This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2552516, IEEE Transactions on Cloud Computing

Stochastic Modeling and Optimization of Stragglers

Farshid Farhat, Diman Zad Tootaghaj, Yuxiong He, Anand Sivasubramaniam, Mahmut Kandemir, and Chita R. Das

Abstract—MapReduce framework is widely used to parallelize batch jobs since it exploits a high degree of multi-tasking to process them. However, it has been observed that when the number of servers increases, the map phase can take much longer than expected. This paper analytically shows that the stochastic behavior of the servers has a negative effect on the completion time of a MapReduce job, and continuously increasing the number of servers without accurate scheduling can degrade the overall performance. We analytically model the map phase in terms of hardware, system, and application parameters to capture the effects of stragglers on the performance. Mean sojourn time (MST), the time needed to sync the completed tasks at a reducer, is introduced as a performance metric and mathematically formulated. Following that, we stochastically investigate the optimal task scheduling which leads to an equilibrium property in a datacenter with different types of servers. Our experimental results show the performance of the different types of schedulers targeting MapReduce applications. We also show that, in the case of mixed deterministic and stochastic schedulers, there is an optimal scheduler that can always achieve the lowest MST.

Index Terms— MapReduce, Stochastic Modeling, Optimal Scheduling, Performance Evaluation, Queuing Theory.

1 Introduction

MAPREDUCE has become a popular paradigm for structuring large scale parallel computations in datacenters. By decomposing a given computation into (one or more) Map and Reduce phases, the work within each phase can be accomplished in parallel without worrying about data dependencies, and it is only at the boundaries between these phases where one needs to worry about issues such as data availability and dependency enforcement. At the same time, with the possibility of elastically creating tasks of different sizes within each phase, these computations can adjust themselves to the dynamic capacities available in the datacenter. There has been a lot of prior work in the past decade to leverage this paradigm for different applications [1,2,3], as well as in the systems substrate needed to efficiently support their execution at runtime [4,5,6].

While each phase is highly parallel, the inefficiencies in MapReduce execution manifest at the boundaries between the phases as data exchanges and synchronization stalls, which ensure completion of the prior phases. One of these inefficiencies is commonly referred to as the straggler problem of mappers, where a reduce phase has to wait until all mappers have completed their work [4]. Even if there is one such straggler, the entire computation is consequently slowed down. Prior work [7,8,9,10] has identified several reasons for such stragglers including load imbalance, scheduling inefficiencies, data locality, communication overheads, etc. There have also been efforts looking to address one or more of these concerns to mitigate the straggler problem [7,8,11,12,13]. While all these prior efforts are important, and useful to address this problem, we believe that a rigorous set of analytical tools is needed in order to: (i) understand the consequences of stragglers on the performance slowdown in MapReduce execution; (ii) be able to quantify this slowdown as a function of different hardware (processing speed, communication bandwidth, etc.), system (scheduling policy, task to node assignment, data distribution, etc.), and application (data size, computation needs, etc.) parameters; (iii) study the impact of different scaling strategies (number of processing nodes, the computation to communication and data bandwidths, tasks per node, etc.) on this slowdown; (iv) undertake "what-if" studies for different alternatives (alternate scheduling policies, task assignments to nodes, etc.) beyond what is available to experiment with on the actual platform/system; and (v) use such capabilities for a wide range of optimizations: they could include determining resources (nodes, their memory capacities, etc.) to provision for the MapReduce jobs, the number of tasks to create and even adjust dynamically, the assignment of these tasks to different kinds of nodes (since datacenters could have heterogeneous servers available at a given time), adjusting the scheduling policies, running redundant versions of the tasks based on the trade-offs between estimated wait times and additional resources mandated, executing a MapReduce computation under a budgetary (performance, power, cost) constraint, etc.

To our knowledge, there are no rigorous stochastic analysis tools available today with these capabilities for modeling and understanding the straggler problem in MapReduce for the purposes listed above. This paper intends to fill this critical gap by presenting an analytical model and optimization framework for capturing the waiting time at

- *Farshid Farhat, Diman Zad Tootaghaj, Anand Sivasubramaniam, Mahmut Kandemir, and Chita R. Das are with the school of electrical engineering and computer science, the Pennsylvania State University, University Park, PA, 16802, USA. Email: {fuf111,dxz149,anand,kandemir,das}@cse.psu.edu.*
- *Yuxiong He is with the Cloud Computing Futures group, the Microsoft Research, Redmond, WA 98052 USA. Email: yuxhe@microsoft.com.*

xxxx-xxxx/0x/$xx.00 © 200x IEEE Published by the IEEE Computer Society

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2552516, IEEE Transactions on Cloud Computing

the end of the Map phase due to any stragglers. We also demonstrate the benefits of having such a tool with a few case studies. Specifically, this paper makes the following contributions towards presenting and exploiting an analytical model for the stragglers in MapReduce computations:

- We demonstrate that our delayed tailed distribution can be used to capture the service time of the tasks at a given node. We then show that, with such service times, the aggregate completion time of the tasks across all the nodes of the cluster also follows our delayed tailed distribution. This is shown against a spectrum of stragglers' completion times of 10 production workloads studied in prior research.
- With this result, we develop a closed-form queuing model of the time expended before a reducer node can begin its part of the computation, i.e., the time for all mappers to finish, referred to as Mean Sojourn Time (MST). Parameterized by the task inter-arrival times to the mappers, the delayed tailed service times, and the number of mappers, this model helps us conveniently study the impact of different parameters--whether job characteristics, hardware capabilities or system configuration--on the delays before a reducer can start.
- This model can be used for a variety of purposes as explained above. In this paper, we specifically illustrate two use cases. First, we show how the model can be used to schedule tasks on different (possibly heterogeneous) nodes of a datacenter to reduce the MST. This is demonstrated on the average to be 129% more effective than the JobTracker scheduling of the current Hadoop distribution [5], and 51% better than ideal deterministic approaches.
- We show that increasing the number of nodes assigned to the tasks is not always helpful, since the variance of completion times can increase the total completion time. There are a critical number of nodes that should be assigned to a job, and we illustrate how our model can be used to determine it with respect to the service times of the nodes and the computational complexity of the job.

2 Preliminaries

2.1 MapReduce Framework

The MapReduce framework [4] is a programming paradigm that can be used to execute data-intensive jobs. This framework can be applied to a large class of algorithms, known as MapReduce Class (MRC) [14], with high levels of parallelism. One of MapReduce implementation is the open-source Hadoop/MapReduce framework [47] as a scalable implementation built upon fault-tolerant Hadoop file system (HDFS) [5].

Fig. 1 shows a high-level view of the MapReduce framework. A job arrives with mean rate λ, and is partitioned into '*map*' tasks. More specifically, the JobTracker module in Hadoop assigns map/reduce tasks to TaskTracker nodes. Each map task tracker node (mapper) has threads to perform the map tasks. Once the map tasks are completed, a set of intermediate key/value pairs is generated and passed to the associated reducer node in the shuffling stage shown in the middle of Fig. 1. In fact, each reducer node may receive values with the same intermediate key assigned to that node. Each mapper node employs reducers to compute and merge the received intermediate values. After the reduce phase, the final values are merged into the HDFS-based storage.

For example, in a word-count application (where the goal is finding the frequency of the words in a huge document), a mapper may count the frequency of each word in its received text in the map phase, and then send (word, frequency) tuples for assigned reducers in the shuffling phase. It may send words that begin with letter 'A' to the first reducer, words that begin with letters 'B, C or D' (having balanced number of words for each) to the second reducer, etc. It is important to note that the map and shuffle phases may overlap. Each reducer calculates the total frequency of its words by summing the corresponding frequencies of each word in reduce phase.

For a reducer to start its execution, all mappers that have data to send to that reducer should finish sending. During execution, one may observe various imbalances across mappers, due to resource contention, unbalanced tasks scheduled on mapper nodes, or heterogeneity across computational resources [8]. Clearly, the start time of a reduce job is dictated by the slowest mapper node, that is, the slowest mapper node determines how soon a reduce task can start its execution. One of the major reasons for excessively long execution latencies of MapReduce jobs is stragglers, i.e., some servers that complete their assigned tasks in a longer time than usual.

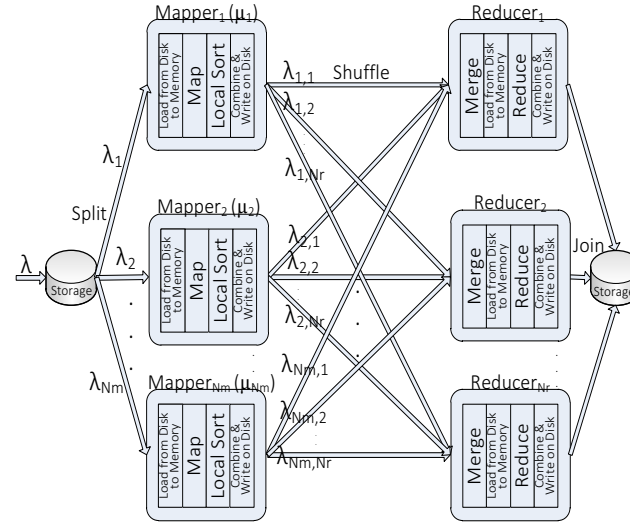


Fig. 1. MapReduce framework.

2.2 Delayed Tailed Distribution

We start by giving the definition of our Delayed Tailed Distribution (DTD) since it is highly used in our model.

Definition 1: *Given random variable X with rate λ and offset time T, $F_X(x)$ and $f_X(x)$, respectively the cumulative distribution function (CDF) and the probability density function (PDF) of delayed tailed distribution, can be defined as follows:*

$$F_X(x) = \begin{cases} 0 & ; x < T \\ 1 - e^{-\Lambda(x-T)}; & x \geq T \end{cases} = \left(1 - e^{-\Lambda(x-T)}\right) U(x-T) \quad (1)$$

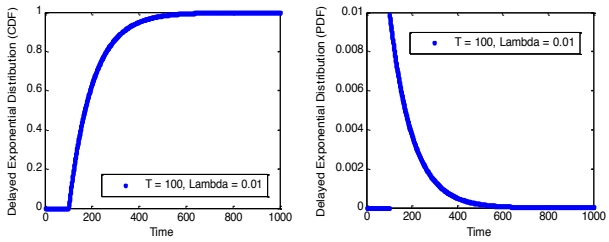
$$f_X(x) = \begin{cases} 0 & ; x < T \\ \Lambda' (x-T) e^{-\Lambda(x-T)}; & x \geq T \end{cases}, \quad (2)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2552516, IEEE Transactions on Cloud Computing

where $U(x)$ is unit step function, $\Lambda(x) = ax + b\,ln(x+1)$ is a monotonically increasing rate function from zero for $a, b \geq 0$ (i.e., $\Lambda(x) = 0; x \leq 0$ and $\Lambda'(x) \geq 0; x \geq 0$), and rate λ is the reciprocal of the mean of the distribution (i.e. $\lambda^{-1} = \int_0^\infty e^{-\Lambda(x)}dx$).

Fig. 2a and Fig. 2b plot CDF and PDF of a DTD respectively, where an exponential distribution is obtained by a linear rate function such as $\Lambda(x) = 0.01$x, and by a logarithmic rate function $\Lambda(x) = 0.01\log(x+1)$, a Pareto, heavy-tailed or power-law distribution can be obtained. Generally $f_X(x)$ can capture any generalized Pareto distribution. An important observation is that the completion times of the tasks exhibit a heavy-tailed distribution [8,9,10]. We show in this paper that DTD completion time is nearly coincident with empirical data derived from prior work.

As illustrated in Fig. 2b, most servers finish their tasks right after a threshold, but a fraction of servers finish their tasks only after a longer time. Since reducers can start only after completion of all their related map tasks, they will be delayed because of these delayed servers. Note that the intrinsic properties of architecture-level heterogeneity (e.g., big core versus small core) can further magnify the impact of stragglers. In this paper, we analytically model stragglers and show their effect on completion time. We also optimize the delay using the schedulers depending on task mapping and the number of servers.



(a) (b)

Fig. 2. (a) CDF and (b) PDF of a DTD.

2.3 Problem Definition

The response time in the critical path of a MapReduce job includes the delays in storage, network communications, map-task computation, synchronization of the map tasks, reduce-task computation, and aggregation of the reduce tasks. While most of these delays are known deterministic, the delay from map task scheduling (mapping) to synchronization at a reducer is known as a stochastic phenomenon. Heterogeneity in cluster resources makes the synchronization delay of the tasks higher. In fact, the problem of uneven task completion times has been shown to be a serious impediment to the scalability of MapReduce computations [8,11,19]. While this has been studied experimentally, it has not been investigated from a stochastic perspective. So we want to minimize this synchronization delay by scheduling the coming job among heterogeneous mappers with stochastic completion times.

A MapReduce job (e.g., word count) can be seen as a set of tasks that must be completed to obtain the desired result (e.g., the frequency of each word). In our problem, the jobs are submitted to the system with the mean arrival rate λ, which can be interpreted as the average number of CPU instructions coming to the system per second. As shown in Fig. 1, a job with arrival rate λ comes to the MapReduce framework, and can be divided into a set of map tasks with arrival rates of $\lambda_1, \lambda_2, \ldots \lambda_{N_m}$, where N_m mapper nodes are involved.

If the job is split into K map tasks, the mapper nodes receive their map tasks proportionally with the ratios $p_1, p_2, \ldots p_{N_m}$ determined by the scheduler. Consequently, $p_i K$ map tasks are sent to the i^{th} mapper node. In other words, the mean value of the task arrival rate to the i^{th} mapper node is $\lambda_i = p_i\lambda$ where $\lambda = \sum_{i=1}^{N_m}\lambda_i$. In addition to the skewness of map tasks captured by λ_i, the heterogeneity of the mean service rates of the mapper nodes is captured by μ_i (instructions per second).

Stragglers problem *captures the skewness of the completion times of the stragglers for a single job. In other words, the completion times of all tasks for a single job have a long-tailed or more generally delayed tailed distribution denoted in Eq. (1,2).*

We are interested to model and formulate this problem as a stochastic scheme, and then we want to find the optimal scheduling or task arrival rates ($\lambda_i\ \forall i$ in Fig. 1) to mappers as well as the optimal number of mappers (N_m) to minimize the completion time of all map tasks, where $\lambda = \sum_{i=1}^{N_m}\lambda_i$ and the service rates ($\mu_i\ \forall i$) and completion time of the mappers are heterogeneous following our verified delayed tailed distribution. The notations used in this paper are listed in Table 1.

Throughout the paper, the stochastic behavior of the i^{th} mapper node with its communication links is modeled as a single queue with a mean service rate (μ_i). The completion time of each mapper node is independent of the other mapper nodes, but the completion time of a map task may be dependent on the completion time of the other map tasks that reside in the same mapper node. Furthermore, the deterministic delay of mapper node computation and its communication links are captured by an offset time.

TABLE 1. NOTATION USED IN OUR FORMULATION.

PARAMETER	SIGN	EXPLANATION
NUMBER OF MAPPER NODES	N_m	THE NUMBER OF NODES IN THE DATA-CENTER ASSIGNED FOR MAP TASKS
MEAN INTER-ARRIVAL TIME OF ith MAPPER	$1/\lambda_i$	THE AVERAGE TIME BETWEEN THE ARRIVALS OF TASKS COMING TO ith MAPPER
MEAN SERVICE TIME OF ith MAPPER	$1/\mu_i$	THE AVERAGE SERVICING AND ROUTING TIME OF THE TASKS BY ith MAPPER
MEAN JOB INTER-ARRIVAL TIME	$1/\lambda$	$\lambda = \sum_{i=1}^{N_m}\lambda_i = \sum_{i=1}^{N_m} p_i\lambda = p_i\,\lambda_i^{-1}$
UNIT STEP FUNCTION	$U(t)$	$U(t) = \begin{cases}1; t \geq 0\\ 0; t < 0\end{cases}$
DIRAC DELTA FUNCTION	$\delta(t)$	$\delta(t) = dU(t)/dt$
OFFSET OF DTD FOR ith MAPPER	T_i	THE MINIMUM AMOUNT OF TIME REQUIRED TO COMPLETE A TASK
COMPLETION TIME OF ALL MAP TASKS	R_M	THE REQUIRED TIME TO FINISH ALL MAP TASKS BY ALL MAPPERS
COMPLETION TIME OF ith MAPPER	R_{M_i}	THE REQUIRED TIME TO FINISH A TASK BY ith MAPPER
MEAN COMPLETION RATE OF ith MAPPER	γ_i	THE AVERAGE REQUIRED TIME TO FINISH A TASK BY ith MAPPER

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2552516, IEEE Transactions on Cloud Computing

Small stochastic variances across communication link delays add a nearly-deterministic delay overhead on all end-to-end paths, because orchestrated traffic from mappers to reducers is nearly-deterministic, and the shuffling phase overlaps with the map phase of MapReduce. Also prior studies [16,23] show task mapping from mappers to reducers via pipelining or multi-level hash-based mechanism can be balanced. Thus, the difference across the deterministic delays of the paths from mappers to a reducer is negligible without loss of generality, and straggling reducers problem is a deterministic problem.

3 Modeling of Stragglers

We showed the mean arrival rate of the job (λ) could capture the computational overhead of the MapReduce job, and the task arrival rates (λ_i) could capture the task scheduling or the amount of data chunks mapped to the servers. Also these task arrival rates (λ_i) can capture the skewness of the data computation across the servers. Now, we describe the server model as a single queue with mean service rate μ_i (instructions per second), and we justify this model can be captured by our delayed tailed distribution, and then we validate the model by using the maximum likelihood method [49] for 10 MapReduce workloads.

3.1 Server as a Single Queue

The completion time of all map tasks is a function of the inter-arrival times of tasks and the service times of the servers. In our model, we investigate the DTD service time. Later, we validate the model using DTD for real workloads. Also we study different distributions for task inter-arrival times and service times. Note that we are not restricted to a simple queue; we investigate the other inter-arrival times and service times that correspond to other potential scenarios in modern datacenters [27] as a general form of job inter-arrival time.

A server can be modeled as a FIFO infinite-buffer single queue with a DTD service time. Because CPU clock rate of a server has a periodic characteristic and is proportional to the mean service rate μ_i as a linear coefficient (say C), i.e., sequential instructions can be executed successively with a time difference not less than $t_i \propto 1/(\mu_i C)$, and other instructions that involve memory or I/O requests can have an even longer time difference from the previous instruction. Consequently, the distribution of CPU clock rate is $\delta(t - t_i)$ (the Dirac Delta function as $\delta(t) = dU(t)/dt$). The distribution of the service time of the instructions is known as an exponential distribution by many prior studies [28,29,30]. The distribution of the combination of these two random variables is equivalent to the convolution ($*$) of these two distributions. The resulting service time distribution of a server is thus a DTD where its PDF can be expressed as:

$$\mu_i e^{-\mu_i(t)} U(t) *_{conv} \delta(t - t_i) = \mu_i e^{-\mu_i(t-t_i)} U(t - t_i), \quad (3)$$

where t_i is the minimum time required to process a CPU instruction. Each task contains a number of sequential instructions (say I), each having a minimum time to execute and route to a reducer. Therefore, the total service time of a server is not less than $T_i = \sum_{j=1}^{I} t_j$ giving the offset of a DTD in the service times, and capturing the deterministic part of computation delay. Note that the different servers can have different mean service rates (μ_i) and offset times T_i which can make a datacenter heterogeneous, and in designing a scheduler for a heterogeneous datacenter, we only need these parameters (μ_i and T_i). As a result, the DTD service time is a generalization of the exponential and Pareto service time and we will show that it matches with real data.

3.2 Delayed Tailed Completion Time of Tasks

The completion time of the tasks is the waiting time in the buffer (queue) plus the time required to service them. Our defined DTD completion time denoted in Eq. (1,2) for all map tasks coming to a reducer matches with the empirical completion times of map tasks that have been published in prior studies [8,9,10]. We use the completion times of different MapReduce applications (in Table 2) as a reference, and then we try to fit the data on a CDF of a DTD represented by Eq. (1,2) as follows:

$$\left(1 - e^{-\Delta_i(t-T_i)}\right) U(t - T_i).$$

The maximum likelihood estimate of rate function has been described in appendix of [49]. Note that the offset time T_i (constant for an application) can be directly taken from the empirical data, and it is only the rate function (Δ_i) believed [20,39,40,41] to be a logarithmic function resembling a heavy-tailed distribution. The mean squared error is not greater than 5% across all those workloads as shown in Table 2, strengthening our rationale for modeling completion times as a DTD. Accordingly, Table 3 lists important parameters and their values used in our subsequent analysis and simulations. These values are extracted from the prior studies data listed in Table 2 by means of quantitative methods discussed in chapter 3 of [45].

TABLE 2. RATES USED IN THE PUBLISHED DATA.

BENCHMARK	REF.	RATE	MEAN SQUARED ERROR
BING SEARCH ENGINE	[8]	1.5196	0.0465
FACEBOOK	[10]	1.6715	0.0345
CLOUDERA CUSTOMER (A)	[10]	1.7122	0.0328
CLOUDERA CUSTOMER (B)	[10]	1.7868	0.0273
CLOUDERA CUSTOMER (C)	[10]	1.6491	0.0334
CLOUDERA CUSTOMER (D)	[10]	1.7123	0.0345
CLOUDERA CUSTOMER (E)	[10]	1.7671	0.0266
OPENCLOUD	[9]	1.6862	0.0352
M45	[9]	1.7162	0.0361
WEBMINING	[9]	1.7805	0.0360

TABLE 3. PARAMETER VALUES USED IN EXPERIMENTS.

PARAMETER	RANGE
MEAN INTER-ARRIVAL TIME	0.1S-2S
MEAN SERVICE TIME	0.5S-2S
OFFSET TIME	0.1S-100S
MEAN COMPLETION TIME	0.5S-1000S
UTILIZATION	0.1-0.95

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2552516, IEEE Transactions on Cloud Computing

4 Formulation of Stragglers

We define a new performance metric known as Mean Sojourn Time (MST), and express it as a closed-form formula in terms of the completion time distributions of the map tasks which are dependent on task inter-arrival rates and service rates. Then by lemmas 1 and 2, we analytically proved that the distribution of sojourn time at a reducer asymptotically follows DTD. Also the closed-form MST for a special case has been derived in Lemma 3.

4.1 Mean Sojourn Time at a Reducer

Mean sojourn time at a reducer represents the average time required to synchronize all completed map tasks before a reducer can start its execution. MST can be a reasonable metric to represent the mean delay from job split to merge in a reducer as a fork-join queue for the first part of the end-to-end delay of a MapReduce job.

Definition 2 *(Mean Sojourn Time): Given general CDFs of independent and identically distributed (i.i.d) completion times of map tasks as* $F_{R_{M_i}}(t) = P(R_{M_i} \le t); i = 1 \dots N_m$*, by using maximum order statistics (MOS), the required time for synchronization of all completed map tasks at* i^{th} *reducer* (S_{R_i}) *is the maximum of the completion times of all map tasks, i.e., we can express* $S_{R_i} = max(R_{M_1}, R_{M_2}, \dots, R_{M_{N_m}})$ *as:*

$$F_{S_{R_i}}(t) = P(S_{R_i} \le t) = \prod_{i=1}^{N_m} P(R_{M_i} \le t) = \prod_{i=1}^{N_m} F_{R_{M_i}}(t). \quad (4)$$

The corresponding PDF can be expressed as follows:

$$f_{S_{R_i}}(t) = \frac{\partial}{\partial t} F_{S_{R_i}}(t) = F_{S_{R_i}}(t) \sum_{j=1}^{N_m} \frac{f_{R_{M_j}}(t)}{F_{R_{M_j}}(t)}. \quad (5)$$

Eq. (6)-(8) given below are used in our formulation to be presented shortly. The mean sojourn time by using the inclusion–exclusion principle [31] can be expressed with respect to the expected-value of the minimum of completion times of every subset of map tasks as:

$$MST = E\{S_{R_i}\} = E\left\{max\left(R_{M_1}, R_{M_2}, \dots, R_{M_{N_m}}\right)\right\}$$
$$= \sum_{i=1}^{N_m} \left\{ (-1)^{i+1} \sum_{\forall \{j_1, j_2, \dots, j_i\} \subset \{1,2,\dots,N_m\}} E\left\{min\left(R_{M_{j_1}}, R_{M_{j_2}}, \dots, R_{M_{j_i}}\right)\right\} \right\} \quad (6)$$

Furthermore, MST can be expressed in another way, with respect to the distribution of completion time of each server using Eq. (5) as:

$$MST = \int_{t=0}^{\infty} t f_{S_{R_i}}(t) dt = \int_0^{\infty} t \frac{\partial}{\partial t}\left(\prod_{i=1}^{N_m} P(R_{M_i} \le t)\right) dt. \quad (7)$$

And, using Eq. (6), we have:

$$MST = \sum_{j=1}^{N_m} \int_0^{\infty} t\, F_{S_{R_i}}(t) \frac{f_{R_{M_j}}(t)}{F_{R_{M_j}}(t)} dt. \quad (8)$$

Note that the mean sojourn time (MST) is dependent on completion time CDF of each map task ($P(R_{M_i} \le t)$) expressed by the inter-arrival time and the service time. For heterogeneous queues (servers with different service times or computational performances) with a general form joint distribution of task inter-arrival time, we are not aware of any published closed-form formulations, bound or approximation, for the MST. However, in the case of two homogeneous queues, there is an approximation and lower/upper bounds [32]. Also for the exponentially distributed completion time of homogeneous queues, there are some approximations and boundaries in the statistics literature [33].

4.2 Asymptotic Delayed Tailed Completion Time

Before going for optimization case studies, we experimentally validated our model in section 3.2, now we are interested to analytically justify the usage of our DTD-based model. Using Lemma 1 and Lemma 2, we show that a sufficient condition for having an asymptotic DTD completion time of all map tasks to a reducer is to have a DTD service time for each mapper node.

Lemma 1: *If the service time distribution of the mapper is a DTD, then the completion time distribution of the map task is asymptotically a DTD (see Appendix A for the proof).*

Lemma 2: *If the completion time distributions of each map task is a DTD, then the completion time distribution of all map tasks or sojourn time at a reducer* ($P(R_M \le t)$) *is also asymptotically DTD.*

Proof. Assume that R_M is the completion time of all map tasks, and R_{M_i} is the completion time of the i^{th} mapper node (where $1 \le i \le N_M$) which has a DTD of the form $\left(1 - e^{-\Delta_i(t-T_i)}\right)U(t - T_i)$, then we have:

$$F_{R_M}(t) = P(R_M \le t) = P\left(R_{M_1} \le t, \dots, R_{M_{N_m}} \le t\right)$$
$$= \prod_{i=1}^{N_m} P(R_{M_i} \le t) = \prod_{i=1}^{N_m} \left(1 - e^{-\Delta_i(t-T_i)}\right) U(t - T_i)$$
$$= 1 - \sum_{i=1}^{N_m} e^{-\Delta_i(t-T_i)} U(t - T_i)$$
$$+ \sum_{i,j=(1,1)}^{(N_m,N_m)} e^{-\left(\Delta_i(t-T_i)+\Delta_j(t-T_j)\right)} U\left(t - min(T_i, T_j)\right) - \cdots$$

The PDF can be derived by computing the derivative of the CDF, and finally the asymptotic term of the PDF can be written as follows:

$$f_{R_M}(t) = P(R_M = t) \sim e^{-\min_i \Delta_i(t-T_i)} U\left(t - T_{\underset{i}{\operatorname{argmin}}\, \Delta_i(t-T_i)}\right) \quad (9)$$

Because the higher-order terms have a higher decay-rate than the first-order terms, and having equal decay-rate terms does not change the exponent while just resulting higher base coefficient. If the completion time has exponential distribution, the resulting completion time is also exponential, i.e., if one of completion time distribution has a Pareto distribution (logarithmic part of DTD), sadly the total completion time at a reducer would be asymptotically a Pareto or long-tailed distribution. □

Consequently, in the case of exponential service time, the sojourn time distribution is also asymptotically exponential, and we derive the MST closed formula for the servers with different M/M/1 queues in Lemma 3.

Lemma 3: *Given M/M/1 mapper nodes with arrival rate* λ_i *and service rate* μ_i *where* $i = 1 \dots N_m$*, using maximum order statistics, the MST of map tasks at a reducer is:*

$$MST_{M/M/1} = \sum_{i=1}^{N_m} \left\{ (-1)^{i+1} \sum_{\forall \{j_1, j_2, \dots, j_i\} \subset \{1,2,\dots,N_m\}} \frac{1}{\sum_{k=1}^{i} (\mu_{j_k} - \lambda_{j_k})} \right\} \quad (10)$$

Proof. The completion time of $j_k{}^{th}$ M/M/1 mapper node is $P\left(R_{M_{j_k}} \le t\right) = \left(1 - e^{-\left(\mu_{j_k} - \lambda_{j_k}\right)t}\right) U(t)$, so we have:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2552516, IEEE Transactions on Cloud Computing

$$E\left\{min\left(R_{M_{j_1}},\dots,R_{M_{j_i}}\right)\right\} = \int_0^\infty tP\left(min\left(R_{M_{j_1}},\dots,R_{M_{j_i}}\right)\le t\right)dt$$
$$= \int_0^\infty t\left(1-\prod_{k=1}^{i}P\left(R_{M_{j_k}}>t\right)\right)dt \qquad (11)$$
$$= \frac{1}{\sum_{k=1}^{i}(\mu_{j_k}-\lambda_{j_k})}.$$

Substituting Eq. (11) in Eq. (7), Eq. (10) can be obtained. □

4.3 MST Variation in DTD Model

This section wants to give some visual intuition about MST metric variation in terms of total arrival rate and number of mapper nodes in the system. The rationale behind these behaviors motivates us to explore optimization case studies in next sections.

Using validated DTD completion time in section 3.2, the variation of the MST with respect to the mean arrival rate of the job can be derived and is shown in Fig. 3a, when the mean service rate of each mapper is $\mu_i = 1$, and the number of mappers is $N_m = 60$.

When the arrival rate of the job increases, the task arrival rates of the mappers grow, and the MST at a reducer homographically tends to infinity. This homographical growth similarly happens in congested network by increasing the number of sources [43] and also the Kingman's formula [46] shows this phenomenon analytically.

Fig. 3b gives an intuition about the variation of the MST in terms of the number of mappers (N_m), when the service rate of each mapper node is $\mu_i = 1$, and the mean arrival rate of the job is $\lambda = 2$.

When the number of mapper nodes increases more than 10, the amount of task assigned to each mapper reduces, but the time required to sync the mappers grows. Finally the MST asymptotically tends to infinity with the order of $O(log_e(N))$ where N is the number of mappers.

The reason is that the derivative of the MST (8) is proportional to the reciprocal of the number of mappers. Intuitively, there is a logarithmic algorithm to sync and merge completed tasks at a reducer, when a sync/merge operation can only happen between two completed tasks, as it has been also verified in [44].

Based on the discussion above, we can conclude that the MST (the average time necessary to synchronize map tasks before the reduce phase) can be stochastically expressed using the completion time of the map tasks. As such, MST can show the overall behavior of the system well in terms of job inter-arrival rate and number of mappers. And it is reasonable that our goal is to minimize the MST with respect to task scheduling.

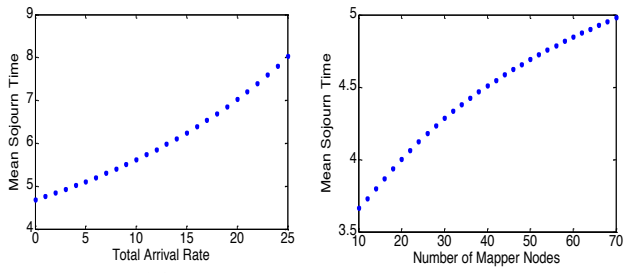


Fig. 3. MST with respect to (a) the total mean arrival rate and (b) the number of mapper nodes.

5 Potential Uses of the Model

Using our model, performance metrics of a parallel-running job such as response time or throughput can be optimized under power or budget constraints, and a corresponding scheduler can be obtained. A homogeneous or heterogeneous datacenter can be expressed using uniform ($\mu_1 = \mu_2 = \cdots = \mu_{N_m}$) or non-uniform ($\mu_1 \neq \mu_2 \neq \cdots \neq \mu_{N_m}$) service rates (Fig. 1). We envision three potential uses of our model with respect to architecture, system, and application parameters of a datacenter:

- One can model the performance parameters of nodes in a datacenter using inter-arrival times and service times of task trackers' queues more accurately, because it captures the deterministic behavior of task completion and the stochastic notion of the straggler problem.
- One can model the performance parameters of a MapReduce job as a fork-join network in a fashion that is more detailed than an M/M/1 queuing model but easier than a general distribution-based model. It can be used globally to improve the performance of a MapReduce job in a datacenter.
- Our model can also be used to optimize other target metrics such as throughput or utilization with power or budget constraints in a heterogeneous datacenter.

The delayed tailed model of completion time of mappers is an extended version of exponential or Pareto distributed model. It has the flexibility of being able to represent the arrival and service rates of different classes of workloads and cluster nodes. For example, the offset time parameter in Eq. (3) of a CPU-bound job is higher than that of a memory-bound job. Since the memory access times are random and take much longer than CPU access times, they can also be modeled by DTD.

In addition, there is high degree of freedom to adjust the departure rates of mappers to reducers ($\lambda_{i,j}; i = 1..N_m$ and $j = 1..N_r$) as shown in Fig. 1, and we can solve linear independent equations to find reducers' arrival rates. By applying a DTD completion time model to reducers, we can have similar MST formulations for reducers to optimize MST for the end to end job as well.

The non-delayed (pure Pareto or exponential) distribution model, having its highest value at zero (time=0), cannot capture the probability of two consecutive completed jobs with minimum inter-arrival time, because minimum inter-arrival time must be a positive value not zero. The DTD completion time model is also a good approximation of the completion time of a typical datacenter server as a single queue, and can be employed in the analysis of different types of distributed computing networks. Based on Lemma 1, the DTD completion time can be obtained using the DTD service time, and this makes most of the analytical formulas much simpler.

Since the focus of our paper is on performance, we use MST as an end-to-end delay-aware metric; other analyses may use different metrics depending on their specific focus. The potential metrics of interest could be money budget, power budget, power/performance, or other combinations. First, the specification of the clusters and workloads should be evaluated. Considering Fig. 2 and Eq. (3), the service rate of the server and offset time related to the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2552516, IEEE Transactions on Cloud Computing

TABLE 4. SUMMARY OF THE LEMMAS.

LEMMA	EXPLANATION
1	DTD SERVICE TIME → DTD COMPLETION TIME FOR A MAPPER.
2	DTD SERVICE TIME → DTD COMPLETION TIME FOR ALL MAP TASKS.
3	MST FORMULA FOR M/M/1 MAPPER NODE.
4	EQUILIBRIUM PROPERTY FOR D/D/1.
5	EQUILIBRIUM PROPERTY FOR M/M/1.
6	EQUILIBRIUM PROPERTY FOR G/M/1.
7	SUFFICIENT CONDITIONS FOR OPTIMAL SCHEDULING.
8	OPTIMAL MAPPING FROM MOMENTS OF THE DISTRIBUTION.
9	LOWER BOUND AND UPPER BOUND OF MST.
10	OPTIMAL NUMBER OF M/M/1 MAPPER NODES.
11	OPTIMAL NUMBER OF HOMOGENEOUS MAPPER NODES.
12	OPTIMAL NUMBER OF MAPPER NODES FOR A FIXED BUDGET.

application can be obtained from the specifications. One can therefore derive a DTD model for the service time. Then, the completion time model can be obtained using the distribution of job inter-arrival time that is discussed in the proof of Lemma 1 in Appendix A.

We are interested in investigating the behavior of different schedulers via the single queue model of mappers with the DTD completion time. This can be extended to a generalized multiple-queue model when we have multiple classes of jobs submitted to a datacenter. Table 4 gives a quick summary of the lemmas used in this work, and the proofs can be found in appendices or [50].

6 OPTIMAL JOB SCHEDULING FOR STRAGGLERS

We investigate the optimal job scheduling with respect to mean sojourn time. Our analysis is carried out for different mapper types to address the straggler problem. We introduce two new schedulers *"pure-deterministic"* and *"pure-stochastic"* schedulers after fair job scheduler [5] which gradually improve the performance of the MapReduce job in case of having DTD completion time. The formulation of each scheduler is subsequently followed by its analytical results and then its simulation results.

We have employed COTSON full-system simulator [34] for multi-node cluster. Our COTSON implementation [25], using parallel discrete-event model, provides networking by a mediator for the system of up to two hundred nodes, half of which set low-performance (2.4GHz) and the rest set high-performance (3.0GHz), in our experiments. We have installed Hadoop 2.6.0 [47] on all nodes, and configured to support a heterogeneous cluster. The offline Wikipedia [48] is used to do *Grep (G)*, *WordCount (W)*, and *TeraSort (T)* operations. To derive the DTD-like characteristic curve of each node type for completion time, the mentioned operations are performed on randomly chosen pages of Wikipedia. The resulting DTD of completion time is used to obtain the scheduling results on real workloads. Note that, the stochastic behavior in our environment stem from data skew and network congestion.

6.1 Fair Job Scheduler

Hadoop fair job scheduler as the simplest way of load balancing sends the same amount of task to each mapper node with task arrival rate λ_i and service rate μ_i for N_m mapper nodes, i.e., if there is no data computation skew, we have:

$$\lambda_1 = \lambda_2 = \lambda_3 = \cdots = \lambda_{N_m} = \lambda/N_m. \quad (12)$$

The fair job scheduler is optimal when we have a completely homogeneous cluster and no data skew. The "no bottleneck system necessary condition" means that the mean job arrival rate should be less than the total service rate of the mapper nodes (i.e., $\lambda < \sum_{i=1}^{N_m} \mu_i$), which here is:

$$\lambda < N_m \mu_{min}. \quad (13)$$

A fair job scheduler makes the mean task arrival rates equal for all mappers as in Eq. (12), i.e., it gives each mapper the same amount of work. Fig. 4 plots the analytical results of fair job scheduler for a cluster with the same number (from 40 to 200) of low-performance (mean service time $\lambda_i = 0.8$) and high-performance ($\lambda_i = 1$) nodes. The MST of the fair job scheduler in Eq. (12), with respect to the total arrival rate and the number of heterogeneous mappers, always increases when the total arrival rate increases. However, the MST with respect to the number of nodes has a minimum, i.e., there is the optimal number of nodes, given a job arrival rate. However, increasing the number of nodes decreases the amount of tasks for each node, and the synchronization time to commit the whole job increases. Also, if the job arrival rate increases, this minimum number of mappers also increases.

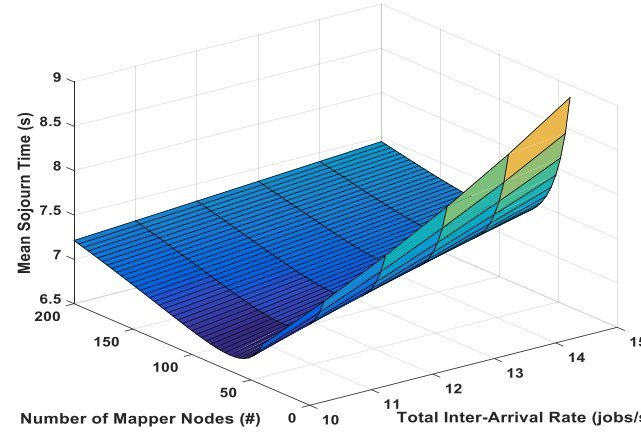


Fig. 4. MST with respect to the total arrival rate and the number of mapper nodes for the fair job scheduler.

The results of the real workloads shown in Fig. 5 are the MST of heterogeneous nodes (half low-performance half high-performance) with the fair job scheduler in terms of seconds. While they match with the analytical results in Fig. 4, they indicate that the response time or job completion time goes up rapidly with elevating the data size, and the system response time needs more time to merge the outputs, but the high-performance nodes finish their job much earlier than the others.

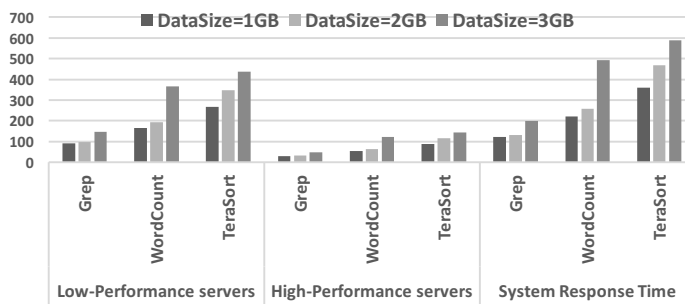


Fig. 5. The MST of heterogeneous nodes with the fair job scheduler.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2552516, IEEE Transactions on Cloud Computing

6.2 Pure-Deterministic Scheduler

We start by giving the definition of the optimal mapping with respect to MST, and then we deduct an equilibrium property that can be used to mathematically express the behavior of the different types of schedulers. In the case of having deterministic completion times (no distribution no ambiguity), we investigate pure-deterministic scheduler, and then in the case of having random completion times, we derive pure-stochastic scheduler.

Definition 3 (Optimal mapping based on MST): Optimal mapping for a known number of mappers (N_m) identifies the optimal task arrival rates ($\lambda_1,\lambda_2,\dots,\lambda_{N_m}$) making MST minimum. The constraint here is the mean job arrival rate to the system (λ). The service rate of each mapper is assumed to have a known distribution with a mean of μ_i. In mathematical terms, we have:

$$\begin{aligned} \min_{\underline{\lambda}}(MST) &= \min_{\underline{\lambda}} \int_0^\infty t \frac{\partial}{\partial t}\Big(\prod\nolimits_{i=1}^{N_m} F_{R_{M_i}}(t)\Big)\, dt \\ S.t.\, \lambda &= \textstyle\sum_{i=1}^{N_m} \lambda_i\,;\ \underline{\lambda} = \left[\lambda_1\lambda_2\lambda_3\ \dots\lambda_{N_m}\right]^T, \end{aligned} \tag{14}$$

where $F_{R_{M_i}}(t)$ is the CDF of the completion time of the i^{th} mapper as a function of μ_i and λ_i. The optimal solution of the mapping problem is derived by using the Lagrange Multipliers method and solving the following set of non-linear equations:

$$\nabla_{\underline{\lambda},\alpha}\left(MST - \alpha\left(\lambda - \textstyle\sum_{i=1}^{N_m}\lambda_i\right)\right) = 0, \tag{15}$$

where $\nabla_{\underline{\lambda},\alpha}(.)$ is the multi-dimensional gradient operator with respect to $\underline{\lambda}$ and α. We now give the following definition as a property for having optimal MST.

Definition 4 (Equilibrium Property): To optimize the mapping of the tasks, the set of non-linear equations derived from Eq. (15) have the *Equilibrium Property* (EP) with the linear constraint $\lambda = \sum_{i=1}^{N_m}\lambda_i$. Solving the set of non-linear equations gives the optimal solution for $\underline{\lambda}$. The equilibrium property can be expressed as:

$$\frac{\partial MST}{\partial \lambda_1} = \frac{\partial MST}{\partial \lambda_2} = \dots = \frac{\partial MST}{\partial \lambda_{N_m}}. \tag{16}$$

The above property cannot be easily solved or practically used to optimize MapReduce job scheduling. It is an N_m-dimensional non-linear optimization problem with a linear constraint. We will shortly present several sufficient conditions to satisfy the equilibrium property captured in Lemma 7. The following lemmas handle certain special cases where Equation (8) can be easily solved.

Lemma 4: *Given Def.2-4 for D/D/1 mapper nodes ($i = 1 \dots N_m$), equilibrium property can be expressed as follows:*

$$\frac{\mu_1}{\lambda_1} = \frac{\mu_2}{\lambda_2} = \dots = \frac{\mu_{N_m}}{\lambda_{N_m}}. \tag{17}$$

We call the above property the deterministic equilibrium property (D-EP), and the optimal scheduler for this property is pure-deterministic scheduler where λ_i is:

$$\lambda_i = \lambda\frac{\mu_i}{\sum_{j=1}^{N_m}\mu_j}\ ;i = 1 \dots N_m. \tag{18}$$

Proof. Eq. (17) can be derived by making the deterministic completion times of all tasks equal using:

$$\frac{\mu_1}{\lambda_1} = \frac{\mu_2}{\lambda_2} = \dots = \frac{\mu_{N_m}}{\lambda_{N_m}} = \frac{\sum_{j=1}^{N_m}\mu_j}{\sum_{j=1}^{N_m}\lambda_j}, \tag{19}$$

as $\lambda = \sum_{j=1}^{N_m}\lambda_j$, Eq. (18) is deduced. □

Fair queue, shortest queue-length first or μ-proportional (service-time proportional) scheduling make the queue-lengths equal and are the same as the pure-deterministic scheduling in Eq. (17) as we have:

$$Q_1 = Q_2 = \dots = Q_{N_m} \Leftrightarrow \rho_1 = \rho_2 = \dots = \rho_{N_m} \tag{20}$$

Considering $T_i \propto \lambda_i/\mu_i$, i.e., the offset of the DTD completion time has a linear relationship with λ_i/μ_i; we can say that pure-deterministic scheduling and shortest queue first scheduling are equivalent. This is because we have:

$$\frac{\lambda_1}{\mu_1} = \frac{\lambda_2}{\mu_2} = \dots = \frac{\lambda_{N_m}}{\mu_{N_m}} \Leftrightarrow T_1 = T_2 = \dots = T_{N_m}. \tag{21}$$

Fig. 6 plots the MST of the pure-deterministic scheduler with respect to the total arrival rate and the number of heterogeneous mapper nodes. The MST growth given by the total arrival rate is homographic versus the MST growth given by the number of mapper nodes is $O(ln(n))$ as the overhead rate of the whole job synchronization. Comparing Fig. 4 with Fig. 6, one can see that the pure-deterministic scheduler has a lower MST when the incoming job rate increases compared to the fair job scheduler. However, cloning the number of nodes, the trend of MST elevation in pure-deterministic scheduler is nearly the same as the fair job scheduler.

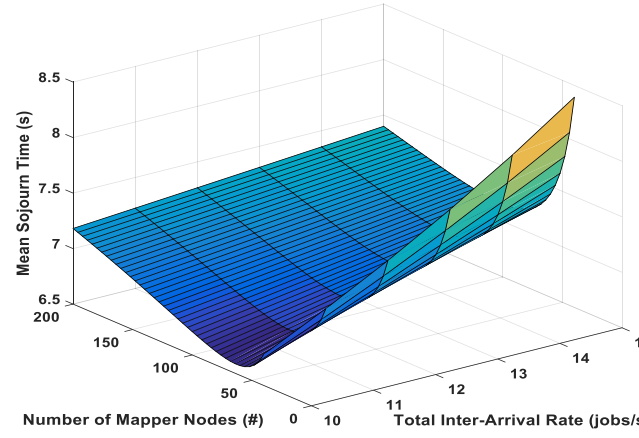


Fig. 6. MST with respect to the arrival rate and the number of mapper nodes for a pure-deterministic scheduler.

Fig. 7 shows real workload results for heterogeneous nodes with the pure-deterministic scheduler. One can observe from these results that pure-deterministic scheduler reduces the MST difference between low-performance and high-performance nodes rather than the fair job scheduler, but there is still a non-negligible MST skew among them.

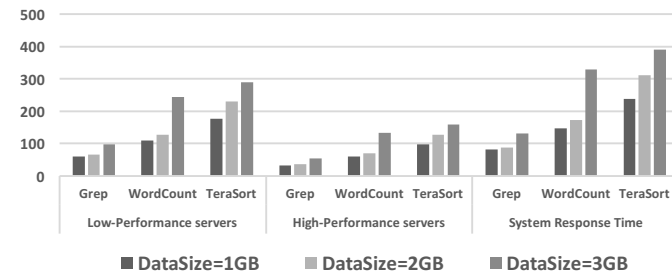


Fig. 7. The MST of heterogeneous nodes with the pure-deterministic scheduler.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2552516, IEEE Transactions on Cloud Computing

6.3 Pure-Stochastic Scheduler

When the completion times of the servers are stochastic by some distributions, we investigate pure-stochastic scheduler to get the optimal MST. The following lemma shows the closed-form formula to adjust pure-stochastic scheduler in the case of having Markovian mappers.

Lemma 5: *Given Def.2-4 for M/M/1 mapper nodes ($i = 1 \dots N_m$), the EP would be (see the proof in Appendix B):*

$$\mu_1 - \lambda_1 = \mu_2 - \lambda_2 = \cdots = \mu_{N_m} - \lambda_{N_m}. \quad (22)$$

We call above property stochastic equilibrium property (S-EP), and the optimal scheduler knobs for S-EP are the following optimal λ_is:

$$\lambda_i = \mu_i + \frac{\lambda - \sum_{j=1}^{N_m} \mu_j}{N_m} ; i = 1 \dots N_m. \quad (23)$$

Having no bottleneck system as the inequality $\lambda < \sum_{i=1}^{N_m} \mu_i$, there would be no bottleneck mapper node (i.e., $\lambda_i < \mu_i$), but we may have negative values for some λ_i, i.e., some jobs should be migrated from slow nodes (i) to other nodes. Thus, the no negative λ_i necessary condition is:

$$\sum_{i=1}^{N_m} (\mu_i - \mu_{min}) < \lambda. \quad (24)$$

Q-proportional (queue-length proportional) scheduling and pure-stochastic scheduling are equivalent, because $Q_i = \lambda_i/(\mu_i - \lambda_i)$, so we have:

$$\frac{\lambda_1}{Q_1} = \frac{\lambda_2}{Q_2} = \cdots = \frac{\lambda_{N_m}}{Q_{N_m}} \Leftrightarrow 1/(\mu_1 - \lambda_1) = \cdots = 1/(\mu_{N_m} - \lambda_{N_m}). \quad (25)$$

Fig. 8 plots the MST of the pure-stochastic scheduler with respect to the total arrival rate and the number of heterogeneous mapper nodes. The MST growth trend is similar to that of the other scheduler. Comparing Fig. 8 against Fig. 4 and Fig. 6, one can observe that the pure-stochastic scheduler has a lower MST under total arrival rate and the number of nodes.

Fig. 9 shows the simulation results for heterogeneous nodes (half low-performance; half high-performance) under the pure-stochastic scheduler. In this case, the MST of the nodes is nearly equal and consequently the MST of the system improves. Actually we have done for other percentages, and we have derived the similar results.

From the data size or job rate angle, pure-stochastic mapping has a lower MST compared to the fair job and pure-deterministic mappings shown in Fig. 10. The MST of the pure-stochastic strategy is lower than the others even under very low arrival rates.

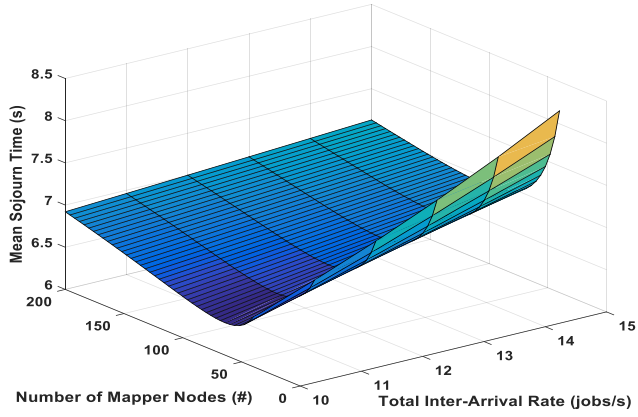


Fig. 8. MST with respect to the arrival rate and the number of mapper nodes for a pure-stochastic scheduler.

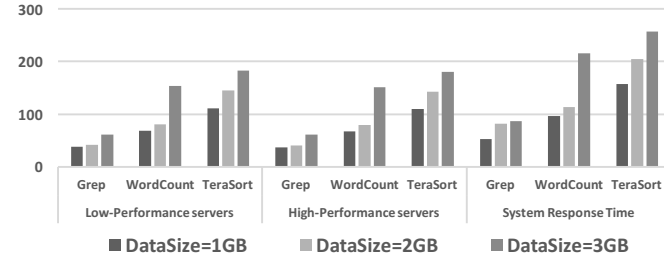


Fig. 9. The MST of heterogeneous nodes with the pure-stochastic scheduler.

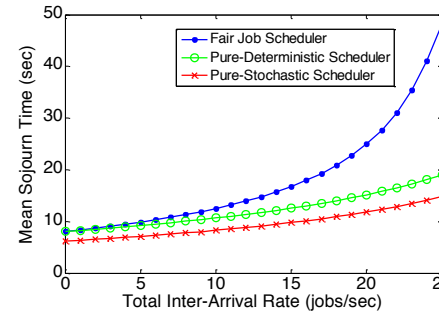


Fig. 10. MST comparison between fair job scheduler, pure-deterministic scheduler, and pure-stochastic scheduler.

6.4 More Advanced Schedulers

We investigate more sophisticated stochastic schedulers in terms of general job arrival rates and service rates. Based on the result of each lemma, we come up with its equilibrium property (EP), and then based on the EP, an algorithm is described that converges to the task arrival rates to the mappers, and can be used in an advanced scheduler based on that EP to extract task arrival rates.

Lemma 6: *Given Def.2-4 for G/M/1 mappers ($i = 1 \dots N_m$), the equilibrium property for optimal MST would be as follows (proof in Appendix C):*

$$\mu_1(1 - r_{0_1}) = \mu_2(1 - r_{0_2}) = \cdots = \mu_{N_m}\left(1 - r_{0_{N_m}}\right), \quad (26)$$

where r_{0_i} is the unique real root of $z = A_i^*[\mu_i(1 - z)]$ in (0,1) and A_i^* is the LST (Laplace–Stieltjes transform) of $A_i(t)$ (CDF of inter-arrival time). The solution of the MST for G/M/1 mappers is not trivial and the following algorithm shows how we can find the optimal arrival rates.

Algorithm 1: Given G/M/1 mapper nodes, Alg. 1 finds CDFs of optimal inter-arrival time (A_i) for the equilibrium property in Eq. (26).

Given Def.2, the LST of A_i is the function of the mean task inter-arrival time (λ_i). Let $A_i^* = g_i(\lambda_i)$. Also, $g_i^{-1}(A_i^*)$ exists in $(0, \lambda)$ where λ is total mean arrival rate of the system, e.g. in M/M/1 case we have $A_i^*(s) = \lambda_i/(\lambda_i + s)$ and $\lambda_i = A_i^* s/(1 - A_i^*)$. The cost function that we are interested to find the root by Secant Method [35] with small error ϵ is as follows:

$$f(\alpha) = \lambda - \sum_{i=1}^{N_m} g_i^{-1}\left(A_i^*(\alpha)\right) \quad (27)$$

1. Let $\alpha_0 = 0.5\mu_{min}, \alpha_1 = \mu_1\left(1 - g_1(\lambda/N_m)\right), n = 1$
2. $n = n + 1$
3. $\alpha_n = \alpha_{n-1} - \frac{f(\alpha_{n-1})(\alpha_{n-1} - \alpha_{n-2})}{f(\alpha_{n-1}) - f(\alpha_{n-2})}$
4. if $(|f(\alpha_n)| > \epsilon)$ then go to 2, else $\alpha = \alpha_n$.

Before deriving the corresponding expressions for other distributions, let us first give the sufficient conditions to achieve the optimal mapping.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2552516, IEEE Transactions on Cloud Computing

Lemma 7: *Given Def.2-4, if one of the following sufficient conditions is satisfied, the solution of the equations set of the sufficient condition and linear constraint ($\lambda = \sum_{i=1}^{N_m} \lambda_i$) is guaranteed to be optimal $\forall t \in \mathcal{R}^+, \forall i,j \in \{1,2,\dots,N_m\}$:*

$$F_{R_{M_j}} \frac{\partial}{\partial \lambda_i} F_{R_{M_i}}(t) = F_{R_{M_i}} \frac{\partial}{\partial \lambda_j} F_{R_{M_j}}(t) \quad (28)$$

$$F_{R_{M_i}}(t) = F_{R_{M_j}}(t) \quad (29)$$

$$M_{R_{M_i}}(t) = M_{R_{M_j}}(t);\ M_X(t) = E\{e^{tX}\} \quad (30)$$

$$\forall s \in \mathcal{R}, \forall i,j \in \{1,2,\dots,N_m\}: W_i^*(s) = W_j^*(s), \quad (31)$$

where $F_{R_{M_i}}(t)$ is CDF, $M_{R_{M_i}}(t)$ is moment-generating function, and $W_i^*(s)$ is the LST of the completion time of the i^{th} mapper node (see the proof in Appendix D). Also, for M/G/1 servers with arrival rate λ_i and service time distribution $g_i(t)$ ($E\{g_i(t)\} = \mu_i$), the sufficient condition to have optimal mapping by inspiring from Eq. (31) and Pollaczek–Khinchine formula [38] is:

$$\frac{(1-\lambda_i/\mu_i)s g_i(s)}{s-\lambda\left(1-g_i(s)\right)} = \frac{\left(1-\lambda_j/\mu_j\right)s g_j(s)}{s-\lambda\left(1-g_j(s)\right)} \quad (32)$$

where $g_i(s)$ and $W_i^*(s)$ are respectively the LST of $g_i(t)$ and the completion time of i^{th} mapper node. This conclusion can be easily derived from the Pollaczek-Khintchine transform. Also for G/G/1 mapper nodes with LST of inter-arrival time distribution $A_i^*(s)$ and LST of service time distribution $B_i^*(s)$, the sufficient condition to have optimal mapping, by inspiring from Eq. (31), is:

$$W_{q_i}(0)B_i^*(s) + \left(1-W_{q_i}(0)\right)B_i^*(s)W_{q_i}^*(s) = W_{q_j}(0)B_j^*(s) + \left(1-W_{q_j}(0)\right)B_j^*(s)W_{q_j}^*(s), \quad (33)$$

whose $W_{q_i}^*(s)$ is the LST of waiting time distribution and satisfies Lindley's equation $W_{q_i}(t) = -\int_0^\infty W_{q_i}(x)dU_i(t-x)$ such that $U_i(x) = \int_{max\,(0,x)}^{\infty} B_i(t)dA_i(t-x)$.

Corollary 8: *Given Def.2-4 of MST problem, if we have the moments of the distribution of task inter-arrival time ($A(t)$), the optimal mapping exists.*

Proof. Similar to Eq. (26) and Eq. (27) you can assume the following relation between the LST of task inter-arrival times ($A_i^*(s)$) and the LST of completion times ($W_i^*(s)$):

$$W_i^*(s) = f\left(A_i^*(s)\right);\ \forall s \in \mathcal{R}, \forall i \in \{1,2,\dots,N_m\}, \exists f \quad (34)$$

At this point, we just need to adjust the moments of inter-arrival times to satisfy Eq. (31) and the constraint while we have:

$$1 + \frac{s}{1!}W_i^{*\prime}(0) + \frac{s^2}{2!}W_i^{*\prime\prime}(0) + \cdots = f(1 + \frac{s}{1!}A_i^{*\prime}(0) + \frac{s^2}{2!}A_i^{*\prime\prime}(0) + \cdots), \quad (35)$$

As the number of unknown variables ($A_i^*(s)$) and the related Eq. (35) and the linear constraint are equal, the optimal mapping can be explored. □

In the next lemma, we obtain results suitable to be employed in practice, because it is difficult to get the best mapping of any inter-arrival time distribution. Instead, we can try making the first moment (mean) of the completion time distribution equal. It is equivalent to make the first terms of Taylor's series of the LST of completion time distribution ($W_i'(s=0)$) equal. The lemma below gives the boundaries of the MST, as well as an approximate solution, which can be very useful in most of the practical cases.

Lemma 9: *Given Def.2-4 the lower bound and upper bound of MST can be found as follows (proof in Appendix E):*

$$max\left(E\{R_{M_1}\}, E\{R_{M_2}\}, \dots, E\left\{R_{M_{N_m}}\right\}\right) \le MST \le \sum_{i=1}^{N_m} E\{R_{M_i}\} \quad (36)$$

The equilibrium property of lower bound can be expressed as:

$$E\{R_{M_1}\} = E\{R_{M_2}\} = \cdots = E\left\{R_{M_{N_m}}\right\}, \quad (37)$$

that we call it "*means equilibrium property*" (M-EP) and approximate optimal scheduler can perform the M-EP. But the "*derivative of means equilibrium property*" (DM-EP) optimizes MST upper bound:

$$\frac{d}{d\lambda_1}E\{R_{M_1}\} = \frac{d}{d\lambda_2}E\{R_{M_2}\} = \cdots = \frac{d}{d\lambda_{N_m}}E\left\{R_{M_{N_m}}\right\}. \quad (38)$$

M-EP approximately optimizes MST and gives sub-optimal mean arrival rates ($\underline{\lambda}$) by making the first moments of completion times equal. The only degrees of freedom are the mean arrival rates ($\underline{\lambda}$), so in an approximate approach, we can only have the M-EP and not the other higher moments equal. This approximation is correct when the other moments of the completion time of the tasks are negligible or the deviation in Eq. (30) is insignificant, i.e., when there is no bottleneck server in the system. In fact, the gap between the optimal solution and this approximate solution is insignificant. We now present the algorithms to have M-EP for DTD/DTD/1, M/G/1 and G/G/1 mapper nodes. MST of a DTD mapper can be expressed as $T_i + 1/(\mu_i - \lambda_i)$, where $T_i = D\lambda_i/\mu_i$ for some constant D. Consequently the M-EP for DTD mapper nodes would be:

$$\forall i,j \in \{1,2,\dots,N_m\}: T_i + 1/(\mu_i - \lambda_i) = T_j + 1/(\mu_j - \lambda_j) \quad (39)$$

Algorithm 2: Given DTD mapper nodes Alg. 2 will find sub-optimal arrival rates (λ_i) for the equilibrium property in Eq. (39), where it is just needed to iterate Alg.1 with the cost function as:

$$f(\alpha) = \lambda - \sum_{i=1}^{N_m} \frac{(\alpha+D)\mu_i - \sqrt{\left(\mu_i(\alpha-D)\right)^2 + 4D\mu_i}}{2D} \quad (40)$$

M-EP for M/G/1 mapper nodes ($\forall i,j \in \{1,\dots,N_m\}$) is:

$$\frac{\lambda_i\left(\sigma_{B_i}^2 + \frac{1}{\mu_i^2}\right)}{2\left(1-\frac{\lambda_i}{\mu_i}\right)} + \frac{1}{\mu_i} = \frac{\lambda_j\left(\sigma_{B_j}^2 + \frac{1}{\mu_j^2}\right)}{2\left(1-\frac{\lambda_j}{\mu_j}\right)} + \frac{1}{\mu_j}. \quad (41)$$

where λ_i, μ_i, and $\sigma_{B_i}^2$ are respectively the i^{th} mapper's arrival rate, mean service rate, and variance of service rate.

Algorithm 3: Given M/G/1 mappers, Alg. 3 finds sub-optimal arrival rates (λ_i) for the equilibrium property in Eq. (41), where it is just needed to iterate over Alg.1 with the following cost function:

$$f(\alpha) = \lambda - \sum_{i=1}^{N_m} 1/\left(\frac{\sigma_{B_i}^2 + \frac{1}{\mu_i^2}}{2\left(\alpha - \frac{1}{\mu_i}\right)} + \frac{1}{\mu_i}\right). \quad (42)$$

Using Kingman's approximation [46], M-EP for G/G/1 is:

$$\forall i,j \in \{1,2,\dots,N_m\}$$
$$\frac{\lambda_i}{\mu_i}\left(\frac{\sigma_{A_i}^2}{\lambda_i^2} + \frac{\sigma_{B_i}^2}{\mu_i^2}\right) + \frac{1}{\mu_i} = \frac{\lambda_j}{\mu_j}\left(\frac{\sigma_{A_j}^2}{\lambda_j^2} + \frac{\sigma_{B_j}^2}{\mu_j^2}\right) + \frac{1}{\mu_j} \quad (43)$$

Algorithm 4: Given G/G/1 mappers for the defined problem in Def. 2-4, Alg. 4 finds the sub-optimal mean arrival

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2552516, IEEE Transactions on Cloud Computing

rates (λ_i) for the equilibrium property in Eq. (43), where it is just needed to iterate Alg.1 with knowing $\forall i: \sigma_{A_i}^2/\lambda_i^2 = \sigma_A^2/\lambda^2$ by the cost function as follows:

$$f(\alpha) = \lambda - \sum_{i=1}^{N_m} \mu_i/(1 + \frac{1}{\mu_i}\left(\frac{\sigma_A^2}{\lambda^2} + \frac{\sigma_{B_i}^2}{\mu_i^2}\right)/\left(2\alpha - \frac{2}{\mu_i}\right)). \quad (44)$$

In the following graphs (Fig. 11a-11d) we use our DTD model of the map phase for a CPU-bound job (when T_i as the deterministic coefficient is big, i.e. $D \gg 1$) and a memory-bound job (when T_i is small, i.e. $D \approx 0$). The graphs show the comparison between different schedulers as mentioned before with respect to total mean arrival rate and number of heterogeneous mapper nodes.

When the target MapReduce job is memory-bound, it is more stochastic and the deterministic coefficient for this type of job is close to zero. For memory-bound jobs, the optimal scheduler that tries to find λ_is minimizing MST is nearly coincident with pure-stochastic, M-EP, and DM-EP schedulers. For CPU-bound jobs, the optimal scheduler is nearly coincident with pure-deterministic and M-EP schedulers. As a result, M-EP is a good approximation of optimal solution in this model. When the input job is CPU-bound, the pure-stochastic scheduler cannot track the optimal completion time well. Other schedulers, in some cases, outperform the non-negative optimal scheduler, because they output negative λ_i, i.e. job migration. The non-negative optimal scheduler always gives positive non-migratory λ_i, but in a non-negative region the optimal scheduler outperforms the others. The same scenario in simulation results (Fig. 12) indicates that the pure-stochastic scheduler can make all mappers' completion times nearly equal, and the other schedulers' deviation is higher. Since MST is a function of the slowest node, here M-EP is near to the pure-stochastic scheduler, and it is on the average 129% more effective than the fair job scheduler and 51% better than ideal deterministic approaches.

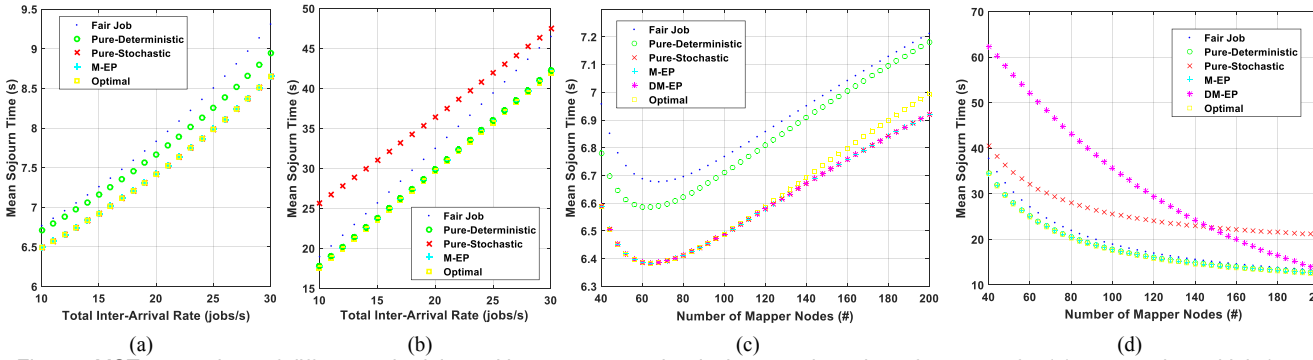


Fig. 11. MST comparison of different schedulers with respect to total arrival rate and number of mappers for (a) memory-bound job ($\boldsymbol{D \approx 0}$), (b) CPU-bound job ($\boldsymbol{D = 100}$), (c) memory-bound job ($\boldsymbol{D \approx 0}$), and (d) CPU-bound job ($\boldsymbol{D = 100}$).

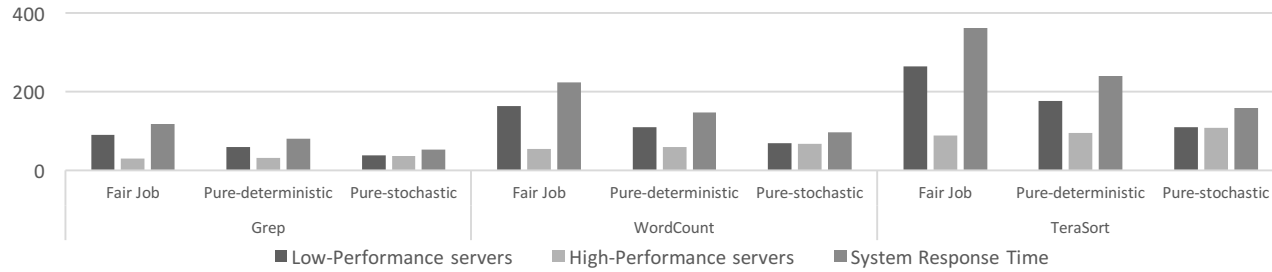


Fig. 12. The MST of heterogeneous nodes with different schedulers.

7 Optimal Number of Mappers

It is interesting to note that there is the optimal number of mappers that makes the mean sojourn time minimum. Because, when we increase the number of mappers in the system, the amount of scheduled task for each mapper decreases. As a result, the smaller jobs can be completed sooner. However, blindly increasing the number of mappers is not a good strategy, because more mappers need more time to be synced. Therefore, there is a tradeoff between smaller job completion time and higher sync time. A general closed form solution for the optimal number of mappers is very difficult. The following definition gives the description of the optimal number of mapper nodes, and the next lemma solves the problem.

Definition 5 *(Optimal Number of Mappers): Given Def.2-4 the optimal number of mappers is an integer number that makes the mean sojourn time minimum. If $F(t)$ is CDF of completion time of any map task, then we have:*

$$\min_{N_m}(MST) = \min_{N_m}\left\{N_m \int_0^\infty t f(t) F^{N_m-1}(t) dt\right\} \quad (45)$$

Lemma 10: *Given Def.5 for M/M/1 mapper nodes with μ as the service rate and λ as the total arrival rate to the system, the optimal number of mapper nodes to minimize MST is equal to the minimum value of the following function with respect to n:*

$$f(n) = \frac{1}{\mu - \lambda/n}\sum_{i=1}^{n}\left((-1)^{i+1}\frac{n!}{(n-i)!\,(i-1)!}\right) \quad (46)$$

Proof. In Eq. (10), we make all $\mu_{j_k} - \lambda_{j_k} = \mu - \lambda/N_m$:

$$MST_{M/M/1} = \sum_{i=1}^{N_m}\left\{(-1)^{i+1}\sum_{\forall\{j_1,\dots,j_i\}\subset\{1,2,\dots,N_m\}}\frac{1}{\sum_{k=1}^{i}(\mu_{j_k} - \lambda_{j_k})}\right\}$$
$$= \sum_{i=1}^{N_m}\left\{(-1)^{i+1}\sum_{\forall\{j_1,\dots,j_i\}\subset\{1,2,\dots,N_m\}}\frac{1}{i(\mu - \lambda/n)}\right\}$$
$$= \frac{1}{\mu - \lambda/N_m}\sum_{i=1}^{N_m}\left\{(-1)^{i+1}\frac{1}{i}\binom{N_m}{i}\right\},$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2016.2552516, IEEE Transactions on Cloud Computing

when N_m is the parameter, it is equivalent with Eq. (46). □

Fig. 13 shows the existence of a knee point for the optimal number of mapper nodes in a heterogeneous datacenter. In this result, we have different nodes with uniform service rates of 1, 0.9, 0.8, and 0.7. The number of nodes is varied between 40 and 200. Also, the arrival rate is 10 and offset time is set to 3.5.

Lemma 11: *Given Def.5 for homogeneous mapper nodes with μ as the service rate and λ as the total arrival rate to the system, the optimal number of mapper nodes is equal to the minimum of the following function with respect to n:*

$$f(n)=\frac{1}{\mu-\frac{\lambda}{n}}\left[H_n+\left(\left(\sum_{i=1}^{n}\binom{n}{i}(-1)^{i+1}\sum_{j=1}^{i}\binom{i}{j}\frac{(j-1)!}{i^{j+1}}\right)-H_n\right)\frac{\lambda}{n\mu}\right] \quad (47)$$

where $H_n=\sum_{i=1}^{n}1/i$.

Proof. This is a direct result of the approximation of MST [36] based on the interpolation of lower bound and upper bound for low/medium/high traffic and it can be used for a general form distribution. We can find the optimal number of mappers by refining it as Eq. (47). □

Lemma 12: *Given Def.5 for mapper nodes with K different service rates, prices, and numbers as (μ_1,P_1,N_1), (μ_2,P_2,N_2), … (μ_K,P_K,N_K), the total arrival rate to the system as λ and the budget constraint as $Budget=\sum_{i=1}^{K}P_iN_i$ where $N_m=\sum_{i=1}^{K}N_i$, the optimal number of mapper nodes of each type to minimize MST is equal to the minimum of the following function with respect to $N_1,N_2,\dots,N_K$:*

$$f(N_1,\dots,N_K)=\frac{\sum_{i=1}^{K}N_i}{\sum_{i=1}^{K}(N_i\mu_i)-\lambda}\sum_{i=1}^{N_m}\left((-1)^{i+1}\binom{N_m}{i}/i\right) \quad (48)$$

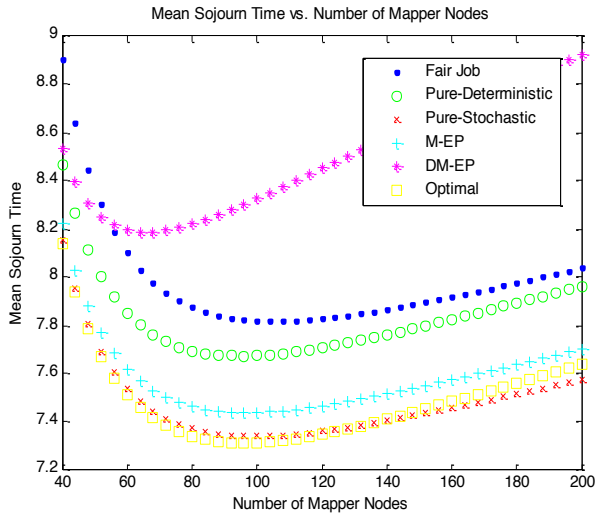


Fig. 13. Comparison of optimal number of heterogeneous mapper nodes based on MST with $\boldsymbol{D=3.5}$.

The proof is straightforward like the proof of Lemma 10. One can derive a similar corollary for the general case with respect to Lemma 11. Fig. 14 (a,b,c) gives the optimal number of low-performance (LP) and high-performance (HP) servers in a heterogeneous datacenter with a fixed amount of budget for low/medium/high traffic. The service rate ratio and price ratio of LP to HP are based on [37]. The budget is changed between 2000 and 40000 where price/service rate for LP servers and HP servers are respectively 400/1 and 800/1.25, and job arrival rates for low/medium/high traffic are respectively 1/5/25.

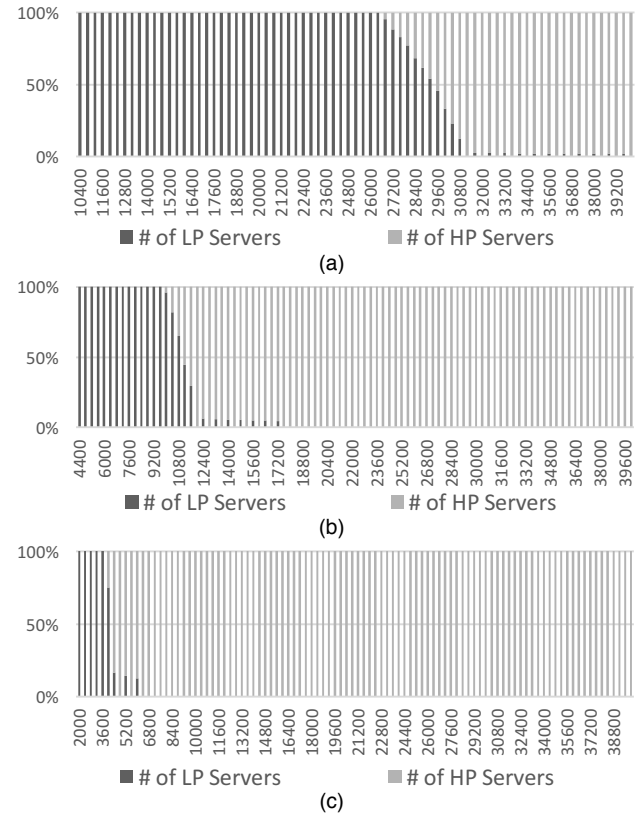


Fig. 14. Stack ratio of LP servers to HP servers with respect to Budget in (a) high, (b) medium, and (c) low traffic.

8 Related Work

The mathematical modeling of MapReduce framework has been investigated in several studies [15,16,17,18]. The main difference between these models and the current study is that the prior models are not sufficiently rigorous in handing the stragglers problem. Also, most of the published studies assume deterministic execution times for mappers and reducers. Li et al. [15] introduce an analytical model for the I/O cost, the number of I/O requests, and the startup cost in Hadoop. They also propose a hash-based mechanism to allow incremental processing and in-memory processing of frequent keys investigating the best merge factor for the jobs larger than memory size.

Ananthanarayanan et al. [11] show that the stragglers can slow down small jobs by as much as 47%. They propose a system called Dolly for cloning small jobs. They also claim that a delay assignment can improve resource contention initiated by cloning. However, their method does not work for stragglers of large tasks. Ahmad et al. [16] propose an analytical model of MapReduce to minimize the execution time and find the optimal map granularity. In comparison, Karloff et al. [17] present a performance model to estimate the cost of map and reduce functions in MapReduce. However, the model assumes all mappers finish at the same time and they do not consider the stochastic behavior of the execution times of the mappers and reducers. Krevat et al. [18] proposed a simple analytical model for MapReduce and compared the performance of MapReduce with other similar frameworks in good conditions.

LATE [7] tries to optimize MapReduce job performance

in a heterogeneous cluster by restarting slow tasks in fast mapper nodes. Tarazu [12] addresses the poor performance of MapReduce in heterogeneous clusters and shows that the traffic contention between the remote tasks is the main problem in heterogeneous clusters. Motivated by this, they propose a communication-aware and dynamic load balancing technique to reduce the network traffic contention between the remote tasks and the shuffling stage. SkewTune [19] interactively manages the skew in non-uniform input data at runtime. It finds an idle server in the cluster and assigns a slow task to that node. Ananthanarayanan et al. [8] discuss the main causes of the outliers (stragglers) and propose Mantri to restart or duplicate the task at the beginning of its lifetime. Scarlett [13] replicates popular blocks across different memory components to reduce interference with the running jobs.

Detailed analysis of various workloads [9] such as OpenCloud, M45 and WebMining, show highest task duration (stragglers runtime) to median task duration has a nearly long-tailed distribution. Similarly, Chen et al. [10] evaluate the task lengths for Cloudera and Facebook workloads and derive similar conclusions. Tan et al. [20] propose a coupling scheduler in MapReduce, and show its performance has a lower exponent as a power-law distribution than FIFO and fair schedulers. They extend their work [21] by upgrading reducers as a multi-server queue. Lin et al. [22] address the challenge of overlapping map and shuffle in MapReduce. They demonstrate that the optimal solution is NP-hard in the offline mode, and suggest MaxSRPT and SplitSRPT schedulers for the online mode, reaching optimal scheduling. Condie et al. [23] modify the framework to support pipelining between map/shuffle and reduce phases. There are some compiler-based structures [24] for SQL-like queries in the MapReduce framework to speedup computations on large data sets. They study a computational DAG (directed acyclic graph) representation of large queries and require multiple rounds of MapReduce for their optimizations to be effective. A deadline-aware scheduler has been proposed by Li et al. [26] to practically address scheduling of deadline-sensitive jobs in a satisfactory range.

9 Concluding Remarks

Targeting MapReduce applications, in this paper, we model the service time of mapper nodes as a single queue with the delayed tailed distribution (DTD), and also show that their completion time has a similar behavior. Next, using this analytical result, we model the map phase of a MapReduce job and formulate the mean sojourn time (MST) at a reducer node by means of task arrival rates and service rates of mapper nodes. MST is a potential metric for optimizing end-to-end delay in a MapReduce framework. Based on different types of inter-arrivals and service rates, we optimize the MST parameter and investigate the advanced schedulers based on the equilibrium property for different types of queues in a heterogeneous datacenter. Our results show that the optimal stochastic scheduler is better than any deterministic scheduler, and it gives the optimal mapping of the job and the number of mappers.

REFERENCES

[1] T. Gunarathne, W. Tak-Lon, J. Qiu, G. Fox, "MapReduce in the clouds for science," 2nd IEEE Conference on Cloud Computing Technology and Science, CloudCom-2010, pp. 565–572, 2010.

[2] R. Ananthanarayanan, K. Gupta, P. Pandey, H. Pucha, P. Sarkar, M. Shah, R. Tewari, "Cloud analytics: Do we really need to reinvent the storage stack?," In Proc. of the HotCloud Workshop, San Diego, 2009.

[3] R. L. Grossman, "The Case for Cloud Computing," IT Professional, vol.11, no.2, pp.23-27, March-April 2009.

[4] J. Dean, S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," In Proc. of the 6th Symposium on Operating Systems Design and Implementation, San Francisco CA, 2004.

[5] K. Shvachko, H. Huang, S. Radia, R. Chansler, "The hadoop distributed file system," in Proc. of the 26th IEEE (MSST2010) Symposium on Massive Storage Systems and Technologies, 2010.

[6] M. Isard, M. Budiu, Y. Yu, A. Birrell, D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," In Proc. of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007 (EuroSys '07), 2007.

[7] M. Zaharia, A. Konwinski, A.D. Joseph, R. Katz, I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments," In USENIX OSDI, 2008.

[8] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, E. Harris, "Reining in the outliers in map-reduce clusters using Mantri," In Proc. of the 9th USENIX OSDI Symposium, 2010.

[9] K. Ren, Y. Kwon, M. Balazinska, B. Howe, "Hadoop's adolescence: a comparative workload analysis from three research clusters," Tech. Report UW-CSE-12-06-01, University of Washington, 2012.

[10] Y. Chen, S. Alspaugh, R. H. Katz, "Design insights for MapReduce from diverse production workloads," Technical Report UCB/EECS-2012-17, EECS Dep., University of California, Berkeley, 2012.

[11] G. Ananthanarayanan, A. Ghodsi, S. Shenker, I. Stoica, "Effective straggler mitigation: attack of the clones," In Proc. of the 10th Symp. on Networked Systems Design and Implementation (NSDI), 2013.

[12] F. Ahmad, S. T. Chakradhar, A. Raghunathan, T. N. Vijaykumar, "Tarazu: optimizing mapreduce on heterogeneous clusters," In Proc. of the 17th ASPLOS Conf., 2012.

[13] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, E. Harris, "Scarlett: Coping with Skewed Popularity Content in MapReduce Clusters," In ACM EuroSys, 2011.

[14] H. Karloff, S. Suri, S. Vassilvitskii, "A model of computation for MapReduce," In Proc. ACM-SIAM Sympos. Discrete Algorithms (SODA), pp. 938–948, 2010.

[15] B. Li, E. Mazur, Y. Diao, A. McGregor, P. Shenoy, "A Platform for Scalable One-Pass Analytics using MapReduce," In Proceedings of ACM SIGMOD Conf., 2011.

[16] X. Yang, J. Sun, "An analytical performance model of mapreduce," In Proc. of Cloud Computing and Intelligence Systems (CCIS), 2011.

[17] X. Lin, Z. Meng, C. Xu, M. Wang, "A practical performance model for hadoop mapreduce," In Proc. Of CLUSTER Workshops, 2012.

[18] E. Krevat, T. Shiran, E. Anderson, J. Tucek, J.J. Wylie, G.R. Ganger, "Applying Performance Models to Understand Data-intensive Computing Efficiency," Technical Report CMU-PDL-10-108, Carnegie Mellon University, Pittsburgh, 2010.

[19] Y. Kwon, M. Balazinska, B. Howe, J. Rolia, "SkewTune: Mitigating skew in MapReduce applications," In Proc. of SIGMOD Conf., pages 25–36, 2012.

[20] J. Tan, X. Meng, L. Zhang, "Delay tails in MapReduce scheduling," in Proc. of the SIGMETRICS/PERFORMANCE Conf., 2012.

[21] J. Tan, Y. Wang, W. Yu, and L. Zhang, "Non-work-conserving effects in MapReduce: diffusion limit and criticality," In Proc. of the 14th ACM SIGMETRICS, Austin, Texas, USA, 2014.

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

[23] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmleegy, and R. Sears, "Mapreduce online," In Proc. of NSDI, 2010.

[24] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, J. Zhou, "Scope: Easy and efficient parallel processing of massive data sets," In Proc. of VLDB, 2008.

[25] D. Z. Tootaghaj, F. Farhat, M. Arjomand, P. Faraboschi, M. T Kandemir, A. Sivasubramaniam, C. R. Das, "Evaluating the combined impact of node architecture and cloud workload characteristics on network traffic and performance/cost," In Proc. of IISWC, pp 203–212, Oct. 2015.

[26] S. Li, S. Hu, S. Wang, L. Su, T. Abdelzaher, I. Gupta, R. Pace. "WOHA: Deadline-Aware MapReduce Workflow Scheduling Framework over Hadoop Cluster," In Proc. of 34th International Conference on Distributed Computing Systems (ICDCS), 2014.

[27] V.A. Saletore, K. Krishnan, V. Viswanathan, M.E. Tolentino, "HcBench: Methodology, Development, and Characterization of a Customer Usage Representative Big Data/Hadoop Benchmark," IEEE International Symposium on Workload Characterization, 2013.

[28] M. A. Marsan, G. Chiola, "On Petri Nets with Deterministic and Exponentially Distributed Firing Times," Advances in Petri Nets, LNCS, vol. 266, Springer, pp. 132-145, 1987.

[29] A. Feldmann, W. Whitt, "Fitting mixtures of exponentials to long-tail distributions to analyze network performance models", In Proc. of IEEE INFOCOM, 1997.

[30] J. Li , Y.S. Fan, M.C. Zhou, "Performance modeling and analysis of workflow," IEEE Trans. on Systems, Man, Cybernetics—Part A: Systems and Humans, vol. 34, no. 2, pp. 229-242, 2004.

[31] R. B. J. T. Allenby and A. B. Slomson, "How to count: An introduction to combinatorics," 2nd ed. CRC Press, pp. 51-60, 2011.

[32] B. Kemper, M. Mandjes, "Mean sojourn times in two-queue fork-join systems: bounds and approximations," OR Spectrum 34(3), 2012.

[33] A. S. Lebrecht, W. J. Knottenbelt, "Response Time Approximations in Fork-Join Queues," in Proc. of the 23rd UK Performance Engineering Workshop (UKPEW), July, 2007.

[34] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, D. Ortega, "COTSon: infrastructure for full system simulation," ACM SIGOPS Operating Systems Review, v.43 n.1, Jan 2009.

[35] A. Kaw, E. Kalu, "Numerical Methods with Applications," Holistic Numerical Methods Institute, Dec. 2008.

[36] A. Makowski, S. Varma, "Interpolation approximations for symmetric fork-join queues," Performance Evaluation, vol.20, 145-165, 1994.

[37] V. J. Reddi, B. Lee, T. Chilimbi, K. Vaid, "Web Search Using Mobile Cores: Quantifying and Mitigating the Price of Efficiency," in Proc. of ISCA, 2010.

[38] D. Gross, J. F. Shortle, J. M. Thompson, C. M. Harris, "Fundamentals of Queueing Theory," 4th ed. John Wiley&Sons, 2008.

[39] S. Kavulya, J. Tany, R. Gandhi, P. Narasimhan, "An analysis of traces from a production mapreduce cluster," In 10th IEEE/ACM CCGrid, pages 94–103, 2010.

[40] Y. Chen, S. Alspaugh, and R. H. Katz, "Design insights for mapreduce from diverse production workloads," Technical Report UCB/EECS-2012-17, EECS Dept, University of California, Berkeley, Jan 2012.

[41] G. Ananthanarayanan, M. Hung, X. Ren, I. Stoica, A. Wierman, and M. Yu, "GRASS: Trimming Stragglers in Approximation Analytics," In Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2014.

[42] S. Nadarajah, S. Kotz, "The generalized Pareto sum," HYDROLOGICAL PROCESSES 22, doi:10.1002/hyp.6602, pp.288–294, 2008.

[43] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "DCTCP: Efficient packet transport for the commoditized data center," In SIGCOMM, 2010.

[44] J. Dean and L. A. Barroso, "The tail at scale," Communications of the ACM, Vol. 56 No. 2, pp.74-80, 2013.

[45] E. Lazowska, J. Zahorjan, G. Graham, K. Sevcik, "Quantitative System Performance: Computer System Analysis Using Queueing Network Models," Prentice-Hall, Englewood Cliffs, NJ, 1984.

[46] J. F. C. Kingman, "The single server queue in heavy traffic," Proc. Camb. Phil. Soc. 57, pp. 902-904, 1961.

[47] https://hadoop.apache.org/docs/stable/.

[48] http://en.wikipedia.org/wiki/Wikipedia:Database_download.

[49] A. Clauset , C. R. Shalizi , M. E. J. Newman, "Power-Law Distributions in Empirical Data," SIAM Review, vol.51 no.4, pp.661-703, 2009.

[50] F. Farhat, D. Z. Tootaghaj, A. Sivasubramaniam, M. Kandemir, C. R. Das, "Modeling and Optimization of Straggling Mappers," Technical Report CSE-14-006, Pennsylvania State University, 2014.

Farshid Farhat received his B.S. and M.S. degrees in electrical engineering from Sharif University of Technology. He is Ph.D candidate at the EECS department, Pennsylvania State University. His current research interests include resource allocation in parallel and distributed systems.

Diman Zad Tootaghaj received her B.S. and M.S. degrees in electrical engineering from Sharif University of Technology. She is Ph.D candidate at the EECS department, Pennsylvania State University. Her current research interests include perfromance evaluation, distributed systems, and resource allocation.

Yuxiong He received her B.Eng in computer engineering from NTU, and. her Ph.D. in computer science from Singapore-MIT Alliance. She is a researcher in cloud computing group of MSR. Her research interests include resource management, algorithms, modeling and performance evaluation of parallel and distributed systems.

Anand Sivasubramaniam is a professor at Pennsylvania State University. His research interests are in computer architecture, operating systems, and performance evaluation. He has been on the editorial boards of IEEE Transactions on Parallel and Distributed Systems and IEEE Transactions on Computers, and program committees of several conferences. He is a fellow of the IEEE and an ACM Distinguished Scientist.

Mahmut Kandemir is a professor in the Computer Science and Engineering Department at The Pennsylvania State University. His research interests are in optimizing compilers, embedded systems, I/O and high performance storage, and power-aware computing. He has served in the program committees of 40 conferences and workshops.

Chita R. Das received his Ph.D. in computer science from the University of Louisiana, Lafayette, in 1986. Since 1986, he has been with the Pennsylvania State University where he is currently a distinguished professor in the EECS Department. His research interests include parallel and distributed computing, performance evaluation and faulttolerant computing.