On the Latency-Accuracy Tradeoff in Approximate MapReduce Jobs

Juan F. Pérez
Universidad del Rosario
Bogotá, Colombia
juanferna.perez@urosario.edu.co

Robert Birke
IBM Research Zurich
Rüschlikon, Switzerland
bir@zurich.ibm.com

Lydia Y. Chen
IBM Research Zurich
Rüschlikon, Switzerland
yic@zurich.ibm.com

***Abstract*—To ensure the scalability of big data analytics, approximate MapReduce platforms emerge to explicitly trade off accuracy for latency. A key step to determine optimal approximation levels is to capture the latency of big data jobs, which is long deemed challenging due to the complex dependency among data inputs and map/reduce tasks. In this paper, we use matrix analytic methods to derive stochastic models that can predict a wide spectrum of latency metrics, e.g., average, tails, and distributions, for approximate MapReduce jobs that are subject to strategies of input sampling and task dropping. In addition to capturing the dependency among waves of map/reduce tasks, our models incorporate two job scheduling policies, namely, exclusive and overlapping, and two task dropping strategies, namely, early and straggler, enabling us to realistically evaluate the potential performance gains of approximate computing. Our numerical analysis shows that the proposed models can guide big data platforms to determine the optimal approximation strategies and degrees of approximation.**

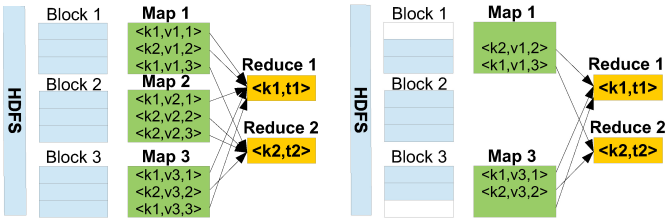
I. Introduction

MapReduce (MR) has become the key programming paradigm for scalable big data analysis through the parallel execution of map and reduce tasks. While the basic concept of MapReduce is simple, its implementation involves many complex steps, starting from the lower level of data storage/access [1], [2] to the higher level of task scheduling [3], [4] and cluster management [5], [6]. Great efforts have been made to optimize the performance of MapReduce jobs with the objective of accelerating the overall job execution times. With the recent trend of running MapReduce jobs in an on-line fashion [7] meeting the response time requirement becomes another critical optimization criterion [8]. To fulfill the ever stringent latency requirements for MapReduce-like jobs in a resource conserving manner, emerging approximate processing platforms, such as BlinkDB [9] and ApproxHadoop [10], advocate to trade off the job accuracy for the job latency, by only processing part of the input data.

The key questions to be answered in approximate MapReduce jobs are where, when and how much input data to drop, given performance targets on accuracy and job latency, and the available resources. A common accuracy metric is the width of the confidence interval for the quantity being computed, whereas the job latency can be the average or tail percentiles of the execution or response times. As MapReduce jobs process input data via map tasks and (a smaller amount of) intermediate data via reduce tasks, reducing the amount of data to process can significantly reduce task and job execution times.

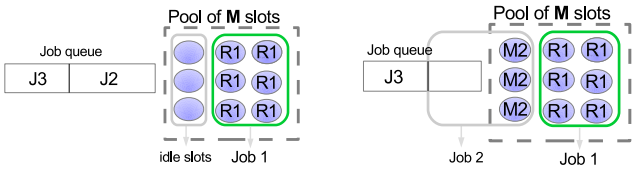
Statistical sampling [11], [12] provides good answers to the question of where and how much data to drop given an accuracy target. In particular, ApproxHadoop applies two stage sampling to determine how much input data should be processed, either by dropping entire map tasks or only a portion of each map task's input data. However, only the average execution time has been explored to find the accuracy-latency optimal tradeoff due to the high complexity in modeling other types of latency measurements, such as tail execution/response times. State-of-the-art latency models [10], [13]–[16] either capture the task execution times in a batch-like setting or the average response times assuming known execution times. To strike a more sophisticated tradeoff between accuracy and latency, models that can precisely capture the distribution of execution and response times under the complex dependencies among data inputs, tasks and job arrivals, are strongly needed.

In this paper, we develop stochastic models, based on matrix analytic techniques, for approximate MapReduce jobs which are executed in an online fashion and whose map tasks and their data inputs are subject to drop decisions. Our contribution lies in that the proposed models incorporate the detailed execution flow of parallel map and reduce tasks, the dynamics of jobs arrivals, and, most importantly, the impact of task dropping and input sampling. We particularly model the early and straggler task dropping strategies, which discard a certain number of tasks either right before the start of or after the completion of fast tasks. We also consider two scheduling policies, namely, exclusive First-Come-First-Serve (FCFS), where the cluster executes one job at a time, and overlapping FCFS, where we partially allow jobs to execute concurrently. Our analysis derives two key latency metrics for interactive MapReduce clusters: the distributions of the job execution times and of the response times. The analysis therefore supports the selection of approximation strategies and the specific task dropping and input sampling levels to achieve average or tail latency targets while fulfilling accuracy requirements. We evaluate the accuracy-latency tradeoff under various system settings, considering among others different data loss levels, task dropping strategies, job scheduling policies and number of map tasks.



(a)original job (b) approximate job

Figure 1: An example of an original MapReduce job (a) with 3 map tasks, each processing 3 data items, and 2 reduce tasks; and an approximate MapReduce job (b) with only 2 map tasks, each processing 2 data items, and 2 reduce tasks.



(a) Exclusive FCFS (b) Overlapping FCFS

Figure 2: An example of MapReduce cluster with 9 slots, serving jobs via two scheduling policies: (a) only one job at the cluster, and (b) map tasks of the next job can start as soon as the first job at the cluster only needs to finish reduce tasks.

II. System

In this section, we first explain the background of MapReduce jobs and their clusters, followed by a highlight on approximate MapReduce jobs.

A. MapReduce Jobs and Clusters

MapReduce is a parallel programming paradigm able to process data at scale, and Hadoop [17] is its best-known open source implementation. A typical MapReduce job needs to process input data and return analysis results via parallel tasks executed in two phases, namely, map and reduce. Since many parameters exist for both map and reduce tasks, we use the subscript m in reference to map tasks and the subscript r in reference to reduce tasks. During the map phase, N_m map tasks, are spawned to process one input block each and produce intermediate results that are stored as key-value pairs in the file system. Afterwards, N_r reduce tasks access and aggregate the intermediate key-value pairs to produce the final result. Reduce tasks can either start after the completion of a certain percentage of map tasks or after the last map task. Fig. 1 (a) illustrates an example of a MapReduce job.

Users submit job requests to the MapReduce cluster at rate λ, providing configuration parameters to split the input data and to set up the map/reduce tasks. A MapReduce cluster consists of a total of C computing slots. Job tasks are scheduled onto slots, where the typical practice is to differentiate between slots for map and reduce tasks. However, recently emerging platforms, such as Spark [18], have a single type of slots for both task types, and we therefore focus on this configuration. When the numbers of map or reduce tasks is greater than the number of slots, e.g., $N_m \geq C$, tasks are executed in multiple waves.

The default scheduling policy at a MapReduce cluster is first-come-first-serve (FCFS), where jobs are processed sequentially depending on their arrival times, as shown in Fig. 2(a). Note that a job is only allowed to start execution once the job in front has completely finished its map and reduce phases. It has therefore been observed [13] that overlapping job executions can significantly improve the performance. This is illustrated in Fig. 2(b), where the job at the head of the queue can start its map phase as soon as the job in front starts to free up slots, i.e., when it has no tasks queued and at most $C-1$ tasks in execution. Other popular schedulers are capacity and fair [15], which aim to improve the potentially long queuing times of small jobs that wait behind large jobs under the FCFS policy.

B. Approximate MR Jobs

To accelerate the execution time of MapReduce jobs, the very recent platform ApproxHadoop [10], implements a two-stage sampling mechanism that enables the tradeoff between accuracy and execution time. Particularly, ApproxHadoop uses map task dropping and input sampling to reduce the amount of data processed while adhering to a given accuracy target, e.g., the confidence interval of the job analysis should be within certain thresholds. Map tasks can be either dropped before execution or after being identified as stragglers, i.e. exceptionally long running tasks.

Fig. 1(b) illustrates the task dropping mechanism, as here we drop one task out of three, such that the resulting map task dropping ratio $1-\theta_m$ (number of dropped tasks divided by the original number of tasks) is $1/3$. Similarly, in Fig. 1(b) we sample two out of the three data items in each of the remaining map tasks, which makes the input sampling ratio η_m (percentage of data items processed per block) equal to 66.67%. The task dropping and input sampling ratios, θ and η, are thus key tuning parameters in ApproxHadoop which determine the effective number of tasks executed $\overline{N} = \lceil N\theta \rceil$ and the effective processing rate $\overline{\mu} = \mu/\eta$ at each phase.

ApproxHadoop incorporates a simple latency model via off-line profiling of a first wave of tasks, assumes that the variation of execution times across parallel tasks is very small and that the job execution time is linear in the number of map tasks and block sizes. The accuracy model is based on the estimated confidence interval computed as

$$t_{\overline{N}_m-1,1-\alpha/2}\sqrt{N_m\left(\left(\frac{1}{\theta_m}-1\right)s_u^2+\left(\frac{1}{\eta_m}-1\right)\overline{s}_i^2\right)}, \quad (1)$$

where $t_{n,p}$ is the p-th percentile of the Student's t distribution with n degrees of freedom, s_u^2 is the inter-task variance and $\overline{s}_i^2$ is the mean intra-task variance. We will adopt this error function in the remaining of the paper, although the latency model we introduce can be used in combination with any error function $g(\theta_m, \eta_m)$ for the map phase or with more sophisticated error functions that consider both map and reduce phases.

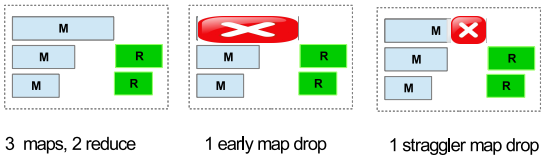


Figure 3: Illustration of early map task drop and straggler task drop for a job of 3 map tasks and 2 reduce tasks.

Note that on the one hand dropping one map task can not only save the time to process a data block, but also the overhead of initializing a map task. On the other hand, dropping few parallel map tasks may not result into a significant reduction in job execution time, since this highly depends on the longest execution times across all parallel tasks, the execution of tasks in multiple waves given the number slots, and the dependency between map and reduce phases.

III. Reference System Model

We propose the following reference system model of approximate MapReduce jobs. Our objective is to derive means and percentiles of the execution and response times to guide the selection of task dropping and input sampling ratios.

A MapReduce cluster is composed of a job queue and a pool of C homogeneous slots that can process one task at a time. Job arrivals follow a Markovian Arrival Process (MAP) [19] with parameters $(d, \boldsymbol{D}_0, \boldsymbol{D}_1)$, where d is the number of arrival phases. Each job consists of N_m map tasks and N_r reduce tasks, which have exponentially distributed processing times with mean $1/\mu_m$ and $1/\mu_r$, respectively. Upon arrival, jobs are served in FCFS order, with either exclusive or overlapping scheduling, as described in the previous section and illustrated in Fig. 2. Jobs are subject to task dropping and input data sampling, and these mechanisms operate at both map and reduce phases. For the map and reduce phases, the task dropping ratios are $1-\theta_m$ and $1-\theta_r$, while the input sampling ratios are η_m and η_r, respectively. Finally, we focus on the case where all map tasks complete before reduce tasks start.

Early v.s. straggler dropping We consider two types of task droppings: early dropping and straggler dropping. In early dropping $\overline{N} = N\theta$ tasks are scheduled and executed whereas the remaining $N - \overline{N}$ tasks are never scheduled. Instead, in straggler dropping all N tasks are scheduled and executed until the first $\overline{N}$ map tasks complete, moment at which the remaining $N - \overline{N}$ tasks are terminated. Fig. 3 illustrates an example of how to drop one map task out of three using early and straggler dropping.

IV. Approximate MR with Exclusive Scheduling

In this section we introduce a *latency* model for approximate MapReduce jobs, following the reference system model introduced in Section III and using matrix analytic techniques. We consider both early and straggler dropping strategies. Since the analysis of both cases is similar, we describe in detail the early dropping case, while for straggler dropping we focus on the main differences.

A. Early Dropping

To model the approximate MR execution we describe the job execution time as a phase-type (PH) distribution, where phase n indicates that the job still has n tasks to complete, $1 \leq n \leq \overline{N}_m + \overline{N}_r$. A job thus starts service in phase $\overline{N}_m + \overline{N}_r$ and moves from phase n to phase $n-1$ with rate $f(n)$ given by

$$f(n) = \begin{cases} C\overline{\mu}_m, & \overline{N}_r + C < n \leq \overline{N}_r + \overline{N}_m, \\ (n - \overline{N}_r)\overline{\mu}_m, & \overline{N}_r < n \leq \overline{N}_r + C, \\ C\overline{\mu}_r, & C < n \leq \overline{N}_r, \\ n\overline{\mu}_r, & 0 < n \leq C. \end{cases}$$

This expression captures the availability of C slots, which limit the number of tasks executing in parallel, as well as the requirement that all map tasks must complete before the start of the reduce phase. To put this description in standard PH notation [19] we define the state space of this distribution as the set $\mathcal{S} = \{\overline{N}_m + \overline{N}_r, \overline{N}_m + \overline{N}_r - 1, \ldots, 1\}$, in descending order. Thus the initial probability vector $\boldsymbol{\alpha}$ has a single non-zero entry, namely $\alpha_{\overline{N}_m + \overline{N}_r} = 1$, indicating that the job starts service in phase $\overline{N}_m + \overline{N}_r$ with probability one. The PH sub-generator $\boldsymbol{G}$ is a bidiagonal matrix with non-zero entries $G_{n,n-1} = f(n)$ and $G_{n,n} = -f(n)$, according to the evolution from phase n to phase $n-1$ with rate $f(n)$. The mean *job* service time is $\tau = \boldsymbol{\alpha}(-\boldsymbol{G})^{-1}\mathbf{1}$, where $\mathbf{1}$ is a column vector of ones.

With the above definitions we can obtain the response-time distribution as in [20], defining an age process $X(t)$ that keeps track of the age of the job in service, and a phase process $J(t) = (A(t), S(t))$ that keeps track of the arrival process phase $A(t)$ and the service process phase $S(t)$. Since the job arrival rate is $\lambda = \boldsymbol{\gamma}\boldsymbol{D}_1\mathbf{1}$, where $\boldsymbol{\gamma}$ is the stationary distribution of the Markov chain with generator $\boldsymbol{D}_0 + \boldsymbol{D}_1$, this system is stable whenever $\rho = \lambda\tau < 1$. If the system is stable, the stationary distribution of the process $(X(t), J(t))$ has a matrix-exponential representation, that is, it can be expressed as $\boldsymbol{\pi}(x) = \boldsymbol{\pi}_0 \exp(\boldsymbol{T}x)$ for $x > 0$. Here $\boldsymbol{\pi}_0$ is the stationary distribution of the phase $J(t)$ at the start of a busy period. Since a busy period starts with the arrival of a new job, which starts service according to $\boldsymbol{\alpha}$, $\boldsymbol{\pi}_0$ is given by $\overline{\boldsymbol{\gamma}} \otimes \boldsymbol{\alpha}$, where $\otimes$ denotes the Kronecker product, and $\overline{\boldsymbol{\gamma}}$ is the stationary distribution of the arrival process phase *just after an arrival*, thus it is the stationary distribution of the Markov chain (MC) with transition matrix $-\boldsymbol{D}_0^{-1}\boldsymbol{D}_1$. Furthermore, we can obtain the matrix $\boldsymbol{T}$ by solving the equation [20]

$$\boldsymbol{T} = \boldsymbol{I}_d \otimes \boldsymbol{G} + \int_0^\infty \exp(\boldsymbol{T}x) \left(\exp(\boldsymbol{D}_0 x)\boldsymbol{D}_1 \otimes \tilde{\boldsymbol{G}}\right) dx, \quad (2)$$

where $\boldsymbol{I}$ is an identity matrix of size d and $\tilde{\boldsymbol{G}} = (-\boldsymbol{G}\mathbf{1})\boldsymbol{\alpha}$ is a matrix that captures the service phase transition when a job completes service, with rates in vector $(-\boldsymbol{G}\mathbf{1})$, allowing a new job to start service according to $\boldsymbol{\alpha}$. Equation (2) can be solved efficiently by exploiting the structure of the matrices $\boldsymbol{G}$ and $\tilde{\boldsymbol{G}}$ as in [21]. In fact, the size $m = d\,(\overline{N}_m + \overline{N}_r)$ of this system makes it very scalable and suitable to consider jobs with hundreds or thousands of tasks.

From $\boldsymbol{\pi}_0$ and $\boldsymbol{T}$ we can obtain the probability of waiting ϕ and the PH representation of the waiting time distribution $(\boldsymbol{\beta}_w, \boldsymbol{B}_w)$, though we avoid the details as these are provided in Section V for the more general case with overlapping jobs. Finally, we obtain the PH representation $(\boldsymbol{\beta}_r, \boldsymbol{B}_r)$ of the response time distribution with parameters

$$\boldsymbol{\beta}_r = [\phi\boldsymbol{\beta}_w \quad (1-\phi)\boldsymbol{\alpha}], \quad \boldsymbol{B}_r = \begin{bmatrix} \boldsymbol{B}_w & (-\boldsymbol{B}_w\mathbf{1})\boldsymbol{\alpha} \\ \mathbf{0} & \boldsymbol{G} \end{bmatrix},$$

where we sum waiting and service times for those jobs that wait, with probability ϕ, and consider only the service time for those jobs that do not wait. This representation thus provides us with latency moments and percentiles for the approximate MR execution with early task dropping.

B. Straggler Dropping

For straggler dropping we can also represent the *job* service time as a PH distribution, but in this case the phase n keeps track of the number of tasks scheduled but not completed. Thus, the state space is given by $\mathcal{S} = \{N_m + N_r, N_m + N_r - 1, \ldots, N_m - \overline{N}_m + 1 + N_r, N_r, N_r - 1, \ldots, N_r - \overline{N}_r + 1\}$, reflecting the fact that initially all $N_m + N_r$ tasks are scheduled but we only require $\overline{N}_m$ and $\overline{N}_r$ to complete for the map and task phases, respectively. The number of service phases is thus $\overline{N}_m + \overline{N}_r$, and we can write the transition rate in phase n as

$$f(n) = \begin{cases} \min\{n - N_r, C\}\overline{\mu}_m, & N_r + N_m - \overline{N}_m + 1 \leq n \\ & \leq N_r + N_m, \\ \min\{n, C\}\overline{\mu}_r, & N_r - \overline{N}_r + 1 \leq n \leq N_r. \end{cases}$$

The PH representation of the job service has initial probability vector $\boldsymbol{\alpha}$ with $\alpha_{\overline{N}_m + \overline{N}_r} = 1$. Similar to the early dropping case, the PH sub-generator matrix $\boldsymbol{G}$ has non-zero off-diagonal entries $G_{n,n-1} = f(n)$ for all $n \in \mathcal{S}$, except for phase $n = N_r + N_m - \overline{N}_m + 1$, where the transition occurs with rate $f(n)$ to phase N_r rather than to phase $n-1$, as this is the last phase of the map stage. With this PH representation of the job service time we can proceed as in the previous section to find the response time distribution.

Remark 1. *Here and in the previous section we assume that either early or straggler dropping are implemented in both map and reduce phases. These models can be easily combined to have, for instance, early dropping in the map phase and straggler dropping in the reduce phase.*

V. Introducing Job Overlap

The model in Section IV captures well clusters where a single job is processed at any point in time. This type of operation means that the job in front of the queue must wait until all the reduce tasks of the job currently in service finish execution. Thus, there will be idle slots from the moment the job in service has $C - 1$ reduce tasks remaining until it completes the reduce phase. Allowing the job in front of the queue to start its map phase as soon as there are idle slots can therefore reduce the job response times. In this section we introduce a model to assess the potential benefits of allowing jobs to overlap.

A. The Waiting Times

Whereas we want to introduce the flexibility of allowing a job to start service if there are idle slots, allowing jobs to enter in a FCFS manner without any constraint would require keeping track of the exact phase of each of the up to C jobs in service. Such a model would suffer from state-space explosion, limiting its usefulness. We therefore opt for an intermediate solution where we allow the job at the head of the queue to start service as long as there are idle slots, but we do not allow this job to continue to the reduce phase until the job in front completes its own reduce phase. This means that there can be up to two jobs in execution, where the youngest one is not allowed to move on to the reduce phase until the oldest one completes its own reduce phase.

Remark 2. *In the following we assume that early task dropping is adopted and that $\overline{N}_r \geq C$, such that a job starts its reduce phase by using all slots. Modifications to cover the $\overline{N}_r < C$ and late dropping cases can be defined similarly, but we omit the details in the interest of clarity and space.*

To model this setup we define a bi-variate Markov process $(X(t), J(t))$, as in the previous section, but now the age $X(t)$ keeps track of the age of the *youngest* job in service, whereas the phase $J(t) = (A(t), M(t), R(t), Y(t))$ keeps track of the phase of the arrival process $A(t)$, the number of map $M(t)$ and reduce $R(t)$ tasks of the oldest job in service, and the number of remaining map tasks $Y(t)$ of the second (youngest) job in service, if any. We describe the service phase $(M(t), R(t), Y(t))$ with a tuple (n_m, n_r, n_y), and distinguish three steps of execution:

- M $(n_m, \overline{N}_r, -)$, where there is one job in execution in the map phase with $1 \leq n_m \leq \overline{N}_m$ map tasks to complete.
- R $(0, n_r, -)$, where there is one job in execution with $C \leq n_r \leq \overline{N}_r$ reduce tasks to complete. A second job is not allowed because the job in service still requires at least all the C servers.
- R/M $(0, n_r, n_y)$, where there are two jobs in execution: the oldest still has $1 \leq n_r \leq C-1$ reduce tasks to complete; whereas the youngest has $0 \leq n_y \leq \overline{N}_m$ map tasks to complete.

The service phase space size is thus $m_s = \overline{N}_m C + \overline{N}_r - N_m$.

In this model, the servers will be blocked in all service phases of the form $(0, n_r, 0)$ with $n_r > 0$, which indicate the youngest job already finished all its map tasks, while the oldest job still has n_r reduce tasks to process. However, if the number of map tasks is at least C, and map and reduce tasks have similar processing times, such blocking states will have a small probability since they require that $C - 1$ reduce tasks already in service take longer to process than the $\overline{N}_m \geq C$ map tasks that only have $C - n_r$ servers to execute. This is further exacerbated if map tasks require several waves to execute (such that $\overline{N}_m$ is many times larger than C) and if map processing times are larger than reduce processing times, as is common in many MR applications.

Table I: Transition rates for the $\boldsymbol{Q}$ and $\boldsymbol{Z}$ matrices in the overlapping model.

Matrix	Step	Phase	Dest	Rate	Condition
$\boldsymbol{Q}$	M	$(n_m, \overline{N}_r, -)$	$(n_m - 1, \overline{N}_r, -)$	$\min\{n_m, C\}\mu_m$	$1 < n_m \leq \overline{N}_m$
	$M \to R$	$(1, \overline{N}_r, -)$	$(-, \overline{N}_r, -)$	μ_m	
	R	$(0, n_r, -)$	$(0, n_r - 1, -)$	$C\mu_r$	$C + 1 \leq n_r \leq \overline{N}_r$
	R/M	$(-, n_r, n_y)$	$(-, n_r - 1, n_y)$	$n_r\mu_r$	$1 < n_r \leq C - 1$
		$(-, n_r, n_y)$	$(-, n_r, n_y - 1)$	$\min\{n_y, C - n_r\}\mu_m$	$1 \leq n_r \leq C - 1$, $n_y \geq 1$
	$R/M \to M$	$(-, 1, n_y)$	$(n_y, \overline{N}_r, -)$	μ_r	$n_y \geq 1$
	$R/M \to R$	$(-, 1, 0)$	$(-, \overline{N}_r, -)$	μ_r	
$\boldsymbol{Z}$	$R \to R/M$	$(-, C, -)$	$(-, C - 1, \overline{N}_m)$	$C\mu_r$	

At this point we should note two key differences between the overlapping case and its non-overlapping counterpart in Section IV. First, with overlapping a job can start service in the "standard" condition $(\overline{N}_m, \overline{N}_r, -)$ if it finds the cluster idle, otherwise it starts service in an "overlapping" state $(0, n_r, \overline{N}_m)$. Second, due to the overlapping a job service time will be affected by the evolution of the job in front, as its own tasks can only use the slots freed by the job in front. As a result, the *job* service time cannot be determined in advance and we can only obtain its *stationary* version after finding the system steady state.

To obtain the waiting-time distribution we define the matrices $\boldsymbol{Q}$ and $\boldsymbol{Z}$, which hold transition rates among service phases without and with a new job service start, respectively. Table I summarizes the non-zero entries of these matrices. Here the first row describes transitions within step M, and the second row the transition from step M to step R when the last map task completes. Next, the third row shows transition within the R step, with the transition from this step to step R/M being relegated to the first row for matrix $\boldsymbol{Z}$ as this transition causes a new job to start service. Within step R/M, rows 4 and 5 capture service completions of reduce and map tasks, respectively. Here there are n_r slots processing reduce tasks and $C - n_r$ available to the map tasks of the youngest job. Finally, a transition from step R/M to step M occurs when the last reduce task terminates, and there is at least one map task of the youngest job, which continues its map phase. However, if the youngest job has already completed all its map tasks, when the job in front completes its last reduce task the youngest job starts its reduce phase.

With the above definitions we can proceed as in the non-overlapping case to find the stationary distribution of the process $(X(t), J(t))$, which has a matrix-exponential representation [22] $\boldsymbol{\pi}(x) = \boldsymbol{\pi}_0 \exp(\boldsymbol{T}x)$ for $x > 0$. In this case the matrix $\boldsymbol{T}$ solves the equation

$$\boldsymbol{T} = \boldsymbol{I}_d \otimes \boldsymbol{Q} + \int_0^\infty \exp(\boldsymbol{T}x) \left(\exp(\boldsymbol{D}_0 x)\boldsymbol{D}_1 \otimes \boldsymbol{Z}\right) dx, \quad (3)$$

where the first and second terms consider transitions without and with new jobs starting service, respectively.

Different from the non-overlapping case, here the system has a more complex boundary behavior as a job that finds up to $C - 1$ slots busy with reduce tasks can start service immediately. As a result, we label as *not-full* a period where this condition holds, and its service phases are given by the set $\mathcal{S}_0 = \{0, 1, \ldots, C - 1\}$, of size $m_0 = C$, indicating the number of slots busy with reduce tasks. We also label *full* the remaining periods, where at least C servers are busy or the current job in service is in the map phase. The process $(X(t), J(t))$ thus describes well the evolution during full periods while during not-full periods the system evolves on the set $\mathcal{S}_0$. Starting from a full period, the system may jump to a not-full period if the service phase is $(-, C, -)$ and the next reduce task completes before the next arrival. We thus define the $m_s \times m_0$ matrix $\boldsymbol{Z}^*$, which has a single non-zero entry equal to $M\mu_r$ corresponding to a transition from phase $(-, C, -)$ to phase $C - 1$.

During a not-full period the service phase evolves according to the $m_0 \times m_0$ matrix $\boldsymbol{Q}^0$, which has non-zero entries $Q^0_{i,i-1} = i\mu_r$ for $i \in \mathcal{S}_0$, indicating the completion of the remaining reduce tasks. A not-full period terminates with an arrival as it triggers a transition to the full period, and the service phase transitions according to the $m_0 \times m_s$ stochastic matrix $\boldsymbol{Z}^0$. Its non-zero entries mark a transition with probability 1 from phase 0 to phase $(\overline{N}_m, \overline{N}_r, -)$ if the system is idle, and from any other phase i in $\mathcal{S}_0$ to phase $(-, i, \overline{N}_m)$ as the new arrival starts its map phase.

Now we can find $\boldsymbol{\pi_0}$ as the solution to the equation

$$\boldsymbol{\pi}_0 = \boldsymbol{\pi}_0 \int_0^\infty \exp(\boldsymbol{T}x) \left(\exp(\boldsymbol{D}_0 x) \otimes \boldsymbol{Z}^*\right) dx \left(-(\boldsymbol{D}_0 \otimes \boldsymbol{I}_{m_0} + \boldsymbol{I}_d \otimes \boldsymbol{Q}^0)\right)^{-1} \left(\boldsymbol{D}_1 \otimes \boldsymbol{Z}^0\right),$$

which describes how the system reaches age x with density $\boldsymbol{\pi}_0 \exp(\boldsymbol{T}x)$, from which it may jump to a not-full period if no arrival occurs before x time units, and starts the not-full period according to matrix $\boldsymbol{Z}^*$. Next, the evolution in the not-full period occurs according to matrices $\boldsymbol{D}_0$ and $\boldsymbol{Q}^0$ for the arrival and service phases, and finally a new full period starts when an arrival occurs with rates in $\boldsymbol{D}_1$ and the service phase jumps according to matrix $\boldsymbol{Z}^0$. The above equation can be solved by first obtaining, as a by-product of solving (3), the matrix $\boldsymbol{L}$ given by

$$\boldsymbol{L} = \int_0^\infty \exp(\boldsymbol{T}x) \left(\exp(\boldsymbol{D}_0 x) \otimes \boldsymbol{I}_{m_s}\right) dx,$$

which we can use to find $\boldsymbol{\pi}_0$ solving the linear system

$$\boldsymbol{\pi}_0 = \boldsymbol{\pi}_0 \boldsymbol{L} (\boldsymbol{I}_d \otimes \boldsymbol{Z}^*) \left(-(\boldsymbol{D}_0 \otimes \boldsymbol{I}_{m_0} + \boldsymbol{I}_d \otimes \boldsymbol{Q}^0)\right)^{-1} \left(\boldsymbol{D}_1 \otimes \boldsymbol{Z}^1\right).$$

From $\boldsymbol{\pi}_0$ and $\boldsymbol{T}$ we can obtain the probability of waiting ϕ and the PH representation of the waiting time distribution $(\boldsymbol{\beta}_w, \boldsymbol{B}_w)$ as in [22].

B. The Job Service Time

To describe the job service time we setup a PH representation $(\boldsymbol{\beta}_s, \boldsymbol{B}_s)$ composed of three stages, similar to those defined in the previous section: R/M, where the job is in the map phase, but sharing resource with the job in front; M, where the job is on its own during its map phase; and R^*, where the job is in its reduce phase, either sharing resources or not. Whereas stages R/M and M are the same as those defined in the previous section, the R^* stage is different as we do not care if another job is in service since this has no influence on the evolution of the job already in the reduce phase. The stage R^* thus has phases $\{\overline{N}_r, \ldots, 1\}$. Next, we define the service sub-generator as

$$\boldsymbol{B}_s = \begin{bmatrix} \boldsymbol{Q}^{R/M,R/M} & \boldsymbol{Q}^{R/M,M} & \boldsymbol{Q}^{R/M,R^*} \\ \mathbf{0} & \boldsymbol{Q}^{M,M} & \boldsymbol{Q}^{M,R^*} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{B}^{R^*,R^*} \end{bmatrix}, \quad (4)$$

where we use the sub-matrices of the matrix $\boldsymbol{Q}$ defined in Table I. The transitions into stage R^* are defined as those into stage R for matrix $\boldsymbol{Q}$, but keeping track of the reduce phase n_r only. Further, the non-zero entries of matrix $\boldsymbol{B}^{R^*,R^*}$ show transitions from phase i to phase $i-1$ with rate $\min\{i, C\}\mu_r$ for $1 \leq i \leq \overline{N}_r$, indicating the successive completion of reduce tasks.

The initial job service phase distribution $\boldsymbol{\beta_s}$ can be obtained by partitioning it according to the three service steps

$$\boldsymbol{\beta}_s = \begin{bmatrix} \boldsymbol{\beta}_s^{R/M} & \boldsymbol{\beta}_s^{M} & \boldsymbol{\beta}_s^{R^*} \end{bmatrix}. \quad (5)$$

Since a job cannot start service in the R^* phase, $\boldsymbol{\beta}_s^{R^*}$ is a zero vector. To find the other sub-vectors of $\boldsymbol{\beta}_s$ we write $\boldsymbol{\pi}_0$ as

$$\boldsymbol{\pi}_0 = \begin{bmatrix} \boldsymbol{\pi}_0^{M} & \boldsymbol{\pi}_0^{R} & \boldsymbol{\pi}_0^{R/M} \end{bmatrix},$$

according to the steps defined in the previous section. Thus a job that starts service in an idle system begins in stage M with phase according to

$$\boldsymbol{\beta}_s^{M} = (1-\phi)\boldsymbol{\pi}_0^{M},$$

where we recall ϕ is the probability of waiting. Instead, a job that starts service in stage R/M can start without waiting, according to vector $\boldsymbol{\pi}_0^{R/M}$. Alternatively, it could start after waiting according to the distribution of the phase after a downward jump $\boldsymbol{\kappa} = -\boldsymbol{\pi}_0 \boldsymbol{T}^{-1} \boldsymbol{L}(\boldsymbol{D}_1 \otimes \boldsymbol{Z})$. Partitioning this vector according to the three service stages we obtain $\boldsymbol{\kappa}^{R/M}$ corresponding to stage R/M. Thus we have

$$\boldsymbol{\beta}_s^{R/M} = \phi \boldsymbol{\kappa}^{R/M} + (1-\phi)\boldsymbol{\pi}_0^{R/M}.$$

We summarize the discussion in the following result.

Lemma V.1. *The job service time distribution has PH representation $(\boldsymbol{\beta}_s, \boldsymbol{B}_s)$, where $\boldsymbol{B}_s$ is given by* (4) *and $\boldsymbol{\beta}_s$ by* (5)*.*

C. The Response Time

To obtain a PH representation $(\boldsymbol{\beta}_r, \boldsymbol{B}_r)$ of the response time distribution we define the vectors $\boldsymbol{\beta}_{s,\text{wait}}$ and $\boldsymbol{\beta}_{s,\text{non-wait}}$, which hold the probability distribution of the initial service for jobs that wait and that do not wait, respectively. Partitioning these vectors according to the three service steps, they are given by

$$\boldsymbol{\beta}_{s,\text{wait}} = \begin{bmatrix} \boldsymbol{\kappa}^{R/M} & 0 & 0 \end{bmatrix}, \quad \boldsymbol{\beta}_{s,\text{non-wait}} = \begin{bmatrix} \boldsymbol{\pi}_0^{R/M} & \boldsymbol{\pi}_0^{M} & 0 \end{bmatrix},$$

since jobs that wait start in step R/M according to vector $\boldsymbol{\kappa}$, whereas jobs that do not wait start in steps R/M or M according to vector $\boldsymbol{\pi}_0$. Note that these two vectors are stochastic. Now we obtain the PH representation $(\boldsymbol{\beta}_r, \boldsymbol{B}_r)$ as

$$\boldsymbol{\beta}_r = \begin{bmatrix} \boldsymbol{\beta}_w & (1-\phi)\boldsymbol{\beta}_{s,\text{non-wait}} \end{bmatrix}, \; \boldsymbol{B}_r = \begin{bmatrix} \boldsymbol{B}_w & (-\boldsymbol{B}_w \mathbf{1})\boldsymbol{\beta}_{s,\text{wait}} \\ 0 & \boldsymbol{B}_s \end{bmatrix},$$

where the first block of phases covers the evolution of jobs as they wait, while the second block considers the job service.

VI. Evaluation

We demonstrate how the derived analysis can guide the design of approximate MapReduce mechanisms under a large space of system parameters, scheduling policies, and choices of approximate strategies. We particularly consider the exclusive and overlapping FCFS scheduling policies described in Section II. In addition to evaluating the latency-accuracy tradeoff in both systems, we study the impact of the number of tasks for the exclusive FCFS system, and investigate the performance improvement introduced by the overlapping execution of approximate MapReduce jobs. We verify the correctness of the stochastic analysis by simulation, but in the interest of space we omit this comparison.

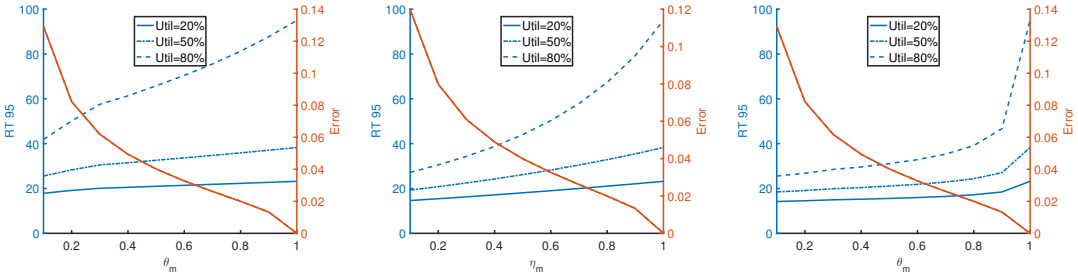
A. System Parameterizations

We first list down the system parameters and their ranges used in the remainder of this section. The number of slots in the MapReduce cluster is $C = [50, 20]$. Jobs consist of $N_m = 2C$ map tasks and $N_r = C$ reduce tasks, except for the scenario where we study the impact of the number of tasks/waves. The mean task execution times are $1/\mu_m = 1$ and $1/\mu_n = .5$, respectively, when no input sampling policy is applied. We compute the accuracy as in Eq. (1), assuming the inter task and the mean intra task standard deviations are $s_u^2 = 0.1$ and $\bar{s} = 0.1$, respectively.

As for the metrics of merit, we report the 95^{th} ($RT95$) and the 99^{th} percentile ($RT99$) of the job response times, and the error introduced by the map task dropping ratio $(1-\theta_m)$ and the map input sampling ratio (η_m). We note that as we consider jobs that have a large number of map tasks, we focus on map tasks and leave exploration on the reduce task as future work. Due to the large combination of system parameters and metrics of merits, we focus on a subset of results to demonstrate the effectiveness of our analysis in guiding the choice of θ_m and η_m against different system operation points.

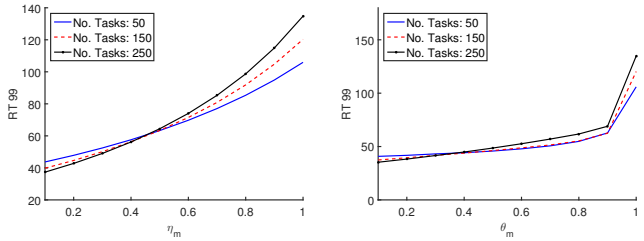
B. Exclusive System

1) How to Drop?: To strike the tradeoff between latency and accuracy, the first question to answer is where and how much to drop. In Fig. 4, we depict the relative error [%] and 95^{th} percentile response time, $RT95$, by applying three approximation strategies, namely, (a) early task dropping, (b) input sampling and (c) straggler dropping. We only apply one policy at a time. For instance, when we drop map tasks, we process all of their data without sampling. To ease the comparison between policies, the values of the x-axes represent the amount of tasks or input data processed, e.g.,



(a) Early dropping (b) Input sampling (c) Straggler dropping

Figure 4: Basic system: accuracy and the 95^{th} percentile response time applying one approximation policy at a time: (a) early task dropping, (b) input sampling, and (c) straggler dropping.



(a) Input sampling (b) Straggler dropping

Figure 5: Impact of number map tasks/waves on effectiveness of straggler task dropping and input sampling.

the value 0.6 shows that only 60% of the number of tasks or input data is processed. We consider three different arrival rates, resulting in utilizations of 20%, 50%, and 80% of the baseline system without any approximation strategy, for which the $RT95$ is around 115, 45, and 28 time units, respectively.

After applying approximate strategies, the $RT95$ decreases monotonically with respect to θ and η and the latency improvement is particularly visible for the high utilization baseline, as observed from the steeper slope for the case of 80% utilization. For all three utilization curves, straggler dropping is best in reducing the $RT95$, followed by input sampling, while early task dropping is last. In other words, given the same amount of data to process, straggler dropping can achieve the lowest latency, particularly for the scenario with high utilization. Another observation worth mentioning is the second order effect of the approximation strategy on $RT95$. While input sampling and straggler dropping result into convex like latency curves, early task dropping shows a concave shape. We reason that straggler dropping benefits greatly from even a little room for approximation, say $1-\theta=x\%$, as it avoids executing the $x\%$ longest tasks. Input sampling also benefits substantially as it reduces the execution of *all* tasks, including the longest, by $x\%$. Instead, early dropping avoids executing $x\%$ tasks but without any guarantee that these are the longest ones. As a result, the latency curves under straggler dropping and input sampling decrease quickly as θ and η decrease, but they decrease much more slowly under early dropping.

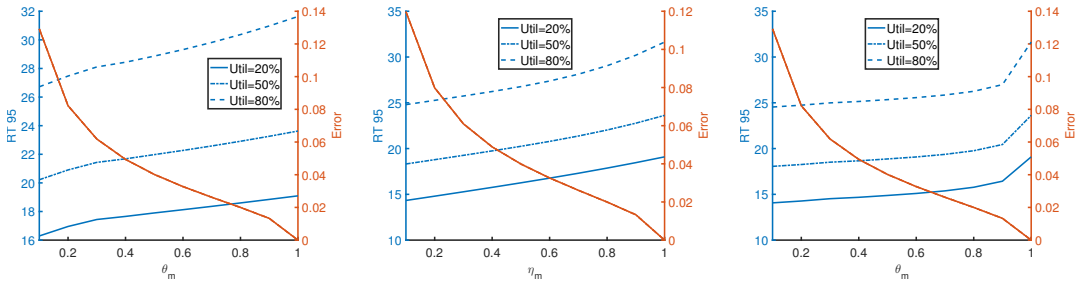
Looking at the accuracy in Fig. 4, one can see that the relative error decreases in θ_m and η_m without any surprise. For the particular inter- and intra- task standard deviation analysis considered here, the resulting accuracy loss is only slightly better under input sampling than under early or straggler task dropping. AS such, for a given relative error target the strategy that results in the minimum $RT95$ is straggler dropping. For example, to bound the relative error within 0.06, the straggler dropping strategy can drop up to 70% of tasks and result into an $RT95$ around 28, while the early dropping and input sampling can only achieve 34 and 56, respectively.

2) Impact of Job Sizes: We now study how the different approximation strategies perform for different job sizes, i.e., for number of map tasks of $[50, 150, 250]$. As the number of slots is 50, the resulting numbers of map waves are $[1, 3, 5]$. We focus on the comparison between straggler dropping and input sampling under the scenarios where the base utilization without any dropping/sampling strategy is 80%. Fig. 5 depicts $RT99$ under different θ_m, η_m, and number of waves.

For big jobs, say having five waves, the latency reduction achieved by either straggler dropping or input sampling is more visible than for small jobs. As shown in Fig. 5, $RT99$ of five-wave jobs can drop from 135 to 35 and 36, respectively, whereas $RT99$ of single-wave jobs drops from 106 to 40 and 45. As a result, big jobs benefit more strongly from approximation strategies, as they start from high response times, which improve with increasing approximation. Until $\eta_m < 0.45$ and $\theta_m < 0.35$, big jobs outperform small jobs. Moreover, similar to our previous findings, straggler dropping is more effective in reducing the latency, as shown by the steep descent around $\theta_m = 0.9$, whereas input sampling has a smoother effect on the latency, across all job sizes.

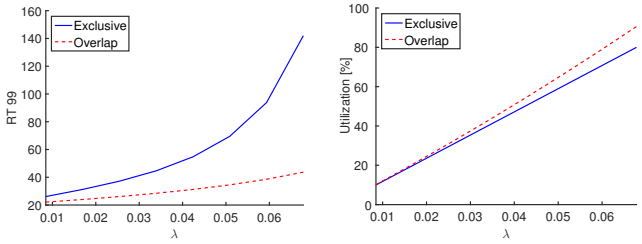
C. Overlapping Systems

Fig. 6 summarizes the accuracy and latency tradeoff for the overlapping system. To ease the comparison with the simple FCFS system, the configurations used here are identical to the ones used in Fig. 4. Clearly, reducing the amount of data processed either through task dropping or input sampling indeed decreases the response times. The reduction of response times shows a linear trend for early dropping and input sampling, whereas straggler dropping shows a steep descent at the beginning and then flattens out. When the reduction is around 90%, i.e., $\eta_m = 0.1$ and $\theta_m = 0.1$, the improvement



(a) Early dropping (b) Input sampling (c) Straggler dropping

Figure 6: Overlap systems: accuracy and the 95^{th} percentile response times applying one approximation policies at a time: (a) early task dropping, (b) input sampling and (c) straggler dropping.



(a) Response times (b) Utilization

Figure 7: Performance difference between overlapping and simple FCFS systems.

in response times is around 25% for all three strategies. Compared to the basic FCFS system, approximate strategies show here lower reductions in response times.

One can also use Fig. 6 to answer the question: in order to bound the relative error within 0.06, which strategy can achieve the minimum response times when the baseline system is roughly 80% utilized? As all three strategies have similar accuracy curves with respect to η_m or θ_m, they can roughly drop up to 70% of tasks and still achieve the error target. Straggler dropping results in the smallest $RT95$, around 18.5, while early dropping and input sampling can only achieve 19.5 and 22, respectively. Our stochastic models therefore serve as an efficient mean to explore the configuration space when searching for the subtle tradeoff between latency and accuracy.

Finally, we present the performance gains achieved by the overlapping system. We particularly consider a cluster with 20 slots and job arrival rate ranging from $\lambda = 0.01$ to 0.07. We do not apply any approximation strategy, keeping the dropping and sampling ratios at one. Fig. 7 presents the response times and cluster utilization for the exclusive and overlapping scheduling policies. We observe that the overlapping system offers much shorter response times than the exclusive system, thanks to the reduction in queueing time achieved by allowing the job at the head of the queue to start as soon as the job in front frees up slots. In fact, this reduction in queueing times compensates the increase in execution time caused by the fact that the job that starts processing can only use the slots made available by the job in front. Another obvious benefit of overlapping jobs is to avoid idle slots, which do not serve any request while other jobs still wait in the queue. In Fig. 7(b), one can see that the better utilization levels achieved by overlapping job executions is particularly visible for high arrival rates. All in all, overlapping executions offer better resource utilization and lower response times, and their benefits are more prominent in highly loaded systems.

VII. Related Work

Motivated by the wide adoption of the MapReduce programming paradigm, there is a plethora of studies that model and optimize the performance of MapReduce jobs, from high level resource managing [23], [24], processing platforms [18], scheduling policies [15], [25], task managements [26], to lower level data block managements [1], [2], [2]. We particularly highlight and summarize the prior art that explores the latency-accuracy tradeoff for MapReduce jobs and develops latency models for capturing the execution times and response times for MapReduce jobs.

Approximate MapReduce Jobs Introducing approximation to MapReduce jobs by only processing a subset of the input data at the cost of analysis accuracy stems from the concern of resource efficiency, query latency [12], and analysis accuracy [9]. Rinard [27] derives the accuracy bound under task dropping; whereas Rinodato et al. [28] develop an sampling algorithm for MapReduce jobs, PARMA, that performs associate rule mining. BlinkDB [9], an approximate query processing framework based on Spark [18], provides accuracy guarantees in short response times by leveraging statistical sampling theory to choose the input data. Approxhadoop [10] provides a framework that combines input sampling with task dropping via the two-stage sampling theory so as to achieve user specified accuracy targets with only a minimum amount of processing. While the related work provides mechanisms to achieve the desirable accuracy target, little is on capturing the impact of input and task dropping on execution times and response times.

Latency Model for MapReduce The existing related work on modeling MapReduce centers on the execution time or the response time of MapReduce jobs, particularly on the average values. Models that capture the execution time at the task level, i.e., data blocks, number of tasks, slots, tend to rely on profiling techniques but overlook aspect of job arrivals. Zhang

et al. [13], [29] propose a set of linear system models and application models that can capture the dependency between data inputs and execution times via a careful parameterization on production installations. Goiri et al. [10] use a linear model that consists of a fixed and variable time overhead, depending on the number of map tasks and input data. Based on their findings, our model assumes the linear relationship between the data input and execution time per task but further captures the complex dynamics of job arrivals, randomness in the single task execution time, as well as, the effect of task dropping and input sampling on the distribution of execution time, particularly the tails.

Another set of work modeling the response time of MapReduce jobs can capture well the impact of job arrivals and also scheduling policies, but tend to abstract the task dynamics, i.e., assuming jobs completion is simply a random variable. Wang et al. [30] focus on modeling the execution time of phases, considering the data locality, and present a scheduler to overcome the performance degradation brought by imbalanced locality. Tan et al. [14] optimize the reduce phase by considering the locality of intermediate key-value pairs so as to optimize the job response time of sequential MapReduce jobs. Lin et al. [15] considers the performance of overlapping map and shuffle phases and explores the designs of scheduling policies via theoretical analysis, as well as, trace driven simulations. As this class of models simplifies the task execution times, they are not immediately applicable to exploit the tradeoff between analysis accuracy and response time.

In contrast to the existing models, our analysis is not only able to model the distribution of response times under three approximation strategies but also for two scheduling systems.

VIII. Concluding Remarks

Motivated by the rise of approximate processing platforms and the lack of suitable latency models, we develop stochastic models to capture the latency distribution for approximate MapReduce jobs that are executed in an on-line fashion. Using matrix analytic techniques, we derive the distribution of execution times and response times of approximate jobs for exclusive and overlapping scheduling. Our analysis can evaluate the tradeoff between different latency metrics, particularly the tail percentiles, and the accuracy, under three approximation strategies, namely early task dropping, input sampling and straggler task dropping. We explore different combinations of system parameters and show that straggler task dropping is the most effective in improving the latency, achieving significant reductions by dropping only a small fraction of tasks. The models also quantify the significant performance gains obtained by allowing the execution of partially-overlapping jobs, which increases the resource utilization while reducing the job response times.

Acknowledgment

This work has been partly funded by the Swiss National Science Foundation (projects 407540_167266 and 200021_1410). Juan F. Pérez has been supported by the ARC Centre of Excellence for Mathematical and Statistical Frontiers (ACEMS).

References

[1] Y. Zhao and J. Wu, "Dache: A data aware caching for big-data applications using the mapreduce framework," in *INFOCOM*, 2013, pp. 35–39.

[2] B. Wang, J. Jiang, and G. Yang, "Actcap: Accelerating mapreduce on heterogeneous clusters with capability-aware data placement," in *INFOCOM*, 2015, pp. 1328–1336.

[3] X. Bu, J. Rao, and C. Xu, "Interference and locality-aware task scheduling for mapreduce applications in virtual clusters," in *HPDC*, 2013, pp. 227–238.

[4] Y. Yuan, D. Wang, and J. Liu, "Joint scheduling of mapreduce jobs with servers: Performance bounds and experiments," in *INFOCOM*, 2014, pp. 2175–2183.

[5] S. Spicuglia, L. Y. Chen *et al.*, "Optimizing capacity allocation for big data applications in cloud datacenters," in *IFIP/IEEE IM*, 2015, pp. 511–517.

[6] Y. Ying, R. Birke *et al.*, "Optimizing energy, locality and priority in a mapreduce cluster," in *IEEE ICAC*, 2015, pp. 21–30.

[7] T. Condie, N. Conway *et al.*, "Mapreduce online," in *NSDI*, 2010, pp. 313–328.

[8] R. Birke, M. Björkqvist *et al.*, "Meeting latency target in transient burst: a caseon spark streaming," in *IEEE IC2E*, 2017.

[9] S. Agarwal, B. Mozafari *et al.*, "Blinkdb: queries with bounded errors and bounded response times on very large data," in *Eurosys*, 2013, pp. 29–42.

[10] I. Goiri, R. Bianchini *et al.*, "Approxhadoop: Bringing approximations to mapreduce frameworks," in *ASPLOS*, 2015, pp. 383–397.

[11] S. Lohr, *Sampling: Design and Analysis*. Cengage Learning, 2009.

[12] S. Agarwal, H. Milner *et al.*, "Knowing when you're wrong: building fast and reliable approximate query processing systems," in *SIGMOD*, 2014, pp. 481–492.

[13] Z. Zhang, L. Cherkasova *et al.*, "Performance modeling and optimization of deadline-driven pig programs," *TAAS*, vol. 8, no. 3, p. 14, 2013.

[14] J. Tan, S. Meng *et al.*, "Improving reducetask data locality for sequential mapreduce jobs," in *INFOCOM*, 2013, pp. 1627–1635.

[15] M. Lin, L. Zhang *et al.*, "Joint optimization of overlapping phases in mapreduce," *SIGMETRICS Performance Evaluation Review*, vol. 41, no. 3, pp. 16–18, 2013.

[16] Z. Qiu, J. F. Pérez, and P. G. Harrison, "Beyond the mean in fork-join queues: Efficient approximation for response-time tails," *Perform. Eval.*, vol. 91, pp. 99–116, 2015.

[17] "Apache Hadoop Distributed File System," http://hadoop.apache.org/.

[18] "Apache Spark," http://spark.apache.org/.

[19] G. Latouche and V. Ramaswami, *Introduction to matrix analytic methods in stochastic modeling*. SIAM, 1999.

[20] B. Sengupta, "Markov processes whose steady state distribution is matrix-exponential with an application to the GI/PH/1 queue," *AAP*, vol. 21, pp. 159–180, 1989.

[21] Z. Qiu and J. F. Pérez, "Evaluating replication for parallel jobs: An efficient approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, 2016.

[22] S. Asmussen and J. R. Møller, "Calculation of the steady state waiting time distribution in GI/PH/c and MAP/PH/c queues," *Queueing Syst.*, vol. 37, pp. 9–29, 2001.

[23] C. Delimitrou and C. Kozyrakis, "Quasar: resource-efficient and qos-aware cluster management," in *ASPLOS*, 2014, pp. 127–144.

[24] B. Hindman, A. Konwinski *et al.*, "Mesos: A platform for fine-grained resource sharing in the data center," in *NSDI*, 2011, pp. 295–308.

[25] Y. Le, J. Liu *et al.*, "Online load balancing for mapreduce with skewed data input," in *INFOCOM*, 2014, pp. 2004–2012.

[26] G. Ananthanarayanan, M. C.-C. Hung *et al.*, "Grass: Trimming stragglers in approximation analytics," in *NSDI*, 2014, pp. 289–302.

[27] M. Rinard, "Probabilistic accuracy bounds for fault-tolerant computations that discard tasks," in *ICS*, 2006, pp. 324–334.

[28] M. Riondato, J. A. DeBrabant *et al.*, "PARMA: a parallel randomized algorithm for approximate association rules mining in mapreduce," in *CIKM*, 2012, pp. 85–94.

[29] Z. Zhang, L. Cherkasova, and B. T. Loo, "Parameterizable benchmarking framework for designing a mapreduce performance model," *Concurr. Comput.*, vol. 26, no. 12, pp. 2005–2026, 2014.

[30] W. Wang, K. Zhu *et al.*, "Map task scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality," in *INFOCOM*, 2013, pp. 1609–1617.