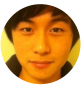
The MDS Queue

Article · November 2012

Source: arXiv



Kang Wook Lee

Korea Advanced Institute of Science and Technology

The MDS Queue: Analysing the Latency Performance of Erasure Codes

Nihar B. Shah, Kangwook Lee, Kannan Ramchandran
Dept. of Electrical Engineering and Computer Sciences
University of California, Berkeley
{nihar, kw1jjang, kannanr}@eecs.berkeley.edu

Abstract

In order to scale economically, data centers are increasingly evolving their data storage methods from the use of simple data replication to the use of more powerful erasure codes, which provide the same level of reliability as replication but at a significantly lower storage cost. In particular, it is well known that Maximum-Distance-Separable (MDS) codes, such as Reed-Solomon codes, provide the maximum storage efficiency. While the use of codes for providing improved reliability in archival storage systems, where the data is less frequently accessed (or so-called "cold data"), is well understood, the role of codes in the storage of more frequently accessed and active "hot data", where latency is the key metric, is less clear.

In this paper, we study data storage systems based on MDS codes through the lens of queueing theory, and term this the "MDS queue." We analytically characterize the (average) latency performance of MDS queues, for which we present insightful scheduling policies that form upper and lower bounds to performance, and are observed to be quite tight. Extensive simulations are also provided and used to validate our theoretical analysis. We also employ the framework of the MDS queue to analyse different methods of performing so-called degraded reads (reading of partial data) in distributed data storage.

I. Introduction

Two of the primary objectives of a storage system are to provide reliability and availability of the stored data: the system must ensure that data is not lost even in the presence of individual component failures, and must be easily and quickly accessible to the user whenever required. The classical means of providing reliability is to employ the strategy of replication, wherein identical copies of the (entire) data are stored on multiple servers. However, this scheme is not very efficient in terms of the storage space utilization. The exponential growth in the amount of data being stored today makes storage an increasingly valuable resource, and has motivated data-centers today to increasingly turn to the use of more efficient erasure codes [1]–[5].

The most popular, and also most efficient storage codes are the Maximum-Distance-Separable (MDS) codes, e.g., Reed-Solomon codes. An MDS code is typically associated to two parameters n and k. Under an (n, k) MDS code, a file is encoded and stored in n servers such that (a) the data stored in *any* k of these n servers suffice to recover the entire file, and (b) the storage space required at each server is $\frac{1}{k}$ of the size of the original file. [1]

While the reliability properties of erasure codes are very well understood, much less is known about their latency performance. In this paper, we study coded data storage systems based on MDS codes through the lens of queueing theory. We term the queue resulting from the use of codes that allow for recovery of the data from any k of the n nodes as "the MDS queue". To understand this queueing-theoretic perspective, consider a simple example with $n = 4$ and $k = 2$. Several files $\{F_i\}$ are to be stored in the 4 servers in a manner that no data is lost upon failure of any $(n - k) = 2$ of the $n = 4$ servers. This is achieved via a $(4,\ 2)$ MDS code under which, each file is partitioned into two halves $F_i = [f_{i,1}\ f_{i,2}]$, and the 4 servers store $f_{i,1}$, $f_{i,2}$, $(f_{i,1} + f_{i,2})$, and $(f_{i,1} + 2f_{i,2})$ respectively for all i. Requests for reading individual files arrive as a stochastic process, which are buffered and served by the system, and the resulting queue is termed an MDS queue (this shall be formalized later in the paper).

An exact analysis of the MDS queue is hard in general: a Markov chain representation of the MDS queue has a state space that is infinite in at-least k dimensions, and furthermore, the transitions are tightly coupled across the k dimensions. In this paper, we present insightful scheduling policies that provide upper and lower bounds to the performance of the MDS queue, and these are observed to be quite tight. Using these bounds we analytically characterize the (average) latency performance of MDS queues, as illustrated in Fig. 1.

The lower bounds (the 'MDS-Reservation(t)' scheduling policies) and the upper bounds (the 'M^{k}/M/n(t)' scheduling policies) presented in this paper are both indexed by a parameter 't'. An increase in the value of t results in tighter bounds, but also

N. B. Shah was supported by a Berkeley Fellowship and K. Lee by a KFAS Fellowship. This work was also supported in part by NSF grant CCF-1116404.

[1] A more generic definition of an MDS code is that it is a code that satisfies the 'Singleton bound' [6].

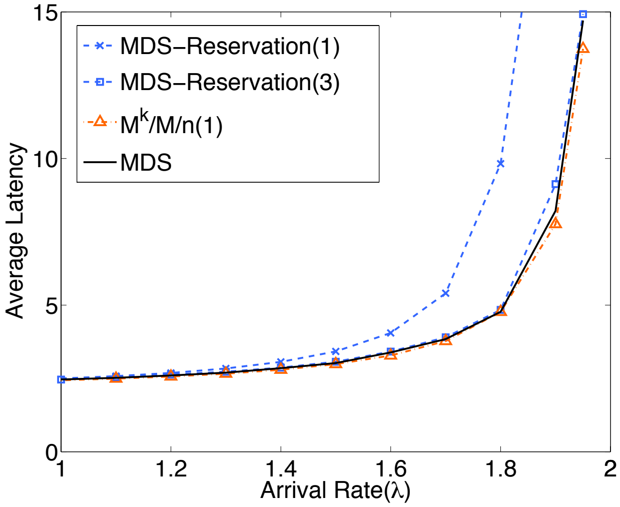


Fig. 1: The average latency of a system using an MDS code with $n = 10$ and $k = 5$. The service of each job is assumed to be drawn from an exponential distribution with rate $\mu = 1$. The curve titled 'MDS' corresponds to simulations of the exact coded system. Also plotted are the analytically computed latencies of the lower bounds (MDS-Reservation(t) queues) and upper bounds (M^{k}/M/n(t) queues) presented in this paper. We also confirmed that these analytically computed latencies closely match the simulated performances of the corresponding scheduling policies.

increases the complexity of analysis. Furthermore, both classes of scheduling policies converge to the MDS scheduling policy as t$\rightarrow\infty$. However, we observe that the performance of the MDS-Reservation(t) queue is very close to that of the MDS queue for very small values of t (as small as t $=$ 3), and the performance of the upper bounds M^{k}/M/n(t) closely follow that of the MDS queue for values of t as small as t $=$ 1. This can be observed in Fig. 1. The MDS-Reservation(t) scheduling policies presented here are themselves practical alternatives to the MDS scheduling policy, since they require maintenance of a smaller state, while offering highly comparable performance.

We also consider the problem of degraded reads (i.e., reading of partial data) in distributed storage systems, that has recently attracted considerable interest in the coding-theory community. We employ the framework of the MDS queue to understand and compare, from a queueing theoretic viewpoint, different methods of performing degraded reads.

The rest of the paper is organized as follows. Section II presents the MDS queue system model. Section III discusses related literature. Section IV describes the general approach and the notation followed in the paper. Section V presents the MDS-Reservation(t) queues that lower bound the performance of the MDS queue. Section VI presents the M^{k}/M/n(t) queues that upper bound the performance. Section VII presents analyses and comparisons of these queues. Section VIII presents conclusions and discusses open problems. The appendix contains proofs of the theorems presented in the paper.

II. The MDS Queue System Model

We shall now describe a queueing theoretic model of a system employing an MDS code. As discussed previously, under an MDS code, a file can be retrieved by downloading data from any k of the servers. We model this by treating each request for reading a file as a *batch* of k *jobs*. The k jobs of a batch represent reading of k encodings of the file from k servers. A batch is considered as served when k of its jobs have been served. For instance, in the example of the $(n = 4,\ k = 2)$ system of Section I, a request for reading file F_i (for some i) is treated as a batch of two jobs. To serve this request, the two jobs may be served by any two of the four servers; for example, if the two jobs are served by servers 2 and 3, then they correspond to reading $f_{i,2}$ and $(f_{i,1} + f_{i,2})$ respectively, which suffice to obtain F_i. We assume homogeneity among files and among servers: the files are of identical size, and the n servers have identical performance characteristics.

Definition 1 (MDS queue): An MDS queue is associated to four parameters (n, k) and $[\lambda, \mu]$.

- There are n identical servers
- Requests enter into a (common) buffer of infinite capacity
- Requests arrive as a Poisson process with rate λ
- Each request comprises a **batch** of k **jobs**
- Each of the k jobs in a batch can be served by an **arbitrary** set of k **distinct** servers

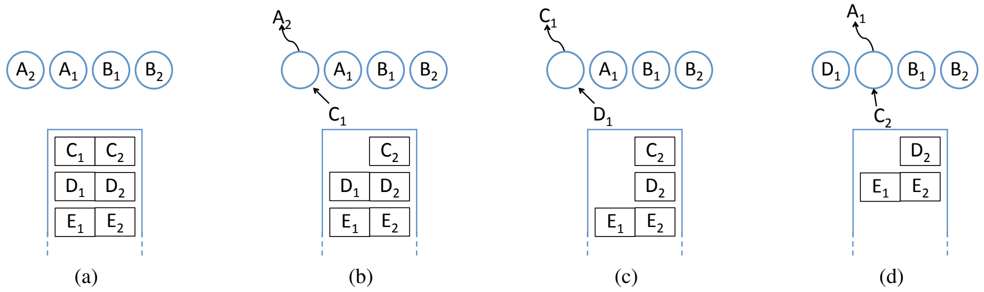


Fig. 2: Functioning of the MDS queue.

- The service time for a job at any server is exponentially distributed with rate μ, independent of all else
- The jobs are processed in order, i.e., among all the waiting jobs that an idle server is allowed to serve, it serves the one which had arrived the earliest.

Algo. 1 formalizes the scheduling policy of the MDS queue.

Algorithm 1 MDS scheduling policy

```
On arrival of a batch
  Assign as many of its jobs as possible to idle servers
  Append remaining jobs (if any) as new batch at end of buffer
On departure from a server (say, server $s$)
  If $\exists$ at least one batch in the buffer such that no job of this batch has been served by $s$
    Among all such batches, find batch that arrived earliest
    Assign a job from this batch to $s$
```

The following example illustrates the functioning of the MDS scheduling policy and the resultant MDS queue.

Example 1: Consider the MDS$(n = 4,\ k = 2)$ queue, as depicted in Fig. 2. Here, each request comes as a batch of $k = 2$ jobs, and hence we denote each batch (e.g., A, B, etc.) as a pair of jobs ($\{A_1, A_2\}$, $\{B_1, B_2\}$, etc.). The two jobs in a batch need to be served by (any) two distinct servers. Denote the four servers (from left to right) as servers 1, 2, 3 and 4. Suppose the system is in the state as shown in Fig. 2a, wherein the jobs A_2, A_1, B_1 and B_2 are being served by the four servers, and there are three more batches waiting in the buffer. Suppose server 1 completes servicing job A_2 (Fig. 2b). This server is now free to serve any of the 6 jobs waiting in the buffer. Since jobs are processed only in order, it begins serving job C_1 (assignment of C_2 would also have been valid). Next, suppose server 1 completes C_1 before any other servers complete their tasks (Fig. 2c). In this case, since server 1 has already served a job of batch C, it is not allowed to service C_2. However, it can service any job from the next batch $\{D_1, D_2\}$, and one of these two jobs is (arbitrarily) assigned to it. Finally, when one of the other servers completes its service, that server is assigned job C_2 (Fig. 2d).

A. Other Applications

The MDS queue also arises in other applications that require diversity or error correction. For instance, consider a system with n processors, with the arriving jobs comprising computational tasks. It is often the case that the processors are not completely reliable [7], and may give incorrect outputs at random instances. In order to guarantee a correct output, a job may be processed at k different servers, and the results aggregated (perhaps by majority rule) to obtain the final answer. Such a system results precisely in an MDS(n,k) queue (with some arrival and service-time distributions). In general, queues where jobs require diversity, for purposes such as security, error-protection etc., may be modelled as an MDS queue. [2] Finally, even in the setting of distributed storage systems, the MDS queue need not be restricted to analysing Maximum-Distance-Separable codes alone, and can be used for any code that supports recovery of the files from 'any k out of the n' nodes.

[2] An analogy that the academic will relate to is that of reviewing papers. There are n reviewers in total, and each paper must be reviewed by k reviewers. This forms an MDS(n,k) queue. The values of λ and μ considered should be such that $\frac{\lambda}{\mu}$ is close to the maximum throughput, modelling the fact that reviewers are generally busy.

B. Exact Analysis

An exact analysis of the MDS queue is hard. The difficulty arises from the special property of the MDS queue, that each of the k jobs of a batch must be served by k *distinct* servers. Thus, a Markov-chain representation of this queue is required to have each state encapsulating not only the number of batches or jobs in the queue, but also the configuration of each batch in the queue, i.e., the number of jobs of each batch currently being processed, the number completed processing, and the number still waiting in the buffer. Thus, when there are b batches in the system, the system can have $\Omega\big(b^k\big)$ possible configurations. Since the number of batches b in the system can take any value in $\{0, 1, 2, \ldots\}$, this leads to a Markov chain which has a state space that has infinite states in at least k dimensions. Furthermore, the transitions along different dimensions are tightly coupled. This makes the Markov chain hard to analyse, and in this paper, we provide scheduling policies for the MDS queue that lower/upper bound the exact MDS queue.

III. Related Literature

A. Queueing-theoretic analysis of coded systems

The study of latency of coded systems was initiated by Huang et al. in [8], which we build upon in this paper. In particular, in [8], a 'block-one-scheduling' policy was presented, that provides a lower bound on the performance of the MDS queue with $k = 2$. This policy is a special case of the MDS-Reservation(t) policies presented in this paper, and corresponds to the case when t $= 1$. While the block-one-scheduling policy was analysed only for $k = 2$ in [8], the analysis in this paper applies to all values of (n, k), and recovers the corresponding results of [8] as a special case. In addition, the MDS-Reservation(t) policies of the present paper, when t > 1, provide significantly tighter bounds to the performance of the MDS queue (as can be seen in Fig. 1). The present paper also provides upper bounds to the performance of the MDS queue.

The blocking probability of such systems in the absence of a buffer was previously studied in [9].

B. Fork-join queues for parallel processing

A class of queues that are closely related to the MDS queue is the class of *fork-join queues* [10], and in particular, fork-join queues with variable subtasks [11]. The classical setup of fork-join queues assumes each batch to consist of n jobs, while the setup of fork-join queues with variable subtasks assumes k jobs per batch for some parameter $k \leq n$. However, under a fork-join queue (with variable subtasks), each job must be served by a particular *pre-specified* server, while under an MDS queue, the k jobs of a batch may be processed at any *arbitrary* set of k servers and this choice is governed by the scheduling policy.

C. Redundant Requests

In a system that employs a (n, k) erasure code, the latency of serving the requests can potentially be reduced by sending the requests redundantly to more than k servers. The request is deemed served when it is served in any one of these ways. Following this, the other copies of this request may be removed from the system. A theoretical analysis when redundant requests help is initiated in [12] under the MDS Queue model (and also under a distributed counterpart). It is shown in [12] that, surprisingly, for any arbitrary arrival process, sending each request (redundantly) to all n servers results in the smallest average latency (as compared to any other redundant-requesting policy) when the service time distribution is memoryless or heavier. Bounds on the average latency when the requests are sent (redundantly) to *all* n servers are derived in an independent work [13]. Approximations and empirical evaluations are performed in [14]–[17].

IV. Our Approach and Notation for Latency Analysis of The MDS Queue

For each of the scheduling policies presented in this paper (that lower/upper bound the MDS queue), we represent the respective resulting queues as continuous time Markov chains. We show that these Markov chains belong to a class of processes known as *Quasi-Birth-Death (QBD)* processes (described below), and obtain their steady-state distribution by exploiting the properties of QBD processes. This is then employed to compute other metrics such as the average latency, system occupancy, etc.

Throughout the paper, we shall refer to the entire setup described in Section II as the 'queue' or the 'system'. We shall say that a batch is waiting (in the buffer) if at-least one of its jobs is still waiting in the buffer (i.e., has not begun service). We shall use the term "*i^{th} waiting batch*" to refer to the batch that was the i^{th} earliest to arrive, among all batches currently waiting in the buffer. For example, in the system in the state depicted in Fig. 2a, there are three waiting batches: $\{C_1, C_2\}$, $\{D_1, D_2\}$ and $\{E_1, E_2\}$ are the first, second and third waiting batches respectively.

We shall frequently refer to an MDS queue as MDS(n,k) queue, and assume $[\lambda, \mu]$ to be some fixed (known) values. The system will always be assumed to begin in a state where there are no jobs in the system. Since the arrival and service time distributions have valid probability density functions, we shall assume that no two events occur at exactly the same time. We shall use the notation a^+ to denote $\max(a, 0)$.

Review of Quasi-Birth-Death (QBD) processes: Consider a continuous-time Markov process on the states $\{0, 1, 2, \ldots\}$, with transition rate λ_0 from state 0 to 1, λ from state i to $(i+1)$ for all $i \geq 1$, μ_0 from state 1 to 0, and μ from state $(i+1)$ to i for all $i \geq 1$. This is a birth-death process. A QBD process is a generalization of such a birth-death process, wherein, each state i of the birth-death process is replaced by a set of states. The states in the first set (corresponding to $i = 0$ in the birth-death process) is called the set of *boundary* states, whose behaviour is permitted to differ from that of the remaining states. The remaining sets of states are called the *levels*, and the levels are identical to each other (recall that all states $i \geq 1$ in the birth-death process are identical). The Markov chain may have transitions only within a level or the boundary, between adjacent levels, and between the boundary and the first level. The transition probability matrix of a QBD process is thus of the form

$$\begin{bmatrix} B_1 & B_2 & 0 & 0 & \cdots \\ B_0 & A_1 & A_2 & 0 & \cdots \\ 0 & A_0 & A_1 & A_2 & \cdots \\ 0 & 0 & A_0 & A_1 & \cdots \\ 0 & 0 & 0 & A_0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

Here, the matrices B_0, B_1, B_2, A_0, A_1 and A_2 represent transitions entering the boundary from the first level, within the boundary, exiting the boundary to the first level, entering a level from the next level, within a level, and exiting a level to the next level respectively. If the number of boundary states is q_b, and if the number of states in each level is q_ℓ, then the matrices B_0, B_1 and B_2 have dimensions $(q_\ell \times q_b)$, $(q_b \times q_b)$ and $(q_b \times q_\ell)$ respectively, and each of A_0, A_1 and A_2 have dimensions $(q_\ell \times q_\ell)$. The birth-death process described above is a special case with $q_b = q_\ell = 1$ and $B_0 = \mu_0$, $B_1 = 0$, $B_2 = \lambda_0$, $A_0 = \mu$, $A_1 = 0$, $A_2 = \lambda$. Figures 4, 6 and 9 in the sequel also present examples of QBD processes.

QBD processes are very well understood [18], and their stationary distribution is fairly easy to compute. In this paper, we employ the *SMCSolver* software package [19] for this purpose. In the next two sections, we present scheduling policies which lower and upper bound the performance of the MDS queue, and show that the resulting queues can be represented as QBD processes. This representation makes them them easy to analyse, and this is exploited subsequently in the analysis presented in Section VII.

V. Lower Bounds: MDS-Reservation(t) Queues

This section presents a class of scheduling policies (and resulting queues), which we call the MDS-Reservation(t) scheduling policies (and MDS-Reservation(t) queues), whose performance lower bounds the performance of the MDS queue. This class of scheduling policies are indexed by a parameter 't': a higher value of t leads to a better performance and a tighter lower bound to the MDS queue, but on the downside, requires maintenance of a larger state and is also more complex to analyse.

The MDS-Reservation(t) scheduling policy, in a nutshell, is as follows:

> "*apply the MDS scheduling policy, but with an additional restriction that for any $i \in \{t+1, t+2, \ldots\}$, the i^{th} waiting batch is allowed to move forward in the buffer only when all k of its jobs can move forward together.*"

We first describe in detail the special cases of t= 0 and t= 1, before moving on to the scheduling policy for a general t.

A. MDS-Reservation(0)

1) Scheduling policy: The MDS-Reservation(0) scheduling policy is rather simple: the batch at the head of the buffer may start service only when k or more servers are idle. The policy is described formally in Algorithm 2.

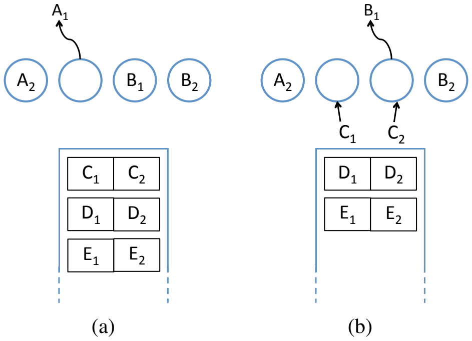


Fig. 3: An illustration of the MDS-Reservation(0) scheduling policy for a system with parameters $(n = 4, k = 2)$. This policy prohibits the servers to process jobs from a batch unless there are k idle servers that can process all k jobs of that batch. As shown in the figure, server 1 is barred from processing $\{C_1, C_2\}$ in (a), but is subsequently allowed to do so when another server also becomes idle in (b).

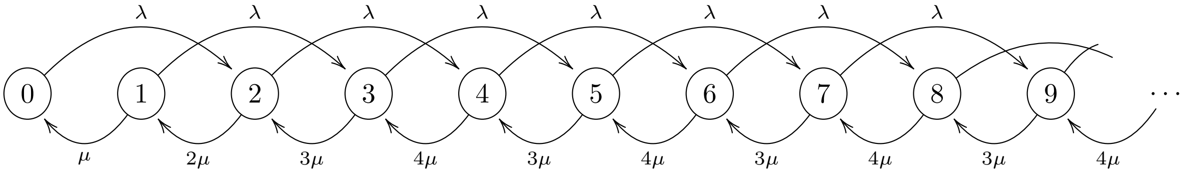


Fig. 4: State transition diagram of the MDS-Reservation(0) queue for $n = 4$ and $k = 2$. The notation at any state is the number of jobs m in the system in that state. The set of boundary states are $\{0, 1, 2\}$, and the levels are pairs of states $\{3, 4\}$, $\{5, 6\}$, $\{7, 8\}$, etc. The transition matrix is of the form (IV) with $B_0 = [0 \quad 0 \quad 3\mu\ ;\ \ 0 \quad 0 \quad 0]$, $B_1 = [-\lambda \quad 0 \quad \lambda\ ;\ \ \mu \quad -(\mu{+}\lambda) \quad 0;\ \ 0 \quad 2\mu \quad -(2\mu{+}\lambda)]$, $B_2 = [0 \quad 0\ ;\ \ \lambda \quad 0\ ;\ \ 0 \quad \lambda]$, $A_0 = [0 \quad 3\mu\ ;\ \ 0 \quad 0]$, $A_1 = [-(3\mu{+}\lambda) \quad 0\ ;\ \ 4\mu \quad -(4\mu{+}\lambda)]$, $A_2 = [\lambda \quad 0\ ;\ \ 0 \quad \lambda]$.

Algorithm 2 MDS-Reservation(0) Scheduling Policy

```
On arrival of a batch
  If number of idle servers $< k$
    append new batch at the end of buffer
  Else
    assign $k$ jobs of the batch to any $k$ idle servers
On departure from server
  If (number of idle servers $\geq k$) and (buffer is non-empty)
    assign $k$ jobs of the first waiting batch to any $k$ idle servers
```

Example 2: Consider the MDS(n=4,k=2) queue in the state depicted in Fig. 2a. Suppose the server 2 completes processing job A_1 (Fig. 3a). Upon this event, the MDS scheduling policy would have allowed server 2 to take up execution of either C_1 or C_2. However, this is not permitted under MDS-Reservation(0), and this server remains idle until a total of at least $k = 2$ servers become idle. Now suppose the third server completes execution of B_1 (Fig. 3b). At this point, there are sufficiently many idle servers to accommodate all $k = 2$ jobs of the batch $\{C_1, C_2\}$, and hence jobs C_1 and C_2 are assigned to servers 2 and 3.

We note that the MDS-Reservation(0) queue, when $n = k$, is identical to a split-merge queue [20].

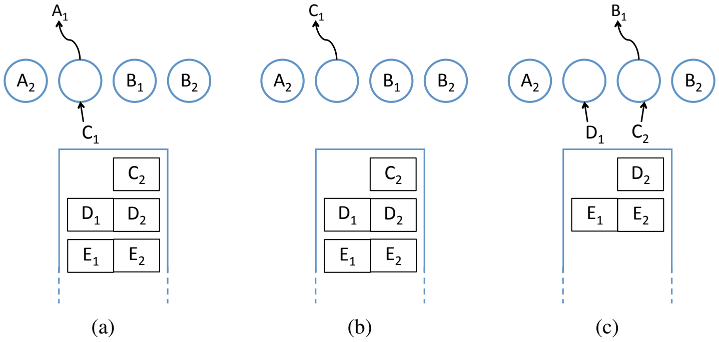
2) Analysis: Observe that under the specific scheduling policy of MDS-Reservation(0), a batch that is waiting in the buffer must necessarily have all its k jobs in the buffer, and furthermore, these k jobs go into the servers at the same time.

We now describe the Markovian representation of the MDS-Reservation(0) queue. We show that it suffices to keep track of only the total number of jobs m in the entire system.

Theorem 1: A Markovian representation of the MDS-Reservation(0) queue has a state space $\{0, 1, \ldots, \infty\}$, and any state $m \in \{0, 1, \ldots, \infty\}$ has transitions to: (i) state $(m + k)$ at rate λ, (ii) if $m \leq n$ then to state $(m - 1)$ at rate $m\mu$, and (iii) if $m > n$ then to state $(m - 1)$ at rate $(n - (n - m) \bmod k))\mu$. The MDS-Reservation(0) queue is thus a QBD process, with boundary states $\{0, 1, \ldots, n - k\}$, and levels $m \in \{n - k + 1 + jk, \ldots, n + jk\}$ for $j = \{0, 1, \ldots, \infty\}$.

The state transition diagram of the MDS-Reservation(0) queue for $(n = 4, k = 2)$ is depicted in Fig. 4.

Theorem 1 shows that the MDS-Reservation(0) queue is a QBD process, allowing us to employ the SMC solver to obtain



(a) (b) (c)

Fig. 5: An illustration of the MDS-Reservation(1) scheduling policy, for a system with parameters $(n = 4, k = 2)$. As shown in the figure, this policy prohibits the servers from processing jobs of the second or later batches (e.g., $\{D_1, D_2\}$ and E_1, E_2 in (b)), until they move to the top of the buffer (e.g., $\{D_1, D_2\}$ in (c)).

its steady-state distribution. Alternatively, the MDS-Reservation(0) queue is simple enough to analyse directly as well. To this end, let $y(m)$ denote the number of jobs being served when the Markov chain is in state m. From the description above, this function can be written as:

$$y(m) = \begin{cases} m, & \text{if} \quad 0 \leq i \leq n \\ n - ((n-m) \bmod k), & \text{if} \quad m > n \ . \end{cases}$$

Let $\pi = [\pi_0 \ \pi_1 \ \pi_2 \ \cdots]$ denote the steady-state distribution of this chain. The global balance equation for the cut between states $(m-1)$ and m gives:

$$\pi_m = \frac{\lambda}{y(m)\mu} \left(\sum_{j=(m-k)^+}^{m-1} \pi_j \right) \qquad \forall \ m > 0. \tag{1}$$

Using these recurrence equations, for any given (n, k), the distribution π of the number of jobs in steady-state can be computed easily.

B. *MDS-Reservation(1)*

1) Scheduling policy: The MDS-Reservation(0) scheduling policy discussed above allows the batches in the buffer to move ahead only when all k jobs in the batch can move together. The MDS-Reservation(1) scheduling policy relaxes this restriction for (only) the job at the head of the buffer. This is formalized in Algorithm 3.

Algorithm 3 MDS-Reservation(1) Scheduling Policy

```
On arrival of a batch
  If buffer is empty
    assign one job each from new batch to idle servers
  append remaining jobs of batch to the end of the buffer
On departure from server (say, server $s$):
  If buffer is non-empty and no job from first waiting batch has been served by $s$
    assign a job from first waiting batch to $s$
    If first waiting batch had only one job in buffer & there exists another waiting batch
      to every remaining idle server, assign a job from second waiting batch
```

Example 3: Consider the MDS(n=4,k=2) queue in the state depicted in Fig. 2a. Suppose server 2 completes processing job A_1 (Fig. 5a). Under MDS-Reservation(1), server 2 now begins service of job C_1 (which is allowed by MDS, but was prohibited under MDS-Reservation(0)). Now suppose that server 2 finishes this service before any other server (Fig. 5b). In this situation, since server 2 has already processed one job from batch $\{C_1, C_2\}$, it is not allowed to process C_2. However, there exists another batch $\{D_1, D_2\}$ in the buffer such that none of the jobs in this batch have been processed by the idle server 2. While the MDS scheduling policy would have allowed server 2 to start processing D_1 or D_2, this is not permitted under MDS-Reservation(1), and the second server remains idle. Now, if server 3 completes service (Fig. 5c), then C_2 is assigned to server 3, allowing batch $\{D_1, D_2\}$ to move up as the first batch. This now permits server 2 to begin service of job D_1.

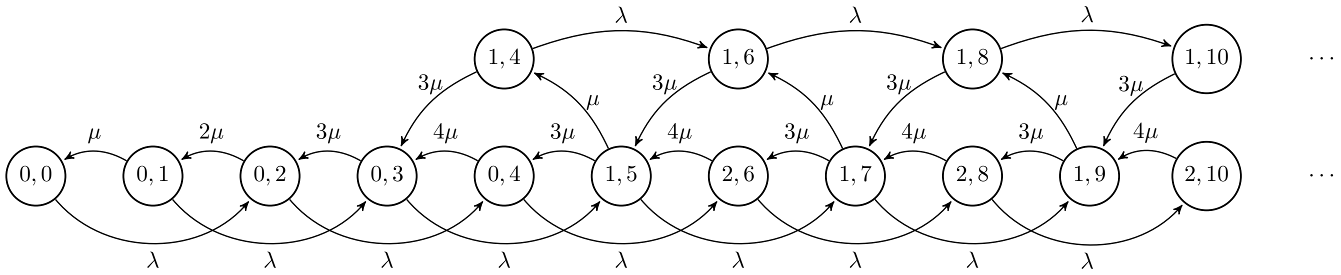


Fig. 6: State transition diagram of the MDS-Reservation(1) queue for $n = 4$ and $k = 2$. The notation at any state is (w_1, m). The subset of states that are never visited are not shown. The set of boundary states are $\{0, 1, 2, 3, 4\} \times \{0, 1, 2\}$, and the levels are sets $\{5, 6\} \times \{0, 1, 2\}$, $\{7, 8\} \times \{0, 1, 2\}$, etc.

The MDS-Reservation(1) scheduling policy is identical to the block-one-Scheduling policy proposed in [8]. While this scheme was analysed in [8] only for the case of $k = 2$, in this paper, we present a more general analysis that holds for all values of the parameter k.

2) Analysis: The following theorem describes the Markovian representation of the MDS-Reservation(1) queue. Each state in this representation is defined by two quantities: (i) the total number of jobs m in the system, and (ii) the number of jobs w_1 of the first waiting batch, that are still in the buffer.

Theorem 2: The Markovian representation of the MDS-Reservation(1) queue has a state space $\{0, 1, \ldots, k\} \times \{0, 1, \ldots, \infty\}$. It is a QBD process with boundary states $\{0, \ldots, k\} \times \{0, \ldots, n\}$, and levels $\{0, \ldots, k\} \times \{n - k + 1 + jk, \ldots, n + jk\}$ for $j = \{1, 2, \ldots, \infty\}$.

The state transition diagram of the MDS-Reservation(1) queue for $(n = 4, k = 2)$ is depicted in Fig. 6.

Note that the state space $\{0, 1, \ldots, k\} \times \{0, 1, \ldots, \infty\}$ has several states that will never be visited during the execution of the Markov chain. For instance, the states $(w_1 > 0, m \le n - k)$ never occur. This is because $w_1 > 0$ implies existence of some job waiting in the buffer, while $m \le n - k$ implies that k or more servers are idle. The latter condition implies that there exists at least one idle server that can process a job from the first waiting batch, and hence the value of w_1 must be smaller than that associated to that state, thus proving the impossibility of the system being in that state.

C. MDS-Reservation(t) for a general t

1) Scheduling policy: Algorithm 4 formally describes the MDS-Reservation(t) scheduling policy.

Algorithm 4 MDS-Reservation(t) Scheduling Policy

```
On arrival of a batch
  If buffer has strictly fewer than t batches
    Assign jobs of new batch to idle servers
  Append remaining jobs of batch to end of buffer
On departure of job from a server (say, server $s$)
  Find $\hat{\imath}$=min{$i$≥1: $s$ has not served job of $i^{\text{th}}$ waiting batch}
  Let $b_{t+1}$ be the $(t+1)^{\text{th}}$ waiting batch (if any)
  If $\hat{\imath}$ exists & $\hat{\imath} \le t$
    Assign a job of $\hat{\imath}^{\text{th}}$ waiting batch to $s$
    If $\hat{\imath} = 1$ & the first waiting batch had only one job in the buffer & $b_{t+1}$ exists
      To every remaining idle server, assign a job from batch $b_{t+1}$
```

The following example illustrates the MDS-Reservation(t) scheduling policy when t= 2.

Example 4: (t=2). Consider the MDS(n=4,k=2) queue in the state depicted in Fig. 2a. Suppose the second server completes processing job A_1 (Fig. 7a). Under the MDS-Reservation(2) scheduling policy, server 2 now begins service of job C_1. Now suppose that server 2 finishes this service as well, before any other server completes its respective service (Fig. 7b). In this situation, while MDS-Reservation(1) would have mandated server 2 to remain idle, MDS-Reservation(2) allows it to start processing a job from the next batch $\{D_1, D_2\}$. However, if the server also completes processing of this job before any other server (Fig. 7c), then it is not allowed to take up a job of the third batch $\{E_1, E_2\}$. Now suppose server 3 completes service

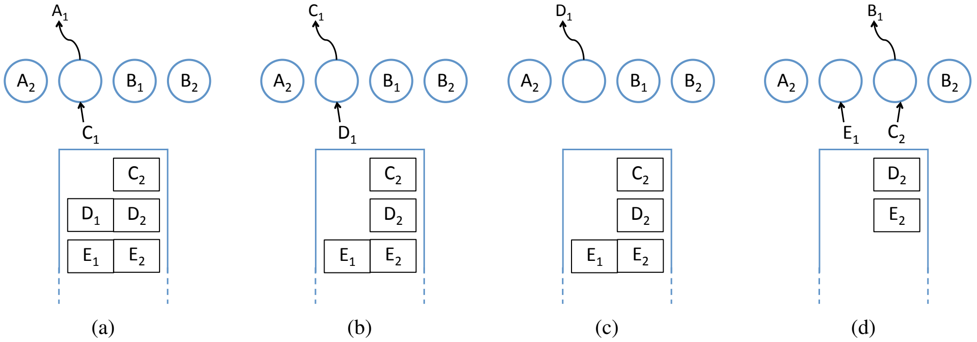


Fig. 7: An illustration of the working of the MDS-Reservation(2) scheduling policy, for a system with parameters $(n = 4, k = 2)$. As shown in the figure, this policy prohibits the servers from processing jobs of the third and later batches (e.g., batch $\{E_1, E_2\}$ in (c)), until they move higher in the buffer (e.g., as in (d)).

(Fig. 7d). Server 3 can begin serving job C_2, thus clearing batch $\{C_1, C_2\}$ from the buffer, and moving the two remaining batches up in the buffer. Batch $\{E_1, E_2\}$ is now within the threshold of t= 2, allowing it to be served by the idle server 2.

2) Analysis:

Theorem 3: The Markovian representation of the MDS-Reservation(t) queue has a state space $\{0, 1, \ldots, k\}^t \times \{0, 1, \ldots, \infty\}$. It is a QBD process with boundary states $\{0, \ldots, k\}^t \times \{0, \ldots, n-k+tk\}$, and levels $\{0, \ldots, k\}^t \times \{n-k+1+jk, \ldots, n+jk\}$ for $j = \{t,\, t+1, \ldots, \infty\}$.

The proof of Theorem 3 also describes how one can obtain the configuration of the entire system from only the number of jobs in the system, under the MDS-Reservation(t) scheduling policies.

One can see that the sequence of MDS-Reservation(t) queues, as t increases, becomes closer to the MDS queue. This results in tighter bounds, and also increased complexity of the transition diagrams. The limit of this sequence is the MDS queue itself.

Theorem 4: The MDS-Reservation(t) queue, when t $= \infty$, is precisely the MDS queue.

VI. Upper bounds: $M^k/M/n(t)$ Queues

In this section, we present a class of scheduling policies (and resulting queues), which we call the M^k/M/n(t) scheduling policies (and M^k/M/n(t) queues), whose performance upper bounds the performance of the MDS queue. The scheduling policies presented here relax the constraint of requiring the k jobs in a batch to be processed by k distinct servers. While the M^k/M/n(t) scheduling policies and the M^k/M/n(t) queues are not realizable in practice, they are presented here only to obtain upper bounds to the performance of the MDS queue.

The M^k/M/n(t) scheduling policy, in a nutshell, is as follows:

> "*apply the MDS scheduling policy whenever there are t or fewer batches in the buffer; when there are more than t batches in the buffer, ignore the restriction requiring the k jobs of a batch to be processed by distinct servers.*"

We first describe the M^k/M/n(0) queue in detail, before moving on to the general M^k/M/n(t) queues.

A. *M^k/M/n(0)*

1) Scheduling policy: The M^k/M/n(0) scheduling policy operates by completely ignoring the restriction of assigning distinct servers to jobs of the same batch. This is described formally in Algorithm 5.

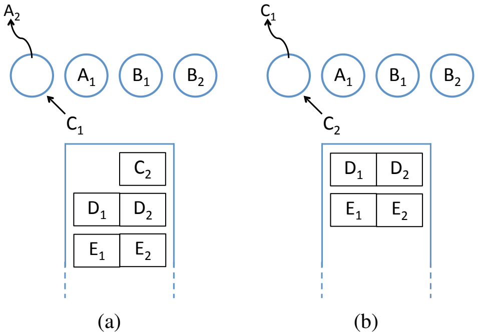


Fig. 8: Illustration of the working of the M^k/M/n(0) scheduling policy. This policy allows a server to process more than one jobs of the same batch. As shown in the figure, server 1 processes both C_1 and C_2.

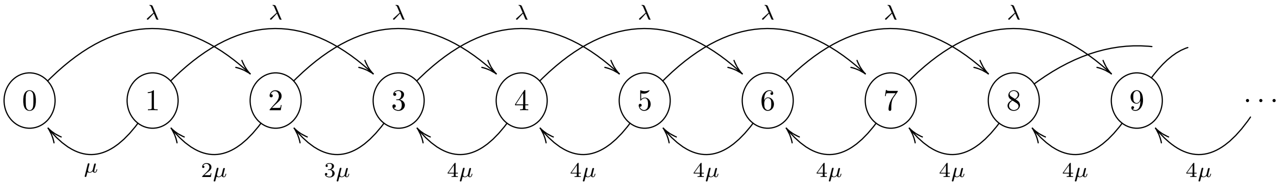


Fig. 9: State transition diagram of the M^k/M/n(0) queue for $n = 4$ and $k = 2$. The notation at any state is the number of jobs m in the system in that state. The set of boundary states are $\{0, 1, 2, 3, 4\}$, and the levels are pairs of states $\{5, 6\}$, $\{7, 8\}$, etc. The transition matrix is of the form (IV) with $B_0 = [0 \ \ 0 \ \ 0 \ \ 0 \ \ 4\mu \ ; 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0]$, $B_1 = [-\lambda \ \ 0 \ \ \lambda \ \ 0 \ \ 0; \ \ \mu \ \ -(\mu{+}\lambda) \ \ 0 \ \ \lambda \ \ 0; \ \ 0 \ \ 2\mu \ \ -(2\mu{+}\lambda) \ \ 0 \ \ \lambda; \ \ 0 \ \ 0 \ \ 3\mu \ \ -(3\mu{+}\lambda) \ \ 0; \ \ 0 \ \ 0 \ \ 0 \ \ 4\mu \ \ -(4\mu{+}\lambda)]$, $B_2 = [0 \ \ 0 \ ; \ \ 0 \ \ 0 \ ; \ \ 0 \ \ 0 \ ; \ \ \lambda \ \ 0 \ ; \ \ 0 \ \ \lambda]$, $A_0 = [0 \ \ 4\mu \ ; \ \ 0 \ \ 0]$, $A_1 = [-(4\mu{+}\lambda) \ \ 0 \ ; \ \ 4\mu \ \ -(4\mu{+}\lambda)]$, $A_2 = [\lambda \ \ 0 \ ; \ \ 0 \ \ \lambda]$.

Algorithm 5 M^k/M/n(0)

```
On arrival of a batch
  assign jobs of this batch to idle servers (if any)
  append remaining jobs at the end of the buffer
On departure from a server
  If buffer is not empty
    assign a job from the first waiting batch to this server
```

Note that the M^k/M/n(0) queue is identical to the M^k/M/n queue, i.e., an $M/M/n$ queue with batch arrivals.

The following example illustrates the working of the M^k/M/n(0) scheduling policy.

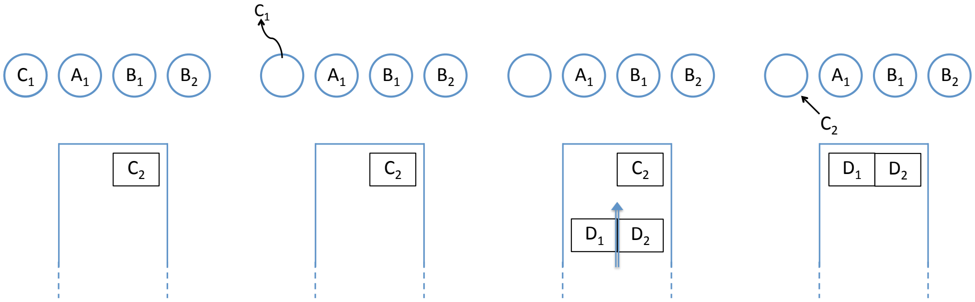
Example 5: Consider the MDS(n=4,k=2) queue in the state depicted in Fig. 2a. Suppose the first server completes processing job A_2, as shown in Fig. 8a. Under the M^k/M/n(0) scheduling policy, server 1 now takes up job C_1. Next suppose server 1 also finishes this task before any other server completes service (Fig. 8b). In this situation, the MDS scheduling policy would prohibit job C_2 to be served by the first server. However, under the scheduling policy of M^k/M/n(0), we relax this restriction, and permit server 1 to begin processing C_2.

2) Analysis: We now describe a Markovian representation of the M^k/M/n(0) scheduling policy, and show that it suffices to keep track of only the total number of jobs m in the system.

Theorem 5: The Markovian representation of the M^k/M/n(0) queue has a state space $\{0, 1, \ldots, \infty\}$, and any state $m \in \{0, 1, \ldots, \infty\}$, has transitions (i) to state $(m + k)$ at rate λ, and (ii) if $m > 0$, then to state $(m - 1)$ at rate $\min(n, m)\mu$. It is a QBD process with boundary states $\{0, \ldots, k\} \times \{0, \ldots, n\}$, and levels $\{0, \ldots, k\} \times \{n - k + 1 + jk, \ldots, n + jk\}$ for $j = \{1, 2, \ldots, \infty\}$.

The state transition diagram of the M^k/M/n(0) queue for $n = 4, k = 2$ is shown in Fig. 9.

Theorem 5 shows that the MDS-Reservation(0) queue is a QBD process, allowing us to employ the SMC solver to obtain its steady-state distribution. Alternatively, the M^k/M/n(0) queue is simple enough to allow for a direct analysis as well. Let π_m denote the stationary probability of any state $m \in \{0, 1, \ldots, \infty\}$. Then, for any $m \in \{1, \ldots, \infty\}$, the global balance equation



(a) (b) (c) (d)

Fig. 10: Illustration of the working of the M$^{\text{k}}$/M/n(1) scheduling policy. This policy allows a server to begin processing a job of a batch that it has already served, unless this batch is the only batch waiting in the buffer. As shown in the figure, server 1 cannot process C_2 in (b) since it has already processed C_1 and C is the only waiting batch; this restriction is removed upon on arrival of another batch in the buffer in (d).

for the cut between states $(m-1)$ and m gives:

$$\pi_m = \frac{\lambda}{\min(m,n)\mu} \sum_{j=(m-k)^+}^{m-1} \pi_j \ . \tag{2}$$

The stationary distribution of the Markov chain can now be computed easily from these equations.

B. M$^{\text{k}}$/M/n(t) for a general t

1) Scheduling policy: Algorithm 6 formally describes the M$^{\text{k}}$/M/n(t) scheduling policy.

Algorithm 6 M$^{\text{k}}$/M/n(t) Scheduling Policy

```
On arrival of a batch
  If buffer has strictly fewer than t batches
    Assign jobs of new batch to idle servers
  Else if buffer has t batches
    Assign jobs of first batch to idle servers
    If first batch is cleared
      Assign jobs of new batch to idle servers
  Append remaining jobs of new batch to end of buffer
On departure of job from a server (say, server $s$)
  If number of batches in buffer is strictly greater than t
    Assign job from first batch in buffer to this server
  Else
    Among all batches in buffer that $s$ has not served, find the one that arrived earliest; assign a job of this batch to $s$
```

Example 6: (t=1). Consider a system in the state shown in Fig. 10a. Suppose server 1 completes execution of job C_1 (Fig. 10b). In this situation, the processing of C_2 by server 1 would be allowed under M$^{\text{k}}$/M/n(0), but prohibited in the MDS queue. The M$^{\text{k}}$/M/n(1) queue follows the scheduling policy of the MDS queue whenever the total number of batches in the buffer is no more than 1, and hence in this case, server 1 remains idle. Next, suppose there is an arrival of a new batch (Fig. 10c). At this point there are two batches in the buffer, and the M$^{\text{k}}$/M/n(1) scheduling policy switches its mode of operation to allowing any server to serve any job. Thus, the first server now begins service of C_2 (Fig. 10d).

2) Analysis:

Theorem 6: The state transition diagram of the M$^{\text{k}}$/M/n(t) queue has a state space $\{0,1,\ldots,k\}^t \times \{0,1,2,\ldots\}$. It is a QBD process with boundary states $\{0,\ldots,k\}^t \times \{0,\ldots,n+tk\}$, and levels $\{0,\ldots,k\}^t \times \{n-k+1+jk,\ldots,n+jk\}$ for $j = \{t+1, t+2, \ldots, \infty\}$.

The proof of Theorem 6 also describes how one can obtain the configuration of the entire system from only the number of jobs in the system, under the M$^{\text{k}}$/M/n(t) scheduling policies.

As in the case of MDS-Reservation(t) queues, one can see that the sequence of M$^{\mathrm{k}}$/M/n(t) queues, as t increases, becomes closer to the MDS queue. On increase in the value of parameter t, the bounds become tighter, but the complexity of the transition diagrams also increases, and the limit of this sequence is the MDS queue itself.

Theorem 7: The M$^{\mathrm{k}}$/M/n(t) queue, when t$=\infty$, is precisely the MDS(n,k) queue.

Remark 1: The class of queues presented in this section have another interesting intellectual connection with the MDS queue: the performance of an MDS(n,k) queue is lower bounded by the M$^{\mathrm{k}}$/M/(n-k+1)(t) queue for all values of t.

VII. Performance Comparison of Various Scheduling Policies

In this section we analyse the performance of the MDS-Reservation(t) and the M$^{\mathrm{k}}$/M/n(t) queues. The analysis is performed by first casting these queues as quasi-birth-death processes (as shown in Theorems 3 and 6), and then building on the properties of quasi-birth-death processes to compute the average latency and throughput of each of these queues. Via simulations, we then validate these analytical results and also look at the performance of these queues with respect to additional metrics. We see that under each of these metrics, for the parameters simulated, the performance of the lower bound MDS-Reservation(t) queues (with $t = 3$) closely follows the performance of the upper bound M$^{\mathrm{k}}$/M/n(t) queues (with $t = 1$).

A. Maximum throughput

The maximum throughput is the maximum possible number of requests that can be served by the system per unit time.

Theorem 8: Let $\rho^*_{\text{Resv(t)}}$, ρ^*_{MDS}, and $\rho^*_{M^k/M/n(t)}$ denote the maximum throughputs of the MDS-Reservation(t), MDS, and M$^{\mathrm{k}}$/M/n(t) queues respectively. Then,

$$\rho^*_{MDS} = \rho^*_{M^k/M/n(t)} = \frac{n}{k} \ .$$

When k is treated as a constant,

$$\left(1 - O(n^{-2})\right) \frac{n}{k} \leq \rho^*_{\text{Resv(t)}} \leq \frac{n}{k} \ .$$

In particular, for any $t \geq 1$, when $k = 2$,

$$\left(1 - \frac{1}{2n^2 - 2n + 1}\right) \frac{n}{k} = \rho^*_{\text{Resv(1)}} \leq \rho^*_{\text{Resv(t)}} \ ,$$

and when $k = 3$,

$$\left(1 - \frac{4n^3 - 8n^2 + 2n + 4}{3n^5 - 12n^4 + 22n^3 - 29n^2 + 26n - 8}\right) \frac{n}{k} = \rho^*_{\text{Resv(1)}} \leq \rho^*_{\text{Resv(t)}}$$

Note that the special case of Theorem 8, for the MDS-Reservation(1) queue with $k = 2$, also recovers the throughput results of [8]. Moreover, as compared to the proof techniques of [8], the proofs in this paper are much simpler and do not require computation of the stationary distribution. Using the techniques presented in the proof of Theorem 8, explicit bounds analogous to those for $k = 2,\ 3$ in Theorem 8 can be computed for $k \geq 4$ as well.

Fig. 11 plots loss in maximum throughput incurred by the MDS-Reservation(1) and the MDS-Reservation(2) queues, as compared to that of the MDS queue.

B. System occupancy

The system occupancy at any given time is the number of jobs present (i.e., that have not yet been served completely) in the system at that time. This includes jobs that are waiting in the buffer as well as the jobs being processed in the servers at that time. The distribution of the system occupancy is obtained directly from the stationary distribution of the Markov chains constructed in sections V and VI. Fig. 12 plots the complementary cdf of the the number of jobs in the system in the steady state. Observe that the analytical upper and lower bounds from the MDS-Reservation(2) and the M$^{\mathrm{k}}$/M/n(1) queues respectively are very close to each other.

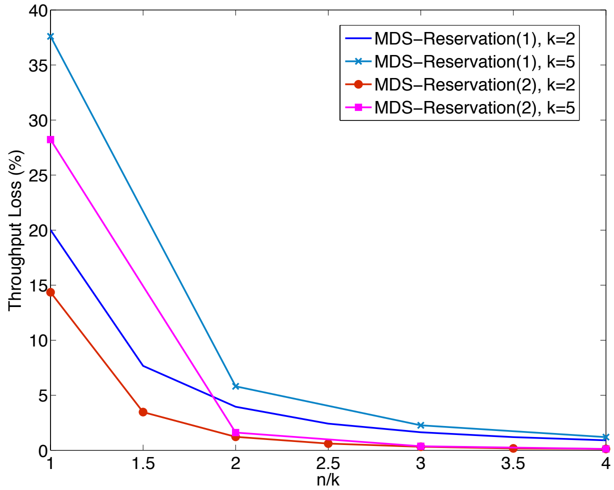


Fig. 11: Loss in maximum throughput incurred by the MDS-Reservation(1) and the MDS-Reservation(2) queues as compared to that of the MDS queue.

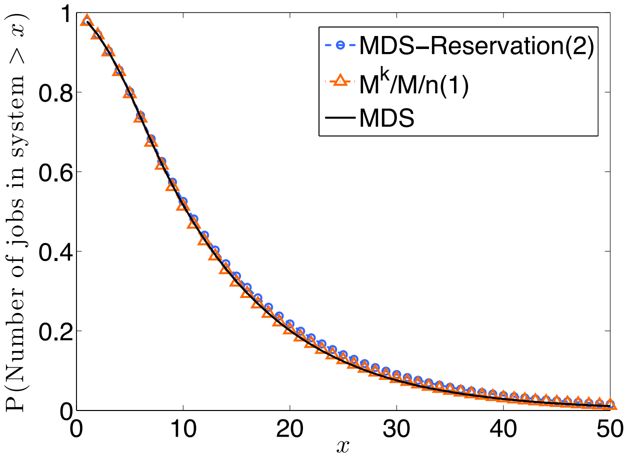


Fig. 12: Steady state distribution (complementary cdf) of the system occupancy when $n = 10$, $k = 5$, $\lambda = 1.5$ and $\mu = 1$.

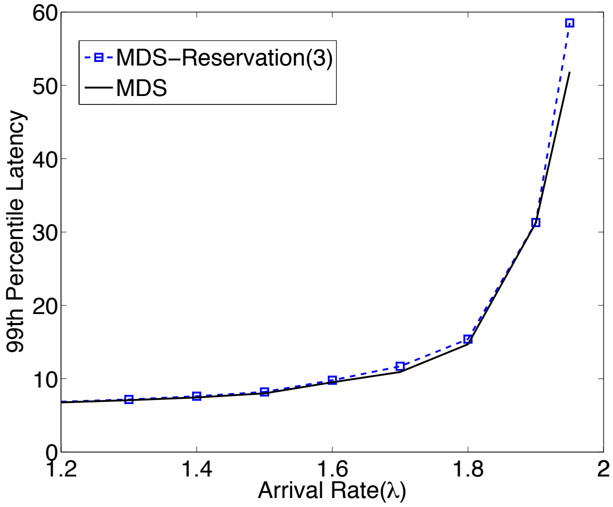


Fig. 13: The 99th percentile of the distribution of the latency in a system with parameters $n = 10$, $k = 5$ and $\mu = 1$. For any arrival rate λ, a curve takes value $y(\lambda)$ if exactly 1% of the batches incur a delay greater than $y(\lambda)$

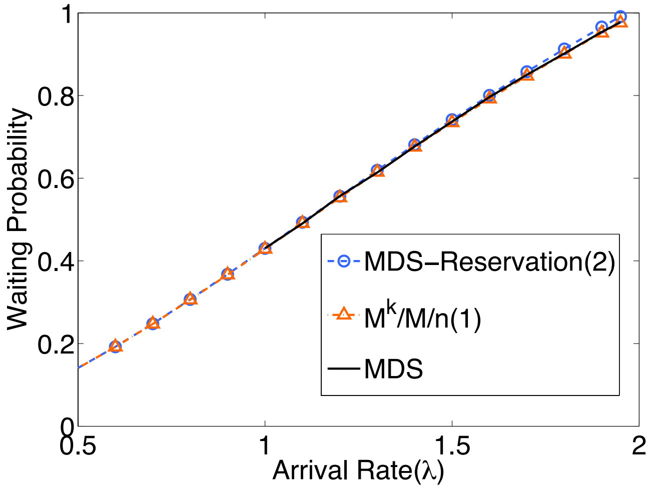


Fig. 14: Waiting probability in a system with $n = 10$, $k = 5$ and $\mu = 1$.

C. Latency

The latency faced by a batch is the time from its arrival into the system till the completion of service of k of its jobs.

Average: We have analytically computed the average batch latencies of the MDS-Reservation(t) and the M^{k}/M/n(t) queues in the following manner. We first compute the steady state distribution π of the number of jobs in the system. As discussed in the previous sections, each state i in the corresponding Markov chain is associated to a unique configuration of the jobs in the system. Now, the average latency d_i faced by a batch entering when the system is in state i can be computed easily by employing the Markovian representations of the queues presented previously. Since Poisson arrivals see time averages [21], the average latency faced by a batch in the steady state is given by $\sum_i \pi_i d_i$.

Fig. 1 plots the average latency faced by a batch in the steady state. Observe that the performance of the MDS-Reservation(t) scheduling policy, for t as small as 3, is extremely close to that of the MDS scheduling policy and to the upper bounding M^{k}/M/n(1) scheduling policy.

Tails (simulations only): We also look at the tails of the latency-distribution via simulations. Fig. 13 plots the 99th percentile of the distribution of the latency. Also observe how closely the bound MDS-Reservation(3) follows the exact MDS.

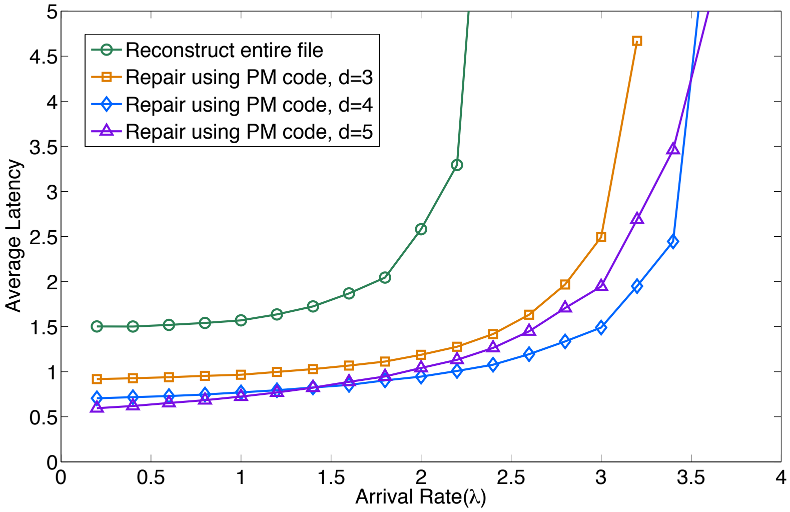


Fig. 15: Average latency during degraded reads. The parameters associated to this system are $n = 6$, $k = 2$. The service time at any server is exponentially distributed with a mean proportional to the amount of data to be read. The performance of the product-matrix (PM) codes are compared for various values of the associated parameter d.

D. *Waiting probability*

The waiting probability is the probability that, in the steady state, one or more jobs of an arriving batch will have to wait in the buffer and will not be served immediately. As shown previously, under the MDS-Reservation(t) and the M^{k}/M/n(t) scheduling policies, the system configuration at any time is determined by the state of the Markov chain, which also determines whether a newly arriving batch needs to wait or not. Thus, the waiting probability can be computed directly from the steady state distribution of the number of jobs in the system. Fig. 14 plots the waiting probability for the different queues considered in the paper. Observe how tightly the M^{k}/M/n(1) and the MDS-Reservation(2) queues bound the waiting probability of the MDS queue.

E. *Degraded Reads*

The system considered so far assumed that each incoming request desires one entire file (from any k servers). In certain applications, incoming requests may sometimes require only a part of the file, say, the part that is stored in one of the servers. In the event that a server is busy or has failed, one may serve requests for that part from the data stored in the remaining servers. This operation is termed a ‘degraded read’. Under an MDS code, a degraded read may be performed by obtaining the entire file from the data stored in any k of the $(n-1)$ remaining servers, and then extracting the desired part. Such an operation is generally called ‘data-reconstruction’.

Dimakis et al. recently proposed a new model, called the *regenerating codes model*, as a basis to design alternative codes supporting faster degraded reads. Several explicit codes under this model have been constructed subsequently, e.g., [22]–[25]. In particular, the *product-matrix (PM) codes* proposed in [23] are practical codes that possess several appealing properties. One feature of the product-matrix codes is that they are associated to an additional parameter d $(\leq n-1)$, and can recover the desired data by reading and downloading a fraction $\frac{1}{d-k+1}^{\text{th}}$ of the requisite data from **any** d of the remaining $(n-1)$ servers. This method of performing degraded reads entails a smaller total download but requires connectivity to more nodes, and hence the gains achieved by this method in a dynamic setting are unclear. This operation is generally called a ‘node-repair’ operation in the literature.

We employ the framework of the MDS queue to compare these two methods of performing degraded reads: the data-reconstruction method results in an MDS(n-1,k) queue, while the node-repair operation leads to an MDS(n-1,d) queue (with a different service rate). Fig. 15 depicts average latency incurred under the two methods when $n = 6$, $k = 2$ and $d = 3$. We see that the product-matrix/regenerating codes perform consistently better in terms of the latency performance. The key insight is that the property of being able to read from **any** d servers provides a great deal of flexibility to the degraded-read operations under a product-matrix code, enabling it to match up (and beat) the performance of the data-reconstruction operation.

VIII. Discussion and Open Problems

We aim to use the results of this paper as a starting point for analysis of more complex systems that mimic the real world more closely. In particular, we intend to build upon the "MDS-Queue" framework presented here to analyse queues that relax one or more of the assumptions made in the paper, e.g., having general service times, heterogeneous requests or servers, non-MDS codes, and decentralized queues.

In this paper, while we characterized the metrics of average latency and throughput analytically, we also used simulations to examine the performance of the queues in terms of several additional metrics. We observed in these simulations that the MDS-Reservation(t) and the M^k/M/n(t) scheduling policies result in a performance very close to that of the exact MDS queue for values of t as small as 3. Thus, even respect to these metrics, an analysis of the (simpler) scheduling policies of MDS-Reservation(t) and M^k/M/n(t) may provide rather-accurate estimations of the performance of MDS codes.

References

[1] D. Borthakur, "HDFS and Erasure Codes (HDFS-RAID)," 2009. [Online]. Available: http://hadoopblog.blogspot.com/2009/08/hdfs-and-erasure-codes-hdfs-raid.html

[2] "Erasure codes: the foundation of cloud storage," Sep. 2010. [Online]. Available: http://blog.cleversafe.com/?p=508

[3] D. Ford, F. Labelle, F. Popovici, M. Stokely, V. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems," in *9th USENIX Symposium on Operating Systems Design and Implementation*, Oct. 2010.

[4] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in Windows Azure Storage," in *USENIX Annual Technical Conference (ATC)*, Jun. 2012.

[5] K. V. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the Facebook warehouse cluster," in *Proc. USENIX HotStorage*, Jun. 2013.

[6] F. MacWilliams and N. Sloane, *The Theory of Error-Correcting Codes, Part I.* North-Holland Publishing Company, 1977.

[7] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *Micro, IEEE*, vol. 25, no. 6, pp. 10–16, 2005.

[8] L. Huang, S. Pawar, H. Zhang, and K. Ramchandran, "Codes can reduce queueing delay in data centers," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, Cambridge, Jul. 2012.

[9] U. Ferner, M. Medard, and E. Soljanin, "Toward sustainable networking: Storage area networks with network coding," in *50th Annual Allerton Conference on Communication, Control, and Computing*, Oct. 2012.

[10] F. Baccelli, A. Makowski, and A. Shwartz, "The fork-join queue and related systems with synchronization constraints: stochastic ordering and computable bounds," *Advances in Applied Probability*, pp. 629–660, 1989.

[11] E. Varki, A. Merchant, and H. Chen, "The M/M/1 fork-join queue with variable sub-tasks," *Unpublished*, 2008.

[12] N. B. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?" in *Annual Allerton Conference on Communication, Control, and Computing*, 2013.

[13] G. Joshi, Y. Liu, and E. Soljanin, "Coding for fast content download," in *50th Annual Allerton Conference on Communication, Control, and Computing*, Oct. 2012.

[14] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[15] G. Liang and U. Kozat, "Fast cloud: Pushing the envelope on delay performance of cloud storage with coding," *arXiv:1301.1294*, Jan. 2013.

[16] A. Vulimiri, O. Michel, P. Godfrey, and S. Shenker, "More is less: Reducing latency via redundancy," in *11th ACM Workshop on Hot Topics in Networks*, Oct. 2012, pp. 13–18.

[17] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Why let resources idle? Aggressive cloning of jobs with Dolly," in *USENIX HotCloud*, Jun. 2012.

[18] R. Feldman and C. Valdez-Flores, *Applied Probability and Stochastic Processes (Chapter 13).* Springer, 2010.

[19] D. Bini, B. Meini, S. Steffé, and B. Van Houdt, "Structured markov chains solver: software tools," in *Workshop on Tools for solving structured Markov chains*, 2006.

[20] P. Harrison and S. Zertal, "Queueing models with maxima of service times," *Computer Performance Evaluation. Modelling Techniques and Tools*, pp. 152–168, 2003.

[21] R. Wolff, "Poisson arrivals see time averages," *Operations Research*, vol. 30, no. 2, pp. 223–231, 1982.

[22] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran, "Explicit construction of optimal exact regenerating codes for distributed storage," in *Proc. 47th Annual Allerton Conference on Communication, Control, and Computing*, Urbana-Champaign, Sep. 2009, pp. 1243–1249.

[23] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Optimal exact-regenerating codes for the MSR and MBR points via a product-matrix construction," *IEEE Transactions on Information Theory*, vol. 57, no. 8, pp. 5227–5239, Aug. 2011.

[24] D. Papailiopoulos, A. Dimakis, and V. Cadambe, "Repair optimal erasure codes through hadamard designs," in *Proc. 47th Annual Allerton Conference on Communication, Control, and Computing*, Sep. 2011, pp. 1382–1389.

[25] I. Tamo, Z. Wang, and J. Bruck, "Access vs. bandwidth in codes for storage," in *IEEE International Symposium on Information Theory (ISIT)*, Cambridge, Jul. 2012, pp. 1187–1191.

Appendix
Proofs of Theorems

Additional Notation used in the proofs: Table I enumerates notation for various parameters that describe the system at any given time. To illustrate this notation, consider again the system depicted in Fig. 2a. Here, the parameters listed in Table I take values $m = 10$, $z = 0$, $b = 3$, $s_1 = 0$, $s_2 = 0$, $s_3 = 0$, $w_1 = 2$, $w_2 = 2$ and $w_3 = 2$. One can verify that keeping track of these parameters leads to a valid Markov chain (under each of the scheduling policies discussed in this paper). Note that we do not keep track of the jobs of a batch once all k jobs of that batch have begun to be served, nor do we track what servers are serving what jobs. This is to ensure a smaller complexity of representation an2d computation. Further note that in terms of the parameters listed in Table I, the number of servers that are busy at any given time is equal to $(n-z)$. For batch i in the buffer ($i \in \{1,\ldots,b\}$), the number of jobs that have completed service is equal to $(k-s_i-w_i)$. For any integer i, $w_i=0$ will mean that there is no i^{th} waiting batch in the buffer.

Proof of Theorem 1: Since the scheduling policy mandates all k jobs of any batch to start service together, the number of jobs in the buffer is necessarily a multiple of k. Furthermore, when the buffer is not empty, the number of servers that are idle must be strictly smaller than k (since otherwise, the first waiting batch can be served). It follows that when $m \leq n$, the buffer is empty ($b=0$), and all m jobs are being served by m servers (and $z=(n-m)$ servers are idle). When $m>n$, the buffer is not empty. Assuming there are z idle servers, there must be $(n-z)$ jobs currently being served, and hence there are $m-(n-z)$ jobs in the buffer. However, since the number of jobs in the buffer must be a multiple of k, and since $z \in \{0,1,\ldots,k-1\}$, it must be that $z=(n-m) \bmod k$. Thus, when $m>n$, there are $b=\frac{m-n+z}{k}$ batches waiting in the buffer, and $w_i=k,\ s_i=0\ \forall\ i \in \{1,\ldots,b\}$. We have thus shown that the knowledge of m suffices to completely describe the system.

Once we have determined the configuration of the system as above, it is now easy to obtain the transitions between the states. An arrival of a batch increases the total number of jobs in the system by k, and hence the transition from state m to $(m+k)$ at rate λ. When $m \leq n$, all m jobs are being served, and the buffer is empty. Thus, the total number of jobs in the system reduces to $(m-1)$ at rate $m\mu$. When $m>n$, the number of jobs being served is $n-z=n-((n-m) \bmod k)$, and thus there is a transition from state m to $(m-1)$ at rate $(n-((n-m) \bmod k))\mu$. ■

Proof of Theorem 2: Results as a special case of Theorem 3. As a side note, in any given state $(w_1,m) \in \{0,1,\ldots,k\} \times \{0,1,\ldots,\infty\}$ of the resulting Markov chain, the number of idle servers is given by $z=n-m$ if $m \leq n-k$, and $z=(n+w_1-m) \bmod k$ otherwise. The state (w_1,m) has transitions to state:

- $((m+k-n)^+, m+k)$ at rate λ, if $w_1=0$.
- $(w_1, m+k)$ at rate λ, if $w_1 \neq 0$
- $(w_1, m-1)$ at rate $m\mu$, if $w_1=0$.
- $(w_1, m-1)$ at rate $(k-w_1-z)\mu$, if $w_1 \neq 0$
- $(w_1-1, m-1)$ at rate $(n-k+w_1)\mu$, if $(w_1>1$ or $(w_1=1$ & $m \leq n+1))$
- $((k-z)^+, m-1)$ at rate $(n-k+w_1)\mu$, if $(w_1=1$ & $m>n+1)$.

■

Proof of Theorem 3: For any state of the system $(w_1,w_2,\ldots,w_t,m) \in \{0,1,\ldots,k\}^t \times \{0,1,\ldots,\infty\}$, define

$$q=\begin{cases}0 & \text{if } \ w_1=0\\ t & \text{else if } \ w_t \neq 0\\ \arg\max\{t' : w_{t'} \neq 0,\ 1 \leq t' \leq t\} & \text{otherwise.}\end{cases} \tag{3}$$

It can be shown that

$$b=\begin{cases}0 & \text{if } \ q=0\\ q & \text{if } \ 0<q<t\\ t+\left\lfloor \frac{m-\sum_{j=1}^{t} w_i - n}{k} \right\rfloor & \text{otherwise,}\end{cases} \tag{4}$$

$$z=n-(m-\sum_{j=1}^{t} w_j-(b-t)^+k)\ , \tag{5}$$

Value	Meaning	Range
m	number of jobs in the entire system	0 to ∞
z	number of idle servers	0 to n
b	number of waiting batches	0 to ∞
$\{s_i\}_{i=1}^{b}$	number of of jobs of i^{th} waiting batch, in the servers	0 to $k-1$
$\{w_i\}_{i=1}^{b}$	number of of jobs of i^{th} waiting batch, in the buffer	0 to k

TABLE I: Notation used to describe state of the system.

$$\text{for } i \in \{1,\ldots,b\}, \quad s_i = \begin{cases} w_{i+1} - w_i & \text{if } \; i \in \{1,\ldots,q-1\} \\ k - z - w_q & \text{if } \; i = q \\ 0 & \text{if } \; i \in \{q+1,\ldots,b\}, \end{cases} \tag{6}$$

and

$$\text{for } i \in \{t+1,\ldots,b\}, \quad w_i = k \; . \tag{7}$$

Given the complete description of the state of the system as above, the characterization of the transition diagram is a straightforward task.

It is not difficult to see that the MDS-Reservation(t) queue has the following two key features: (a) any transition event changes the value of m by at most k, and (b) for $m \geq n-k+1+tk$, the transition from any state $(w_1,\, m)$ to any other state $(w_1',\, m' \geq n-k+1+tk)$ depends on $m \bmod k$ and not on the actual value of m. This results in a QBD process with boundary states and levels as specified in the statement of the theorem. Intuitively, this says that when $m \geq n-k+1+tk$, the presence of an additional batch at the end of the buffer has no effect on the functioning of the system. (In contrast, when $m < n-k+1+tk$, the system may behave differently if there was to be an additional batch, due to the possibility of this batch being within the threshold t. For instance, a job of this additional batch may be served upon completion of service at a server, which is not possible if this additional batch was not present).

■

Proof of Theorem 4: The MDS-Reservation(t) scheduling policy treats the first t waiting batches in the buffer as per the MDS scheduling policy, while imposing an additional restriction on batches $(t+1)$ onwards. When t$=\infty$, every batch is treated as in MDS, thus making MDS-Reservation(∞) identical to MDS. ■

Proof of Theorem 5: Under M$^{\text{k}}$/M/n(0), any job can be processed by any server, and hence a server may be idle only when the buffer is empty. Thus, when $m \leq n$, all m jobs are in the servers, and the buffer is empty. When $m > n$, all the n servers are full and the remaining $(m-n)$ jobs are in the buffer. The transitions follow as a direct consequence of these observations. It also follows that when $m \leq n$, $b = 0$ and $z = n - m$. In addition, in state $m\,(> n)$ it must be that $w_1 = (m-n)\text{mod}k$, $b = \lceil \frac{m-n}{k} \rceil$, and for $i \in \{2,\ldots,b\}$, $w_i = k$. Thus the knowledge of m suffices to describe the configuration of the entire system. ■

Proof of Theorem 6: For any state $(w_1$,w_2,...,w_t,$m)$, define q as in (3). The values of b, z, w_i, are identical to that in the proof of Theorem 3. Given the complete description of the state of the system as above, the characterization of the transition diagram is a straightforward task. It is not difficult to see that the M$^{\text{k}}$/M/n(t) queues have the following two key features: (a) any transition event changes the value of m by at most k, and (b) for $m \geq n+1+tk$, the transition from any state $(w_1$,$m)$ to any other state $(w_1'$,$m' \geq n+1+tk)$ depends on $m \bmod k$ and not on the actual value of m. This results in a QBD process with boundary states and levels as specified in the statement of the theorem. Intuitively, this says that when $m \geq n+1+tk$, the total number of waiting batches is strictly greater than t. In this situation, the presence of an additional batch at the end of the buffer has no effect on the functioning of the system. ■

Proof of Theorem 7: The M$^{\text{k}}$/M/n(t) scheduling policy follows the MDS scheduling policy when the number of batches in the buffer is less than or equal to t. Thus, M$^{\text{k}}$/M/n(∞) is always identical to MDS. ■

Proof of Theorem 8: In the MDS queue, suppose there are a large number of batches waiting in the buffer. Then, whenever a server completes a service, one can always find a waiting batch that has not been served by that server. Thus, no server is ever idle. Since the system has n servers, each serving jobs with times i.i.d. exponential with rates μ, the average number of jobs exiting the system per unit time is $n\mu$. The above argument also implies that under no circumstances (under any scheduling policy), can the average number of jobs exiting the system per unit time exceed $n\mu$. Finally, since each batch consists of k jobs, the rate at which batches exit the system is $\lambda^*_{MDS} = \frac{n\mu}{k}$ per unit time. Since the M$^{\text{k}}$/M/n(t) queues upper bound the performance of the MDS queue, $\lambda^*_{\text{M}^k\text{/M/n(t)}} = \frac{n\mu}{k}$ for every t.

We shall now evaluate the maximum throughput of MDS-Reservation(1) by exploiting properties of QBD systems. In general, the maximum throughput λ^* of any QBD system is the value of λ such that: $\exists$ $\mathbf{v}$ satisfying $\mathbf{v}^T(A_0 + A_1 + A_2) = 0$ and $\mathbf{v}^T A_0 \mathbf{1} = \mathbf{v}^T A_2 \mathbf{1}$, where $\mathbf{1} = [1 \;\, 1 \;\, \cdots \;\, 1]^T$. Note that the matrices A_0, A_1 and A_2 are affine transformations of λ (for fixed values of μ and k). Using the values of A_0, A_1, A_2 in the QBD representation of MDS-Reservation(1), we can show that $\lambda^*_{\text{Resv(1)}} \geq (1 - O(n^{-2}))\frac{n}{k}\mu$. For t$\geq$ 2, each of the MDS-Reservation(t) queues upper bound MDS-Reservation(1), and are themselves upper bounded by the MDS queue. It follows that $\frac{n}{k}\mu \geq \lambda^*_{\text{Resv(t)}} \geq (1 - O(n^{-2}))\frac{n}{k}\mu$ for t$\geq$ 1.

The value of $\lambda^*_{\text{Resv(t)}}$ can be explicitly computed for any value of n, k and t via the method described above. We perform this computation for $k = 2$ and $k = 3$ when t$=1$ to obtain the result mentioned in the statement of the theorem. We show the computation for $k = 2$ here.

When $k = 2$ and t$= 1$, the j^{th} level of the QBD process consists of states $\{0,1,2\} \times \{n-1+2j, n+2j\}$ for $j \geq 1$. However, as seen in Fig. 6, several of these states never occur. In particular, in level j, only the states $(1, n-1+2j)$, $(1, n+2j)$ and $(2, n+2j)$ may be visited. Thus, to simplify notation, in the following discussion we consider the QBD process assuming the existence of only these three states (in that order) in every level. Under this representation, we have

$$A_0 = \begin{bmatrix} 0 & \mu & (n-1)\mu \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad A_1 = \begin{bmatrix} -n\mu - \lambda & 0 & 0 \\ (n-1)\mu & -(n-1)\mu - \lambda & 0 \\ n\mu & 0 & -n\mu - \lambda \end{bmatrix}, \quad A_2 = \begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{bmatrix}.$$

$$\Rightarrow A_0 + A_1 + A_2 = \begin{bmatrix} -n & 1 & n-1 \\ n-1 & -(n-1) & 0 \\ n & 0 & -n \end{bmatrix} \mu \tag{8}$$

One can verify that the vector

$$\mathbf{v} = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix}^T = \begin{bmatrix} n-1 & 1 & \frac{(n-1)^2}{n} \end{bmatrix}^T$$

satisfies

$$\mathbf{v}^T (A_0 + A_1 + A_2) = 0. \tag{9}$$

Thus,

$$\mathbf{v}^T A_0 \mathbf{1} = n(n-1)\mu, \tag{10}$$

and

$$\mathbf{v}^T A_2 \mathbf{1} = \lambda \left(n + \frac{(n-1)^2}{n} \right) . \tag{11}$$

According to the properties of QBD processes, the value of $\lambda = \lambda^*_{\text{Resv(1)}}$ must satisfy $\mathbf{v}^T A_0 \mathbf{1} = \mathbf{v}^T A_2 \mathbf{1}$. Thus,

$$\lambda^*_{\text{Resv(1)}} = \frac{n^2(n-1)}{2n^2 - 2n + 1} = \left(1 - \frac{1}{2n^2 - 2n + 1} \right) \frac{n}{2} \mu. \tag{12}$$

■