Modeling and Optimization of Straggling Mappers

F. Farhat, D. Zad Tootaghaj, A. Sivasubramaniam, M. T. Kandemir, C. R. Das
Computer Science and Engineering, Pennsylvania State University, University Park, USA
{fuf111,dxz149,anand,kandemir,das}@cse.psu.edu

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 mappers increases, the map phase can take much longer than expected. This paper analytically shows that stochastic behavior of mapper nodes has a negative effect on the completion time of a MapReduce job, and continuously increasing the number of mapper nodes can degrade the overall performance. We analytically capture the effects of stragglers (delayed mappers) problem on the performance. Based on observed delayed exponential CDF of response time of mappers, we then model the map phase by means of hardware, system and application parameters. Mean sojourn time (MST), the time needed to sync the completed map tasks at one reducer, is mathematically formulated. Following that, we optimize MST by finding the task inter-arrival rate to each mapper node. Optimal mapping problem leads to equilibrium property investigated for many types of inter-arrival and service time distributions in a heterogeneous datacenter. Our experimental results show the performance and important parameters of the different types of schedulers. 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.**

I. Introduction

MapReduce has become a popular paradigm for structuring large scale parallel computations in the datacenter. By decomposing the 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 embarrassingly parallel, the inefficiencies in MapReduce execution manifest at the boundaries between the phases as data exchanges and synchronization stalls to 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 (in the mappers), 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 the MapReduce execution, (ii) be able to quantify this slow-down 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 -- determine 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), adjust the scheduling policies, run redundant versions of the tasks based on the trade-offs between estimated wait times and additional resources mandated, run a MapReduce computation with a budgetary (performance, power, cost) constraint, etc.

However, there are no rigorous analysis tools available today with these capabilities for modeling and understanding the straggler problem in MapReduce computations for the purposes listed above. This paper intends to fill this critical void by presenting a novel analytical model for capturing the waiting time at the end of the Map phase due to any straggling mappers. 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 a delayed exponential distribution can be used to capture the service time of the map tasks at a given node. We then show that with such service times at each node, the aggregate completion time of mapper tasks across all the nodes of the cluster also follows a delayed exponential distribution. This is validated (less than 5% least square error) against a spectrum of mapper completion times of 10 production workloads published in prior research.
- With this result, we develop a closed-form queuing model of the time expended (the Mean Sojourn Time) before a reducer node can begin its part of the computation, i.e., waiting time for all mappers to finish. Parameterized by the arrival rate of the mappers, the delayed exponential service times and the number of nodes, this model helps

conveniently study the impact of different parameters -- whether job characteristics, hardware capabilities or system operation -- 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 a use case. We show how the model can be used to schedule map tasks on different (possibly heterogeneous) nodes of a datacenter cluster to reduce the Mean Sojourn Time. This is demonstrated to be much more effective than how the JobTracker does it today in the Hadoop distribution.

II. Background and Related Work

In this section we describe how MapReduce [4] works, and explain the *straggler* problem that affects the efficiency of the framework negatively. We also discuss the related work.

A. MapReduce Framework

MapReduce framework [4] is a programming environment 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 and low job completion times. Open-source Hadoop MapReduce is a fault-tolerant scalable implementation built upon Hadoop file system (HDFS) [5].

Figure 1 shows a high-level view of the MapReduce framework. A job in this framework arrives with some 'mean rate', and is partitioned into 'map' tasks. More specifically JobTracker module in Hadoop assigns map/reduce tasks to TaskTracker nodes. Each map task tracker node (called mapper node) has threads to perform the map tasks (called mappers). 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. Each reducer node may receive values with the same intermediate key assigned to that node. Each reducer node employs reduce task trackers (called reducers) to compute and merge the received intermediate values. After finishing the reduce phase, the final values are merged into the HDFS storage again.

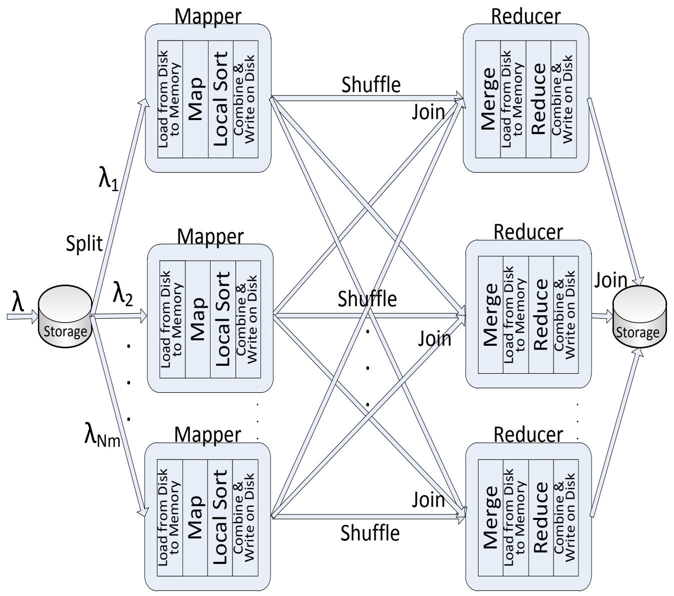


Fig. 1. MapReduce framework including map, shuffle, and reduce phases.

For example, in a word-count application, the i^{th} mapper node counts the frequency of each word in its received text in the map phase and, then calculates and sends corresponding tuple $\left(mapper_i, word_j, count_{i,j}\right)$ to k^{th} reducer node in the shuffle phase. If $i = 1..N_m$, $j = 1..N_w$ and $k = 1..N_r$, on average N_w/N_r words go to each reducer node. For example, words that begin with letter "A" go to 1st reducer node, words that begin with letters "B, C or D" (having balanced number of words for each) go to 2nd reducer node, etc. It is important to note that the map and shuffle phases may overlap. Each reducer node calculates the total frequency of its words by summing the counts in reduce phase, as k^{th} reducer node does $\sum_{i=1}^{N_m} count_{i,j}\ ; j = \frac{(k-1)N_w}{N_r} + 1, \frac{(k-1)N_w}{N_r} + 2, \ldots, \frac{k.N_w}{N_r}$.

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 in a mapper node, unbalanced jobs scheduled on mapper nodes or heterogeneity across computational resources [8]. Clearly, the start time of a reduce job is dictated by slowest mapper, that is, the slowest mapper 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 cluster nodes that complete their assigned tasks in a time longer than usual. Node hardware failure, resource contention of tasks running on a node, and limited resource availability on a node can make it a straggler.

B. Related Work

The mathematical modeling of MapReduce framework has recently been researched in studies such as [15,16,17,18]. The main difference between these models and our work 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 I/O cost, number of I/O requests, and startup cost in Hadoop. They also propose a hash-based mechanism to allow incremental processing and in-memory processing of frequent keys. They investigate the best merge factor for MapReduce jobs larger than memory size.

Ananthanarayanan et al [11] show that 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, this method does not work for stragglers of large tasks. In [16], the authors try to find an analytical model of MapReduce to minimize the execution time and find optimal map granularity. In comparison, [17] presents a performance model to estimate the cost of map and reduce functions in MapReduce. However, the model assumes that all mappers finish at the same time and they do not consider the stochastic behavior of execution times of mappers and reducers. Krevat et al [18] proposed a simple analytical model for MapReduce and compared the performance results of MapReduce with other similar frameworks in some good conditions.

LATE [7] tries to optimize MapReduce performance in a heterogeneous cluster by restarting slow tasks in fast mapper nodes, and Tarazu [12] addresses the poor performance of MapReduce in heterogeneous clusters and shows that the traffic contention between remote tasks is the main problem in heterogeneous clusters. Motivated by this, they then propose a communication aware and dynamic load balancing technique to reduce the network traffic contention between remote tasks and the shuffling stage. SkewTune [19] manages interactively skew of non-uniform input data of user-defined applications at runtime. It finds an idle node in the cluster and assigns slow task to that node. Ananthanarayanan et al. [8] discuss the main causes of the outliers (stragglers) and, based on an identified cause, they propose to restart or duplicate the task at the beginning of their lifetime. Scarlett [13] replicates popular blocks in different machine's memory to reduce interference with running jobs.

Ananthanarayanan et al. [8] propose Mantri to reduce the effect of outliers of Bing clusters, where outliers' duration is compared to the duration of the median task in some figures fractionally. Also, e.g. the analysis of the workloads [9] from various research clusters, OpenCloud, M45 and WebMining, shows stragglers runtime to median task duration has a nearly delayed exponential distribution. Similarly, Chen et al. [10] evaluate the task lengths for Cloudera and Facebook workloads and reach similar conclusions. Tan et al [20] propose a coupling scheduler in MapReduce based on an analytical approach also modeling FIFO and fair scheduler with an index range for delay distribution tail. Under some circumstances, this scheduler delay distribution can have better performance than a fair scheduler, as its index is one order lower.

Lin et al [21] try to address map and shuffle overlapping challenge in MapReduce. They demonstrate that the optimal solution is NP-hard in offline mode and they suggest MaxSRPT and SplitSRPT schedulers for online mode reaching optimal scheduling. MaxSRPT is comparable to optimal scheduling when the "remaining map task" to "remaining shuffle task" ratio is small and SplitSRPT performance is good when the ratio is near one. Condie et al [22] change the MapReduce framework to support pipelining between map, shuffle and reduce phases. There are some compiler-based architecture [23,24] for SQL-like queries in MapReduce framework to setup and speedup the execution of computation on large data sets. They work on computational DAG (directed acyclic graph) of large data sets queries and they need multiple rounds of MapReduce to be optimized.

C. *Delayed Exponential Distribution*

We start by giving the mathematical definition of *Delayed Exponential Distribution* since it is heavily used in our model.

Definition 1: For a delayed exponential distribution with rate λ and offset T, $F(t)$ is cumulative distribution function (CDF) and $f(t)$ is probability density function (PDF) as follows:

$$F(t) = \begin{cases} 0 & ; t < T \\ 1 - e^{-\lambda(t-T)} & ; t \geq T \end{cases} = \left(1 - e^{-\lambda(t-T)}\right) U(t-T) \qquad (1)$$

$$f(t) = \begin{cases} 0 & ; t < T \\ \lambda e^{-\lambda(t-T)} & ; t \geq T \end{cases} = \lambda e^{-\lambda(t-T)} U(t-T), \qquad (2)$$

where $U(t)$ is unit step function. Figure 2(a) shows the cumulative distribution of a delayed exponential function. Figure 2(b) shows a probability density function of delayed exponential distribution.

An important observation is that the completion times of map tasks exhibit a delayed exponential distribution [8,9,10]. In fact, we show in the next section that delayed exponential distribution is nearly coincident with empirical data derived from some other papers. As illustrated in Figure 2(b), most of mappers finish their tasks right after threshold, but a fraction of mappers finish their tasks after a long time. Since reducers can start only after completion of all their map tasks, they will be delayed because of their delayed mappers.

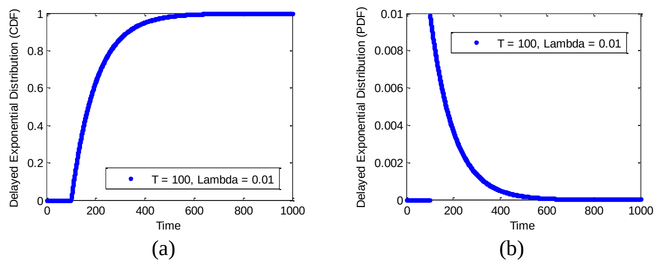


Fig. 2. CDF (a) and PDF (b) of delayed exponential distribution.

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 the end-to-end delay. We also optimize the delay using scheduling algorithms that make the execution look more homogeneous.

III. Map Phase Modeling

This section gives the assumptions we make and explains the parameters used in our model. The delayed exponential distribution is justified in each sub-section based on general knowledge and real observations, and the response time distribution of mapper nodes is proved analytically and verified using the available data.

A. *Assumptions*

A MapReduce job (e.g., word count) can be seen as a set of tasks that must be completed to get the desire results (e.g., the frequency of each word). In our model, we assume that the different jobs are submitted to the system with a mean rate or we can say the jobs have a total mean inter-arrival time. The mean inter-arrival rate of the job can be interpreted as the average number of CPU instructions coming to the system per second and it is denoted by λ parameter as shown in Figure 1. A job comes to the MapReduce framework as a bunch of tasks and then they are split into M map tasks with the nearly equal number of instructions and sent to mapper nodes by the scheduler. There are N_m mapper nodes which receive map tasks respectively with the probabilities $p_1, p_2, \dots, p_{N_m}$. Alternately, given a scheduler, $p_i.M$ map tasks are sent to i^{th} mapper node. In other words, the mean inter-arrival rate of the task to i^{th} mapper node is $\lambda_i = p_i.\lambda$ where $\lambda = \sum_{i=1}^{N_m} \lambda_i$.

TABLE I. Notation used in our formulations.

Parameter	Notation	Explanation
Number of mapper nodes ($i \in \{1,2,\dots,N_m\}$)	N_m	The number of nodes in the datacenter assigned for map tasks
Mean task inter-arrival rate to i^{th} mapper node	λ_i	The average rate of the tasks coming to i^{th} mapper node
Mean service rate of i^{th} mapper node ($\mu_i = 1/s_i$)	μ_i	The average rate of the tasks departing from i^{th} mapper node
Mean job inter-arrival rate or total inter-arrival rate	λ	$\lambda = \sum_{i=1}^{N_m} \lambda_i$
Unit step function	$U(t)$	$U(t) = \begin{cases} 1; t \geq 0 \\ 0; t < 0 \end{cases}$
Unit impulse function	$\delta(t)$	$\delta(t) = dU(t)/dt$
Offset of delayed exponential for i^{th} mapper node	T_i	The minimum amount of time required to complete a task
Response time of all map tasks	R_M	The required time to finish all

(random variable)		map tasks by all mapper nodes
Response time of ith mapper node (random variable)	R_{M_i}	The required time to finish a task by ith mapper node
Mean response rate of ith mapper node	γ_i	The average required time to finish a task by ith mapper node

A mapper node may run multiple map tasks. Mapper node i is modeled as a single queue with a mean service rate of μ_i, i.e., on an average, it can service μ_i tasks concurrently. 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. Communication links (arrows in Figure 1) are assumed to have a nearly-deterministic delay or containing routers with a nearly-deterministic service rate. Small variances across communication link delays add a nearly-constant delay overhead on all end-to-end delay, and this overhead is small compared to the computation delay. This is because routing service rate is much higher than any map/reduce task service rate, and shuffling phase has overlap with map phase of MapReduce. Table I shows the notation used in this paper.

B. *Mapper Node as a Single Queue*

Mapper node can be modeled as a FCFS (first-come first-served) infinite-buffer single queue. The distribution of response time of a mapper node is a function of the inter-arrival rates (with a mean of λ_i) as well as their service rates (with a mean of μ_i). In our model we investigate delayed exponential service rate. Later in Section 5, we study different distributions for task inter-arrival time and mapper node service time. Note that, we are not restricted to a simple M/M/1 queue; we investigate the other inter-arrival rates and service rates that correspond to other potential scenarios in modern datacenters like Gamma distribution or Erlang-k distribution as a general rational form of job inter-arrival time distribution in datacenters [25].

The clock rate of mapper node has a periodic characteristic and is proportional to service rate (μ_i) with a linear deterministic coefficient (say C), i.e., most instructions can be executed successively with a time difference not less than $t_i = 1/\mathrm{C}\mu_i$ and other instructions that involve memory or I/O requests can have an even longer time difference from the previous instructions. Thus the distribution of clock rate is $\delta(t - t_i)$ where $\delta(t) = dU(t)/dt$, and the distribution of service rate for different instructions is exponential, as assumed by many prior papers [26,27,28]. The distribution of summation of these two random variables is equivalent to the convolution $(*)$ of these two distributions. The resulting service rate of a mapper node is thus a delayed exponential PDF which can be expressed as follows:

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

where t_i is the minimum time required to execute an instruction. Naturally each map task contains a number of instructions (say I), each having a minimum time to execute. Therefore, the total service time of a map task is longer than $T_i = \sum_{j=1}^{I} t_j$ giving the offset of the delayed exponential distribution in the service times. Note that the different mapper nodes can have different service rates (μ_is) and offset times (T_is) which can make a datacenter heterogeneous. As far as the running job is concerned, we need only these parameters (μ_i and T_i) of each mapper node to derive our desired scheduler. As a result, the delayed exponential service time is a generalization of the exponential service time and we show below that it matches with real data.

C. *Response Time of a Mapper Node*

Response time (completion time) of a mapper node is the time required for its map tasks to wait in the mapper node buffer (queue) plus the time required to service them. In Lemma 1, we show that, given the delayed exponential service time of each mapper node, the total completion time of map tasks has a delayed exponential property. To show this, the following lemmas are crucial.

Lemma 1: Delayed exponential service rate approximately results in delayed exponential response time for a mapper node. (See [29] for the proof).

Lemma 2: Given the distribution of response time of each mapper node is delayed exponential, the completion time of all mapper nodes (tasks) across the cluster also has a delayed exponential CDF. (See [29] for the proof).

D. *Validation of Delayed Exponential Completion Time*

We show that our defined delayed exponential distribution matches with the empirical completion times of map tasks that have been published in prior studies. We use the CDFs of published completion times of different MapReduce applications (Table II) and try to fit a delayed exponential distribution on this data represented by the following function:

$$\left(1 - e^{-\gamma(t-T)}\right) U(t - T). \tag{4}$$

TABLE II. CURVE-FITTING OF THE COMPLETION TIMES OF THE PUBLISHED DATA TO DELAYED EXPONENTIAL DISTRIBUTION.

Benchmark	**Reference**	**Exponential Rate (γ)**	**Least Square Error**
Bing Search Engine	[8]	0.894498571	0.04861312
Facebook	[10]	0.011891914	0.04178871
Cloudra Customer (a)	[10]	0.014479546	0.05567882
Cloudra Customer (b)	[10]	0.037436986	0.01997069
Cloudra Customer (c)	[10]	0.009922954	0.03162061
Cloudra Customer (d)	[10]	0.01703502	0.03640228
Cloudra Customer (e)	[10]	0.025387299	0.04502734
OpenCloud	[9]	0.459106034	0.04565715
M45	[9]	0.232328761	0.05662841
WebMining	[9]	1.154299592	0.01424641

Note that the offset time T (constant for an application) can be directly taken from the empirical data, and it is only the mean response rate (γ) that we need to estimate. Newton's method for fast convergence has been used to find the response rate that reduces the mean square error of the fit. As shown in Table II, the least square error is not greater than 5% across all those workloads, strengthening our rationale for modeling completion times as a delayed exponential distribution. Table III lists important parameters and their values used in our subsequent experiments. These values are based on the data from the prior studies listed in Table II.

TABLE III. PARAMETER VALUES USED IN EXPERIMENTS.

Parameter	**Range**
Total mean inter-arrival time ($1/\lambda$)	0.1s-2s
Mean service time ($1/\mu$)	0.5s-2s
Offset time (T_i)	0.1s-100s
Mean response time ($1/\gamma$)	0.5s-1000s
Utilization ($\rho = \lambda/\mu$)	0.1-0.95

E. *Mean Sojourn Time at a Reducer*

Mean sojourn time (MST) at a reducer represents the average time required to synchronize all completed map tasks before a reducer can start its execution. Assuming uniform mapping from mapper nodes to reducer nodes via homogeneous hash-based (key, value) pair mechanism, there is no difference amongst the delay of all paths from mapper nodes to a reducer node, and thus we can choose one of them without loss of generality. MST can be a reasonable metric to represent the mean delay from job split to merge in a reducer as a fraction 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) response times of mapper nodes as $F_{R_{M_i}}(t) = P\left(R_{M_i} \le t\right); i = 1 \dots N_m$, using *maximum order statistics (MOS)*, the required time for synchronization of the completed map tasks at i^{th} reducer (S_{R_i}) is the maximum of the response times of all mapper nodes, i.e., we can find $S_{R_i} = max\left(R_{M_1}, R_{M_2}, \dots, R_{M_{N_m}}\right)$ as:

$$F_{S_{R_i}}(t) = P\left(S_{R_i} \le t\right) = \prod_{i=1}^{N_m} P(R_{M_i} \le t) = \prod_{i=1}^{N_m} F_{R_{M_i}}(t). \qquad (5)$$

The corresponding PDF can be easily expressed as follows:

$$f_{S_{R_i}}(t) = \frac{\partial}{\partial t} F_{S_{R_i}}(t) = F_{S_{R_i}}(t). \sum_{j=1}^{N_m} \frac{f_{R_{M_j}}(t)}{F_{R_{M_j}}(t)}. \qquad (6)$$

The following equations (Eq.7 and Eq.9) are used in our next formulations. The expected value of random sojourn time (MST) using the inclusion–exclusion principle [30] can be expressed respect to expected value of minimum of random response times of every subset of mapper nodes as follows:

$$MST = E\{S_{R_i}\} = E\left\{max\left(R_{M_1}, R_{M_2}, \dots, R_{M_{N_m}}\right)\right\}$$
$$= \sum_{i=1}^{N_m} \left\{ (-1)^{i+1} \sum_{\forall\{j_1, j_2, \dots, j_i\} \subset \{1,2,\dots,N_m\}} E\left\{min\left(R_{M_{j_1}}, R_{M_{j_2}}, \dots, R_{M_{j_i}}\right)\right\} \right\}. \qquad (7)$$

Further, MST can be expressed in other ways, with respect to distribution of random response time of each mapper node using Eq.5 as:

$$MST = \int_{t=0}^{\infty} t. f_{S_{R_i}}(t) dt = \int_0^{\infty} t. \frac{\partial}{\partial t}\left(\prod_{i=1}^{N_m} P\left(R_{M_i} \le t\right)\right) dt \qquad (8)$$

And using Eq.6 we have:

$$MST = \int_0^{\infty} t. F_{S_{R_i}}(t). \left(\sum_{j=1}^{N_m} \frac{f_{R_{M_j}}(t)}{F_{R_{M_j}}(t)}\right) dt$$
$$= \sum_{j=1}^{N_m} \int_0^{\infty} t. F_{S_{R_i}}(t) . \frac{f_{R_{M_j}}(t)}{F_{R_{M_j}}(t)} dt. \qquad (9)$$

Note that, the mean sojourn time (MST) is dependent on response time CDF of each mapper node (P$\left(R_{M_i} \le t\right)$), and it depends on the inter-arrival rate and service rate of each mapper node. For heterogeneous queues with general form distribution of response 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 [31]. To our knowledge, the exponentially distributed response time of heterogeneous queues has no closed formula, no approximation, and no bound. In fact, only for exponentially distributed response time of homogeneous queues (M/M/1 queues with the same μ), there are some approximations and boundaries in the statistics literature [32]. We derive the MST closed formula for the mapper nodes as M/M/1 queues using MOS in Lemma 3.

Lemma 3: Given M/M/1 mapper nodes with λ_i arrival rate and μ_i service rate where $\mathrm{i} = 1 \dots N_m$, using *maximum order statistics*, the MST of map tasks in a reducer is (proof in [29]):

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

As an example, Figure 3(a) gives the sensitivity of the mean sojourn time to the total mean inter-arrival rate to the system, when the service rate of each mapper node is $\mu_i = 1$ and the number of mapper nodes is $N_m = 60$. When the inter-arrival rate of the job increases, the mean sojourn time at a reducer homographically tends to infinity, if the number of mapper nodes is fixed. Figure 3(b) gives an intuition about the variation of MST (Eq.9) versus the number of mapper nodes of the system (N_m), when the service rate of each mapper node is $\mu_i = 1$ and the total inter-arrival rate to the system is constant ($\lambda = 2$). When the number of mapper nodes increases, if the total mean inter-arrival rate is fixed, the mean sojourn time tends to $O(ln(n))$ asymptotically, where n is the number of mapper nodes. Intuitively, there is a logarithmic algorithm to sync and merge completed tasks in one place, when a sync/merge operation can only happen between two completed tasks.

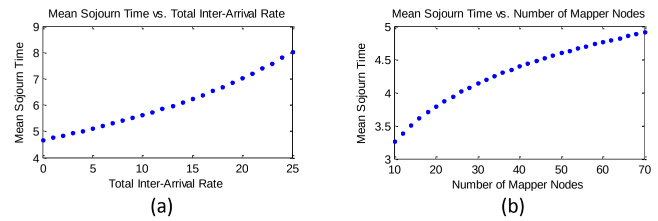


Fig. 3. MST with respect to total mean inter-arrival rate (a) and the number of mapper nodes (b).

Based on the discussion above, MST is the average time necessary to synchronize completed map tasks before reducer phase. As such, it can be expressed using the response times of each mapper node. We aim to minimize MST with respect to job mapping and number of mappers.

IV. Potential Uses of the Model

To optimize performance of a datacenter running a MapReduce job, we need the performance model of the datacenter. Using the performance model, critical parameters such as response time or throughput of the MapReduce job can be optimized and a corresponding scheduler can be obtained. One of the important decisions that datacenter designers need to make is whether to employ homogeneous or heterogeneous nodes. As far as our formulation is concerned, the difference between two options is uniform ($\mu_1 = \mu_2 = \dots = \mu_{N_m}$) versus non-uniform ($\mu_1 \ne \mu_2 \ne \dots \ne \mu_{N_m}$) service rates. We envision three potential uses of our model with respect to architecture, system and application parameters of a heterogeneous datacenter:

- It is able to model the performance parameters of nodes in a datacenter as service times and completion times of single queues more accurately, because it captures the deterministic behavior of task completion and the stochastic notion of straggler problem jointly. It can be used to optimize the performance of each node in a heterogeneous datacenter.
- It is able to 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 to globally improve performance of a MapReduce job in a datacenter.
- It can be used to optimize other target metrics such as power, and utilization of a heterogeneous datacenter.

The delayed exponential model of service time of mapper node is an extended version of the exponential model. It has the flexibility to be used for representing the service times of different classes of workloads. For example, the offset time parameter 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 an exponential distribution. However, both jobs have a fixed delay of execution that cannot be reduced stochastically.

The exponential distribution model cannot capture such situations, as the probability of having two consecutive completed jobs with zero delay in reality is zero but in exponential response time model is the highest. The delayed exponential response time model is also a good approximation of the response time of a typical datacenter server as a single queue, and can be used in the analysis of different types of distributed computing networks. Based on Lemma 1, the delayed exponential completion time can be obtained using the delayed exponential 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 based on their focus. The potential metrics of interest could be power, budget, power/performance, and other similar combinations. First, the specification of the cluster and workload should be evaluated. Considering Figure 2 and Eq.3, the service rate of the server and offset time related to the application can be obtained from the specs. So, the delayed exponential model of service time can be derived. Then, the response time model can be obtained by having the distribution of workload inter-arrival time that is discussed rigorously in the proof of Lemma 1 in [29]. We are interested in investigating the behavior of different schedulers by single queue model of mapper node with the delayed exponential response time when one type job is submitted to the datacenter. This can be extended to a generalized multiple queues model when multiple classes of jobs are submitted to a datacenter. Table IV gives a quick summary of the lemmas used in this work.

TABLE IV. SUMMARY OF THE LEMMAS (LEMMAS 7 THROUGH 13 CAN BE FOUND IN [29]).

Lemma	Explanation
1	delayed exponential service rate → delayed exponential response time for a mapper node.
2	delayed exponential service rate → delayed exponential completion times for all map tasks.
3	MST formula for M/M/1 mapper node
4	Joint optimization of MST is equivalent to two separate optimizations.
5	equilibrium property for D/D/1.
6	equilibrium property for M/M/1.
7	equilibrium property for G/M/1 [29].
8	sufficient conditions for optimal scheduling [29].
9	optimal mapping from moments of the distribution [29].
10	lower bound and upper bound of MST [29].
11	optimal number of M/M/1 mapper nodes [29].
12	optimal number of homogeneous mapper nodes [29].
13	optimal number of mapper nodes for a fixed budget [29].

V. MAP PHASE OPTIMIZATION

The delays 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 merge/aggregate of the reduce tasks. Heterogeneity in cluster resources makes the synchronization delay of map tasks higher. In fact, the problem of uneven map task completion times has been shown to be a serious impediment to the scalability of MapReduce computations [7,8,11,12]. While this has been studied experimentally, it has not been investigated from an analytical perspective.

We want to minimize this synchronization delay by dividing the total inter-arrival rate between mapper nodes. The unknown parameters are the residual inter-arrival rate to each mapper node and the number of mapper nodes. The cost function in our problem is the mean sojourn time (MST) given by the task inter-arrival rates of mapper nodes, and the sum of these inter-arrival rates is a constant (mean job inter-arrival rate) being the constraint of the problem. The MST of the map tasks is the average time taken between the start of a map task and reaching to a reducer. MST is the first moment (mean) of completion times' distribution at a reducer. We are interested in reducing MST with respect to the number of mapper nodes and their task inter-arrival rates.

Optimizing task inter-arrival rates (say mapping) and the number of mapper nodes jointly is in general a hard problem. JobTracker of MapReduce knows which mapper nodes are idle, i.e., it knows the state of the cluster. The task inter-arrival rates of mapper nodes can be also controlled by JobTracker to assign each mapper node the desired rate in mapping stage. The following lemma indicates that the optimal mapping and the optimal number of mapper nodes can be *separately optimized*. Consequently, we can separate these two problems and solve each of them independently.

Lemma 4: Joint optimization of MST with respect to the number of mapper nodes and their task inter-arrival rates is approximately equivalent to two separate optimizations, subject to, respectively the number of mapper nodes and their task inter-arrival rates. We have (see the proof in [29]):

$$\min_{N_m,\ \underline{\lambda}}(MST) = \min_{N_m}\left(\min_{\underline{\lambda}}(MST)\right) = \min_{\underline{\lambda}}\left(\min_{N_m}(MST)\right) \qquad (11)$$
$$S.t.\ \lambda = \sum_{i=1}^{N_m}\lambda_i\,;\ \underline{\lambda} = \left[\lambda_1\lambda_2\lambda_3\ \ldots\lambda_{N_m}\right]^T.$$

A. Optimal Mapping

In this section, we investigate the optimal mapping of tasks to mapper nodes with respect to the mean sojourn time. Our analysis is carried out for different mapper node queues. We then use this to investigate efficient schedulers that address the straggler problem.

Hadoop fair job scheduler divides and sends the same amount of job to each mapper node with λ_i task inter-arrival rate and μ_i service rate for N_m mapper nodes, i.e.

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

The fair job scheduler is optimal when we have a completely homogeneous cluster. The "*no bottleneck system necessary condition*" means that the mean job inter-arrival should be less than the total service rates of the mapper nodes ($\lambda < \sum_{i=1}^{N_m} \mu_i$). Depending on the scheduling algorithm, we may have some tighter bounds, as in this case no bottleneck node necessary condition is:

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

A fair job scheduler (like the Hadoop scheduler) makes the mean task inter-arrival rates equal for all mapper nodes (as captured by Eq.12), i.e., it gives each mapper node the same amount of work. Figure 4 shows the results from a cluster with the same number (from 40 to 200) of low-performance (service time $s_i = 1.25$) and high-performance ($s_i = 1$) nodes. The MST of fair job scheduler (Eq.12) with respect to the total inter-arrival rate and the number of M/M/1 heterogeneous mapper nodes always increases when the total inter-arrival rate increases. However, the MST with respect to the number of nodes has a minimum, i.e., there is an optimal number of nodes given by a job inter-arrival rate. If the job inter-arrival rate increases, this minimum number of nodes also increases.

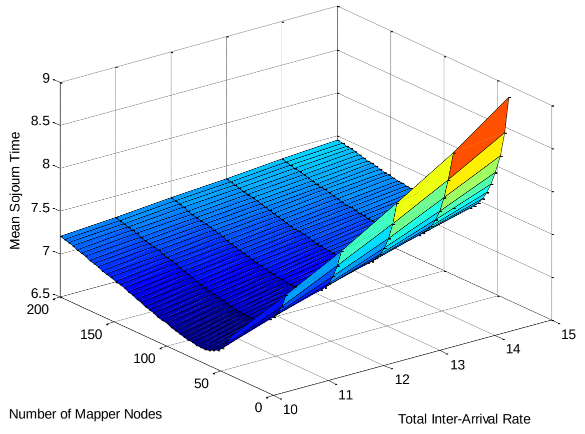


Fig. 4. MST with respect to total inter-arrival rate and the number of M/M/1 mapper nodes for fair job scheduler.

Assume that we have the same number (10) of low-performance (service time $s_i = 1.5$) and high-performance ($s_i = 1$) mapper nodes in a cluster. The simulation results plotted in Figure 5 are based on CSIM (a queueing system simulator [33]) implementation of these heterogeneous mapper nodes with the fair job scheduler. They show that the throughput of the different type nodes in the cluster remains almost the same, but the other parameters like utilization, queue length and response time change based on the inter-arrival time to the system and service time of mapper nodes. Next, we define a metric that can be used to mathematically compare the performance of the different types of schedulers. We start by giving the definition of the optimal MST.

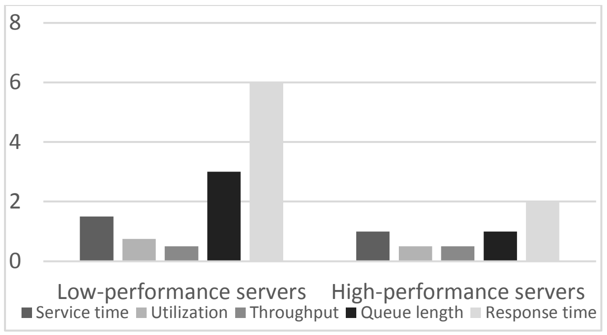


Fig. 5. The mean value of important parameters of a heterogeneous datacenter with the fair job scheduler.

Definition 3 (Optimal mapping based on MST): Optimal mapping for known number of mapper nodes (N_m) finds the optimal values of the mean task inter-arrival rates ($\lambda_1, \lambda_2, \ldots, \lambda_{N_m}$) which make MST minimum. The constraint is the mean job inter-arrival rate to the system (λ). The service rate of each mapper node is assumed to have a known distribution with the mean of μ_i. In mathematical terms, we have:

$$\min_{\underline{\lambda}}(MST) = \min_{\underline{\lambda}} \int_0^\infty t . \frac{\partial}{\partial t}\left(\prod_{i=1}^{N_m} F_{R_{M_i}}(t)\right) dt \qquad (14)$$
$$S.t. \; \lambda = \sum_{i=1}^{N_m} \lambda_i \, ; \; \underline{\lambda} = \left[\lambda_1 \lambda_2 \lambda_3 \; \ldots \lambda_{N_m}\right]^T,$$

where $F_{R_{M_i}}(t)$ is CDF of response time of the ith mapper node 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:

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

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:

$$\frac{\partial MST}{\partial \lambda_1} = \frac{\partial MST}{\partial \lambda_2} = \cdots = \frac{\partial MST}{\partial \lambda_{N_m}}. \qquad (16)$$

The above property cannot be easily solved or practically used to optimize a MapReduce job scheduling. It is an N_m-dimensional non-linear optimization problem with a linear constraint. Also, the following Lemmas handle certain special cases where Eq.9 can be easily solved and then used.

Lemma 5: Given Def.2-4 for D/D/1 mapper nodes ($i = 1 \ldots N_m$), equilibrium property can be expressed as follows:

$$\frac{\mu_1}{\lambda_1} = \frac{\mu_2}{\lambda_2} = \cdots = \frac{\mu_{N_m}}{\lambda_{N_m}}. \qquad (17)$$

We call the above property as the deterministic equilibrium property (D-EP), and the optimal solution for λ_is is:

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

And, the "no bottleneck node necessary" condition is:

$$\lambda < \frac{\mu_{min}}{\mu_{max}} \sum_{i=1}^{N_m} \mu_i. \tag{19}$$

Proof: Eq.17 can be derived by making the deterministic response times of all mapper nodes equal, and using:

$$\frac{\mu_1}{\lambda_1} = \frac{\mu_2}{\lambda_2} = \dots = \frac{\mu_{N_m}}{\lambda_{N_m}} = \frac{\sum_{j=1}^{N_m} \mu_j}{\lambda}.$$

Eq.18 is deduced, also $\lambda < \mu_i$; $\forall i$ will give Eq.19. □

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

$$\mathrm{Q}_1 = \mathrm{Q}_2 = \dots = \mathrm{Q}_{\mathrm{N_m}} \Leftrightarrow \frac{\rho_1}{1-\rho_1} = \dots = \frac{\rho_{\mathrm{N_m}}}{1-\rho_{\mathrm{N_m}}} \Leftrightarrow \rho_1 = \rho_2 = \dots = \rho_{\mathrm{N_m}} \tag{20}$$

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

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

Figure 6 plots the MST of the pure-deterministic scheduler with respect to the total inter-arrival rate and the number of M/M/1 heterogeneous mapper nodes. The MST growth given by the total inter-arrival rate is homographic versus MST growth given by the number of mapper nodes is $O(ln(n))$. Comparing Figure 4 with Figure 6, we can find that the pure-deterministic scheduler has a lower MST with respect to the fair job scheduler.

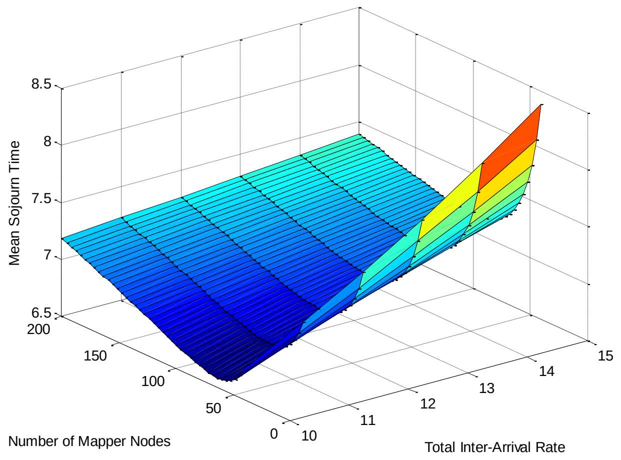


Fig. 6. MST with respect to the total inter-arrival rate and the number of M/M/1 mapper nodes for the pure-deterministic scheduler.

Figure 7 shows the simulation results of CSIM for heterogeneous mapper nodes with the pure-deterministic scheduler where only the utilization of the different type nodes is equal but all other parameters like response time are different, and it still makes the total delay higher.

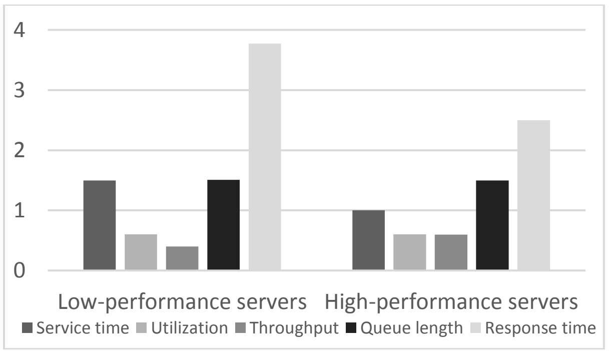


Fig. 7. Mean values of important parameters of a heterogeneous datacenter with the pure-deterministic scheduler.

Lemma 6: Given Def.2-4 for M/M/1 mapper nodes ($i = 1 \dots N_m$), the equilibrium property would be as follows:

$$\mu_1 - \lambda_1 = \mu_2 - \lambda_2 = \dots = \mu_{N_m} - \lambda_{N_m}. \tag{22}$$

We call above property as stochastic equilibrium property (S-EP), and the optimal λ_is are:

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

Without having bottleneck system as the inequality $\lambda < \sum_{i=1}^{N_m} \mu_i$, there is no bottleneck node necessary condition for any mapper node, but we may have negative values for λ_i, i.e., some jobs should be migrated from slow node (i) to other nodes. Then no negative λ_i necessary condition is:

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

See the proof in [29].

Q-proportional scheduling and pure-stochastic scheduling are alike, because:

$$\frac{\lambda_1}{Q_1} = \frac{\lambda_2}{Q_2} = \dots = \frac{\lambda_{N_m}}{Q_{N_m}} \Leftrightarrow \frac{\lambda_1}{\frac{\frac{\lambda_1}{\mu_1}}{1-\frac{\lambda_1}{\mu_1}}} = \frac{\lambda_2}{\frac{\frac{\lambda_2}{\mu_2}}{1-\frac{\lambda_2}{\mu_2}}} = \dots = \frac{\lambda_{N_m}}{\frac{\frac{\lambda_{N_m}}{\mu_{N_m}}}{1-\frac{\lambda_{N_m}}{\mu_{N_m}}}} \tag{25}$$
$$\Leftrightarrow \mu_1\left(1-\frac{\lambda_1}{\mu_1}\right) = \mu_2\left(1-\frac{\lambda_2}{\mu_2}\right) = \dots = \mu_{N_m}\left(1-\frac{\lambda_{N_m}}{\mu_{N_m}}\right)$$
$$\Leftrightarrow 1/(\mu_1 - \lambda_1) = \dots = 1/(\mu_{N_m} - \lambda_{N_m}).$$

Figure 8 plots the MST of the pure-stochastic scheduler with respect to the total inter-arrival rate and the number of M/M/1 (heterogeneous) mapper nodes. The MST growth trend is similar to that of the other scheduler. Comparing Figure 8 with Figures 4 and 6, one can observe that the pure-stochastic scheduler has a lower MST, given any total inter-arrival rate and any number of nodes.

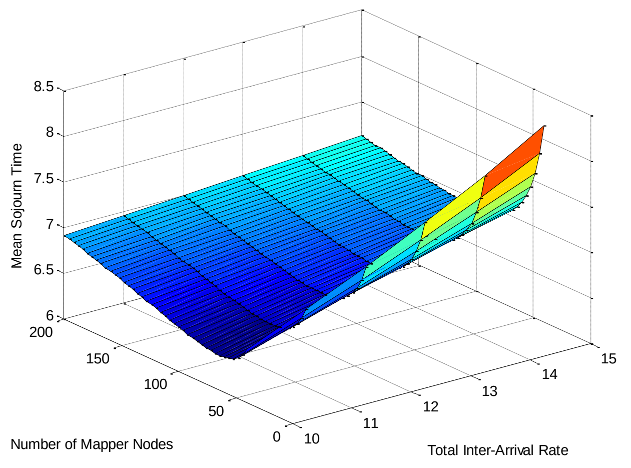


Fig. 8. MST with respect to the total inter-arrival rate and the number of M/M/1 mapper nodes for the pure-stochastic scheduler.

Figure 9 plots the simulation results for heterogeneous mapper nodes with the pure-stochastic scheduler. In this case, the mean response time of all nodes are nearly equal and the minimum delay can be achieved for the system.

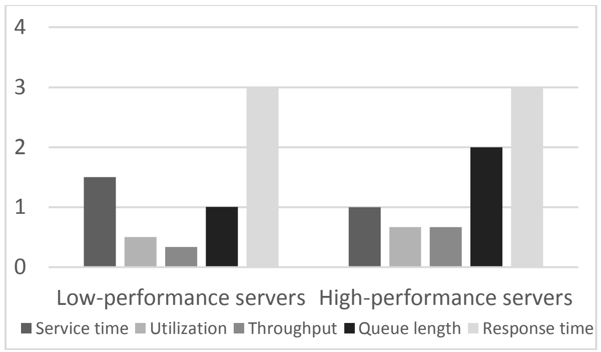


Fig. 9. Mean values of important parameters of a heterogeneous datacenter with the pure-stochastic scheduler.

For the M/M/1 mapper nodes, the pure-stochastic mapping has lower MST compared to the fair job and the pure-deterministic mapping shown in Figure 10. The MST of the pure-stochastic strategy is lower than the others even for very low inter-arrival rates.

Means equilibrium property (M-EP) is more suitable to be employed in practice. It is difficult and time-consuming to find the best mapping of inter-arrival time distributions to have such equilibrium properties. Instead of that, we can imagine to make first moment (mean) of response time distributions equal. The means equilibrium property as a lower bound [29] is:

$$E\{R_{M_1}\} = E\{R_{M_2}\} = \cdots = E\left\{R_{M_{N_m}}\right\}.$$

Also, the derivative of means equilibrium property (DM-EP) optimizing MST upper bound [29] can be expressed as:

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

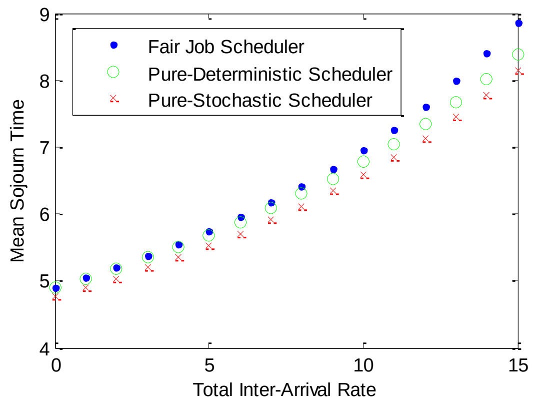


Fig. 10. MST comparison between fair job, pure-deterministic, and pure-stochastic schedulers.

M-EP approximately optimizes MST and gives sub-optimal mean inter-arrival rates ($\underline{\lambda}$) by making first moment of response times equal. The only degrees of freedom are mean inter-arrival rates ($\underline{\lambda}$), so in approximate approach we can only make M-EP not the other higher moments absolutely equal. This approximation is correct when the other moments of the response time of mapper nodes are negligible or the deviation is insignificant i.e. practically there is no bottleneck node in the system. In fact the gap between optimal solution and this approximate solution is insignificant.

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

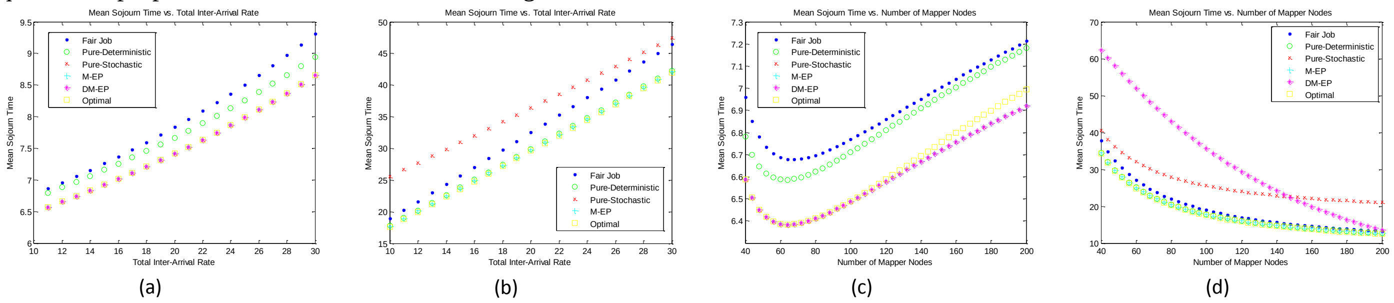
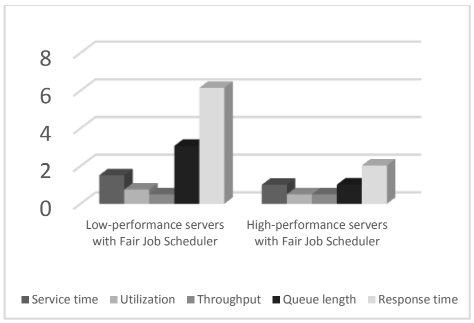
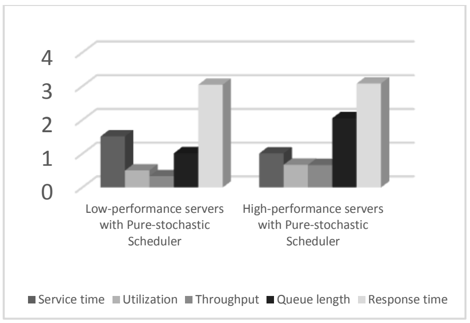


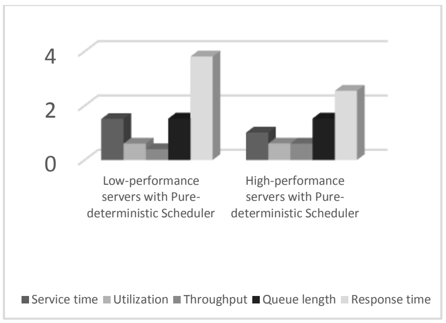
Fig. 11. MST comparison of different schedulers with respect to total inter-arrival rate for (a) memory-bound job ($D \approx 0$), (b) CPU-bound job ($D = 100$), (c) memory-bound job ($D \approx 0$), and (d) CPU-bound job ($D = 100$).



(a)



(b)



(c)

Fig. 12. Comparison of different schedulers' parameters.

When the target MapReduce job is memory-bound or has random memory access, it is more stochastic and the deterministic coefficient for this type of job is close to zero. For memory-bound jobs, 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, 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. Some schedulers sometimes outperform optimal scheduler, because they output negative λ_i but optimal scheduler always gives positive not-migratory λ_i, but in non-negative regions optimal scheduler always outperforms the others.

When the input job is CPU-bound, the pure-stochastic scheduler cannot track the optimal response time well. Figure 12 (a,b,c) indicates that the pure-stochastic scheduler cannot make all nodes' response time completely equal, and the other schedulers' deviation is higher. Since MST is a function of the slowest node, M-EP is near to the pure-stochastic in this case.

VI. CONCLUDING REMARKS

Targeting MapReduce applications, in this paper, the service rate of mapper nodes as a single queue has been modeled with the delayed exponential distribution, and also it has been shown that their response time has a similar behavior. Using this analytical result, next, the map phase of a single-pass MapReduce job has been modeled and we have formulated the mean sojourn time (MST) at a reducer node by means of task inter-arrival rates and service rates of mapper nodes. MST is a potential metric for optimizing end-to-end delay in MapReduce framework. Based on different types of inter-arrivals and service rates, the MST parameter has been optimized and equilibrium property was investigated for many cases. To realize the minimum map phase delay in a heterogeneous datacenter, we have also investigated different types of schedulers.

ACKNOWLEDGMENTS

We thank Amin Jadidi, Narges Shahidi and Mahdy Zolghadr for their feedback on earlier drafts of this paper.

REFERENCES

[1] T. Gunarathne et al., "MapReduce in the clouds for science," IEEE 2nd CloudCom Conf., pp. 565–572, 2010.

[2] R. Ananthanarayanan et al. "Cloud analytics: Do we really need to reinvent the storage stack?," In Proc. of the HotCloud Workshop, 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 OSDI Symp., San Francisco CA, 2004.

[5] K. Shvachko et al., "The hadoop distributed file system," In 26th IEEE MSST Symp., May, 2010.

[6] M. Isard et al., "Dryad: distributed data-parallel programs from sequential building blocks," In Proc. of the 2nd ACM SIGOPS/EuroSys, 2007.

[7] M. Zaharia et al., "Improving MapReduce Performance in Heterogeneous Environments," In USENIX OSDI, 2008.

[8] G. Ananthanarayanan et al., "Reining in the outliers in map-reduce clusters using Mantri," In Proc. of the 9th OSDI Symp., 2010.

[9] K. Ren et al. Hadoop's adolescence: a comparative workload analysis from three research clusters. Technical Report UW-CSE-12-06-01, University of Washington, June 2012.

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

[11] G. Ananthanarayanan et al., "Effective straggler mitigation: attack of the clones," In Proc. of the 10th NSDI Symp., 2013.

[12] F. Ahmad et al., "Tarazu: optimizing mapreduce on heterogeneous clusters," In Proc. of the 17th ASPLOS Conf., pp. 61-74, New York, NY, USA, ACM, 2012.

[13] G. Ananthanarayanan et al., "Scarlett: Coping with Skewed Popularity Content in MapReduce Clusters," In ACM EuroSys, 2011.

[14] H. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In Symposium on Discrete Algorithms (SODA), 2010.

[15] B. Li et al., "A Platform for Scalable One-Pass Analytics using MapReduce," In Proc. of the 2011 ACM SIGMOD, 2011.

[16] X. Yang and J. Sun, "An analytical performance model of mapreduce," In IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS), pages 306-310, 2011.

[17] X. Lin et al., "A practical performance model for hadoop mapreduce," In CLUSTER Workshops, pp. 231–239, 2012.

[18] E. Krevat et al., "Applying Performance Models to Understand Data-intensive Computing Efficiency," Technical Report CMU-PDL-10-108. Carnegie Mellon University, Pittsburgh, 2010.

[19] Y. Kwon et al. SkewTune: Mitigating skew in MapReduce applications. In Proc. of the SIGMOD Conf., pages 25–36, 2012.

[20] J. Tan, X. Meng, and L. Zhang, "Delay tails in MapReduce scheduling," In Proc. of the 12th ACM SIGMETRICS/PERFORMANCE, pp. 5–16, New York, NY, USA, ACM, 2012.

[21] M. Lin et al., "Joint optimization of overlapping phases in MapReduce," Performance Evaluation, 2013.

[22] T. Condie et al., "Mapreduce online," In NSDI, 2010.

[23] R. Chaiken et. al. "Scope: Easy and Efficient Parallel Processing of Massive Data Sets," In Proc. of VLDB, 2008.

[24] A. Thusoo et al., "Hive - a warehousing solution over a Map-Reduce framework," PVLDB, vol. 2, no. 2, pp. 1626–1629, 2009.

[25] V. A. Saletore et al., "HcBench: Methodology, Development, and Characterization of a Customer Usage Representative Big Data/Hadoop

Benchmark," IEEE International Symposium on Workload Characterization, September 2013.

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

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

[28] J. Li, Y. Fan and M. Zhou, "Performance modeling and analysis of workflow," IEEE Trans. Syst., Man, Cybern. A, Syst. Humans, vol. 34, no. 2, pp.229-242 2004.

[29] F. Farhat et al., "Modeling and Optimization of Straggling Mappers," Technical Report PSU-CSE-14-006, Pennsylvania State University, March 2014.

[30] R. B. J. T. Allenby, A.B. Slomson, "How to Count: An Introduction to Combinatorics," Discrete Mathematics and Its Applications (2ed.), CRC Press, pp. 51–60, ISBN 9781420082609, 2010.

[31] B. Kemper, M. Mandjes, "Mean sojourn times in two-queue fork-join systems: bounds and approximations," OR Spectrum 34, 723–742, 2012.

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

[33] H. Schwetman, "CSIM: A C-based, process oriented simulation language," In Proceedings of the 1986 Winter Simulation Conference, pp. 387–396, December 1986.