Optimizing Network Usage in MapReduce Scheduling*

Antonio Blanca Seung Woo Shin

University of California, Berkeley
{ablanca, seungwoo}@eecs.berkeley.edu

ABSTRACT

Recent works [8, 19] have opened up an important optimization space for MapReduce scheduling by proposing models where map and shuffle phases are allowed to overlap. We study the MapReduce scheduling problem in the job-level model proposed in [19], focusing on the question of whether optimizing network usage can lead to better system performance. To achieve this, we introduce a technique called Network Utilization Monitoring (NUM), which is a meta algorithm that combines existing scheduling algorithms to obtain a new scheduling algorithm with improved performance in both network utilization and mean response time. The core idea of the technique is to maintain high network utilization and low network congestion at the same time by constantly monitoring the network usage of the system. We show that NUM partially inherits theoretical guarantees of the source algorithms. Moreover, our empirical studies suggest that NUM can be used to obtain algorithms that significantly outperform existing algorithms in the two metrics of interest. We examine two NUM-based policies in our experiments. It is shown that one is optimal for network utilization and the other is near optimal for mean response time, improving upon existing algorithms by as much as 17%.

1. INTRODUCTION

MapReduce is a widely used programming model for distributed parallel computing on large-scale computer clusters. Implementations of the MapReduce framework, such as Apache's Hadoop [1] and Google's MapReduce [12], have been successfully deployed on clusters consisting of tens of thousands of machines and are in active use in various production settings with applications ranging from web searching to machine learning.

In MapReduce, computation is viewed as consisting of two phases, called 'map' and 'reduce' respectively. In the map phase, data is reorganized in such a way that the desired computation can then be carried out by uniformly applying one algorithm on small portions of the data. This second phase is called the reduce phase. Since each of these two phases can employ massive parallelism, MapReduce systems can exploit the extravagant computing power offered by huge-scale clusters.

When discussing the performance of MapReduce systems, it is customary to view a MapReduce job as consisting of three phases instead of two. The additional phase, which sits between the map phase and the reduce phase, is a data transfer phase called the 'shuffle' phase. In this phase, the output of the map phase is recombined and transferred to the compute nodes that are scheduled to perform corresponding reduce operations. Clearly, the performance of MapReduce systems depends heavily on the scheduling of tasks belonging to these three phases, which has developed into an active area of research [34, 6, 18, 33, 5, 17, 21, 25, 26, 4, 16, 19, 28, 8, 7, 13].

Recent studies [8, 19] have observed that the shuffle phase can begin once the map phase starts producing partial output, causing a large overlap between the two phases in practice. (In contrast, the reduce phase is typically not allowed to begin until all the preceding phases are completed.[1]) In fact, the default setting of Hadoop allows the shuffle phase to start when the progress of the map phase exceeds 5%. Therefore, it is natural to ask whether such 'overlapping' of the map and shuffle phases can be optimized to yield better scheduling performance. Moreover, the fact that those two phases utilize different resources, namely CPU and network bandwith, indicates that such overlapping would indeed be necessary to achieve optimal performance.

*This work was done as a final project for CS262A (Advanced Topics in Computer Systems) in the Fall 2013 semester at UC Berkeley.

[1]We note, however, that the overlap between the reduce phase and the other phases is also a subject of active research [3, 27].

Another important observation is that the amount of shuffle workload available at any given time is closely related to the output rate of the map phase, due to the inherent dependency between the map and shuffle phases. In particular, when the job that is being processed is 'map-heavy,' the available workload of the same job in the shuffle phase is upper-bounded by the output rate of the map phase. Therefore, poor scheduling of map tasks can have adverse effects on the throughput of the shuffle phase, causing the network to be idle and the efficiency of the entire system to decrease.

In this paper, we attempt to address the problem of optimizing network usage in MapReduce scheduling. The reason that we are interested in network usage is twofold. Firstly, network utilization is a quantity of independent interest, as it is directly related to the throughput of the system. Note that the total amount of data processed in unit time is simply (CPU utilization)$\cdot$(CPU capacity)$+$ (network utilization)$\cdot$(network capacity). CPU utilization will always be 1 as long as there are enough jobs in the map queue, but network utilization can be very sensitive to scheduling, as explained in the previous paragraph. In fact, network utilization has been identified as a key component in optimization of MapReduce systems in several previous works [3, 27, 10, 11, 24].

Secondly, we envisioned that the optimization of network usage could lead us to algorithms with smaller mean response time. We find the main motivation for this direction of our work in the results of [19]. The two algorithms proposed in that paper, which are based on the idea of informed control of the aforementioned overlap between map and shuffle phases, are shown to yield significantly better mean response time than Hadoop's fair scheduler. However, we observed that neither of these two algorithms explicitly attempted to optimize network usage, which suggested room for improvement.

The main contribution of this paper is a technique called Network Utilization Monitoring (NUM), which is a meta algorithm that can be used in conjunction with other scheduling algorithms to boost their performance in both network utilization and mean response time. The core idea of the technique is to not only maintain high network utilization at all times but also ensure that the network does not become so congested as to become a bottleneck. We prove that NUM can be tuned to inherit theoretical guarantees of the source algorithms regarding mean response time and network utilization. Our simulation results on both synthetic and Facebook workload traces show that NUM can be used to obtain scheduling policies that approach optimality in these two metrics.

2. RELATED WORK

Since MapReduce has become one of the most popular frameworks for large-scale distributed computing, there exists a huge body of work regarding performance optimization of MapReduce. For instance, researchers have tried to optimize MapReduce systems by efficiently detecting and eliminating the so-called "stragglers" [34, 5], providing better locality of data [16, 17, 21], preventing starvation caused by large jobs [33, 25], analyzing the problem from a purely theoretical viewpoint [7], etc. Many of these works concentrate on optimizing individual phases rather than globally optimizing over all three phases, with a few exceptions [3, 27].

On the other hand, recent papers have proposed ways to model and analyze the effect of overlapping phases in the MapReduce scheduling problem [8, 19]. The granularity of such models can be varied; for instance, [8] proposes a task-level model where dependencies between tasks are specified by a directed acyclic graph and presents LP-based algorithms that guarantee constant factor approximation. However, it is unclear how much practical impact those algorithms can have because their approximation factors are rather large and LP-based algorithms are often too complicated to be effective in practice. It seems that the complexity of these results largely draws from the complexity of the model itself.

In contrast, [19] proposes a job-level model which is based on the assumption that each job contains a large number of tasks and therefore can be broken into arbitrary fractions. Because of the simplicity of the model, they obtain algorithms that are fairly easy to implement and yet outperform fair schedulers by large factors. In the present paper, we adopt the job-level model of [19] to study the effect of network utilization on performance of MapReduce systems.

The main subject of the present paper, namely the optimization of network usage in MapReduce scheduling, has been studied in a few recent works [3, 27, 10, 11, 24], but we note that they approach the problem at a much lower level than we do, tackling such issues as network topology, alternative implementations of MapReduce, etc. To our best knowledge, the present paper is the first to approach optimization of network usage as a scheduling problem in a model where map and shuffle phases overlap.

We note that there are many open questions to be studied in between the high-level approach taken by [7, 8, 19] and this paper and the low-level approach taken by the majority of the existing literature on the subject. However, since these two lines of research seem to address orthogonal sets of problems, we hope that they will complement rather than annul each other when this gap is eventually closed by future works.

As a final remark, we note that similar scheduling-type problems are being studied in other distributed com-

puting models as well, such as Spark [20].

3. MODEL

Since our results largely build upon the job-level model of MapReduce originally proposed in [19], we find it necessary to define the model before we proceed to our main results.

The model, called the overlapping tandem queue model, is a job-level model for MapReduce where the map and shuffle phases of the MapReduce framework are modeled as two queues that are put in tandem. Since it is a job-level model, each job is represented by only two numbers, namely the map size and the shuffle size (often denoted x_i and y_i for the i-th job).[2] This simplification is justified by the introduction of two main assumptions. The first assumption states that *each job consists of a large number of small-sized tasks*, which allows us to represent the progress of each phase by real numbers. The second assumption requires that *the amount of shuffle workload generated per unit map workload is uniform over the tasks within the same job*. Under this assumption, we can estimate the available shuffle workload as $x_i(t) \cdot y_i/x_i$ where $x_i(t)$ is the amount of map workload completed at time t.

In addition, there are two constants involved in the model, which are the map service capacity μ_m and the shuffle service capacity μ_s. These numbers denote the maximum amount of workload that can be processed in unit time within each phase. For example, the sum of total map workloads processed in unit time cannot exceed μ_m at any time. Therefore, μ_m roughly corresponds to the total computing power of the cluster and μ_s to the total network bandwith.

The most important feature of the overlapping tandem queue model is that the map and shuffle phases are allowed to overlap with each other. In this model, the scheduling policy is expected to assign shares $S_{m,t}(i)$ and $S_{s,t}(i)$ to each job i at each time t. These numbers respectively denote the amount of map and shuffle workload of the i-th job that will be processed in unit time, and therefore it is required that $\sum_i S_{m,t}(i) \leq \mu_m$ and $\sum_i S_{s,t}(i) \leq \mu_s$ for all t. Note that the model, by nature, allows the two phases of the job to be processed simultaneously. The only restriction on this simultaneous processing, which captures the inherent dependency between the two phases, is that the shuffle workload of the i-th job is produced at the rate $S_{m,t}(i) \cdot y_i/x_i$. Thus, when there is no 'buffer' of shuffle workload already accumulated for the i-th job, it does not make sense to assign more than $S_{m,t}(i) \cdot y_i/x_i$ to $S_{s,t}$, because any excess will lead to a waste of network capacity.

[2]A caveat is that the model completely ignores the reduce phase. This is not so unreasonable as it may initially seem because the reduce phase does not participate in the "overlapping" which is the main subject of this paper.

Consider an example where we have one job in the queue for which the map size is $x_1 = 10$ and the shuffle size is $y_1 = 5$. Suppose also that $\mu_m = \mu_s = 1$. If 0.7 of μ_m was assigned to this job at time t, it means that the shuffle workload for this job is being generated at the rate of $0.7 \cdot 5/10 = 0.35$. Thus, at most 0.35 of μ_s can be assigned to the shuffle phase of this job at time t before we start wasting the network capacity. For a more formal discussion of the model, we refer the reader to [19] and do not reproduce it here due to space constraint.

In the MapReduce setting, this kind of "overlapping" simply translates to the fact that the shuffle phase of a job can begin as soon as a partial output of its map phase is available. This particular feature of the model was experimentally validated in [19] using real systems, and was demonstrated to be a better model of reality than the traditional tandem queue models where the queues do not overlap.

The job-level model offers two big advantages over the more complicated task-level models. Firstly, it gives rise to algorithms that are much simpler than those of task-level models, which enhances chances of being deployed in an actual system. Secondly, the number of jobs in a system is often smaller than the number of tasks by several orders of magnitude, making the problem computationally much less strenuous. However, we note that there are still some questions to be studied regarding the general applicability of the additional assumptions of the job-level model, which are interesting research questions in their own light.

Finally, we note that the optimal scheduling problem under this model is known to be NP-hard in both online and offline settings [19]. This is in contrast to the models with a single queue, where Shortest Remaining Processing Time (SRPT) first scheduling is known to yield optimal mean response time [23].

4. ALGORITHM

In this section, we describe our main technique, Network Utilization Monitoring.

4.1 Measuring Network Utilization

Before we introduce our technique, it is necessary to define the notion of network utilization we seek to optimize. One potential definition would simply be the percentage of network capacity that is being utilized at each time. For instance, if 0.20 was assigned to each of 5 jobs in a system whose network capacity is $\mu_s = 1$, but in fact only 0.15 of network capacity is being utilized for each job because the output rate of the map phase is not fast enough, then the network utilization at that time would simply be 0.75. Having defined network utilization thus, we can easily compute the mean network utilization by averaging this quantity over time. However,

a careful observation reveals that this naïve notion does not make much sense in the online setting. For instance, if one scheduling policy cleverly allocates resources and finishes jobs faster than another scheduling policy, the network utilization for the first scheduling policy may become 0 for some time only because there is no more job in the queue, whereas the other scheduling policy, thanks only to its inefficiency, still has jobs left in the queue and therefore has nonzero network utilization.

To resolve this issue, we will use an alternative notion of network utilization in this paper, which we will call *relative network utilization*. To define this notion, let UOPT be the greedy algorithm which blindly assigns service capacities so that the network utilization is maximized at each time step, i.e. it assigns all of the map service capacity to the job with the maximum y_i/x_i. The relative network utilization of algorithm A is defined to be the ratio of the network utilization produced by A to the network utilization that would be produced by UOPT in the same situation. To enforce well-definedness, we will define the relative network utilization to be 1 when the latter quantity is zero. Note that this metric does not suffer from the problem mentioned in the previous paragraph. Moreover, it is clear what algorithm is optimal for this metric; UOPT itself always achieves relative network utilization of 1.

Our main technique, which we call "Network Utilization Monitoring (NUM)," consists in constant monitoring of relative network utilization and making of adequate adjustments in map service scheduling.

4.2 NUM(P_1, P_2, θ)

NUM is a meta algorithm that takes two scheduling policies P_1 and P_2 as input and combines them to obtain a new scheduling policy NUM(P_1, P_2). Here, P_1 is expected to have good mean response time and P_2 is expected to have good relative network utilization. The main idea of NUM is to use P_1 and P_2 in parallel in map service scheduling, so that the resulting schedule will attain characteristics of both algorithms. The hope is that the resulting algorithm will be good in both response time and relative network utilization.

In the formal exposition that will follow, we will assume for the sake of simplicity that $\mu_m = \mu_s = 1$. However, we note that it is straightforward to generalize our arguments to arbitrary μ_m and μ_s.

To explain the meta algorithm NUM, we will fix a time t and describe the behavior of NUM(P_1, P_2) in relation to P_1 and P_2. Let S_A denote the map service schedule that the policy A would produce at time t. Formally, a schedule is defined as a function that maps a job to the amount of share that is assigned to that job. Thus, $S_A(i)$ will denote the share of map service capacity that is assigned to the i-th job at time t, and we will have $\sum_i S_A(i) = 1$. Then, one of the following cases holds:

1. Let $\alpha = 1$, if one or more of the following conditions hold:
   - The amount of buffer in the shuffle queue is nonzero.
   - The total output rate of S_{P_1} is greater than or equal to the total output rate of S_{P_2}, i.e.,
   $$\sum_i S_{P_1}(i)\frac{y_i}{x_i} \geq \sum_i S_{P_2}(i)\frac{y_i}{x_i}.$$
2. Let $\alpha = 0$, if the total output rate of S_{P_2} is less than 1, i.e., $\sum_i S_{P_1}(i)\frac{y_i}{x_i} < \sum_i S_{P_2}(i)\frac{y_i}{x_i} < 1$.
3. Otherwise, i.e., if $\sum_i S_{P_1}(i)\frac{y_i}{x_i} < 1 \leq \sum_i S_{P_2}(i)\frac{y_i}{x_i}$, let $\alpha = \frac{r_2-1}{r_2-r_1}$ where $r_1 = \sum_i S_{P_1}(i)\frac{y_i}{x_i}$ and $r_2 = \sum_i S_{P_2}(i)\frac{y_i}{x_i}$.

For map service scheduling, NUM(P_1, P_2) uses $\alpha S_{P_1} + (1-\alpha)S_{P_2}$. For shuffle service scheduling, it always follows P_1.

Although the above algorithm may seem complicated initially, a closer look will reveal that it is entirely based on a single idea, which is to primarily use P_1 but also to mix in P_2 exactly as much as it is necessary to ensure that the resulting relative network utilization is at least as high as that of P_2. Moreover, since P_1 has better mean response time than P_2, the resulting algorithm will have better mean response time than P_2.

The intuitive interpretations of the three cases of the algorithm are as follows. Case 1 says that we should follow P_1 if P_1 has better relative network utilization than P_2. Case 2 says that if P_2 is necessary to guarantee high relative network utilization, then we should follow P_2. Finally, in Case 3, where the output rate of P_2 exceeds the network capacity, we balance P_1 and P_2 such that the overall output rate exactly matches the network capacity. These intuitions are formalized in the following theorem.

THEOREM 1. *For any P_1 and P_2, relative network utilization of NUM(P_1, P_2) is at least as high as that of P_2.*

PROOF. It suffices to analyze the three cases in the definition of the algorithm and show that the relative network utilization of NUM(P_1, P_2) is always greater than or equal to that of P_2. Cases 1 and 2 are trivial. In case 3, since NUM(P_1, P_2) assigns the share of $\frac{r_2-1}{r_2-r_1}$ to P_1 and $1 - \frac{r_2-1}{r_2-r_1}$ to P_2, the output rate of the map service schedule is exactly $\frac{r_2-1}{r_2-r_1} \cdot r_1 + \left(1 - \frac{r_2-1}{r_2-r_1}\right) r_2 = 1$, which guarantees relative network utilization of 1. □

COROLLARY 1. *For any P, NUM(P, UOPT) is optimal for relative network utilization.*

However, when the main interest is in improving mean response time, it may be undesirable to require that the resulting algorithm have such high relative network utilization. To be able to relax this condition when there is a need, we introduce a parametrized version of NUM; NUM(P_1, P_2, θ) is the same as NUM(P_1, P_2) except that it enforces α to be at least $1 - \theta$ at all times. Intuitively, the parameter θ controls how much degradation of mean response time we are willing to accept in exchange of better network utilization. That is, the smaller θ is, the more the resulting algorithm behaves like P_1. We immediately have the following theorems.

THEOREM 2. *The relative network utilization of NUM (P_1, P_2, θ) is greater than or equal to θ times the relative network utilization of P_2.*

THEOREM 3. *The mean response time of NUM(P_1, P_2, θ) under service capacities μ_m, μ_s is less than or equal to that of P_1 under service capacities $(1-\theta)\mu_m, \mu_s$.*

For example, since MaxSRPT is known to be 2-speed 1-competitive [19], NUM(MaxSRPT, P, 1/2) for any P is 4-speed 1-competitive. The proofs of these theorems are straightforward and we omit them due to space constraints.

5. EVALUATION

5.1 Experimental Setup

To evaluate our algorithms, we simulated the overlapping tandem queue model described in Section 3 under various workloads and job arrival distributions. For simulation, we used the discrete-event simulator implemented by the authors of [19] for their experimental studies.

5.1.1 Algorithms

We used the following algorithms in our experiments.

1. **Fair Scheduler**: Fair Scheduler simply divides up each resource fairly among all jobs that are in the queue. For instance, if there is only one job in the map queue, the computing power of the entire cluster is dedicated to that one job. If there are three jobs in the shuffle queue, the network bandwidth is divided into thirds and assigned to each job equally. This policy was implemented to mimic Hadoop's Fair Scheduler [2].
2. **MaxSRPT**
3. **SplitSRPT**: Since MaxSRPT and SplitSRPT of [19] have greatly improved upon fair schedulers in mean response time, they will serve as better points of comparison in that metric. Theoretically, both of these algorithms guarantee 2-approximation of optimal mean response time.[3] It was also shown in [19] that they complement each other in the sense that MaxSRPT is near optimal when jobs mostly have balanced map and shuffle sizes, and SplitSRPT is near optimal when jobs mostly have unbalanced map and shuffle sizes.
4. **NUM(MaxSRPT, UOPT,** 1): By Theorem 2, this policy has optimal mean relative utilization. Since MaxSRPT, which is very good for mean response time, is used as P_1, it is expected that the mean response time of this algorithm will not be degraded too much.
5. **NUM(MaxSRPT, SplitSRPT,** 0.85): Since our experimental investigations of MaxSRPT and SplitSRPT suggested that MaxSRPT is generally better for mean response time and SplitSRPT is generally better for mean relative network utilization, we chose MaxSRPT and SplitSRPT as P_1 and P_2 of NUM. Since these two algorithms are already very good in mean response time, we envisioned that using NUM with these two algorithms may further improve mean response time. The parameter 0.85 was chosen empirically by running small experiments on randomly generated workloads.

5.1.2 Metrics

Clearly, the metrics of interest in our experiments were mean response time and mean relative network utilization. We note that the optimal value of mean relative network utilization is always equal to 1; it cannot be greater than 1 by definition and it cannot be less than 1 because UOPT achieves 1. Hence, this metric automatically tells us how close to optimal a schedule is.

On the other hand, mean response time generally does not reveal any information about how close to the optimum we might be. Moreover, it is impossible to compute the optimal mean response time for large datasets because the problem is known to be NP-hard [19]. Therefore, we resort to computing a theoretical lower bound for mean response time as in [19]. This lower bound can be computed as follows. Suppose x_i, y_i, and r_i denote the map size, shuffle size, and arrival time of the i-th job respectively. Then, if $q(t)$ denotes the number of jobs that are in at least one of the queues at time t, the total response time is given by $\int_{t=0}^{\infty} q(t)$. Now let A be the single server queue model where the i-th job arrives at time r_i with size x_i and B another single server queue model where the i-th job arrives at time r_i with size y_i.

[3]More precisely, 2-approximation in the offline setting and 2-speed 1-competitiveness in the online setting.

Also, let $q_A(t)$ and $q_B(t)$ denote the number of jobs in the queue for each model at time t. It is not hard to see that $q_A(t) \leq q(t)$ and $q_B(t) \leq q(t)$ if SRPT is used in A and B. Therefore, the total response time of the overlapping tandem queue model is lower bounded by $\int_{t=0}^{\infty} \max\{q_A(t), q_B(t)\}$.

5.1.3 Workloads

Our experiments were performed on both synthetic and real workloads. In the synthetic workload, job sizes were sampled from log-normal distributions in the following manner; first, the map size (x_i) was sampled randomly from a log-normal distribution with mean equal to 1 and standard deviation equal to 3.65. Then, the shuffle size (y_i) was generated by multiplying the map size with another sample from a different log-normal distribution with mean equal to 1 and standard deviation equal to 3.29. The reason for sampling shuffle sizes this way was to capture the intuition that map and shuffle sizes of jobs should be to some extent correlated. This process was introduced in [19] as a way to approximate the statistics of the publicly available Google traces [32]. The use of log-normal distributions is also supported by several experimental studies [19, 22]. The job arrival distribution was simulated by a Poisson process with varied values of the parameter λ.

The other workload that was used in our experiments was the Facebook-trace-based workload generated by the authors of [9]. This dataset, which contains 24,442 jobs, was prepared by sampling and concatenating 24 one-hour intervals from a trace of the Facebook cluster. The original trace spans a 1.5-month period from October 2010 to November 2010, and the cluster consisted of about 3,000 machines at that time.

5.2 Experimental Results

5.2.1 Synthetic Workload

In experiments on synthetic data, the parameter λ of the Poisson process was varied in order to study the performance of different algorithms under different job arrival densities. Since λ is the reciprocal of the expected interval between job arrivals, a higher value of λ implies denser arrival of jobs. Eight different values of λ were tried, on each of which the experiment was run 100 times. In each run, the number of jobs was $2,000$ and the service capacities μ_m and μ_s were fixed at 1.

Fig. 1 shows mean response time and mean relative network utilization of various algorithms under different values of λ. Mean response time is reported in the form of the ratio to the theoretical lower bound described in Section 5.1.2. Note that Fair Scheduler was dropped from the experiment after $\lambda = 1.00$. This was necessary because it was causing a significant slowdown to the whole experiment by keeping too many jobs in the queues. The ratio of Fair Scheduler's mean response

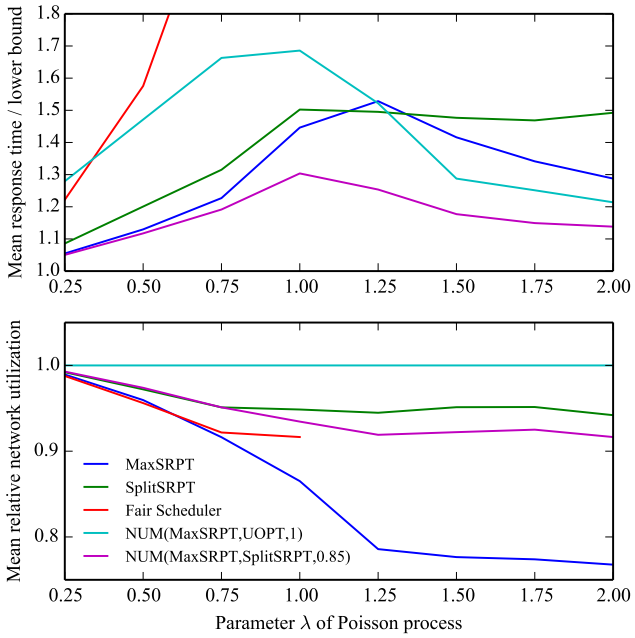

Figure 1: Mean response time and mean relative network utilization of various algorithms on synthetic workloads.

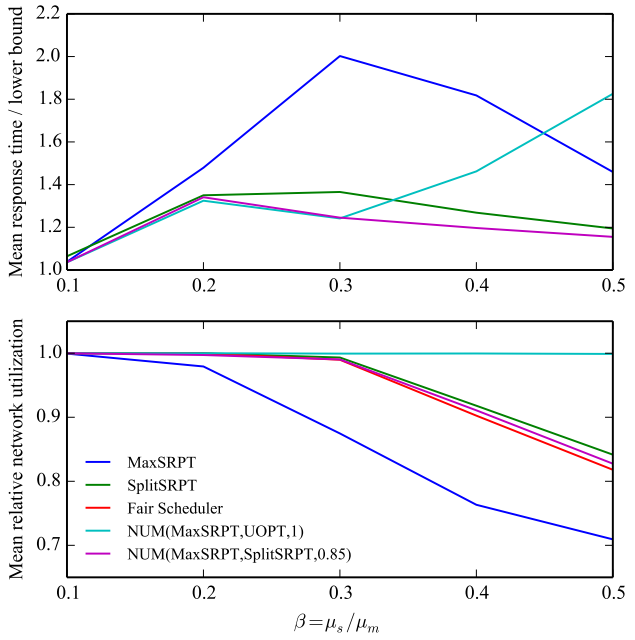

Figure 2: Mean response time and mean relative network utilization of various algorithms on Facebook workload.

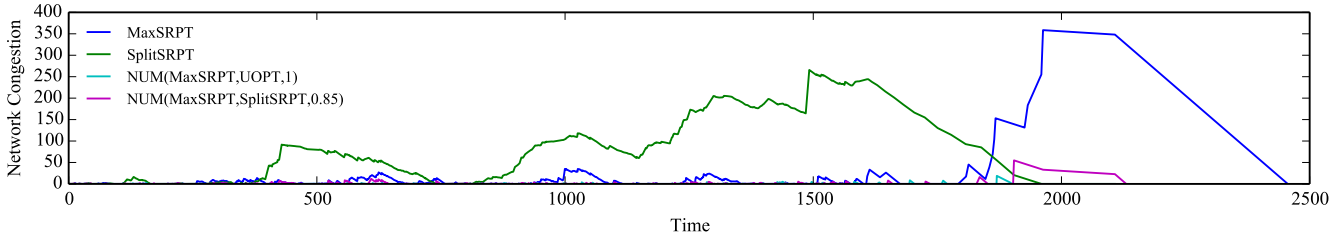

Figure 3: Network congestion of a typical run on synthetic data ($\lambda = 1.25$).

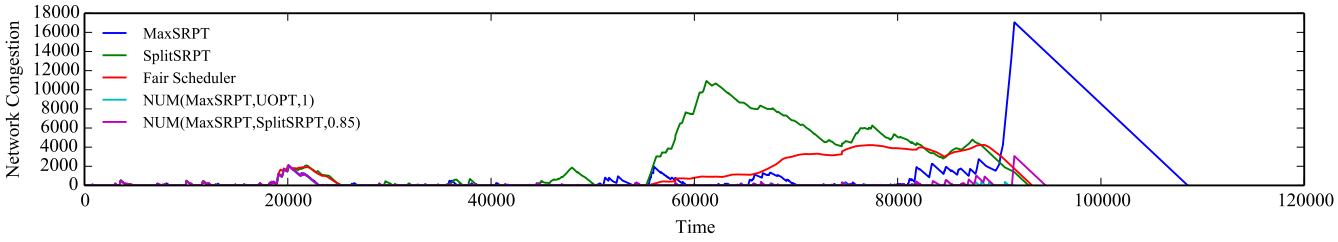

Figure 4: Network congestion on Facebook data ($\beta = 0.5$).

time to the lower bound was 2.2515 at $\lambda = 0.75$, and 3.9443 at $\lambda = 1.00$, suggesting that it is not competitive at all at heavier workloads.

As expected from our theorems, NUM(MaxSRPT, UOPT, 1) is indeed optimal for relative network utilization. Its mean response time is worse than that of MaxSRPT and SplitSRPT when the workload is light, but still within a competitive range as opposed to Fair Scheduler which quickly falls behind as λ increases. As the workload becomes heavy enough, NUM(MaxSRPT, UOPT, 1) begins to outperform MaxSRPT and SplitSRPT.

The most outstanding result is the mean response time of NUM(MaxSRPT, SplitSRPT, 0.85). It consistently outperforms MaxSRPT and SplitSRPT at all values of λ, achieving as much as 17% improvement over those two algorithms at $\lambda = 1.25$. This result suggests that NUM(MaxSRPT, SplitSRPT, 0.85) is a significantly better algorithm than MaxSRPT or SplitSRPT when the workload is on the heavier side.

To understand why NUM-based algorithms perform better than MaxSRPT and SplitSRPT on heavier workloads, we plotted the network congestion of a typical run at $\lambda = 1.25$ in Fig. 3. Network congestion was measured as the total amount of buffer in the shuffle phase, which was summed across all jobs. The plot clearly shows that network bandwidth often becomes a bottleneck for MaxSRPT and SplitSRPT, whereas NUM-based algorithms avoid such a situation by spreading shuffle workloads over time.

5.2.2 *Facebook Workload*

The control variable in experiments on Facebook workload was the ratio of the shuffle service capacity to the map service capacity ($\beta \triangleq \mu_s/\mu_m$). First, μ_m was set to be $\sum_i x_i/86400$, i.e., the capacity that will allow all the map tasks to be processed in exactly a day if they were all submitted at the same time. Then, μ_s was subsequently picked by varying β from 0 to 1. The objective of the experiment was to understand how different algorithms behave on CPU-strong clusters versus network-strong clusters.

Fig. 2 and 5 show mean response time and mean relative network utilization of various algorithms under different values of β. Fair Scheduler is not visible in the response time plot because its mean response time was completely out of scale compared to that of other algorithms. The ratio of Fair Scheduler's mean response time to the theoretical lower bound was 18.8094 at $\beta = 0.1$, and 4.3598 at $\beta = 1.0$.

As before, the experimental data confirms that NUM (MaxSRPT, UOPT, 1) is optimal for relative network utilization and that NUM-based algorithms outperform MaxSRPT and SplitSRPT when the workload is heavy. The improvement is more modest than in synthetic data, peaking at 9% when $\beta = 0.3$. We find the explanation for the smaller improvement in that MaxSRPT, which is P_1 of our meta algorithm, yields much worse mean response time than SplitSRPT on this workload. It is remarkable that NUM(MaxSRPT, SplitSRPT, 0.85) still outperforms MaxSRPT and SplitSRPT consistently.

After $\beta = 0.5$, the gap between SplitSRPT and NUM (MaxSRPT, SplitSRPT, 0.85) continues to close, which we do not show in Fig. 2 because of scale issues. At $\beta = 1.0$, SplitSRPT has mean response time to lower

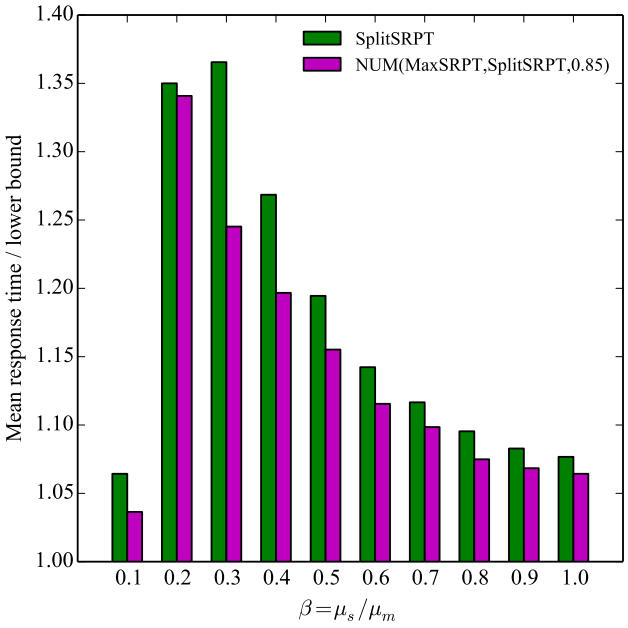

Figure 5: Mean response time comparison between SplitSRPT and NUM(MaxSRPT, SplitSRPT, 0.85) on Facebook workload.

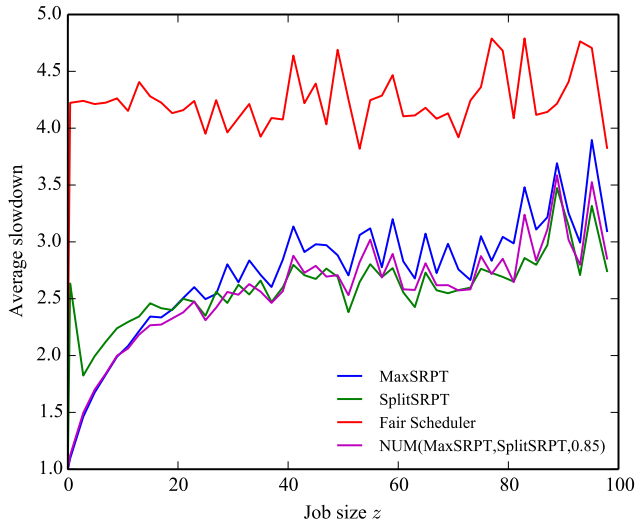

Figure 6: Average slowdown study with 50 buckets.

bound ratio of 1.0767, whereas NUM(MaxSRPT, SplitSRPT, 0.85) has 1.0643. That is, when β is large, all of MaxSRPT, SplitSRPT, and NUM(MaxSRPT, SplitSRPT, 0.85) approach optimality. An explanation for this could be that such large values of β correspond to unrealistic situations where the network is capable of processing much more data in unit time than CPU. Since the job sizes in this workload is recorded in the unit of bytes, it is more realistic to use a value of μ_s that is much smaller than μ_m.

Finally, Fig. 4 plots network congestion of various algorithms on the Facebook workload, which shows a similar behavior to before.

5.3 Fairness

A natural concern with SRPT-based policies is unfairness to large jobs. The unfairness of SRPT has been a subject of many studies [14, 15, 29, 30, 31], and the unfairness of MaxSRPT and SplitSRPT specifically was also studied in [19]. Interestingly, this study shows that MaxSRPT and SplitSRPT have better response times than Fair Scheduler for all job sizes. We study the unfairness of NUM(MaxSRPT, SplitSRPT, 0.85) following the standard approach of [30, 31, 19]. This approach consists in computing the "average slowdown" for each job size z, which is defined as the ratio of mean response time for jobs of size z to z. The job size z_i for the i-th job is defined as $z_i = \max\{x_i/\mu_m, y_i/\mu_s\}$. Note that z_i is precisely the amount of time the i-th job would take if it were the only job in the queue. In the ideal case, the average slowdown would take similar values for all job sizes.

Our experiments were performed on synthetic workloads generated using the method described in Section 5.1.3, but the number of jobs was set to be a rather large 10^6. This was because we needed sufficiently many jobs in each "bucket" for the estimation of the average slowdown to be accurate. Other parameters were set to be $\lambda = 0.75$ and $\mu_m = \mu_s = 1$. The results (Fig. 6) show that the mean response time is better under NUM(MaxSRPT, SplitSRPT, 0.85) than under Fair Scheduler for every job size, indicating that there is no reason to choose Fair Scheduler over NUM(MaxSRPT, SplitSRPT, 0.85) for fairness concerns.

5.4 More For Less?

Another question we studied in these experiments was whether the increased efficiency of network usage allows us to achieve the same mean response time with less resources. To answer this question, we performed two sets of experiments on the Facebook workload using $\beta = 0.3$.

Firstly, we ran NUM(MaxSRPT, SplitSRPT, 0.85) with service capacities $\gamma \cdot \mu_m$ and μ_s and determined the minimum value of γ that allowed us to match the mean response time of MaxSRPT, SplitSRPT, and Fair Scheduler with μ_m and μ_s. The resulting γ was 0.88, 0.98, and 0.70 respectively.

Secondly, we ran NUM(MaxSRPT, SplitSRPT, 0.85) with service capacities μ_m and $\gamma \cdot \mu_s$ and determined the minimum value of γ that allowed us to match the mean response time of MaxSRPT, SplitSRPT, and Fair Scheduler with μ_m and μ_s. The resulting γ was 0.81,

0.94, and 0.55 respectively.

The result indicates that NUM(MaxSRPT, SplitSRPT, 0.85) operating on a cluster of smaller capacity can achieve the same performance as other algorithms would achieve on a cluster of larger capacity. While the improvement over SplitSRPT is relatively modest, it is unclear whether this tendency will persist on other realistic workloads. It is somewhat surprising that NUM (MaxSRPT, SplitSRPT, 0.85), whose performance is based on the idea of optimization of network usage, allows for the saving of CPU power as well.

6. DISCUSSION

We have introduced our main technique Network Utilization Monitoring and demonstrated clear performance improvements over existing algorithms. We have established two use cases; firstly, we used it to combine MaxSRPT and UOPT to get a new scheduling policy which is optimal for network utilization and competitive for response time. Secondly, we used it to combine MaxSRPT and SplitSRPT and obtained a scheduling policy that consistently outperforms those two policies in mean response time. The improvement was generally bigger when the cluster was busier, peaking at 17% in synthetic workloads and 9% in Facebook workloads.

Since our approach is built on top of a high-level model of MapReduce, there may be unforeseen difficulties in implementing our algorithm in real systems. For example, how can we know the size of jobs before we process them? It was out of scope of this paper to try to answer this question, but we note that there exist some results on this subject [34, 33], which mostly focus on utilizing historical data for the estimation. There are also issues regarding how capacity sharing is actually implemented at the low level, which may or may not require a new implementation of the MapReduce framework.

For future work, it may be interesting to establish more use cases for our technique. For instance, how would it behave when applied on algorithms that are not SRPT-based? Can we apply similar ideas to improve system performance in other metrics? Last but not least, we remark that the core idea of our technique does not seem to be crucially dependent on the job-level model on top of which it is defined. This raises an important question; is there a way to bypass the theory of high-level models and apply similar ideas directly to real systems?

7. ACKNOWLEDGMENTS

We thank Minghong Lin and Adam Wierman for sharing their simulator code with us. We thank John Kubiatowicz, Anthony Joseph, and Charles Reiss for insightful discussions.

8. REFERENCES

[1] Apache Hadoop. `http://hadoop.apache.org/`.

[2] Apache Hadoop Fair Scheduler. `https://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html`.

[3] F. Ahmad, S. Lee, M. Thottethodi, and T. N. Vijaykumar. MapReduce with communication overlap (MaRCO). In *JPDC 2012*, 2012.

[4] 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*, 2012.

[5] 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 *In Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*, pages 1–16, 2010.

[6] T. C. Bressoud and M. A. Kozuch. Cluster fault-tolerance: An experimental evaluation of checkpointing and MapReduce through simulation. In *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, pages 1–10, 2009.

[7] H. Chang, M. Kodialam, R. R. Kompella, T. V. Lakshman, M. Lee, and S. Mukherjee. Scheduling in MapReduce-like systems for fast completion time. In *INFOCOM, 2011 Proceedings IEEE*, 2011.

[8] F. Chen, M. Kodialamy, and T. V. Lakshmany. Joint scheduling of processing and shufSSe phases in MapReduce systems. In *INFOCOM, 2012 Proceedings IEEE*, 2012.

[9] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz. The case for evaluating MapReduce performance using workload suites. In *IEEE MASCOTS 2011*, 2011.

[10] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica. Managing data transfers in computer clusters with Orchestra. In *ACM SIGCOMM*, 2011.

[11] P. Costa, A. Donnelly, A. Rowstron, and G. O'Shea. Camdoop: Exploiting in-network aggregation for big data applications. In *NSDI 2012*, 2012.

[12] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, January 2008.

[13] A. D. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca. Jockey: Guaranteed job latency in data parallel clusters. In *EuroSys '12 Proceedings of the 7th ACM european conference on Computer Systems*, pages 99–112, 2012.

[14] M. Gong and C. Williamson. Simulation evaluation of hybrid SRPT scheduling policies. In *Proceedings of IEEE MASCOTS*, pages 3–13, 2004.

[15] M. Gong and C. Williamson. Revisiting unfairness in web server scheduling. *Computer Networks*, 50(13):2183–2203, 2006.

[16] Z. Guo, G. Fox, and M. Zhou. Investigation of data locality in MapReduce. In *Cluster Computing and the Grid, IEEE International Symposium on*, pages 419–426, 2012.

[17] M. Hammoud and M. F. Sakr. Locality-aware reduce task scheduling for MapReduce. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 570–576, 2011.

[18] 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, 2009.

[19] M. Lin, J. Tan, A. Wierman, and L. Zhang. Joint optimization of overlapping phases in MapReduce. *Performance Evaluation*, 2013.

[20] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica. Sparrow: Distributed, low latency scheduling. In *SOSP 2013*, 2013.

[21] B. Palanisamy, A. Singh, L. Liu, and B. Jain. Purlieus: Locality-aware resource allocation for MapReduce in a cloud. In *Proceedings of the International Conference on Supercomputing*, 2011.

[22] Z. Ren, X. Xu, J. Wan, W. Shi, and M. Zhou. Workload characterization on a production Hadoop cluster: A case study on Taobao. In *Workload Characterization (IISWC), 2012 IEEE International Symposium on*, pages 3–13, 2012.

[23] L. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16(3):687–690, 1968.

[24] A. Shiehzy, S. Kandulaz, A. Greenberg, C. Kim, and B. Saha. Sharing the data center network. In *NSDI 2011*, 2011.

[25] 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, 2012.

[26] A. Verma, L. Cherkasova, and R. H. Campbell. Two sides of a coin: Optimizing the schedule of MapReduce jobs to minimize their makespan and improve cluster performance. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), IEEE 20th International Symposium on*, 2012.

[27] A. Verma, N. Zea, B. Cho, I. Gupta, and R. H. Campbell. Breaking the MapReduce stage barrier. In *Cluster 2010*, 2010.

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

[29] A. Wierman. Fairness and classifications. *ACM SIGMETRICS Performance Evaluation Review*, 34(4):4–12, 2007.

[30] A. Wierman and M. Harchol-Balter. Classifying scheduling policies with respect to unfairness in an M/GI/1. In *Proceedings of ACM SIGMETRICS*, 2003.

[31] A. Wierman and M. Harchol-Balter. Classifying scheduling policies with respect to higher moments of conditional response time. In *Proceedings of ACM SIGMETRICS*, 2005.

[32] J. Wilkes. More Google cluster data. `http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html`, 2011.

[33] M. Zaharia, D. Borthakur, J. Sen 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, Apr 2009.

[34] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica. Improving MapReduce performance in heterogeneous environments. In *OSDI'08 Proceedings of the 8th USENIX conference on Operating systems design and implementation*, pages 29–42, 2008.