lemma.

LEMMA 4. *For ϵ_q, η, there exists $0 < \epsilon_m < 1$ and n_0 such that $n > n_0$ implies*

$$\mathbb{P}\left[\mathcal{B}_{\epsilon_q,\epsilon_m}^{(n)}\right] \geq 1 - \eta. \tag{22}$$

Since there are r servers in the reduce queue, at any time t we denote by $\mathcal{T}_t$ the set that contains the running reduce tasks on these servers. If no such task exists at time t, then $\mathcal{T}_t = \emptyset$. Each element in $\mathcal{T}_t$, if not empty, can be denoted by r_i^j, meaning the jth reduce task of job i. For a reduce task r_i^j, denote by $V_m^{(n)}(r_i^j, t)$ the remaining map workload of job i observed at time t (with $V_m^{(n)}(\emptyset, t) \equiv 0$), and by $V_r^{(n)}(r_i^j, t)$ the remaining workload of task r_i^j at time t.

PROOF FOR POLICY I. We only need to prove the result on $D([0, 1], J_1)$, since it can be extended to $D([0, \infty), J_1)$ [30]. The proof is based on constructing a new work-conserving reduce queue. It has a larger workload than the original non-work-conserving reduce queue. Then, we compare the workloads in the new queue and the relaxed MapReduce (defined before Lemma 1), and show that the difference of the normalized queue sizes vanishes.

For any $\epsilon_q, \epsilon_r > 0$ and $0 < \eta < 1$, due to Lemmas 2, 3 and 4, the subset $\mathcal{E}^{(n)} \triangleq \{M^{(n)}(n) \leq m_\eta\sqrt{n}\} \cap \mathcal{A}_{\epsilon_q,\epsilon_r}^{(n)} \cap \mathcal{B}_{\epsilon_q,\epsilon_m}^{(n)}$ satisfies $\mathbb{P}[\mathcal{E}^{(n)}] > 1 - 3\eta$ for $n > n_0$. In the following discussion, we only work on the sample paths within $\mathcal{E}^{(n)}$, since we can pass $\eta \to 0$.

Let t_y^x be the time when ReduceTask r_y^x is first assigned to a reduce server. If $V_m^{(n)}(r_y^x, t_y^x) = 0$ then task r_y^x always satisfies the dependence constraint. Therefore, we only need to take care of the case $V_m^{(n)}(r_y^x, t_y^x) > 0$. Consider the map queue workload $W_m^{(n)}(t)$ observed at time t_y^x with $V_m^{(n)}(r_y^x, t_y^x) > 0$. For $\epsilon_q > 0$, there are two different cases: 1) $W_m^{(n)}\left(t_y^x\right) \leq \epsilon_q\sqrt{n}$ and 2) $W_m^{(n)}\left(t_y^x\right) > \epsilon_q\sqrt{n}$.

For the first case, within $\mathcal{B}_{\epsilon_q,\epsilon_m}^{(n)}$, the event $W_m^{(n)}\left(t_y^x\right) \leq \epsilon_q\sqrt{n}$ implies $W_m^{(n)}\left(t_y^x + \epsilon_m n\right) \leq 2\epsilon_q\sqrt{n}$. Define $\mathcal{C}_{\epsilon_q}^{(n)} = \left\{Q_m^{(n)}(t) \leq K\epsilon_q\sqrt{n}, \forall t \in \left\{s : W_m^{(n)}(s) \leq 2\epsilon_q\sqrt{n}, s \in [0, n]\right\}\right\}$. The state space collapse property [12, 21] shows $\mathbb{P}\left[\mathcal{C}_{\epsilon_q}^{(n)}\right] > 1 - \eta$, for n large enough and some $K > 0$ that does not depend on ϵ_q. Therefore, within $\mathcal{B}_{\epsilon_q,\epsilon_m}^{(n)} \cap \mathcal{C}_{\epsilon_q}^{(n)}$, task r_y^x share with at most $K\epsilon_q\sqrt{n}$ number of jobs in the processor sharing map queue during $[t_y^x, t_y^x + \epsilon_m n]$. Since $\mathbb{P}[B_n \leq \kappa] = 1$, choosing $n > (K\kappa\epsilon_q/\epsilon_m)^2$, we know that task r_y^x can finish before time $t_y^x + \epsilon_m n$. Furthermore, it can only idle for at most $K\kappa\epsilon_q\sqrt{n}$ time when waiting for its map phase to finish. Thus, we can upper bound the remaining processing time of r_y^x by $K\kappa\epsilon_q\sqrt{n} + V_r^{(n)}(r_y^x, t_y^x)$. For the second case, task r_y^x has the smallest remaining map service time among all the jobs in the system at time t_y^x under Policy I. By Lemma 2, we know $V_r^{(n)}(r_y^x, t_y^x) < \epsilon_r$ within $\mathcal{E}^{(n)}$. In addition, $M^{(n)}(n) \leq m_\eta\sqrt{n}$ within $\mathcal{E}^{(n)}$, at most $m_\eta\sqrt{n}$ number of jobs share the map queue. Thus, task r_i^j can idle for at most $\epsilon_r m_\eta\sqrt{n}$ time. For $\epsilon_r = \epsilon_q$ and $\bar{K} = \max\{K\kappa, m_\eta\}$, we obtain an upper bound $\bar{K}\epsilon_q\sqrt{n} + V_r^{(n)}(r_y^x, t_y^x)$ to the remaining processing time of task r_y^x for both cases.

Now, we construct a new work-conserving reduce queue, with $W_{new}^{(n)}(t)$ being its workload at time t. The new queue is constructed by replacing the remaining workload of a ReduceTask r_y^x by $\bar{K}\epsilon_q\sqrt{n} + V_r^{(n)}(r_y^x, t_y^x)$ if $V_m^{(n)}(r_y^x, t_y^x) > 0$ and ignoring the dependence constraint $\Theta_{C_i^j}(t) \leq \Theta_{B_i}(t)$ for all i, j. With this construction, conditional on $\mathcal{E}^{(n)} \cap \mathcal{C}_{\epsilon_q}^{(n)}$, the execution time of each job is no smaller in the new queue than in the original queue. In addition, the new queue is work-conserving with $W_{new}^{(n)}(t) \geq W_r^{(n)}(t)$ for $t \in [0, n]$. Note that outside of the set $\mathcal{E}^{(n)} \cap \mathcal{C}_{\epsilon_q}^{(n)}$, these results may not hold.

We operate the new reduce queue and the relaxed MapReduce under Policy I using the same input and the same order to serve ReduceTasks. Let the workload of the reduce queue in the relaxed MapReduce be $W_{relax}^{(n)}(t)$. Denote by $Q_{new}^{(n)}(t)$ and $Q_{relax}^{(n)}(t)$ the number of ReduceTasks in the reduce queue for the new system and the relaxed MapReduce, respectively. It is clear that $W_{new}^{(n)}(t) \geq W_{relax}^{(n)}(t)$ and $Q_{new}^{(n)}(t) \geq Q_{relax}^{(n)}(t)$. Note that a ReduceTask r_y^x can only increase its workload by $\bar{K}\epsilon_q\sqrt{n}$ when $Q_{new}^{(n)}\left(t_y^x\right) \leq r$ for the new system. Thus, when $Q_{new}^{(n)}(t) \leq r$, we have $W_{new}^{(n)}(t) - W_{relax}^{(n)}(t) \leq r\bar{K}\epsilon_q\sqrt{n}$ since there are r servers. On the other hand, when $Q_{new}^{(n)}(t) > r$, the workloads $W_{new}^{(n)}(t)$ and $W_{relax}^{(n)}(t)$ always change by the same value. Therefore, uniformly for both cases, we obtain, within $\mathcal{E}^{(n)} \cap \mathcal{C}_{\epsilon_q}^{(n)}$,

$$W_{new}^{(n)}(t) - W_{relax}^{(n)}(t) \leq r\bar{K}\epsilon_q\sqrt{n}, \text{ for all t} \in [0, \text{n}]. \tag{23}$$

Repeating the same arguments as for (15), we can connect the workloads and the number of jobs between the new

queue and the relaxed MapReduce. Therefore, by (23), there exist n_0 and a constant $d > 0$ such that for all $n > n_0$ and $s \in [0, 1]$,

$$\mathbb{P}\left[\rho\left(\frac{Q^{(n)}_{new}(sn)}{\sqrt{n}} - \frac{Q^{(n)}_{relax}(sn)}{\sqrt{n}}\right) \le d\epsilon_q, \mathcal{E}^{(n)} \cap \mathcal{C}^{(n)}\right] > 1 - 5\eta. \quad (24)$$

Recalling Lemma 1, we obtain

$$\frac{Q^{(n)}_{relax}(sn)}{\sqrt{n}} \Rightarrow \Phi(W^*_r(s)). \quad (25)$$

Passing $\epsilon_q \to 0, \eta \to 0, m_\eta \to \infty$ and using (25), (24) with Convergence Together Theorem, we finish the proof of (16). The proof of the result on $\hat{Q}^{(n)}(t)$ is a direct consequence of (16) and Theorem 1, in view of (1). □

3.3.2 *Proof for Policy II*

We can define two modes for the reduce queue at each time t. Mode A is when at least one reduce task r^x_y running on a server (i.e., $r^x_y \in \mathcal{T}_t$) has unfinished map phase $V_m(r^x_y, t) > 0$. Mode B is when $V^{(n)}_m(r^x_y, t) = 0$ for every $r^x_y \in \mathcal{T}_t$; recall $V^{(n)}_m(\emptyset, t) = 0$. Let $\mathcal{X}_r(t)$ denote the mode of the reduce queue at time t. It is clearly that the reduce queue can be either in mode A or mode B, i.e., $\mathcal{X}_r(t) \in \{A, B\}$.

Now, we can inductively define successive cycles. Recall that the reduce queue is empty at time $0-$ and new reduce tasks join at 0 ($\mathcal{X}_r(0) = A$). Let $T^{(n)}_1 = 0$ and $T^{(n)}_{A,1} = \inf\{t : \mathcal{X}_r(t) = B, t > 0\}$. Denote by $T^{(n)}_2$ the first time after $T^{(n)}_{A,1}$ when the reduce queue turns from mode B to mode A, i.e., $T^{(n)}_2 = \inf\left\{t : \mathcal{X}_r(t) = A, t > T^{(n)}_{A,1}\right\}$. Inductively, let $T^{(n)}_{A,k} = \inf\left\{t : \mathcal{X}_r(t) = B, t > T^{(n)}_k\right\}$ and $T^{(n)}_{(k+1)} = \inf\left\{t : \mathcal{X}_r(t) = A, t > T^{(n)}_{A,k}\right\}$. Let $L^{(n)}_k$ be the length of the kth cycle in the system indexed by n, i.e., $L^{(n)}_k = T^{(n)}_{k+1} - T^{(n)}_k$. Note that $\{T^{(n)}_k, k = 1, 2, \cdots\}$ are not regenerative.

Proof for Policy II. 2) We break the queueing process of the reduce queue over the successive cycles along the time sequence $T^{(n)}_k, k = 1, 2, \cdots$. Consider the process during the kth cycle, $Q^{(n),k}_r(nt) \triangleq Q^{(n)}_r(t)\mathbf{1}_{[T^{(n)}_k, T^{(n)}_{k+1})}(nt)$ for $t \ge 0$. Thus, $T^{(n)}_k = \sum_{i=1}^{k-1} L^{(n)}_i, k \ge 2$ and

$$\hat{Q}^{(n)}_r(t) = \sum_{k=1}^{\infty} Q^{(n),k}_r(nt)/\sqrt{n}. \quad (26)$$

The proof follows a similar approach in [19]. However, the difference is that in our case $T^{(n)}_k, k = 1, 2, \cdots$ are not regenerative points. The issue is that at the beginning of the kth cycle $T^{(n)}_k, k \ge 2$, some of the ReduceTasks in the system can have remaining service times that depend on the queueing dynamics of cycle $k-1$. Let $R^{(n)}_{max}(k)$ be the maximum of the remaining service times for all the ReduceTasks that are still in the system observed immediately before $T^{(n)}_k$.

If we can show that, for each k,

$$\frac{1}{\sqrt{n}}\left(Q^{(n),1}_r(nt), Q^{(n),2}_r(nt), \cdots, Q^{(n),k}_r(nt), L^{(n)}_1, L^{(n)}_2, \cdots, L^{(n)}_k\right) \Rightarrow \left(\hat{Q}^1_r(t), \hat{Q}^2_r(t), \cdots, \hat{Q}^k_r(t), L_1, L_2, \cdots, L_k\right) \quad (27)$$

and $\mathbb{P}[L_k > 0] = 1$ with $T_k \triangleq L_1 + L_1 + \cdots + L_k \xrightarrow{\mathcal{P}} \infty$ as $k \to \infty$, then, we obtain, as $n \to \infty$,

$$\hat{Q}^{(n)}_r(t) \Rightarrow \sum_{k=1}^{\infty} \hat{Q}^k_r(t). \quad (28)$$

We use mathematical induction on the cycle indexed by k, as illustrated in Fig. 4. Assuming it is true for the first

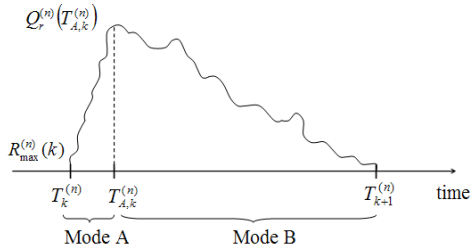

Figure 4: Illustration of the kth cycle

$k-1$ cycles, we prove five results for the kth cycle. (a) $R^{(n)}_{max}(k)/\sqrt{n} \xrightarrow{\mathcal{P}} 0$ (the remaining workload left from cycle $k-1$ vanishes on the scale $\sqrt{n}$) and $\left(T^{(n)}_{A,k} - T^{(n)}_k\right)/n \xrightarrow{\mathcal{P}} 0$ (the time spent in mode A within a cycle vanishes on the scale n). (b) The normalized number of jobs in the reduce queue $Q^{(n)}_r(t)/\sqrt{n}$ observed at $T^{(n)}_{A,k}$ converges in probability to J_k, which has a lower bound $\lambda B^e_k \hat{Q}_m(T_k)/r$. (c) For any $0 < \eta < 1$, there exists $l^\eta_k > 1$ such that $\mathbb{P}\left[T^{(n)}_{k+1} < (l^\eta_k - 1)n\right] > 1 - \eta$ for n large enough. Because of this result, we only need to focus on $t \in [0, l^\eta_k n]$ for studying the first k cycles. On $[0, l^\eta_k n]$, we can use Lemmas 3 and 4, which hold for any fixed $l^\eta_k > 0$. (d) The queueing process $\hat{Q}^{(n)}(t)$ after $T^{(n)}_{A,k}$ converges weakly to a Brownian motion $BM(t)$ starting at J_k with drift $-r\xi/\mu_r$ and variance $\lambda + \sigma^2_r/\mu^3_r$. (e) $T^{(n)}_{k+1}$ converges weakly to the first-passage time to 0 of $BM(t)$. Comparing the proof of Theorem 2.1 in [19] with ours, we only need to prove (a), (b) and (c).

The assumption $\mathbb{P}[B^{(n)} < \kappa]$ makes $R^{(n)}_{max}(k)/\sqrt{n} \to 0$ trivial. Actually, it can be shown that (a) holds even when $B^{(n)}$ has an infinite support. For the first $k-1$ cycles, we distinguish two cases A) $Q^{(n)}_m\left(T^{(n)}_k\right) \le \epsilon_q\sqrt{n}$ and B) $Q^{(n)}_m\left(T^{(n)}_k\right) > \epsilon_q\sqrt{n}$. For case A), using a similar approach to case 1) in the proof of Policy I and the state space collapse property, we can prove that $Q^{(n)}_r\left(T^{(n)}_{A,k}\right) \le K\epsilon_q\sqrt{n}$ for some $K > 0$. Passing $\epsilon_q \to 0$ takes care of this case.

Therefore, we only need to focus on case B). Recall that the tie-breaking rule is uniform random selection. At time $T^{(n)}_k$, the chosen task J^* has a remaining service time $X^{(n)}_k$ with CDF $\nu^{(n)}_k(x), x \ge 0$. Note that $\mathbb{P}[B^{(e)} \le x]$ is a continuous function. We can construct a sequence of simple CDF functions $F_m(x) = \sum_{i=1}^{k_m} f^m_i \mathbf{1}_{[b^m_{i-1}, b^m_i)}(x), b^m_i > 0, 1 \le i \le k_m$ for $m = 1, 2, \cdots$, with $\rho\left(F_m(x), \mathbb{P}[B^{(e)} \le x]\right) < \epsilon/2$ for m large enough. By the state space collapse property of procesor sharing [12], we know $\rho\left(\nu^{(n)}_k(x), \mathbb{P}[B^{(e)} \le x]\right) < \epsilon/2$ for n large enough, which implies $\rho\left(\nu^{(n)}_k(x), F_m(x)\right) < \epsilon$. Thus, we can first assume $X^{(n)}_k \in \{b^m_i, 1 \le i \le k_m\}$ for a fixed m that is large enough.

Define $\mathcal{D}_q^{(n)} = \left\{ Q_m^{(n)} \left(T_k^{(n)} \right) / \sqrt{n} > q \right\}$ for $q \geq \epsilon_q$. Denote by Y_k the length of the time interval when task J^* occupies a server. Using Lemma 4 and the state space collapse property, we can repeat the similar arguments in proving case 1) for Policy I to show, for $1 > \epsilon > 0$ and n large enough,

$$\mathbb{P}\left[Y_k \geq (1-\epsilon) b_i^m q \sqrt{n} | X_k^{(n)} = b_i^m, \mathcal{D}_q^{(n)} \right] > 1 - \eta. \quad (29)$$

Let

$$\mathcal{W}_k^{(n)} = \left\{ Q_r^{(n)} \left(T_k^{(n)} + Y_k \right) > (1-\epsilon)\lambda \min\{b_i^m\} \epsilon_q \sqrt{n} / r \right\}.$$

Due to Poisson departures from the map queue and the law of large numbers, using (29) with $q = \epsilon_q$ on $Y_k \geq (1-\epsilon)\min\{b_i^m\}\epsilon_q\sqrt{n}$, we obtain $\mathbb{P}\left[\mathcal{W}_k^{(n)} | \mathcal{D}_{\epsilon_q}^{(n)} \right] > 1 - 2\eta$, for n large enough. For $\mathcal{E}^{(n)} = \left\{ M^{(n)}(l_{k-1}^{\eta} n) \leq m_\eta \sqrt{n} \right\}$, with $M^{(n)}(l_{k-1}^{\eta} n)$ being the maximum of $W_m^{(n)}(t)$ on the interval $[0, l_{k-1}^{\eta} n]$ (noting that $\mathbb{P}[T_{k-1}^{(n)} < (l_{k-1}^{\eta} - 1)n] > 1 - \eta$), we have, by Lemma 2, for any $0 < \eta < 1$, there exist m_η and n_0 such that $\mathbb{P}\left[\mathcal{E}^{(n)} \right] \geq 1 - \eta$ for all $n > n_0$. We can prove

$$\mathbb{P}\left[\left\{ T_{A,k}^{(n)} - T_k^{(n)} < r\kappa m_\eta \sqrt{n} \right\} \cap \mathcal{E}^{(n)} \cap \mathcal{W}_k^{(n)} | \mathcal{D}_{\epsilon_q}^{(n)} \right] > 1 - 3\eta$$

for n large enough, which, in combination with the upper bound for case A), proves (a). The previous arguments, by (29), at the same time also provide a better lower bound

$$\mathbb{P}\left[Q_r^{(n)} \left(T_k^{(n)} + Y_k \right) > (1-\epsilon)\lambda b_i^m q \sqrt{n} / r | \mathcal{G}^{(n)} \right] > 1 - 2\eta,$$

where $\mathcal{G}^{(n)} = \mathcal{D}_q^{(n)} \cap \{ X_k^{(n)} = b_i^m \}$ with n large enough. It can be used to prove (b). We skip the lengthy technical details and the proof of (c) here. Putting all together and using the M_1 topology, by the arguments in the proof of Theorem 2.1 in [19], we finish the proof. □

4. CRITICALITY PHENOMENON

The criticality phenomenon was first characterized through analysis by the logarithmic asymptotics derived in Theorem 4 of [28]. It assumes that B follows a power law distribution, which represents typical MapReduce workloads [6, 10, 18]. Essentially, it says that if $R^* = \sup\{n : \mathbb{P}[R = n] > 0\}$ is smaller than a critical value $\lambda \mathbb{E}[R]\mathbb{E}[C+D]$ then $\log \mathbb{P}[T > x] / \log x \approx -\alpha + 1$; if $R^* > \lambda \mathbb{E}[R]\mathbb{E}[C+D]$ then $\log \mathbb{P}[T > x] / \log x \approx -\alpha$. It reveals that when the number of ReduceTasks, configured by independent users, reaches a critical value that depends on the statistic characteristics of the jobs running in the system, then the job processing time distribution tail can even change by one order. This can result in much longer job execution times when multiple users configure the number of ReduceTasks for their jobs independently in a shared cluster.

We empirically validate this criticality phenomenon for representative workloads that are similar to Facebook traffic [6]. The details of the testbed are described in Section 5. We can find the best configuration for the number of ReduceTasks that optimizes each job's processing time in a stand-alone environment. However, using these configurations in a shared run-time, we demonstrate that the job execution distribution exhibits much worse performance than using a different configuration, under which every job runs slower in a stand-alone environment. This result provides guidance on configuring the number of ReduceTasks for MapReduce.

To validate the criticality phenomenon, we test a flow consisting of 506 jobs that represent the Facebook traffic as described in [6]. This workload captures two critical features in real traffic [6, 10]. First, the input job sizes exhibit a power law distribution with an exponent equal to 1.6, as measured in [6]. Second, it contains a diverse set of jobs that stress different resource bottlenecks (e.g., CPU, I/O, network) in the cluster. Some jobs such as Wordcount and Termvector consume a large amount of CPU computation while generating little intermediate data. Other jobs such as Sequencecount and Terasort impose heavy pressure on both network and storage systems, straining the copy/shuffle phase.

The job composition of the flow is shown in Table 1. In the table, we sort all the jobs according to their input data sizes and the number of ReduceTasks specified by a given job. All of these jobs are categorized into 9 different groups in increasing order.

Table 1: Composition of 506 jobs that are similar to the Facebook workload [6]

Group	Benchmark	Input Size	Job (#)	ReduceTasks (#)	
				Test-1	Test-2
1	Wordcount	64MB	330	1	1
2	Termvector	128MB	109	4	4
3	Invertedindex	256MB	36	8	18
4	Termvector	512MB	16	12	24
5	Invertedindex	1GB	5	12	32
6	Terasort	2GB	4	16	46
7	Adjancylist	4GB	3	16	46
8	Sequencecount	8GB	2	20	46
9	Sequencecount	16GB	1	20	46

We conduct two sets of tests: all configurations for Test 1 and Test 2 are identical except that the number of ReduceTasks are different, as shown in the column on ReduceTasks for Test 1 and Test 2 in Table 1. In both tests, job submission time points follow the same Poisson process with an average interval of 8 seconds. Test 1 configures each job with a small/medium number of ReduceTasks; none of them takes the full reduce slots. On the contrary, Test 2 sets a large number of ReduceTasks to jobs in Group 6-9. In practice, users optimize their individual jobs in a selfish manner, applying large numbers of ReduceTasks to achieve high parallelism and accelerate job executions. We indeed observe that, with the configuration in Table 1, every job running in a stand-alone environment takes longer time under Test 1 than under Test 2.

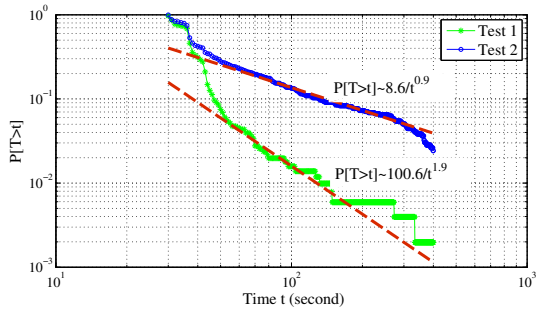

Figure 5: Criticality phenomenon exhibited by workloads similar to Facebook traffic [6]

However, running in a shared environment, as shown in Fig. 5, the job processing time under Test 2 is much worse

than under Test 1, in the sense that they even have one order magnitude of difference for the empirical distribution tail (i.e., 1.9 for Test 1 and 0.9 for Test 2). The average execution time for Test 1 is 42.6 seconds, contrasting to the prolonged average execution time 70.66 seconds under Test 2. In Test 1, we do not observe job starvation, which is defined to be the phenomenon that a job cannot successfully take a reduce slot even after its map phase completes [31]. Nevertheless, in Test 2, though each job runs faster in a stand-alone environment, we observe dramatically increased job delays to many small jobs that are submitted after large ones, since these large jobs can monopolize the availalbe slots. The one order difference of the empirical distributions for the job processing times under the two tests sufficiently validates the criticality phenomenon in Theorem 4 of [28]. The criticality phenomenon shows that that selfish optimization for individual jobs can deviate significantly from a global optimum for configuring MapReduce.

5. EXPERIMENTS

This section describes the test bed and the experiment that illustrates the performance in heavy-traffic.

5.1 Environment

Test-bed Setup: All experiments are conducted on a cluster with 24 nodes using Linux 22.6.18-194.17.4.el5 kernel. Each node is equipped with four 2.67GHz hex-core Intel Xeon X5650 CPUs with Hyper-threading capability, 24GB memory, and two 500GB Western Digital SATA hard drivers. All nodes are connected to the same Top-of-Rack 1Gigabit Ethernet switch.

Hadoop Configuration: One master node is dedicated as the NameNode of the Hadoop Distributed File System and JobTracker of the Hadoop MapReduce. Each of the other slave nodes has 4 map slots and 2 reduce slots, totaling 92 map slots and 46 reduce slots in the cluster. We assign 8GB heap memory to the JobTracker and 1GB heap memory to each map and reduce task, respectively. HDFS block size is 128MB. Given that Hadoop contains hundreds of configuration parameters, we follow the default setting for most of them, such as 3 seconds heartbeat interval, 3 data block replicas, 5% slowstart, etc.

Benchmarks: We employ Tarazu benchmark suite designed in [4] to compose the workloads. It is designed to represent typical workload, including Wordcount, Terasort, Termvector, Invertedindex, Sequencecount and Adjancylist. This benchmark is used to empirically validate the criticality phenomenon in Section 4.

5.2 Detailed desciption

ReduceTasks are launched once a small percentage of MapTasks of a job complete so that the copy/shuffle phase of ReduceTasks can overlap with the map phase. Ideally, as soon as the map phase of a job finishes, ReduceTasks should quickly step into the reduce phase to execute reduce functions without further waiting for intermediate data transfer. However, when the system is heavily loaded, we observe that MapTasks and ReduceTasks cannot overlap in time. To demonstrate this issue, we submit 180 Terasort jobs according to a Poisson process with average arrival interval 18 seconds. The stand-alone execution time of each Terasort job is 120 seconds.

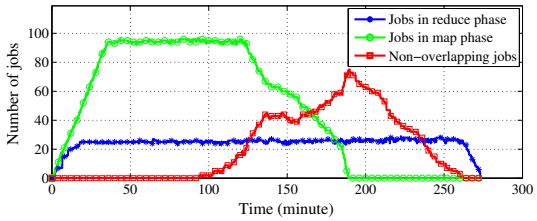

Figure 6: Non-overlap for map and reduce tasks in heavy traffic

In Fig. 6, we profile the number of running jobs that are in the map phase and the reduce phase. In addition, we also monitor the number of non-overlapping jobs, of which all ReduceTasks are launched after all MapTasks of the same job complete. Since a majority of the jobs have not finished their map phases before 100 minutes, these jobs are not non-overlapping jobs. After 100 minutes, we observe a quick increase in the number of non-overlapping jobs in the system. As more and more jobs finish their map phases, a large fraction of them find their copy/shuffle phases severely delayed. The number of non-overlapping jobs reaches 73 (40.5% of total jobs) at around 200 minutes. After 200 minutes, the number of non-overlapping jobs gradually declines due to job departures. This experiment demonstrates that the map phase and the reduce phase of the same job cannot effectively overlap in heavy traffic.

6. CONCLUSION

A MapReduce system serving multiple jobs can be modeled by a processor sharing queue coupled with a multi-server queue. Theoretically understanding its scaling performance can provide guidance in managing the system and configuring job parameters. Through modeling analysis and real experiments, we show that this system, if not carefully dealt with, can cause interesting non-work-conserving effects. First, we derive the diffusion limit when the system is in heavy traffic. We show that, some seemingly innocuous design choices for a tie-breaking rule can result in undesirable performance. Depending on how tasks are scheduled, the diffusion limit can even exhibit spiking behaviors, implying longer job execution times. Second, we empirically validate a criticality phenomenon by real experiments. The results offer insights on configuring MapReduce systems and user job parameters.

References

[1] Capacity scheduler. `http://hadoop.apache.org/mapreduce/docs/r1.2.1/capacity_scheduler.html`.

[2] Fair scheduler. `http://hadoop.apache.org/mapreduce/docs/r1.2.1/fair_scheduler.html`.

[3] Hadoop. http://hadoop.apache.org.

[4] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. N. Vijaykumar. Tarazu: optimizing mapreduce on heterogeneous clusters. In *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '12, pages 61–74, London, England, UK, 2012. ACM.

[5] G. Ananthanarayanan, C. Douglas, R. Ramakrishnan, S. Rao, and I. Stoica. True elasticity in multi-tenant data-intensive compute clusters. In *Proceedings of the*

Third ACM Symposium on Cloud Computing, SoCC '12, pages 24:1–24:7, San Jose, California, 2012. ACM.

[6] G. Ananthanarayanan, A. Ghodsi, A. Wang, D. Borthakur, S. Kandula, S. Shenker, and I. Stoica. PACMan: coordinated memory caching for parallel jobs. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, NSDI'12, San Jose, California, 2012. USENIX Association.

[7] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris. Reining in the outliers in map-reduce clusters using Mantri. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*, pages 1–16, Vancouver, BC, Canada, 2010. USENIX Association.

[8] P. Billingsley. *Convergence of probability measures*. John Wiley & Sons, New York, 1968.

[9] H. Chen and D. D. Yao. *Fundamentals of Queueing Networks*. Springer, 1 edition, 2001.

[10] Y. Chen, A. Ganapathi, R. Griffith, and R. H. Katz. The case for evaluating MapReduce performance using workload suites. In *Proceedings of the 19th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Singapore, July 2011.

[11] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. MapReduce online. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, San Jose, California, 2010. USENIX Association.

[12] H. C. Gromoll. Diffusion approximation for a processor sharing queue in heavy traffic. *Annals of Applied Probability*, 14:555–611, 2004.

[13] D. L. Iglehart and W. Whitt. Multiple channel queues in heavy traffic. I. *Advances in Applied Probability*, 2(1):pp. 150–177, 1970.

[14] D. L. Iglehart and W. Whitt. Multiple channel queues in heavy traffic. II: Sequences, networks, and batches. *Advances in Applied Probability*, 2(2):pp. 355–369, 1970.

[15] D. L. Iglehart and W. Whitt. The equivalence of functional central limit theorems for counting processes and associated partial sums. *The Annals of Mathematical Statistics*, 42(4):1372–1378, 1971.

[16] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 59–72, Lisbon, Portugal, 2007. ACM.

[17] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: fair scheduling for distributed computing clusters. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pages 261–276, Big Sky, Montana, USA, 2009. ACM.

[18] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan. An analysis of traces from a production MapReduce cluster. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 94–103, Washington, DC, USA, 2010. IEEE Computer Society.

[19] O. Kella and W. Whitt. Diffusion approximations for queues with server vacations. *Advances in Applied Probability*, 22(3):706–729, 1990.

[20] F. P. Kelly. *Reversibility and Stochastic Networks*. John Wiley & Sons, 1979.

[21] A. Lambert, F. Simatos, and B. Zwart. Scaling limits via excursion theory: Interplay between Crump-Mode-Jagers branching processes and processor-sharing queues. *The Annals of Applied Probability*, 23:2161–2603, 2013.

[22] M. Lin, L. Zhang, A. Wierman, and J. Tan. Joint optimization of overlapping phases in MapReduce. *Performance Evaluation*, 70(10):720–735, 2013.

[23] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 135–146, Indianapolis, Indiana, USA, 2010. ACM.

[24] D. G. Murray, M. Schwarzkopf, C. Smowton, S. Smith, A. Madhavapeddy, and S. Hand. CIEL: A universal execution engine for distributed data-flow computing. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, Boston, MA, 2011. USENIX Association.

[25] R. Power and J. Li. Piccolo: Building fast, distributed programs with partitioned tables. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–14, Vancouver, BC, Canada, 2010. USENIX Association.

[26] Y. Prokhorov. Convergence of random processes and limit theorems in probability theory. *Theory of Probability & Its Applications*, 1(2):157–214, 1956.

[27] J. Tan, A. Chin, Z. Z. Hu, Y. Hu, S. Meng, X. Meng, and L. Zhang. DynMR: Dynamic MapReduce with ReduceTask interleaving and MapTask backfilling. In *Proceedings of the 9th ACM European Conference on Computer Systems*, EuroSys '14, Amsterdam, The Netherlands, 2014. ACM.

[28] J. Tan, X. Meng, and L. Zhang. Delay tails in MapReduce scheduling. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '12, pages 5–16, London, England, UK, 2012. ACM.

[29] Y. Wang, J. Tan, W. Yu, L. Zhang, and X. Meng. Preemptive ReduceTask scheduling for fair and fast job completion. In *Proceedings of the USENIX International Conference on Autonomic Computing*, 2013.

[30] W. Whitt. *Stochastic-Process Limits: An Introduction to Stochastic-Process Limits and Their Application to Queues*. Springer, February 1, 2002.

[31] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Job scheduling for multi-user mapreduce clusters. Technical Report UCB/EECS-2009-55, EECS Department, University of California, Berkeley, April 2009.

[32] 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 *Proceedings of the 5th European conference on computer systems*, EuroSys '10, pages 265–278, Paris, France, 2010. ACM.