Managing Response Time Tails by Sharding

P. G. HARRISON, Imperial College London
N. M. PATEL, NetApp Inc, Sunnyvale, California
J. F. PÉREZ, Universidad del Rosario, Colombia
Z. QIU, Imperial College London

Matrix analytic methods are developed to compute the probability distribution of response times (i.e. data access times) in distributed storage systems protected by erasure coding, which is implemented by sharding a data object into N fragments, only $K < N$ of which are required to reconstruct the object. This leads to a partial-fork-join model with a choice of canceling policies for the redundant $N - K$ tasks. The accuracy of the analytical model is supported by tests against simulation in a broad range of setups. At increasing workload intensities, numerical results show the extent to which increasing the redundancy level reduces the mean response time of storage reads and significantly flattens the tail of their distribution; this is demonstrated at medium-high quantiles, up to the 99^{th}. The quantitative reduction in response time achieved by two policies for canceling redundant tasks is also shown: for cancel-at-finish and cancel-at-start, which limits the additional load introduced whilst losing the benefit of selectivity amongst fragment service times.

ACM Reference format:
P. G. Harrison, N. M. Patel, J. F. Pérez, and Z. Qiu. 2018. Managing Response Time Tails by Sharding. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 1, 1, Article 1 (December 2018), 33 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

With the amount of new data created doubling every two to three years, enterprise storage costs are increasingly taking a higher proportion of corporate IT budgets. This compounding effect is being addressed by simplifying and consolidating operations whilst maintaining service level objectives (SLOs) for performance, data management and data protection. Scale-out systems that use commodity servers as nodes have become popular for both on-prem and cloud storage but, as with NAND-like storage media, response time variability remains a problem, especially in the tail of its probability distribution – commonly called tail-latency[1]. Networked storage stacks have many sources of tail latency as requests and responses flow from compute servers, across networks, to storage controllers and physical I/O devices. Fortunately, new interface protocols such as NVMe-oF (Nonvolatile Memory express over Fabrics) and RDMA-based networking (such as 100 GbE and beyond) can streamline the path considerably for short-distance data flows. Similarly, back-end storage interface protocols can also use NVMe-oF solutions to minimize tail latencies in comparison to Serial-Attached SCSI (SAS) networks. Clearly, all sources of tail latency along the path need to be addressed for good

[1]This is something of a misnoma since latency usually refers to mean response time

end-to-end response time, and NAND-based I/O devices become a dominant source because of the queuing of requests behind long-running erase operations.

Groups of persistent storage devices, such as Solid State Drives (SSDs) or rotating Hard Disk Drives (HDDs), often use some form of *erasure encoding* to provide protection from one or more node failures. An object is stored by first creating K fragments and then storing $N \geq K$ encoded fragments on different devices so that the original object can be retrieved from any K of the N fragments, typically using a maximum-distance-separable (MDS) code, such as Reed-Solomon [23]. Thus $N-K$ failures can be tolerated. This physical fragmentation of data is often termed *sharding*. The implementation of MDS encoding is one major application, but there are a number of others, for example sharded databases. Besides the failure-protection, many architectures also harvest response time (i.e. data access time) benefits on read operations, for example by issuing N tasks (corresponding to fragments) and constructing the required object after the first K responses have been received. These can be described as (N, K) *partial-join queues* where the special case with $K=1$ is simple replication [4, 8, 9, 31] and $N = K + 1, N = K + 2$ or $N = K + 3$ can represent single, dual or triple parity scenarios.

Partial-join queues can be implemented with various policies, characterized largely by when redundant tasks are deleted. For example: (a) *canceling-at-finish*, which we often abbreviate to just "canceling", where all associated tasks terminate their service immediately after the first K complete[2]; and (b) *canceling-at-start*, abbreviated to "early cancelling", where $N-K$ associated tasks leave the system when any K of the tasks have *started* service, as in [10, 11] and many implementations of MDS codes, for example [20, 21, 36][3]. In practice, the overhead of terminating a task will be higher when the task is already in service and needs to be pre-empted. By design, canceling-at-start ensures that tasks will not require pre-emption, thus incurring a lower overhead than canceling-at-finish. However, there is a trade-off in that canceling-at-start gains significantly less in response time since its choice of servers is restricted to the first k that become available; there can be no "server-race". To a good approximation, we can represent the overhead, in either case, as a high priority background workload that essentially elongates service times; it is present at all times since cancelations occur on every read-access.

The present work obtains stochastic models for response times in sharded storage systems operating the aforementioned policies. Such systems deal with relatively small numbers N of parallel data-accesses – from around 9 up to a maximum of under 30 devices – so that detailed models may be entertained. This is in contrast to distributed databases or computation in the cloud, where the number of nodes involved may be selected from thousands [6]. Here, however, the rather different technique of *replication* is what predominates, where not all nodes – probably only a few – need to be accessed for any composite operation and only one needs to complete successfully. As a running example throughout the paper, we consider a scale-out storage system with K=6 and N between 6 and 9 servers, each with its own set of SSDs (i.e., up to 50% additional servers for handling server failures and improving response time). An object write requires all N fragments to be stored, but this is typically not response time sensitive because writes are acknowledged earlier in the response-path and processed in the background. On the other hand, reads are highly response time sensitive and can take advantage of the (N, K)-sharding.

Replication and sharding may have a further impact on performance through *correlation* between the task-service times associated with the same job. The *demand* on each node may even be identical for every such task, especially in

[2]Or, rather, killing signals are sent to them as soon as possible.

[3]There are several names for these policies and we choose "canceling-at-start" and "canceling-at-finish" to give more precision. In the context of MDS, they are the policies of "Send-to-K" and "Send-to-N" in [20, 30]. "Early canceling" is an alternative to "canceling-at-start" – called "fork-early-cancel" by Joshi in [11].

replication. However, there are also node-dependent components in the task-service times arising from, for instance, the head position on a disk drive, the number of retries required in accessing a triple-level flash cell (TLC)[4], or the current overhead on a core performing a compute-task. NAND Flash based I/O devices have internal housekeeping activities such as cleaning large erase blocks that can create long tail latencies for some incoming requests regardless of size. These I/O devices typically implement complex Flash Translation Layers (FTL) that tend to randomize the timing of the cell-to-cell interference and so make the assumption of independent service times across devices more acceptable. Moreover, drive-specific garbage collection by the drive firmware can create further delays for both reads and writes of all sizes. In fact, in networked storage systems, transfer sizes are often limited to about 64K, the maximum transmission size unit (MTU), and the actual transfer time is estimated to be less than around 20% of the total access time [1, 14]. These characteristics suggest that correlation amongst the service times of tasks from the same job should be negligible, even though this is often not the case with task replication strategies, cf. [6, 26].

The key contributions of this paper are as follows:

- Response time distribution functions, rather than just latencies or mean response times[5], cf. [4, 8, 9, 11, 22]. Distributions have been considered for MDS queues in a purely Markovian context [20], but as far as we know, there are no such results for canceling-at-finish.
- The use of matrix-analytic methods to investigate partial-join systems with MAP arrivals (Markov Arrival Process, defined in Section 2.1). Response time is a K^{th} out of N order-statistic, as opposed to the simpler minimum and maximum extreme order-statistics in the cases of replication and fork-join, respectively.
- Quantitative recommendations as to redundancy policies for capacity planning in distributed storage systems, choice of redundancy level and fine-tuning of system parameters, such as the granularity of fragments.

The main part of the paper investigates partial-joins using matrix-analytic (MA) methods to provide quantiles of job response times. The partial-join operation cannot be modelled as a regular queue since a simple order statistic could not account for the presence of tasks from other jobs in the queues, which break the synchronicity of split-merge. A precise representation of partial-joins requires keeping track of the states of all the queues as well as the progress of the particular tasks of some marked job, the response time of which is sought. This results in a huge raw state space, and the novelty of the present work is judicious state-space reduction that is achieved by restructuring the state and observing certain invariances. Together with a certain truncation of the state, which allows arbitrarily greater accuracy at increasing computational cost, this renders the MA approach tractable.

2 Canceling-at-finish Model

With canceling-at-finish, upon completion of the first K tasks of a job, the remaining $N-K$ are canceled "immediately"[6]. To illustrate the operation of this policy, Figure 1 depicts an example with N=3 servers, where a job is done when any K=2 out of its 3 tasks are finished. The intermediate configurations identified determine the state transitions in the underlying Markov chain constructed below.

[4]TLC Flash needs to distinguish among 8 charge levels to re-create 3 storage bits, possibly after retries as the drive ages. QLC (quad-level cells) flash will need to distinguish among 16 charge levels to re-create 4 storage bits and will wear-out much faster than TLC as well as being more prone to long service time delays because of retries and internal garbage collection.
[5]We use the term "latency" as synonomous with mean response time. Some authors use it as the response time random variable, whereupon it has a distribution and, indeed, a tail.
[6]In practice the cancelations are initiated immediately and the time penalty of cancelation may be included in the model as an overhead, as discussed in the Introduction.

Table 1. Table of terminology and definitions.

Term	Description
Cancel-at-finish	On completion of K out of N tasks of a job, its remaining $N-K$ tasks are canceled. Abbreviated to "canceling" and equivalent to fork-partial-join.
Cancel-at-start	As soon as K out of N tasks of a job have entered service, its remaining $N-K$ tasks are canceled. Abbreviated to "early canceling" and equivalent to fork-early-partial-join.
MAP	Markovian Arrival Process.
PH	Phase-type probability distribution.
(N, K) sharding	Partial-join model with N servers in which a job is complete as soon as K of its tasks are served.
Performance metrics.	
Throughput	Average number of jobs or tasks processed in unit time.
Latency	Average (job- or task-) response time.
ATP(x)	Accelerated Throughput, the ratio of the throughput x and ORT(x).
ORT(x)	Overall Response Time, the average value of the latency over the throughput range $[0, x]$.
Knee	Top of the effective operating range of workload, the point x^* at which ATP(x) is maximum.

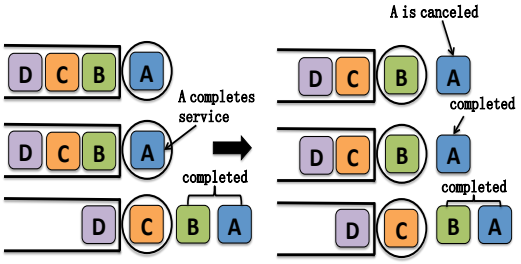


Fig. 1. Canceling-at-finish example. In the initial configuration, one task from job A has already completed service at the bottom server. Next the task from job B completes, also at the bottom server. When the task from job A at the middle server competes, the task in-service at the top server is deleted, leaving the configuration shown, with two B-tasks and one C-task in service. More progress is made than with canceling-at-start (Figure 2) because of the ability to choose the fastest 2 tasks from job A.

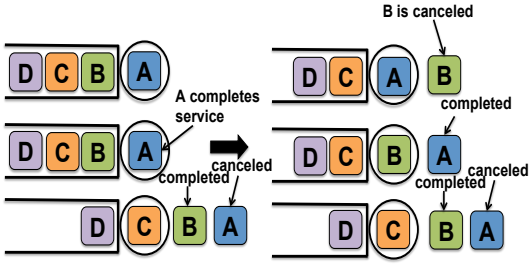


Fig. 2. Canceling-at-start example. In the same initial configuration, the task from job B again completes first at the bottom server, leaving the same configuration as with canceling-at-finish. When the task from job A at the middle server competes, the queuing B-task enters service there. The queuing B-task at the top server is then deleted because two B-tasks have already started at the other servers. Thus, the C-task queuing in the top queue enters service, leaving one A-task, one B-task and one C-task in service.

The main modeling issues arise from the fact that the system is a set of parallel queues, the evolution of which is tightly coordinated as every arriving job sends a task to each queue simultaneously. Furthermore, canceling can occur when redundant tasks are either waiting in-queue or in service, which requires precise tracking of the queues' states. In the general case with $1 \leq K \leq N$, we show that both the job queuing-time and job service-time have PH distributions, given that task-service times are negative exponential, and that these can be combined to yield the required job response-time distribution by means of [28, Theorem 1].

2.1 Arrival processes and service times

We use MAPs to model the arrival of jobs to the storage system because this class of arrival process is much more general than the Poisson process and has the ability to model variable rate and correlated arrivals. The continuous-time MAP [18] is a marked Markov chain (MC) with generator matrix $D=D_0+D_1$. The transition rates *not* associated with arrivals are held in D_0, while those that trigger new arrivals are in D_1. The diagonal entries of D_0 hold the negative

total exit rate from each state, so that $(D_0+D_1)\mathbf{1}=\mathbf{0}$, where $\mathbf{1}$ is a column vector of ones. We denote this process by MAP(m_a, D_0, D_1), where m_a is the number of states, or arrival phases, in the MC. The mean arrival rate is $\lambda=\boldsymbol{d}D_1\mathbf{1}$, where $\boldsymbol{d}$ is the invariant probability distribution of the underlying Markov chain, i.e., $\boldsymbol{d}D=0$ and $\boldsymbol{d}\mathbf{1}=1$.

The task processing-time distribution is taken to be of phase-type (PH), to model the different levels of processing time variability observed in storage systems. A PH random variable X represents the absorption time in a MC with $n+1$ states, where the states $\{1,\ldots,n\}$ are transient and state 0 is absorbing [18]. A phase-type distribution is usually denoted PH$(\boldsymbol{\tau}, S)$, where the vector $\boldsymbol{\tau}$ is the $1\times n$ vector of the MC's initial probability distribution for the transient states, and matrix S is the $n\times n$ sub-generator matrix holding the transition rates amongst the transient states. The vector $\boldsymbol{S}^*=-S\mathbf{1}$ holds the absorption rates from the transient states. The negative exponential random variable is a special case of the PH distribution (comprising just one phase) and is commonly employed in models of coding systems to good effect [4, 8, 9, 31]. In this section, we too assume that individual task-processing times are negative exponential random variables. This also leads to PH job-service-time and job-queueing time distributions but the memoryless property facilitates greater state-reduction, as discussed in the next section.

2.2 The queuing-time distribution

To determine the queuing-time distribution, we perform an age-based analysis [2], observing the N queues only during the *all-busy* periods, i.e., when the N servers are all busy, and defining a bivariate Markov process $\{\mathcal{X}(t), \mathcal{J}(t)|t \geq 0\}$. The *age* $\mathcal{X}(t)$ keeps track of the total time-in-system of the current youngest job in service, thus taking values in $[0,\infty)$, increasing linearly with rate 1 if no *job* service completions occur[7]. The *phase* $\mathcal{J}(t)=(\mathcal{A}(t), \mathcal{S}(t))$ holds the joint state of the MAP arrival process $\mathcal{A}(t)$ and the service process $\mathcal{S}(t)$. The service process keeps track of the N queue lengths, where the length of a queue includes both the tasks in service and waiting in the queue. Before fully defining $\mathcal{S}(t)$ we note the following property.

Proposition 1. *In the system with canceling-at-finish, the longest $N-K+1$ queues always have the same length.*

Proof. The oldest job in service cannot have any older tasks in front of any of its tasks in any queue. In other words, in every queue the oldest job in service must either be in-service or else have already completed service. Moreover, every queue at which this job has a task in-service must have the same length, since each of them has exactly one task waiting to commence service emanating from each of the jobs that arrived after the oldest job in service. Clearly, these are the longest queues. To complete the proof, we note that the maximum number of queues from which a task of the oldest job has departed is $K-1$, for otherwise the job would have finished, with all its remaining tasks canceled. Hence the minimum number in service is $N-K+1$. □

This effect is illustrated in Figure 1 where, with $N=3$, $K=2$, the 2 longest queues decrease their length at the same time when either task of job A completes.

Due to this proposition, instead of keeping track of the states of all queues, we focus on the shortest K queues only. Furthermore, instead of considering the K queue-lengths themselves, we order them by length and obtain the queue-length difference between any queue and the shortest one. Since all the queues are homogeneous, it suffices to

[7] So when the first task of a job starts service, $\mathcal{X}(t)$ resets to 0.

count the number of queues with the same difference, whereupon the service process only needs to keep track of the number of queues with i more tasks than the shortest queue, for $i \geq 0$.

Since these queue-length differences are not bounded, to make the analysis tractable, we follow [28] and truncate the state space artificially by restricting the queue-length difference to be at most some positive integer $C < \infty$. The service process then has state space given by the following.

Definition 2.1. For canceling-at-finish, the service process $\mathcal{S}(t) = (n_0(t), n_1(t), \ldots, n_C(t))$, which takes values in the state space

$$\left\{(n_0, \ldots, n_C) \mid \sum_{i=0}^{C} n_i = K, n_0 \geq 1, n_i \geq 0 \ (0 < i \leq C)\right\},$$

where n_0 is the number of shortest queues.

For example, assume $C=2$ and consider the scenario on the left in Figure 1 with $N=3$ and $K=2$. There, the service process $\mathcal{S}(t)$ is in state $(n_0(t), n_1(t), n_2(t)) = (1, 0, 1)$, as there is one shortest queue, and one queue representing the longest 2 queues, both of which have two more tasks than the shortest one.

Under this truncation, the number of states is

$$m_s = \binom{K + C - 1}{C}, \tag{1}$$

so that the phase process $\mathcal{J}(t)$ takes values in a set of size $m = m_a m_s$, where m_a is the number of arrival phases (see Appendix 2.1).

To determine the PH representation of the queuing-time distribution, the next step is to compute the stationary distribution $\boldsymbol{\pi}(x)$ of the process $(\mathcal{X}(t), \mathcal{J}(t))$, which has a matrix-exponential representation [29]

$$\boldsymbol{\pi}(x) = \boldsymbol{\pi}(0) \exp(Tx), \text{ for age } x > 0,$$

where $\boldsymbol{\pi}(0)$ is the steady-state distribution of the phase at the beginning of an all-busy period. Finding $\boldsymbol{\pi}(x)$ then boils down to finding the matrix T and the vector $\boldsymbol{\pi}(0)$.

Proposition 2. *The $m \times m$ matrix T satisfies the equation*

$$T = S^{MAP} + \int_0^\infty \exp(Tu) A^{MAP}(u) \mathrm{du}, \tag{2}$$

where:

- $S^{MAP} = S \otimes I_{m_a}$, $A^{MAP}(u) = A^{(jump)} \otimes \exp(D_0 u) D_1$, I_n *is the identity matrix of dimension n, and* $\otimes$ *denotes the Kronecker product operation;*
- *The matrices S and $A^{(jump)}$ are $m_s \times m_s$ matrices that hold the transition rates of the service process associated with transitions that, respectively, do not and do cause a new job to start service;*
- *The matrices D_0, D_1 characterize the MAP.*

Proof. The result is an instantiation of the matrices S and $A^{(\mathrm{jump})}$ in equation (7.1) of [2] for the MAP/PH/c queue, which itself is derived from equation (3) of [29] for bivariate Markov processes, with a continuous, increasing "age" component and a discrete component: $T = D + \int_0^\infty \exp(Tu) \mathrm{dA(u)}$. □

The matrices S and $A^{(\mathrm{jump})}$ are given by Table 2 for a model with truncation constant C. It can be seen that $A^{(\mathrm{jump})}$ holds the rates of service completions at any of the *shortest* queues, as a new job starts service if and only if there is a

Table 2. Transition rates for matrices S and $A^{(\text{jump})}$

From	To	Rate	Matrix
$(n_0, n_1, \ldots, n_{C-1}, n_C)$	$(1, n_0{-}1, n_1, \ldots, n_{C-2}, n_{C-1}{+}n_C)$	$n_0\mu$	$A^{(\text{jump})}$
$(n_0, \ldots, n_{\text{longest}}, 0, \ldots, 0)$	$(n_0, \ldots, n_{\text{longest}-1}{+}1, n_{\text{longest}}{-}1, 0, \ldots, 0)$	$(n_{\text{longest}}{+}N{-}K)\mu$	S
$(n_0, \ldots, n_{i-1}, n_i, \ldots, n_C)$	$(n_0, \ldots, n_{i-1}{+}1, n_i{-}1, \ldots, n_C)$	$n_i\mu$, $1{\le}i{<}$longest	

task service completion in a shortest queue. Figure 1 illustrates the situation where a hypothetical job D, arriving after job C, would start service if the task of job C in the bottom server (shortest queue) completed service. Alternately, other task service completions would allow a task of job B to start service, but job B must have started service previously as one of its tasks had already left the bottom server. The first row of Table 2 shows that a service completion in any of the shortest n_0 queues, results in a single shortest queue and $n_0{-}1$ queues with one more task than the shortest queue.

Since we observe only the shortest K queues, assuming n_{longest} queues of these are the longest, there must be $n_{\text{longest}}{+}N{-}K$ queues among all the N queues with this length, as the longest $N{-}K{+}1$ queues always have the same length. This is captured in matrix S, as shown in the second row of Table 2, where a task completes service in one of these longest queues with transition rate $(n_{\text{longest}}{+}N{-}K)\mu$. The third row in this table considers service completions in queues other than the shortest and longest ones, where a task in service in one of the queues with difference n_i completes service, reducing its difference with respect to the shortest queue by one, for $i{\ge}1$. Notice that as the queue-length differences are bounded by C, transitions leading these differences to exceed the limit C are re-assigned to n_C.

Thus, for the configuration illustrated in Figure 1 with $N = 3, K = 2$, the state-space is $\{(1,0,1),(1,1,0),(2,0,0)\}$ and the matrices $A^{(\text{jump})}$ and S are

$$A^{(\text{jump})} = \begin{bmatrix} \mu & 0 & 0 \\ \mu & 0 & 0 \\ 0 & 2\mu & 0 \end{bmatrix} \qquad S = \begin{bmatrix} 0 & \mu & 0 \\ 0 & 0 & \mu \\ 0 & 0 & 0 \end{bmatrix},$$

which can be obtained directly from Table 2.

To find the steady-state distribution $\boldsymbol{\pi}(0)$ of the phase at the beginning of an all-busy period, we keep track of the system evolution during the not-all-busy period and the connecting transitions between the not-all-busy and all-busy periods. During a not-all-busy period, the shortest queues must be empty and the system will remain in the not-all-busy period until a new arrival occurs, sending tasks to all queues and initiating an all-busy period. Thus, the system's evolution is governed by service completions and it suffices to keep track of the phase process $\mathcal{J}(t)$, which is defined as for the all-busy period. To hold the transition rates across service phases, between arrivals, during a not-all-busy period, we introduce the $m_s \times m_s$ matrix $S_{\text{not-all-busy}}$, whose non-negative transition rates are actually the same as in matrix S since this matrix also contains service completion rates between arrivals. Since the shortest queues are empty, they have no service completions. Finally, we set the diagonal entries of matrix $S_{\text{not-all-busy}}$ to make it a Markov process generator. Then the vector $\boldsymbol{\pi}(0)$ is given by the following.

Proposition 3. *When equilibrium exists, $\boldsymbol{\pi}(0)$ is the normalized solution of the equation:*

$$\boldsymbol{\pi}(0) = \boldsymbol{\pi}(0) \int_0^\infty \exp(Tu)(A^{(jump)} \otimes \exp(D_0 u))du(S_{not\text{-}all\text{-}busy} \oplus D_0)^{-1}(I_{m_s} \otimes D_1), \tag{3}$$

where $\oplus$ stands for the Kronecker sum, i.e., $A \oplus B = A \otimes I + I \otimes B$.

Proof. See [2, 29]. □

Table 3. Transition rates for $S_{\text{(full)-(not-full)}}$ and $S_{\text{not-full}}$ (Canceling-at-finish)

From	To	Rate	Matrix
$N, (n_0, \ldots, n_i, \ldots, n_C)$	$N{-}1, (1, n_0{-}1, \ldots, n_{C-2}, n_{C-1}{+}n_C)$	$n_0\mu$	$S_{\text{(full)-(not-full)}}$
$r, (n_0, \ldots, n_{\text{longest}}, 0, \ldots, 0)$	$r, (n_0, \ldots, n_{\text{longest}-1}{+}1, n_{\text{longest}}{-}1, 0, \ldots, 0)$	$(n_{\text{longest}}{+}N{-}K)\mu$	$S_{\text{not-full}}$
$r, (n_0, \ldots, n_{i-1}, n_i, \ldots, n_C)$	$r, (n_0, \ldots, n_{i-1}{+}1, n_i{-}1, \ldots, n_C)$	$n_i\mu$, $1{\le}i{<}\text{longest}$	
$r, (n_0, n_1, \ldots, n_C)$	$r{-}1, (n_0{+}1, n_1{-}1, \ldots, n_C)$	$n_1\mu$, $r{>}N{-}K{+}1$	

Notice that the sub-generator $S_{\text{not-all-busy}} \oplus D_0$ captures the phase process evolution during the not-all-busy period until an arrival triggers the process into a new all-busy period. After $\boldsymbol{\pi}(0)$ has been obtained, we follow the steps in [2] to find the PH representation of the queuing-time distribution.

2.3 The service-time distribution

The job service time under the canceling-at-finish scheme is the time elapsed between the start of service of its first task and the time when its K^{th} task completes. We thus follow a tagged job and let $\mathcal{Y}(t){=}(\mathcal{R}(t), \mathcal{S}(t))$ be the service state of this job at time t. Here $\mathcal{R}(t)$ records the number of tasks in the tagged job that *have not finished service* at time t (recall that $\mathcal{S}(t)$ is the state-vector of the queues). Since the tagged job completes service when any K of its N tasks complete service, we are interested in service phases where at least $N{-}K{+}1$ tagged tasks are still in service or queuing, i.e., when $\mathcal{R}(t){\ge}N{-}K{+}1$. Although similar to the service phase defined for the queuing-time distribution, in this case the variable $\mathcal{S}(t)$ focuses on the queue-lengths *in front of the tagged job only*. Thus, if $\mathcal{R}(t){=}N$, n_0 holds the number of queues with a tagged task in service, and $\sum_{i=0}^{C} n_i(t){=}\mathcal{R}(t)$. Instead, if $\mathcal{R}(t){<}N$, n_0 holds the number of servers that have already finished serving tagged tasks, and $\sum_{i=1}^{C} n_i(t){=}\mathcal{R}(t)$. Note that when $\mathcal{R}(t){<}N$, the number of tagged tasks currently in service is given by n_1. Assuming a bound on the queue-length difference $C{=}2$ and considering the example on the left in Figure 1, if job C in the figure is the tagged job, then $n_0{=}1$ and $\mathcal{S}(t){=}(1, 0, 1)$, as there is one queue among the shortest $K{=}2$ queues that has two more tasks than the shortest one. If instead we focus on job B in the figure, then $\mathcal{S}(t){=}(1, 1, 0)$.

Next, to obtain the service-time PH representation, we split the service states into two sets: the *full* phases, where none of the tagged tasks have completed service, so that $\mathcal{R}(t){=}N$; and the *not-full* phases, where at least one tagged task has already finished service, so that $N{-}K{+}1{\le}\mathcal{R}(t){<}N$. We then put together the transition rates among service states in a sub-generator S_{ser}, which is partitioned according to the above two subsets as

$$S_{\text{ser}} = \begin{bmatrix} S_{\text{full}} & S_{\text{(full)-(not-full)}} \\ 0 & S_{\text{not-full}} \end{bmatrix} \otimes I_{m_a}. \tag{4}$$

Here the matrix S_{full} holds the rates of transitions that do not involve a service completion in the shortest queues while all the servers are busy; thus it is the same as the matrix S. When the tagged job is in a full state and a service completion in the shortest queues occurs, the tagged job jumps to a not-full state, as shown in the first row of Table 3. Once the job is in a not-full state, the matrix $S_{\text{not-full}}$ considers three types of transitions, summarized in rows 2-4 of Table 3: the second row considers the case where a task completes service in one of the longest queues; the third row covers the case with service completions at any queue but the longest and shortest ones; and in the fourth row, a tagged task completes service.

To complete the PH representation of the service-time distribution, it is necessary to find the stationary distribution of the phase $\boldsymbol{\alpha}_{\text{ser}}$ in which a task starts service, which is given by [28, Proposition 1]. Having obtained the PH

Table 4. Transition rates for matrices $S_{\text{(all)-(not-all)}}$ and $S_{\text{not-all-busy}}$ (canceling-at-start)

From	To	Rate	Matrix
$(n_0, n_1, \ldots, n_{C-1}, n_C)$	$(1, n_0{-}1, n_1, \ldots, n_{C-2}, n_{C-1}{+}n_C)$	$n_0\mu$	$S_{\text{(all)-(not-all)}}$
$(n_0, \ldots, n_{\text{longest}}, 0, \ldots, 0)$	$(n_0, \ldots, n_{\text{longest}-1}{+}n_{\text{longest}}, 0, \ldots, 0)$	$n_{\text{longest}}\mu$	$S_{\text{not-all-busy}}$
$(n_0, \ldots, n_{i-1}, n_i, \ldots, n_C)$	$(n_0, \ldots, n_{i-1}{+}1, n_i{-}1, \ldots, n_C)$	$n_i\mu$, $1{\le}i{<}\text{longest}$	

representations of a job's queuing and service times, the PH representation of the response-time distribution follows from [28, Theorem 1], which we summarize in Appendix F

2.4 Utilization

The system utilization, or resource consumption, is the expected fraction of time that a server is busy. Under the canceling-at-finish scheme, assuming $E[r]$ is the expected number of servers used by a job, the utilization can be computed as $\rho = E[r]\lambda/(N\mu_{\text{job}})$, where λ^{-1} is the mean job inter-arrival time, and μ_{job}^{-1} is the mean job service time, given by $\mu_{\text{job}}^{-1}{=}\boldsymbol{\alpha}_{\text{ser}}(-S_{\text{ser}})^{-1}\mathbf{1}$. As requests may finish service using K or more servers depending on their evolution amongst service phases, thus $K \le E[r] \le N$, and $E[r]$ can be obtained as $E[r] = \langle \boldsymbol{\pi}_{\text{ser}}, \boldsymbol{n}_{\text{server}} \rangle$, where $\langle \cdot, \cdot \rangle$ is the inner-product operator. Here $\boldsymbol{\pi}_{\text{ser}}$ is the stationary distribution of the job service phase, given by $\boldsymbol{\pi}_{\text{ser}} = \boldsymbol{\alpha}_{\text{ser}}(-S_{\text{ser}}^{-1})/(\boldsymbol{\alpha}_{\text{ser}}(-S_{\text{ser}})^{-1}\mathbf{1})$. Also, the i^{th} element of the vector $\boldsymbol{n}_{\text{server}}$ holds the number of tagged tasks in service in the i^{th} service phase, which is equal to N for full states, and to n_1 for not-full states.

3 Canceling-at-start model

With canceling-at-start, as soon as the first K tasks of a job have *started* service, the remaining $N{-}K$ tasks are canceled "immediately". This policy is simpler to implement, not requiring pre-emption of tasks, and is appropriate in applications where the latency is mainly due to queuing delays, especially at high load [5]. We illustrate this policy in Figure 2, again for the case with $N{=}3$ and $K{=}2$ and with the same initial configuration as in Figure 1. The changed intermediate configurations distinguish the underlying Markov chain from that defined for canceling-at-finish.

When $K{=}1$ and service times are exponential, for any arrival process, the model simplifies to a $G/M/N$ queue, as only one task of each job starts service, canceling all its siblings, as long as there is a server available. In contrast, in the case with canceling-at-finish the model reduces to a $G/M/1$ queue with N times faster service rate, since job-service time is then the minimum of N exponential random variables.

To analyze the general case with $1 \le K \le N$, we take a similar approach to that of the previous section, determining first the queuing-time distribution and then the service-time distribution. However, the structure of the state-space and the state-transitions differ significantly. We note the following property.

Proposition 4. *In the system with canceling-at-start, the longest $N{-}K{+}1$ queues always have the same length.*

Proof. The proof is inductive and in four parts.

(1) In the state where all the queues are empty and the servers idle, the property holds vacuously, and we assume inductively that it holds at all times.

(2) In any state of the queues, a new arrival has up to K of its tasks start service (by arriving at an idle server), at least $N{-}K{+}1$ tasks join the longest queues, or no tasks join a queue if K tasks started service and $N{-}K$ were canceled. In either case, the longest $N{-}K{+}1$ queues remain at the same length.

(3) After a service completion, if the K^{th} task of a job enters service, the queue length at that server decreases by one and the task's siblings are canceled. Hence the $N-K+1$ longest queues all lose one task and remain at the same length.

(4) If after a service completion a task of a job, that is not its K^{th} task, enters service, it could not have come from one of the $N-K+1$ longest queues. Its queue length therefore reduces by one and the $N-K+1$ longest queues remain the longest and unchanged in length.

As a result, the $N-K+1$ longest queues remain at the same length at all times. □

This result is illustrated in Figure 2 where, with N=3, K=2, the two longest queues decrease their length at the same time when the second task of the youngest job in service (B) starts to be served. Due to this proposition, we again need not keep track of the states of all queues, but instead focus on the shortest K queues only – just as with the canceling-at-finish policy.

3.1 The queuing-time distribution

Based on the observation above we focus on the K shortest queues and proceed with an age-based analysis, as in the previous section, defining the age $\mathcal{X}(t)$ and the phase $\mathcal{J}(t)$, the latter evolving on the same set as in the previous section. Again, we define the matrices S and $A^{(\text{jump})}$.

Proposition 5. *The matrix T given by Eq. (2) is the same for both the canceling-at-start and canceling-at-finish policies.*

Proof. The matrix T is completely determined by $A^{(\text{jump})}, S, D_0$ and D_1, where D_0 and D_1 are the same for both policies by definition. Under both policies, during the all-busy period, a service completion at the shortest queue triggers a new job (if present) to start service, so it is clear that the matrix $A^{(\text{jump})}$ is the same, as defined in Table 2. Further, as highlighted in Proposition 4, the $N-K+1$ longest queues are formed where the youngest job in service still has tasks waiting. Therefore, if a task in any of these queues completes service, it will trigger all the longest queues to decrease at the same time, as shown in the second row of Table 2. Finally, service completions in other queues simply allow a new task to start service, modifying the queue-length differences as in the third row of Table 2. Thus the matrix S is also the same for both canceling policies and the proof is complete. □

Different from canceling-at-finish, in the case of canceling-at-start, more than one arrival may be necessary to initiate an all-busy period. For instance, consider an empty system with N=4 queues and K=3, so that an arrival leads to 3 out of the 4 servers becoming busy, while one server remains empty due to the early canceling, leaving the system in the not-all-busy state. This contrasts with the canceling-at-finish case, where any arrival during a not-all-busy period must initiate an all-busy period. Whilst during all-busy periods we keep track of the states of the shortest K queues only, during a not-all-busy period we must also consider the number of idle servers and the state of all the non-empty queues. We therefore define:

Definition 3.1. For canceling-at-start, the service process $\mathcal{S}^e(t)=\{n_0(t),\ldots,n_C(t)\}$, where $n_0(t)$ is the number of idle servers at time t, which takes values on the state space

$$\left\{(n_0,\ldots,n_C) \mid \sum_{i=0}^{C} n_i=N, n_i\geq 0, n_0\geq 1\right\},$$

which has cardinality $m_{\text{not-all}}$.

Table 5. Transition probabilities for $R_{\text{all-busy}}$ and $R_{\text{not-all}}$ (canceling-at-start)

From	To	Probability	Matrix
$(n_0, n_1, 0, \ldots, 0)$	$(n_0{-}K, n_1{+}K, 0, \ldots, 0)$	$1,\ n_0{>}K$	$R_{\text{not-all}}$
$(n_0, \ldots, n_i, \ldots, n_C)$	$(n_0, \ldots, n_i, \ldots, n_C)$	$1,\ n_0{\leq}K$	$R_{\text{all-busy}}$

Table 6. Transition rates for $S_{\text{(full)-(not-full)}}$ (canceling-at-start)

From	To	Rate
$(n_0, \ldots, n_i, \ldots, n_C)$	$K{-}1, (1, n_0{-}1, \ldots, n_{C-2}, n_{C-1}{+}n_C)$	$n_0\mu$

We now define the $m_s{\times}m_{\text{not-all}}$ matrix $S_{\text{(all)-(not-all)}}$ of service transition rates that trigger the start of a not-all-busy period from an all-busy-period, connecting the process $\mathcal{S}(t)$ to the process $\mathcal{S}^e(t)$. Table 4 shows how the non-zero entries of $S_{\text{(all)-(not-all)}}$ hold the service completion rates at the shortest queues, as the all-busy period terminates if any of these queues completes service before the next arrival.

Next, we focus on the not-all-busy period and introduce an $m_{\text{not-all}}{\times}m_{\text{not-all}}$ matrix $S_{\text{not-all-busy}}$, which holds the service transition rates between arrivals during the not-all-busy period, as summarized in Table 4. The second and third rows of the table consider a service completion in any of the longest queues, and in any of the other queues, respectively.

Since, during the not-all-busy period, several arrivals could occur, we track service phase changes due to arrivals by introducing an $m_{\text{not-all}}{\times}m_{\text{not-all}}$ matrix $R_{\text{not-all}}$, the $(i,j)^{\text{th}}$ entry of which is the probability that the service phase just after an arrival is j, given that it was i just before. As shown in Table 5, these transitions are restricted to cases where a job finds more than K idle servers upon arrival, thus starting service with all its K tasks, so that the system stays in the not-all-busy period.

In the complementary case where an arrival sees at most K idle servers and initiates an all-busy period, we introduce an $m_{\text{not-all}}{\times}m_{\text{all-busy}}$ matrix $R_{\text{all-busy}}$, whose non-zero entries are summarized in Table 5. The only difference is that the phase after the arrival relates to the process $\mathcal{S}(t)$, since the all-busy period has started.

With the definitions above, we can find $\boldsymbol{\pi}(0)$, which is the steady state distribution (when this exists) of the phase at the beginning of an all-busy period.

Proposition 6. *$\boldsymbol{\pi}(0)$ is the normalized solution of the equation:*

$$\boldsymbol{\pi}(0) = \boldsymbol{\pi}(0) \int_0^\infty \exp(Tu) S^*(u) Q_{not\text{-}all}^{-1} (R_{all\text{-}busy} \otimes D_1) du. \tag{5}$$

Proof. In this equation, a transition that triggers the start of a not-all busy period occurs according to $S^*(u) = S_{\text{(all)-(not-all)}} \otimes \exp(D_0 u)$. Once the system is in the not-all-busy period, it evolves according to the sub-generator

$$Q_{\text{not-all}} = I_{m_{\text{not-all}}} \otimes D_0 + S_{\text{not-all-busy}} \otimes I_{m_a} + R_{\text{not-all}} \otimes D_1,$$

where we consider all transitions that do not trigger the start of an all-busy period. Eventually, an arrival starts an all-busy period with rates as in $R_{\text{all-busy}} \otimes D_1$. The rest of the proof is now by [2, 29]. □

Having found T and $\boldsymbol{\pi}(0)$, we utilize [2] to find the PH representation of the queuing-time distribution.

3.2 The service-time distribution

Finally, we seek a PH representation ($\boldsymbol{\alpha}_{\text{ser}}$,$S_{\text{ser}}$) for the job service-time distribution. With canceling-at-start, a job can have at most K tasks in service, as the remaining $N-K$ are canceled when the K^{th} task starts service. Hence the state of a tagged job in service at time t is defined as $\mathcal{Y}(t)=(\mathcal{R}(t),\mathcal{S}(t))$, where $\mathcal{R}(t)\in\{K,K-1,\ldots,1\}$ is now the number of tasks that *have not completed service* and $\mathcal{S}(t)$ keeps track of the queue-lengths in front of the tagged tasks. Similar to the canceling-at-finish case, we identify two subsets for the state space of this process, the full set and the not-full set. The difference here is that a job in service has at most K tasks, thus $\mathcal{R}(t)=K$ and $\mathcal{R}(t) < K$ for phases in the full and not-full sets, respectively. We can therefore describe the service transition rates in a sub-generator matrix S_{ser}, which we partition as in Eq. (4). As the matrix S_{full} holds the rates of service-transitions that exclude service completions of the tagged tasks, it is equal to the matrix S.

The matrix $S_{\text{not-full}}$, on the other hand, is the same as in the canceling-at-finish case, as summarized in Table 3: the service completion of one of the n_1 tagged tasks increases the number of completed tagged tasks by one, accumulating up to job service completion; and other service completions decrease the queue-length observed by the tagged tasks in these queues by one. The only minor difference arises when considering the longest queues. In the canceling-at-finish case, a task that completes service in one of these queues cancels all its siblings, while in the canceling-at-start case, a service completion triggers the task from another job to start service, and this task cancels its siblings. In either case, the result for the tagged job is a decrease by one in the lengths of all the longest queues. Finally, the transition rates in matrix $S_{\text{(full)-(not-full)}}$ are summarized in Table 6, which considers a service completion in one of the shortest queues when the tagged job is in a full state, triggering the tagged job to a not-full state.

To complete the PH representation of the service-time distribution, it remains to determine $\boldsymbol{\alpha}_{\text{ser}}$, which is the stationary distribution of the initial job service phase. Different from the canceling-at-finish case, to find this vector we consider the three different conditions under which a request starts service, thus

$$\boldsymbol{\alpha}_{\text{ser}} = [\boldsymbol{s}_{\text{busy}} + \boldsymbol{s}^{\text{all}}_{\text{not-busy}} + \boldsymbol{s}^{\text{part}}_{\text{not-busy}}, \quad \mathbf{0}]. \tag{6}$$

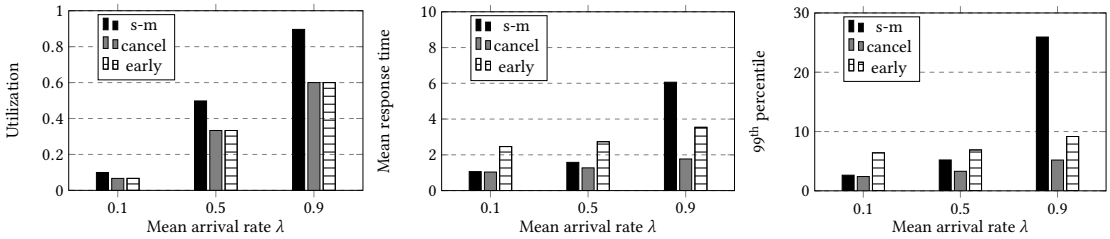
Notice first that the non-zero entries correspond to the full phases, as a job can only start service in phases where zero tasks have finished service, i.e., with $\mathcal{R}(t)=K$, and service phases are ordered such that the full phases come first. The vector $\mathbf{0}$ corresponds to the not-full phases. Next, for the full phases, we consider the following three scenarios:

(1) On arrival, a job finds all the servers busy. At equilibrium, its initial service phase then has the stationary distribution of the phase $\mathcal{J}(t)$ just after a downward jump of the age process $\mathcal{X}(t)$, since these jumps mark the start of a new job service during the all-busy period. The stationary distribution of the arrival's initial service phase is therefore [28, Proposition 1],

$$\boldsymbol{s}_{\text{busy}} = c\gamma\boldsymbol{\alpha}_{\text{busy}}(T-S^{\text{MAP}}),$$

where c is a normalizing constant that ensures this vector is stochastic, and γ is the probability that a job has to wait for service upon arrival, which can be obtained as in [2, Section 6].

(2) A job arrives during a not-all-busy period and starts service without initiating an all busy period. As already noted, more than one arrival may be necessary to initiate the all-busy period. Among these arrivals, all but one starts service with all their tasks, while the last one does initiate an all-busy period and starts service with a number of tasks equal to the number of idle servers just before it arrived. Let η_1 be the number of arrivals in a not-all-busy period, the expected value of which, $E[\eta_1]$, can be obtained from [2, Section 7.2], so that $E[\eta_1]-1$ is the expected number of jobs that arrive during the not-all-busy period and find there are more than K idle servers. The probability that a job arriving



(a) Utilization (b) Mean response time (c) 99$^{\text{th}}$ percentile response time

Fig. 3. Comparison of three policies – (N, K)=(9, 6), C=10.

during the not-all-busy period starts service with all its tasks is therefore $p_K = (E[\eta_1]-1)/E[\eta_1]$. These arrivals start service with all their tasks, and so in the service phase with n_0=K. We can write their initial service phase distribution as

$$\boldsymbol{s}^{\text{all}}_{\text{not-busy}} = (1-\gamma)p_K[\mathbf{0}, \quad 1],$$

where we assume that the full phases are ordered increasingly with respect to n_0, so that the last phase corresponds to n_0=K.

(3) A job arrives in a not-all-busy period, but finds at most K idle servers and so initiates an all-busy period, starting service with initial phase distribution

$$\boldsymbol{s}^{\text{part}}_{\text{not-busy}} = (1-\gamma)(1-p_K)\tilde{\boldsymbol{\pi}}(0),$$

where the components of the vector $\tilde{\boldsymbol{\pi}}(0)$ are the phases of $\boldsymbol{\pi}(0)$ that correspond to full phases, i.e., with $\mathcal{R}(t)$=K, as $\boldsymbol{\pi}(0)$ is precisely the steady state distribution of the phase at the beginning of an all-busy period.

We have therefore obtained the service- and queuing-time distributions, from which we can obtain the PH representation of the response-time distribution by applying [28, Theorem 1], as summarized in Appendix F. The resource consumption in this case is U=$K\lambda/(N\mu)$, where λ is the request arrival rate and μ^{-1} mean *task* service time, respectively. Unlike the canceling-at-finish case, where we had to compute the expected number of servers used by each job, under canceling-at-start, exactly K servers are used by every job .

4 Computational issues

The crucial step in obtaining the response-time distribution by the method of Section 2 is the solution of Eq. (2) to find the matrix T. Whilst efficient methods exist to solve this equation in general, here we are dealing with very large matrices as the service process keeps track of the queue-length differences amongst a set of K servers, resulting in m_s phases as given by Eq. (1). To the best of our knowledge, the most efficient method to solve Eq. (2), without considering additional information about the structure of the matrices S and $A^{(\text{jump})}$, is provided in [28] by re-writing this equation in the form

$$S^{\text{MAP}}\bar{L} + \bar{L}(I_{m_s} \otimes D_1)\bar{L} + \bar{L}(I_{m_s} \otimes D_0) = -A^{(\text{jump})} \otimes I_{m_a}, \tag{7}$$

where $\bar{L} = \int_0^\infty \exp(Tu)(A^{(\text{jump})} \otimes \exp(D_0 u))du$. Equation (7) is a non-symmetric algebraic Riccati equation (NARE) and can be solved efficiently by means of the Schur method [3], which provides good results for small values of K, but as K increases, rapidly becomes very time-consuming or even infeasible. We therefore propose an alternate approach specific to the models of this paper.

4.1 A customized solution method to find T

The key observation is that, by ordering the service phase space lexicographically, the matrix S defined in Table 2 is upper triangular. In fact, S has at most $K-1$ non-diagonal entries in each row, corresponding to service completions in the $K-1$ queues other than the shortest one. Solving a linear system with this structure requires Km_s operations, a property we take advantage of to obtain T. We also note that the matrix $A^{(\text{jump})}$ has $m'_s = \binom{K+C-2}{C-1}$ nonzero columns, as a transition tracked in $A^{(\text{jump})}$ results in a single shortest queue and the other $K-1$ queues holding between 1 and C additional tasks. We focus on the case with Poisson arrivals and notice that we can re-write Eq. (7) as

$$\lambda \bar{L}^2 + (S - \lambda I_{m_s})\bar{L} + A^{(\text{jump})} = 0, \tag{8}$$

which is a matrix-quadratic equation in $\bar{L}$ of the kind that has been well studied in the solution of quasi-birth-death (QBD) processes [18]. However, none of the methods used to solve (8) takes advantage of the particular structure of S. The methods that come closest are those in [34], but these require the matrix $A^{(\text{jump})}$ to be upper-triangular as well, a property that does not hold in our case. We therefore opt for re-writing Eq. (8) as $\bar{L} = -(S-\lambda I_{m_s})^{-1}A^{(\text{jump})} - (S-\lambda I_{m_s})^{-1}\lambda \bar{L}^2$. Note that S is a sub-generator matrix, i.e., a sub-matrix of the generator matrix of a Markov chain that corresponds to a set of transient states. Thus $\Phi = S - \lambda I_{m_s}$ is also a sub-generator, and the inverse Φ^{-1} is well defined and entry-wise positive. This immediately leads to the functional iteration

$$\bar{L}_{n+1} = \Phi^{-1}A^{(\text{jump})} + \lambda \Phi^{-1}\bar{L}_n^2, \tag{9}$$

which can be started with $\bar{L}_0 = \Phi^{-1}A^{(\text{jump})}$ and generates an entry-wise strictly-increasing matrix sequence. Whilst a number of functional iterations exist for QBDs [18], the main advantage of Eq. (9) is that the matrix Φ is upper-triangular with at most $K-1$ non-zero entries per row for all iterations, which allows us to compute L_n efficiently for every n. Further, from (9) we observe that $\bar{L}_n$ has m'_s non-zero columns since the same property holds for $A^{(\text{jump})}$. A disadvantage is that, as with all functional iterations, its convergence rate is linear and may be slow when the utilization is large. Note that (9) is repeated until the difference in norm between two consecutive iterates $||L_{n+1} - L_n||_\infty$ is less than a predefined threshold ϵ.

Solving (9) entails the solution of a linear system with coefficient matrix Φ and m'_s right-hand sides corresponding to the non-zero columns of $\bar{L}_n$, thus requiring $Km_sm'_s$ operations. The result is then multiplied by the m'_s non-zero rows corresponding to the non-zero columns in $\bar{L}_n$, requiring $m_sm'^2_s$ operations. The result is added to the first term, requiring $m_sm'_s$ operations. Assuming M iterations are needed to achieve convergence, the total number of operations is $M(m_sm'^2_s + (K+1)m_sm'_s)$. The storage requirement is $2m_sm'_s + Km_s$ to store $\bar{L}_n$, $\Phi^{-1}A^{(\text{jump})}$, and Φ. Comparing this method with the NARE approach mentioned above is difficult as the latter is estimated to require about $75m_s^3$ operations [19], but this is based on assuming 25% of eigenvalue re-ordering operations when building the real Schur form associated with the Hamiltonian matrix, a fraction that is problem-specific. To make a quantitative comparison, we executed both methods on a commodity computer with $N = 9$ and $K = 6$ (our primary scenario), which results in $m_s = 3003$ and $m'_s = 2002$. We report the times required by NARE and our method for solving Eq. (2) to find the matrix T as follows. The NARE method is fairly consistent across different loads, requiring 860 seconds on average. Under a load of 0.5, our method requires 115 seconds, a reduction of 87% with respect to NARE. A high load of 0.9 increases the execution time to 642 seconds, 25% better than NARE. At the other extreme, smaller loads of 0.1 and 0.3 can be solved in just 39 and 67 seconds, gaining 92% and 95% over NARE, respectively.

4.2 Test cases and faster implementations

For our primary test-suite, we used a set-up with (N, K)=$(9, 6)$, unit mean task-service time (μ=1) and Poisson arrivals with rates λ=0.1, 0.5 and 0.9. We also used MAP arrivals and Erlang-2 service times in some cases. Greater values of K lead to state spaces that are too large for tractable solutions using our current software on a commodity personal computer. However, note that in real storage systems, the value of K would rarely exceed about 16 and the greatest (N, K) pair we know of is $(26, 29)$ [24]. Furthermore, we would not expect significant further qualitative insight from models of ever increasing size, relating to, for example, recommended redundancy level, $N - K$, or the positions of knees (delimiting the effective operating range – see Section 7.4) in graphs as utilization increases.

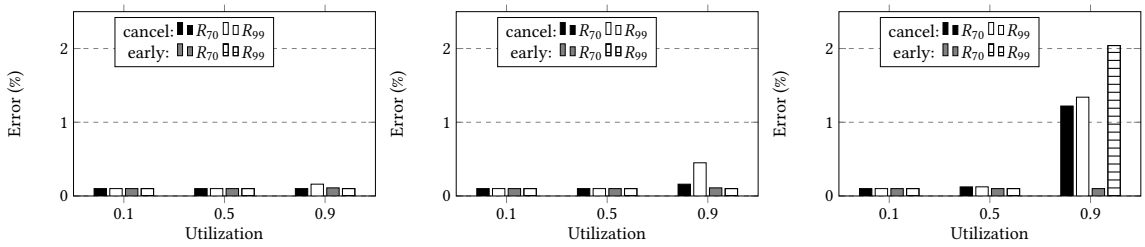
There are two main ways that would substantially reduce the execution time of our models. First, of course, a compiled, production version of our software, written in C for example, would yield major speed-ups and facilitate the solution of bigger problems, with considerably enlarged state-spaces. Secondly, the customized algorithm of section 4.1 is ripe for parallel implementation, either using the multiple cores of a single processor or using many such processors. This is because the matrix S is sparse and upper triangular, and the resultant linear systems of equations are typically solved iteratively by one of several well known algorithms, in which the dominant share of computation time is consumed by the matrix-vector multiply operation. This operation is highly parallelizable and scalable, so that large problems of our type would be good candidates for massively parallel algorithms, which were already able to handle 100 million states over a decade ago, with matrices of arbitrary structure [15–17]. Advances in technology, combined with the special structure our matrices possess, can be expected to make this number over a billion states, and we note that m_s, m'_s are $(254, 186, 856, 183, 579, 396)$ for $K = 27, C = 10$ and still both less than a billion when K increases to 31. One problem in massively parallel computation of this type is the partitioning of the matrix defining the linear equations, typically by some kind of graph or hyper-graph partitioning algorithm, itself probably implemented in parallel [12, 33]. Fortunately, the special structure of S (sparse and upper triangular) would assist this process.

5 Numerical results

The models developed are now applied to evaluate the impact of erasure coding on delivered response time. First, the accuracy of the models is assessed by comparing against simulation.

5.1 Validation

The accuracy of the models' approximations, i.e., the truncation of the difference between any queue and the shortest queue to a constant C, is assessed for both the canceling-at-finish and canceling-at-start cases, considering different utilization levels, namely, low (0.1), medium (0.5), and high (0.9), for the case of (N, K)=$(6, 4)$ as an example. Assuming Poisson arrivals and C=10, we report the errors obtained on the $70^{\text{th}}(R_{70})$ and $99^{\text{th}}(R_{99})$ percentile of response time in Figure 4(a). Note that errors lower than 0.1% are rounded to 0.1%. For each setting, the simulations were run 5,000 times with 200,000 samples each time, from which we estimated the mean response-time as well as its 70^{th} and 99^{th} percentiles. We found the errors on the mean in all test cases to be below 1%; thus we focus on the 70^{th} and 99^{th} percentiles in the following. From the figure, it can be seen that, although C is relatively small, the model provides accurate results for both the 70^{th} and the 99^{th} percentile. Keeping the same C but increasing the values of N to 9 and K to 6, we show in Figure 4(b) the corresponding errors, which increase slightly. This is because having more queues potentially incurs a larger difference between the longest queue and the shortest one. Moreover, we observe larger



(a) (N,K)=(6,4), Exp arrivals (b) (N,K)=(9,6), Exp arrivals (c) (N,K)=(6,4), MAP arrivals

Fig. 4. Approximation error w.r.t. simulation (C=10).

errors at high utilization (0.9), but still lower than 3%, due to the greater queue-length differences at higher loads, thus requiring a larger value of C to achieve the same accuracy.

Figure 4(c) shows the case when the Poisson arrivals in the (6,4) case are replaced by an order-2 MAP, with inter-arrival time having the same mean but a squared coefficient of variation (SCV) of 10, together with an auto-correlation function (ACF) having a decay rate of 0.5, which we obtained by the method of [7]. Clearly, in this setup the model incurs larger errors compared to the case in Figure 4(a). This is simply because its higher variability and auto-correlation leads to burstier traffic, creating longer queues and a higher probability of having large differences between queue lengths. All in all, the errors are remarkably small, and accuracy can be improved, arbitrarily, by increasing the value of C, but at greater computational cost.

To show the extent to which greater accuracy is obtained by increasing the value of C, as well as how to choose C low enough to permit an efficient implementation, but high enough to achieve the desired accuracy for a particular modelling objective, we consider again the mean response time in the MDS implementation of [20], with $N = 10$ and $K = 5$. Latency was plotted against arrival rate for $C = 2, 5, 8, 10$ in Figures 5 and 6 for both canceling-at-finish and canceling-at-start. It can be seen that $C = 5, 8$ and 10 all give latencies that are very close up to an arrival rate of about 1.8 (utilization 0.9) in both cases. Thus, $C = 5$ appears to be an adequate truncation point at these utilizations. Even $C = 2$ is acceptable up to arrival rate around 1.3 (utilization 0.65). To see the extent of the improvement obtained by increasing C from 8 to 10 we zoomed in at high utilizations; see the insets to figures 5 and 6. From these it can be seen that there is really very little improvement and, moreover, the relatively big gap compared with the graph for $C = 5$ suggests that the exact results are approximated very accurately at $C = 10$. In any case, running a system at such a high utilization is non-sensical.

Finally, the graph for canceling-at-start in Figure 6 is seen to match closely that of Lee et. al. [20, Figure 10], certainly lying within their bounds, and the small changes seen in going from $C = 5$ to 8 to 10 suggest good convergence to their simulation results, which themselves may be less accurate at very high utilizations.

5.2 Comparing cancelation policies

5.2.1 Exponential service times We compared the response time statistics obtained from our models under the canceling-at-finish, canceling-at-start and split-merge policies. Specifically, we used our primary set-up with (N, K)=(9, 6), unit mean task-service time (μ=1) and Poisson arrivals with rates λ=0.1, 0.5 and 0.9. Greater values of K can be accommodated with a compiled and/or parallel implementation, but we note again that in real storage systems, the value of K tends to be up to about 16. Figure 3 depicts the utilization, the mean and the 99^{th} percentile of response

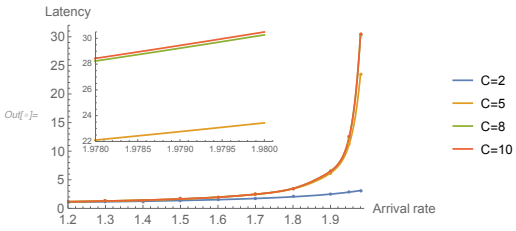


Fig. 5. Mean response times in the ($N = 10$, $K = 5$) model with cancelling-at-finish, computed with $C = 2, 5, 8, 10$. The inset shows the graphs at high utilizations.

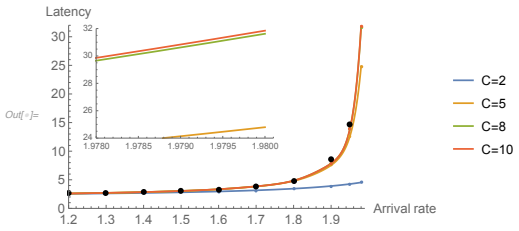


Fig. 6. Mean response times in the ($N = 10$, $K = 5$) model with cancelling-at-start, computed with $C = 2, 5, 8, 10$. The inset shows the graphs at high utilizations. This is the model of Lee et. al. and their simulation results are shown as black dots

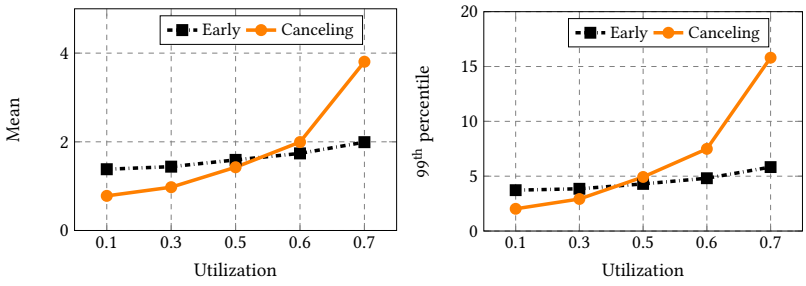
time achieved under different policies. From Figure 3(a), we observe that the utilization is highest in the split-merge case, as servers are blocked by the job in service when it has tasks outstanding at some servers. Canceling-at-finish and canceling-at-start achieve about the same utilization.

The canceling-at-finish policy provides the lowest mean response time and, more emphatically, 99^{th} percentile of response time. This is partly because it limits the extra load introduced by redundant tasks (in common with canceling-at-start) but mainly because it has "power of choice" and can benefit from resource diversity: jobs use the *first* K tasks to finish out of N, whereas they must pre-select a particular set of K servers under canceling-at-start. As we noted in Section 1, the 6^{th} out of 9 order statistic has mean near to 1, compared to the mean of the maximum of 6 exponential, unit-mean random variables of about 2.45. We also observe that, when the load is low, for instance when λ=0.1, canceling-at-start has longer response times than even split-merge. Again, this is because the canceling-at-start policy has no power of choice and cannot benefit from resource diversity as does split-merge. Nevertheless, this is not to say that canceling-at-start is inherently inferior. So far we have only considered exponentially distributed service times, for which canceling-at-finish will indeed always be superior when it incurs no time penalty.

5.2.2 Non-exponential service times Figures 1 and 2 show a sequence of service completions in which canceling-at-start makes less progress. However, suppose now that on arrival to empty queues, job A assigns tasks to the top and middle servers and cancels the other, under the canceling-at-start policy, so that the configuration of in-service tasks becomes A, A, B, assuming the next event is the arrival of job B. With canceling-at-finish, the configuration would be A, A, A. Now suppose that the top A-task completes next, yielding respective configurations B, A, B and B, A, A but with the B-task waiting in the middle queue deleted according to canceling-at-start. Suppose that the next service completion is task A at the middle server – a likely possibility if the variance of task-service times is small. This will lead to respective configurations B, C, B and B, B, B, assuming another job C has arrived; canceling-at-start now wins[8]!

To be more quantitative, consider the simplest case with K=1, exponential service times and Poisson arrivals, as we considered at the beginning of section 3. Then, under canceling-at-finish, the system performs as an $M/M/1$ queue with service rate N times faster than a single server, and under canceling-at-start, as an $M/M/N$ multi-server with single queue. The latter is known to provide inferior performance to the former – although the total service rate is the same in both cases when (and only when) the queue length is greater than $N - 1$.

[8]If the B-task at the middle server completes first in the canceling-at-finish scenario, this policy may be able to catch up, but this is less likely when task-service times have small variance.



(a) Mean (b) 99th percentile

Fig. 7. Comparison of mean response time and its 99th percentile for canceling-at-finish vs. canceling-at-start, with Poisson arrivals, Erlang-2 service times and (N, K)=(4, 2) (C=10).

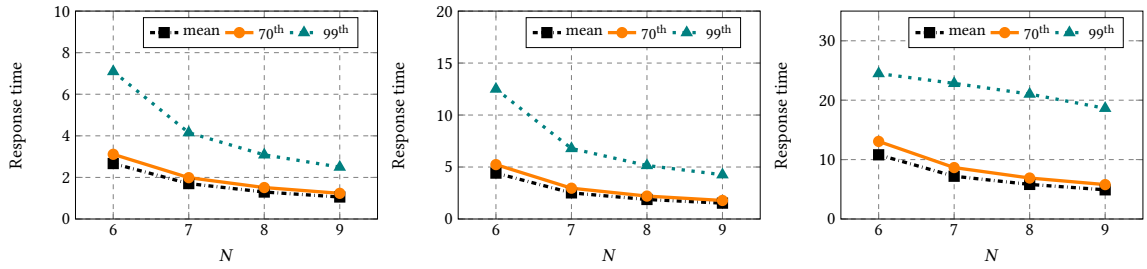
Now consider the same systems with constant service times. With canceling-at-finish, the N servers run in lock-step and there is no benefit in the parallel computation from the performance point of view: both tasks of a job complete service simultaneously and the system behaves as an $M/D/1$ queue with service rate the same as a single server. However, with canceling-at-start, which has no task-parallel computation, the system is just an $M/D/N$ queue, which clearly outperforms the canceling-at-finish alternative.

The key issue is the variability in the service time: up to some point between zero variance (constant service times) and variance equal to the square of the mean (as with exponential service times), one would expect canceling-at-start to be superior at high enough utilization. We therefore conducted simulation experiments with Erlang-2 service times for $N = 4, K = 2$ to test this conjecture. The results are shown in Figure 7, which shows clear crossover points in both the mean response time and its 99th percentile as the utilization increases. To the right of the crossover point, canceling-at-start gives increasingly superior performance. Notice that the crossover point occurs at a lower utilization in the case of the 99th percentile, suggesting canceling-at-start may be the preferred policy for tail reduction. These experiments strongly support the above conjecture, which we expect to be more pronounced in larger systems when the variance of service times is low.

Finally, it should also be remembered that canceling-at-start is easier and more efficient to implement, since it does not require pre-emptive "killing" of tasks in service. Consequently system overhead is reduced, giving better overall performance.

5.3 The impact of redundancy level

Throughout this sub-section, we set K=6 with mean service rate μ=1, and use different values of N=6, 7, 8 and 9, with redundancy levels $N-K$=0, 1, 2, 3, respectively. We show, in Figure 8(a)-(c), the mean and the 70th and 99th percentiles of response time obtained for the canceling-at-finish set-up, at utilizations of 0.1, 0.5 and 0.9, respectively. Clearly, all the response time metrics decrease with increasing N, at all the utilization levels. This is because a job only waits for the first K tasks to complete service, thus benefiting from the more diverse response times with increasing N. The largest decrease in the response time metrics occurs when N increases from 6 to 7, while additional redundant tasks have a smaller impact. Further, these gains become more limited as the utilization increases. Similar results can be observed under the canceling-at-start policy.



(a) Utilization:0.1 (b) Utilization:0.5 (c) Utilization:0.9

Fig. 8. Effect of redundancy level $N-K$ (Canceling-at-finish, K=6, C=10).

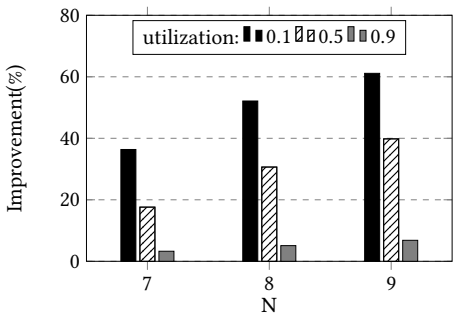
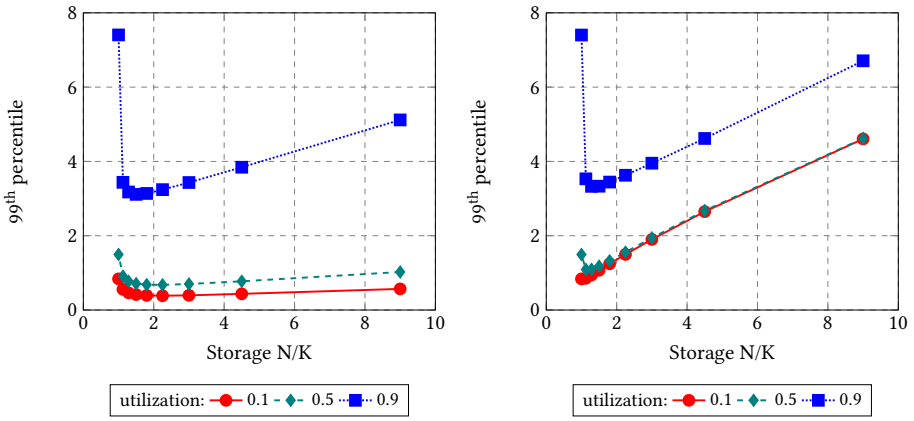


Fig. 9. Improvement on response time's 99th percentile for canceling-at-finish over canceling-at-start (Poisson arrivals, exponential service times, C=10).

To compare the canceling-at-finish and canceling-at-start policies, Figure 9 shows the percentage improvement on the 99th percentile of response time for canceling-at-finish over canceling-at-start. The first observation is that the improvement decreases with the load, as we have already noted, because when the utilization is low, queuing time is small, so that any reduction in it offered by canceling-at-start is very limited. Instead, at low loads, coding with canceling-at-finish can still benefit from power of choice in service times, thus achieving a significant improvement over canceling-at-start. However, when the utilization increases, canceling-at-start has the advantage of avoiding the extra load introduced by redundant tasks under canceling-at-finish, when the first K have started service. Hence the difference between the two canceling schemes becomes smaller and we would expect there to be a cross-over point in the case of non-exponential service times with lower variance, according to Section 5.2.2. Further, we observe that the difference between the two schemes amplifies as N increases, as this allows canceling-at-finish to exploit more diverse resources to reduce job service time. We have obtained similar results by fixing N and varying K, observing consistent superiority of the canceling-at-finish set-up over canceling-at-start, the difference decreasing with the load and the value of K. We explore this further in Section 5.4.

5.4 Storage-latency trade-off

For an object of K fragments, the storage space needed for each fragment is $1/K$ storage units (chosen so that the object has size one) and so, if the object is encoded to have N fragments, the total storage space needed is N/K units. Any



(a) Canceling-at-finish (b) Canceling-at-start

Fig. 10. Trade-off between storage space (corresponding to decreasing K horizontally) and 99$^{\text{th}}$ percentile of response time (N=9, K={1, . . . , 9}, C=10).

gain in response time due to redundancy is therefore achieved at a cost of storage space. When the number of servers is fixed to N=9, Figures 10(a)-(b) show the 99$^{\text{th}}$ percentile of response time and the storage space usage for K={1, . . . , 9}, under canceling-at-finish and canceling-at-start, respectively. We observe that for both policies, when the utilization is 0.5 and 0.9, the response time percentile plotted against K has a clear minimum when the storage requirement is 1.5 (K=6), a little less (maybe for K = 7) in the case of canceling-at-start. This is because when K is small, the sojourn time per task (including queueng time) is relatively large, because more storage is accessed at each node, leading to longer job response time; when K is large (especially when K=N), waiting for many tasks to complete increases the probability of having a straggler with exceptionally long service time (e.g. due to a deteriorating SSD cell). However, when the load is 0.1, we observe that for the canceling-at-finish case the 99$^{\text{th}}$ percentile decreases with N/K (increases with K) up to a point where this trend changes. Instead, with canceling-at-start the lowest response time is achieved using the least storage, i.e., at K=N or N/K=1. To provide some insight into these observations, we use the following two lemmas.

Lemma 5.1. *Given exponential task-service times with mean* $1/(K\mu)$*, the mean job-service time at low load under canceling-at-finish is minimum when* $K = 1$.

Lemma 5.2. *Given exponential task-service times with mean* $1/(K\mu)$*, the mean job-service time at low load under canceling-at-start is minimum when* K=N.

The proofs can be found in Appendices D and E, respectively. At low utilization, job queuing times are small so that both the canceling policies approach split-merge as the utilization approaches zero. Thus, all the eligible tasks of a job can be allocated to nodes (almost) immediately. These results indicate that at low loads, where the service time is the dominant component of response time, it is optimal to select K=1 for canceling-at-finish and K=N for canceling-at-start. This is consistent with what we observed above when considering percentiles as the objective rather than the mean. When the load increases, queuing times become more significant but coding reduces queuing time with increasing K, i.e., as the size of tasks decreases.

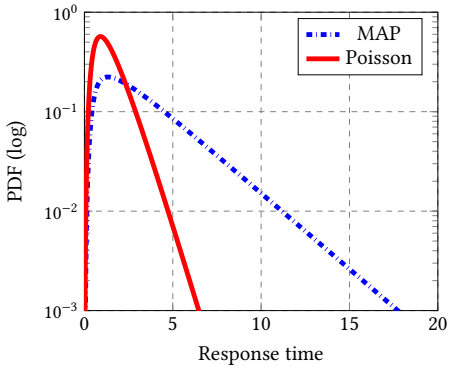


Fig. 11. Response time PDF under Possion and MAP arrivals; (N,K)=(6,4), utilization 0.5, C=10.

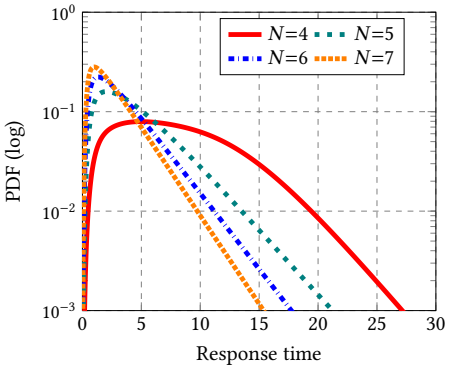


Fig. 12. Response time PDF (K=4, MAP arrivals, utilization 0.5, N=4, 5, 6, 7, C=10).

Table 7. Response time percentiles and the improvement (%) over $N = 4$

Percentile	N=4	N=5	N=6	N=7
70^{th}	10.78	5.96 (44.71%)	4.43 (58.91%)	3.61 (66.51%)
95^{th}	18.10	12.17 (32.76%)	9.57 (47.13%)	7.99 (55.86%)
99^{th}	23.48	17.57 (25.17%)	14.16 (39.69%)	11.93 (49.19%)
99.9^{th}	30.81	25.27 (17.98%)	20.72 (32.75%)	17.56 (43.01%)

5.5 MAP arrivals

We now consider the MAP of Section 5.1, with SCV 10 and ACF decay-rate 0.5, in the canceling-at-finish set-up with (N, K)=(6, 4) at 0.5 utilization. Figure 11 shows the response-time PDFs obtained under both Poisson and MAP arrivals, and we note a much longer tail in the latter case. This suggests that coding might be more beneficial with bursty and auto-correlated arrivals in reducing a long tail.

Figure 12 shows the response-time PDF obtained for K=4 and N=4, 5, 6, 7. We see that even a redundancy of just 1 (i.e. N=K+1) is highly effective in flattening the response-time tail. Table 7 shows the 70^{th}, 95^{th}, 99^{th} and 99.9^{th} percentiles for different values of N, and the corresponding improvement that N=5, 6, 7 has over the N=4 case. Notice that the biggest reduction in the tail comes from introducing just one level of redundancy (N=5 instead of 4), with decreasing benefits as N increases to 5 then 6.

5.6 Canceling-at-finish with general processing times

The PH distribution, defined in Appendix 2.1, is a good candidate for modeling processing times with a range of levels of variability. However, models quickly become computationally expensive since the size of the phase space increases exponentially with the number of task service phases – see Section 5.2.2. We therefore turn to an approximate approach to determine bounds on the response time in the canceling-at-finish set-up, which we choose in view of its superior performance over the canceling-at-start policy in the exponential case.

6 Simple lower and upper bounds

An intuitive lower bound in the Markovian case (Poisson job arrivals, rate λ and exponential servers with unit rate) can be constructed by considering the n processing nodes as *independent* M/M/1 queues, each with arrival rate $k\lambda/n$, this being the throughput of each node at equilibrium. This can be expected to give a lower-than-exact latency because in reality the arrivals are synchronized, giving batch-arrivals to the overall system, which tend to increase latency over the case where arrivals are more "spread out". We do not prove this formally here since the preceding matrix analytical approach already provides a lower bound through its truncation and there are other lower bounds available in the literature, e.g. [20].

For canceling-at-finish, the lower bound is then the (n, k) order statistic of the response times in n of these M/M/1 queues, each having exponential distribution with parameter $1 - k\lambda/n$. In the case that $n = 10, k = 5$, as in the running example of [20], this gives the bounds plotted against the total job arrival rate shown in Figure 13, where the upper bound is given by the split-merge approximation of the previous subsection.

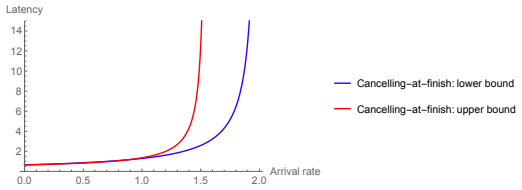


Fig. 13. Upper and lower bounds for the latency in the ($n = 10$, $k = 5$) cancel-at-finish model at arrival rates between 0 and 2.

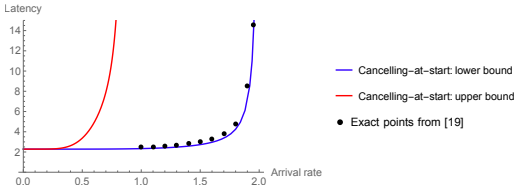


Fig. 14. Upper and lower bounds for the latency in the ($n = 10$, $k = 5$) cancel-at-start model at arrival rates between 0 and 2. Exact values are taken from Lee *et. al.* [20].

For canceling-at-start, the same approach is applied to the queueing time at non-idle servers to obtain a lower bound. Given that there are $i < k$ idle servers at an arrival instant, only $k - i$ tasks from the arriving job have to queue – at $n - i$ servers. Each of their queueing times has the same exponential probability distribution as the response time in a steady state M/M/1 queue under our independence assumption. Hence, because of the memoryless property of both the queueing times and the service times, their response time random variable can be obtained recursively as the maximum of: (1) the sum of the shortest of $n - i$ independent queueing times and one service time, and (2) the conditional response time for i increased by 1, i.e. for $k - i - 1$ out of $n - i - 1$ queues, for $i < k - 1$. The base case of the recursion, $i = k - 1$, is just the sum of the shortest of $n - k + 1$ independent queueing times and one service time. If $i \geq k$, the response time is just the (k, k) order statistic of the service times. The mean response times conditioned on i are then weighted with the equilibrium probabilities that i servers are idle and $n - i$ are not idle out of n independent M/M/1 queues, by the Random Observer Property.

An upper bound is provided by the split-merge model, where servers are blocked by a single job until any K out of its N tasks finish service, thus introducing a synchronization point that makes the analysis more tractable, cf. [11]. In the above $n = 10, k = 5$ scenario, we used split-merge except that when an arriving job finds 5 or more idle servers under canceling-at-start, the latency is just the mean of the maximum of 5 service times. The resulting bounds are shown in Figures 13 and 14. For canceling-at-start, the upper bound is all but useless at even moderate utilizations, and gives an unstable queue at an arrival rate of around 0.7, i.e. utilization of less than 0.4, due to excessive spin-wait times which dominate the model. Fortunately tight bounds can be found on the exact latency curve in this case, and

the near-exact results are also included from [20] in Figure 14. These are the values calculated by our MA method in section 3 and we note the tightness of our simple, quite naive lower bound.

The split-merge result is commonly used to bound the *mean* response time, e.g. [10, 11], but our split-merge model is also able to provide the full response time *distribution*; see section B for Poisson arrivals and general service times, and Appendix C for MAP arrivals and PH service times. Rather unsurprisingly, we conclude from numerical results in Figure 18 (Appendix C) that the split-merge model provides a tight bound at all percentiles at low utilization, but does not perform well at high loads, especially when the arrival process is highly variable and auto-correlated.

7 Related work

A number of recent works have studied the effectiveness of coding on latency reduction in (N, K) partial-join queues [4, 8, 9, 22, 31, 35]. In one variant, each job creates K tasks, which can be served by any K out of the N servers, and departs only after all its tasks complete service[8, 35]. Models that generate N tasks per job and wait for any K of them to complete service, with various policies for handling the remaining $N-K$ tasks, are considered by [4, 9, 22, 31, 32]. In particular, [31] argues that, under certain conditions, the mean response time is minimized when tasks are sent to all servers. Among these works, [4, 8, 22, 35] consider a centralized set-up, where a central queue buffers jobs when all the servers are busy. On the other hand, [9] derives upper and lower bounds on the mean response time for a distributed setup like ours, where upon arrival of a job, each task is dispatched immediately to a different server, each of which maintains a local queue. This approach simplifies the dispatching logic and is often used in server farms at scale. The canceling-at-start policy has also been considered for replication to reduce latency, as in [5, 10], for example. However existing studies usually obtain the *mean* response time only [4, 8, 9, 22, 31, 35], whereas our work compares policies in the context of response time *distribution*, and medium-high quantiles of response time in particular.

The closest work to ours is [20], considered in section 6, which deals with canceling-at-start implementations of MDS. Although it is restricted to Poisson arrivals and exponential service times, it does provide arbitrarily tight upper and lower bounds on mean response time and percentiles. However, it does not deal with canceling-at-finish. Xiang *et. al.* propose a novel *probabilistic* allocation scheme of k_i out of n_i encoded "chunks" for heterogeneous files, with sizes $n_1, n_2, \ldots$ chunks [36]. Whilst the probabilistic scheme is not what we have considered, nor even a common implementation, this work proves that it does provide a tight upper bound and admits service times with arbitrary probability distribution.

A limitation of all these papers, including our own, is that the methods do not scale to large n and k. Li *et. al.* address this problem by a form of mean field analysis, whereby any fixed number of queues are assumed to be independent of each other as the number of servers goes to infinity [21]. Simulation results support the validity of the independence assumption, and the mean-field approach certainly looks worthy of further investigation in large scale cloud systems, for example. However, this is not necessary in the small scale storage systems considered in the present paper and, in fact, the independence assumption would be hard to justify.

Four of the areas in which we plan further related research are the following.

7.1 Family of canceling policies

Rather than just canceling-at-start and canceling-at-finish, there is actually a *family* of canceling policies; tasks can be canceled in-queue when $K, K+1, K+2, \ldots, N$ have started service, followed by pre-emptive cancelation of a further $0, 1, \ldots, N-K$ tasks in-service when K tasks have finished. We considered only the first (canceling-at-start) and last

(canceling-at-finish) of these in this paper. The limiting "lack of power of choice" in the canceling-at-start policy may be largely overcome by allowing even just one more than K to start service before deleting the remaining tasks in-queue.

It is also important to assess the impact of cancellation delays on the scheduling strategies; we have assumed these to be small and considered them as overhead. Whilst this may well be effective when cancelations occur routinely along with other background processes, it would be interesting to see the impact when this is not the case, for example at low levels of redundancy in dedicated storage networks.

7.2 Implementation enhancements

The performance of the various cancelation policies should be evaluated in the context of both MAP arrivals and non-exponential service times so that it can be established whether static policy choices are robust enough to handle a wider set of scenarios or whether policies need to be changed based on dynamic workload characteristics. This will require parallel processing techniques along the lines outlined in Section 4.2. Moreover, we wish to investigate non-synchronous policies approximately to improve the bound provided by split-merge (see Section 5.6). Such approximations would be compared against the above extended implementation.

Furthermore, it would also be interesting to consider non-Markovian arrivals and service times, such as may occur with self-similar traffic common in the internet. For example, we might ask whether the comparison of canceling-at-start and canceling-at-finish examined in Section 5.2.2 generalises to new-longer-than-used (NLU) and new-shorter-than-used (NSU) service time distributions. NLU distributions can be used to characterize scenarios where the downloading time is a constant value followed by a short latency tail. For instance, the data downloading time of Amazon AWS has been observed thus, with a roughly exponential tail, in [22]. In contrast, NSU distributions can characterize occasional slow responses, as considered here with MAPs.

7.3 Correlated task-service times

In certain systems with forking, correlation amongst the tasks comprising a job does have a significant impact. As noted in the introduction, this is particularly relevant in replication, but there may also be such cases in storage-sharding; for example if the sizes of the fragments of a storage object were very large and the same for all tasks, outweighing the variability in device-access overheads. Our approach to this problem is to use the hyper-erlang special case of PH-service times. This allows a choice between different classes of service time, e.g. large or small in the simplest case. The key additional step is to ensure that the same choice is made for every replica of a job, or fragment of an object. This can be done along the lines of [27], which studies similar issues in a system with no canceling at all, where all redundant tasks are allowed to run to completion.

7.4 Performance metrics

Although latency vs. throughput charts are good for visually comparing policies, ideally we would prefer a single numeric value as a measure of goodness for ranking each policy. We have found the *ATP value of the knee* (see [25]) to be a good metric for practical use since it combines the top of the operating throughput range and the corresponding latency. The knee of a performance graph delimits the effective operating range that one can expect in production environments and so is somewhat subjective. A knee may be defined in terms of a system's latency *and* its throughput, which is the same as the arrival rate in a system at equilibrium. For a given latency versus throughput curve, the *Accelerated Throughput* (ATP) metric is defined to be the ratio of the throughput x and the *overall response time*, $\mathrm{ORT}(x)$,

Table 8. ATP-based comparison

Arrivals/Services	Policy	ATP
Possion/Expo	Split-merge (6,6)	0.0199
	Split-merge (9,6)	0.1415
	Canceling-at-finish (9,6)	**0.2837**
	Canceling-at-start (9,6)	0.1675
MAP/Expo	Split-merge (6,6)	0.0086
	Split-merge (9,6)	0.0389
	Canceling-at-finish (9,6)	**0.0819**
	Canceling-at-start (9,6)	0.0794

which is the average value of the latency over the throughput range $[0, x]$. In [25], the point x^* at which the maximum value of ATP occurs is adopted as the location of the knee and the interval $[0, x^*]$ is called the *operational throughput range*. For $M/M/1$ queues, the ATP maximum indicates an operational utilization range of zero up to the knee at 71.5%, whereas Kleinrock's power metric suggests a more conservative range of zero to 50%, [13].

Our selected policies are compared using the higher-is-better ATP metric in Table 7. For exponential service times, this clearly demonstrates the superiority of the canceling-at-finish policy when arrivals are Poisson. The same applies to correlated and bursty MAP-arrivals but here the case is less clear cut. The reason for this is that the knee occurs at a lower utilization with MAP-arrivals than with Poisson, possibly because a sudden large influx of jobs that cannot take advantage of power of choice leads to more rapidly increasing response times. Variations on the ATP metric could make it applicable in a wide range of applications – not only in IT but also in management and business processes.

8 Conclusion

We have evaluated two policies – canceling-at-finish and canceling-at-start – for managing (N, K) partial-join queues and compared them with a "benchmark", split-merge policy in terms of job response time distribution. In general, long response times can occur on exceptionally large service times, when arrivals become more intense (high utilization), or when both occur roughly at the same time. These effects are compounded when a job has to wait for all its parallel tasks. This is common in erasure encoded storage systems but consistent performance is still required as the system ages and cells deteriorate electronically in SSDs.

Some of the "take-away" conclusions we draw from our experimental results are the following:

- In (N, K) partial-join systems with MAP arrivals and exponential service times, both canceling policies significantly reduce the high quantiles of job response time, especially with highly variable and correlated MAP arrival streams.
- Canceling-at-finish achieves the lowest response times across the range of utilizations in most of the models we tested, since it has power of choice in service times, benefiting from resource diversity, whilst also limiting the extra load introduced. However, when service times have lower variance (compared to an exponential random variable), canceling-at-start gives the better response times at higher utilizations, because the performance benefits of the order statistics are then reduced. Analytical models yield the crossover point and so recommend which policy is the better in a given scenario.

- Canceling-at-start has a lower overhead and is easier to implement since in-queue canceling requires less modification of the system than canceling in-service. The best choice, which is also affected by the utilization, can be made on a sound, quantitative basis, in terms of performance improvement and cost.
- Job response time metrics decrease with increasing N, at all utilization levels, for both the canceling schemes. However, any gain in response time is achieved at a cost of storage space: if an object is encoded to have N fragments, the total storage space needed is N/K units - i.e. more than 1.

From the modeling point of view, we have produced the first analysis of partial-join-queues that provides response time PDFs, rather than just mean values. We were able to compare alternate policies through judicious, approximate representation of the state space, to an arbitrary degree of accuracy at increasing computational expense – an instance of "approximate computation".

References

[1] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. Why let resources idle? aggressive cloning of jobs with dolly. In *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Ccomputing*, HotCloud'12, Berkeley, CA, USA, 2012.

[2] Søren Asmussen and Jakob R Møller. Calculation of the steady state waiting time distribution in GI/PH/c and MAP/PH/c queues. *Queueing Syst.*, 37:9–29, 2001.

[3] D.A. Bini, B. Iannazzo, and B. Meini. *Numerical Solution of Algebraic Riccati Equations, SIAM Book Series Fundamentals of Algorithms*. SIAM, 2012.

[4] Shengbo Chen, Yin Sun, Ulas C Kozat, Longbo Huang, Pradeep Sinha, Guanfeng Liang, Xin Liu, and Ness B Shroff. When queueing meets coding: Optimal-latency data retrieving scheme in storage clouds. In *IEEE INFOCOM*, pages 1042–1050, 2014.

[5] Jeffrey Dean and Luiz André Barroso. The tail at scale. *CACM*, 56:74–80, 2013.

[6] Kristen Gardner, Mor Harchol-Balter, and Alan Scheller-Wolf. A better model for job redundancy: Decoupling server slowdown and job size. In *IEEE MASCOTS*, 2016.

[7] Armin Heindl, Gábor Horváth, and Karsten Gross. Explicit inverse characterizations of acyclic MAPs of second order. In *EPEW*, 2007.

[8] Longbo Huang, Sameer Pawar, Hao Zhang, and Kannan Ramchandran. Codes can reduce queueing delay in data centers. In *IEEE ISIT*, pages 2766–2770, 2012.

[9] Gauri Joshi, Yanpei Liu, and Emina Soljanin. On the delay-storage trade-off in content download from coded distributed storage systems. *IEEE JSAC*, 32(5):989–997, 2014.

[10] Gauri Joshi, Emina Soljanin, and Gregory Wornell. Efficient replication of queued tasks for latency reduction in cloud systems. In *Allerton*, 2015.

[11] Gauri Joshi, Emina Soljanin, and Gregory Wornell. Efficient redundancy techniques for latency reduction in cloud systems. *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 2(2):12:1–12:30, April 2017.

[12] George Karypis and Vipin Kumar. Multilevel algorithms for multi-constraint graph partitioning. In *Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, SC '98, pages 1–13, Washington, DC, USA, 1998. IEEE Computer Society.

[13] Leonard Kleinrock. On flow control in computer networks. In *Proceedings of the International Conference on Communications*, volume 2, pages 27–2, 1978.

[14] Ana Klimovic, Christos Kozyrakis, Eno Thereska, Binu John, and Sanjeev Kumar. Flash storage disaggregation. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, pages 29:1–29:15, New York, NY, USA, 2016. ACM.

[15] W.J. Knottenbelt. *Parallel Performance Analysis of Large Markov Models*. PhD thesis, Imperial College London, London, United Kingdom, February 2000.

[16] W.J. Knottenbelt, P.G. Harrison, M.A. Mestern, and P.S. Kritzinger. A probabilistic dynamic technique for the distributed generation of very large state spaces. *Performance Evaluation Journal*, 39:127–148, February 2000.

[17] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1994.

[18] Guy Latouche and Vaidyanathan Ramaswami. *Introduction to matrix analytic methods in stochastic modeling*. SIAM, 1999.

[19] A. J. Laub. A schur method for solving algebraic Riccati equations. *IEEE TACON*, 24:913–921, 1979.

[20] K. Lee, N. B. Shah, L. Huang, and K. Ramchandran. The mds queue: Analysing the latency performance of erasure codes. *IEEE Transactions on Information Theory*, 63(5):2822–2842, May 2017.

[21] Bin Li, Aditya Ramamoorthy, and R. Srikant. Mean-field analysis of coding versus replication in large data storage systems. *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 3(1):3:1–3:28, February 2018.

[22] Guozheng Liang and Ulas C Kozat. Fast cloud: Pushing the envelope on delay performance of cloud storage with coding. *IEEE/ACM TON*, 22(6):2012–2025, 2014.

[23] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error Correcting Codes*. N. Holland, 1977.

[24] Netapp. Netapp fas8000 series technical specifications. Technical report, 2016. URL = http://www.netapp.com/us/products/storage-systems/fas8000/fas8000-tech-specs.aspx.

[25] Naresh M Patel. Half-latency rule for finding the knee of the latency curve. *ACM SIGMETRICS PER*, 43:28–29, 2015.

[26] Z. Qiu, J. F. Perez Bernal, R. Birke, L. Chen, and P. G. Harrison. Cutting latency tail: Analyzing and validating replication without canceling. *IEEE Transactions on Parallel and Distributed Systems*, PP(99):1–1, 2017.

[27] Z. Qiu, J. F. Pérez, R. Birke, L. Chen, and P. G. Harrison. Cutting latency tail: Analyzing and validating replication without canceling. *IEEE Transactions on Parallel and Distributed Systems*, 28(11):3128–3141, Nov 2017.

[28] Zhan Qiu, Juan F. Pérez, and Peter G. Harrison. Beyond the mean in fork-join queues: Efficient approximation for response-time tails. *Perform. Eval.*, 91:99–116, 2015.

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

[30] N. B. Shah, K. Lee, and K. Ramchandran. When do redundant requests reduce latency? *IEEE Transactions on Communications*, 64(2):715–722, Feb 2016.

[31] Nihar B Shah, Kangwook Lee, and Kannan Ramchandran. The MDS queue: Analysing latency performance of codes and redundant requests. Technical report, Technical Report, 2012.

[32] Y. Sun, Z. Zheng, C. E. Koksal, K. Kim, and N. B. Shroff. Provably delay efficient data retrieving in storage clouds. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 585–593, April 2015.

[33] Aleksandar Trifunović and William J Knottenbelt. Parallel multilevel algorithms for hypergraph partitioning. *Journal of Parallel and Distributed Computing*, 68(5):563–581, 2008.

[34] Benny Van Houdt and Johan S. H. van Leeuwaarden. Triangular M/G/1-type and tree-like quasi-birth-death markov chains. *INFORMS J. on Computing*, 23(1):165–171, 2011.

[35] Yu Xiang, Tian Lan, Vaneet Aggarwal, and Yih Farn R Chen. Joint latency and cost optimization for erasurecoded data center storage. *ACM SIGMETRICS PER*, 42(2):3–14, 2014.

[36] Yu Xiang, Tian Lan, Vaneet Aggarwal, Yih-Farn R. Chen, Yu Xiang, Tian Lan, Vaneet Aggarwal, and Yih-Farn R. Chen. Joint latency and cost optimization for erasure-coded data center storage. *IEEE/ACM Trans. Netw.*, 24(4):2443–2457, August 2016.

A Split-merge processing

Under a split-merge policy, all a job's tasks are allocated to servers simultaneously immediately after the completion of service of the previous job, but even here the problem is not straightforward. A K out of N *split-merge model with Poisson arrivals* is precisely an M/G/1 queue with job service time that is the K^{th} order statistic out of N task service times. Note that, in this case, the first 1 to $K-1$ tasks to complete will continue to occupy their servers until the merge occurs (when the K^{th} task completes), so this approach has some inherent idling, but serves as an upper bound on other policies and configurations. Such a bound would obviously become increasingly loose as K increases above 1, as well as at increasing job arrival rate. Indeed, because servers are sometimes idle even when there is work in the queue, the system will become saturated at a lower arrival rate when $K > 1$. Of course, when $K = 1$ under canceling-at-finish, the job service time is the minimum of the N task service times and model is exact.

A.1 M/G/1 split-merge model

When the task service times are negative exponential with parameter μ, we can derive a closed-form expression for the job response time distribution in this split-merge queue. Taking $(N, K) = (9, 6)$, typical for storage applications, the service time is the 6^{th} order statistic out of 9 independent exponential random variables with parametrer μ. The service time PDF is routinely derived as $504\mu e^{-4\mu t}(1 - e^{-\mu t})^5$, with Laplace transform $S^*(\theta) = \frac{60480\mu^6}{(\theta+4\mu)(\theta+5\mu)(\theta+6\mu)(\theta+7\mu)(\theta+8\mu)(\theta+9\mu)}$. This can be used in the Pollaczek-Khinchine transform formula[9] to obtain the Laplace transform of the job response time, $R^*(\theta)$. For simplicity, we take the mean task service time to be 1 so that the mean (9,6) order statistic is just under 1. The Laplace transform of the job response time is then

$$R^*(\theta) = \frac{48(6300 - 2131\lambda)}{5\left(\theta^6 + 39\theta^5 + 625\theta^4 + 5265\theta^3 + 24574\theta^2 - (\theta + 13)(\theta(\theta + 13)(\theta(\theta + 13) + 118) + 4632)\lambda + 60216\theta + 60480\right)},$$

which depends on the utilization level, λm (through λ), where $m = -\lambda S^{*\prime}(0)$ is the mean job service time[10]. This expression can be decomposed into partial fractions, in terms of all the roots of the degree-6 polynomial denominator, and inverted to give the PDF. Given the utilization, these roots are easily calculated numerically, and the Laplace transform inverted term by term to give the PDF $r(t)$ of job response time. For example, at 70% utilization, the cumulative distribution function (CDF) of the response time is (after integration):

$$R(t) = 1 + 0.239e^{-10.8t} - 1.37e^{-0.554t} + 0.295e^{-8.88t}\cos(3.07t) - 0.161e^{-4.59t}\cos(3.65t) + 0.485e^{-8.88t}\sin(3.07t) + 0.607e^{-4.59t}\sin(3.65t)$$

Figure 15 shows three high percentile curves as job utilization increases, and the "knees" of the curves, which delimit the maximum effective operating range that one can expect in production environments; see Section 7.4. In this split-merge model, the utilization at the knee is around 70% and the 99.5^{th} percentile of the job response time is 9.8 units, which is significantly less than the 28.9 units corresponding to the same percentile in the (6,6) model (the case of no redundancy). Figure 16 shows the response time densities at utilizations 0.01, 0.5, 0.78, and 0.9. At low utilizations, response time is close to job service time, i.e. the 6^{th} out of 9 order statistic of the task processing times, since queuing

[9] The Pollaczek-Khinchine transform formula for the response time in an M/G/1 queue is $R^*(\theta) = \frac{(1-\rho)\theta S^*(\theta)}{\theta - \lambda(1 - S^*(\theta))}$, where λ is the Poisson arrival rate, $S^*(\theta)$ is the Laplace transform of the service time PDF and $\rho = -\lambda S^{*\prime}(0)$ is the utilization.

[10] In the split-merge model, utilization is inflated by the spin-waiting (or, idling) time for merging. However, one can think of utilization as the fraction of the maximum throughput achieved.

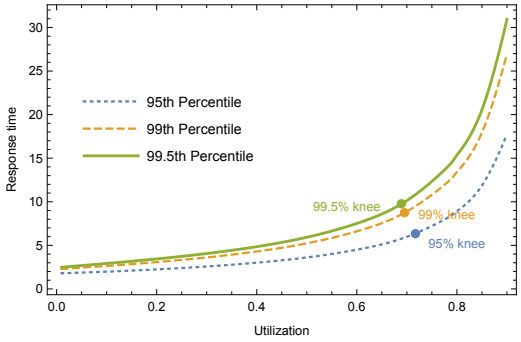


Fig. 15. Response time 99.5th, 99th and 95th percentiles at utilizations between 0.01 and 0.9, showing knees at 0.689, 0.695, 0.717 respectively, in the (9, 6) split-merge model.

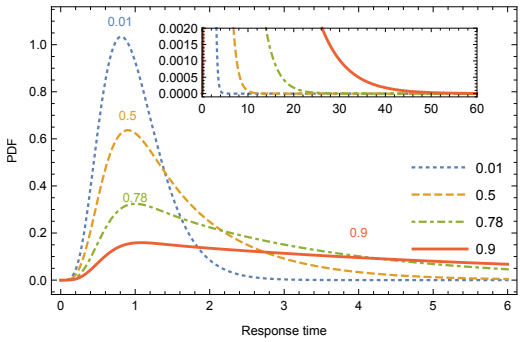


Fig. 16. Response time density for 6 from 9 tasks at utilizations 0.01, 0.5, 0.78, 0.9. Longer time-scale shown in the inset.

times are negligible. As the load increases, the peak drops and the area under the density curve pushes out to the right. This is as we would expect, but we also observe a surprisingly much shorter tail, even at a utilization as high as 0.9[11].

This model is simple to analyze and provides an upper bound on response time statistics; Appendix C extends it to MAP arrivals and phase-type (PH) service times. Although in this example we have only considered the simplest, benchmark set-up, with exponential task-service times and Poisson arrivals, we observe that it is already effective in reducing the response time tail. In Appendix B it is shown that the tail is also greatly reduced when the task service time density has a "lump" in its tail[12] – i.e. a region where the density is exceptionally high due to rare events like multiple reads of an ageing Flash storage cell. We observe that in both cases, the utilization of roughly 70% at the 99.5th percentile's knee includes about 34% of spin-waiting time (by completed tasks awaiting a merge). This indicates an opportunity to improve the job processing rate further with partial-join policies, which is the main topic of the present paper.

B Lumpy tails

One of the characteristics of the Flash storage technology is that aging and the repeated use of storage cells can cause orders of magnitude increases in device IO time. This can lead to rare but exceptionally long processing times at some of the servers, especially as the system ages and cells need to be replaced. When the tail of the task-service-time distribution has such a "lump", we can still reduce the split-merge upper bound model to an M/G/1 queue. Suppose the server processing-time for a task is now a mixture of two random variables, one negative exponential and one Erlang-4, with respective means 0.5 and 50.5 units and weights 99% and 1%. This generates mostly short processing times but a small fraction will be orders of magnitude longer. The mean processing time is one unit but the 99.5th percentile is 46.4 units. Such a lumpy tail is highly undesirable in parallel systems, where tails can become magnified, and we wish to determine whether sharding can protect read-access times from these tail characteristics.

The Laplace transform of the PDF of the service time can be computed numerically and this can then be used as input to the Pollaczek-Khinchine formula for the Laplace transform of the response time density in an M/G/1 queue.

[11]The inset plot with wider response time range shows the tail remaining close to zero at higher response times.
[12]Note that such a "lumpy tail" is not necessarily a *fat tail* (or *heavy tail*), well known in the context of self-similar data. A fat tail is super-exponential, e.g. polynomial, whereas the lumpy tail considered in Appendix B is asymptotically exponential. Conversely, there are many fat tails that do not have lumps, e.g. when they decay purely polynomially.

Numerical inversion then provides the PDF. As with exponential task-service times in Section A.1, the utilization at the knees is again around 70% but the 99.5$^{\text{th}}$ job response time is 5.1 units, which is significantly less than the 46.4 units corresponding to the task processing time and 88.0 units in the (6,6) model. Figure 17 shows the response time densities at utilizations 0.01, 0.5, 0.78, and 0.9, and we see that we can indeed remove the lumpy characteristics and deliver significantly better job response times. Note, however, that these results are only upper (pessimistic) bounds, which become increasingly loose as the load increases.

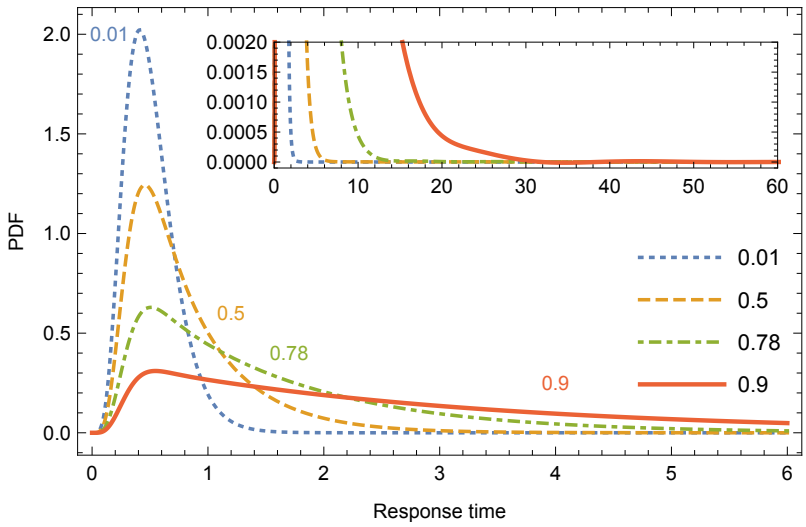


Fig. 17. Response time density when task-service-times have lumpy tails, for 6 from 9 tasks at utilizations 0.01, 0.5, 0.78, 0.9. Longer time-scale shown in the inset.

C MAP-PH upper bounds

In this appendix, we consider the upper bound given by split-(partial-)merge operation with general PH task-processing times and MAP arrivals. Although the simpler M/G/1 split-merge model considered in the previous appendices can, in principle, handle arbitrary service times, including non-PH, it is restricted to Poisson arrivals, which is what we relax here.

To solve the model with PH task processing times, the key is to keep track of the phase of all tasks in service. Assuming the task processing-time distribution is PH($\boldsymbol{\alpha}_{\text{task}}, S_{\text{task}}$) with m_{task} phases, we define the service process of the job in service as $\mathcal{S}^{\text{PH}}(t) = (n_1(t), \ldots, n_{m_{\text{task}}}(t))$, where $n_p(t)$ is the number of tasks in service in phase p at time t, for $1 \le p \le m_{\text{task}}$. Thus $\mathcal{S}^{\text{PH}}(t)$ takes values in the set $S^{\text{PH}} = \{\boldsymbol{n} = (n_1, \ldots, n_{m_{\text{task}}}) | n_p \in \{0, \ldots, N\}, N{-}K{+}1 \le \sum_{p=1}^{m_{\text{task}}} n_p \le N\}$. Note that we can represent exponential task processing times with mean processing time $1/\mu$, by setting $\boldsymbol{\alpha}_{\text{task}} = 1$ and $S_{\text{task}} = \mu$, with m_{task}=1 phase. The job service time is the interval between the instant that all the tasks of a job enter service, and the instant the K^{th} task completes service.

Table 9. Transition rates for matrix S_{ser}

From	To	Rate	Condition
$(n_1,\ldots,n_{m_{\text{task}}})$	$(n_1,\ldots,n_i{-}1,\ldots,n_j{+}1,\ldots,n_{m_{\text{task}}})$	$n_i S_{\text{task}}(i,j)$	–
$(n_1,\ldots,n_{m_{\text{task}}})$	$(n_1,\ldots,n_i{-}1,\ldots,n_{m_{\text{task}}})$	$n_i S^*_{\text{task}}(i)$	$\sum_{i=1}^{m_{\text{task}}} n_i{>}N{-}K{+}1$

C.1 The Job Service-time Distribution

Following this definition, we first seek the PH representation PH($\boldsymbol{\alpha}_{\text{ser}}, S_{\text{ser}}$) of the job service time. As all N tasks of a job start service at the same time, and each of them selects its initial phase independently, the initial distribution of the job service phase $\boldsymbol{\alpha}_{\text{ser}}$ follows a multinomial distribution, where the probability that a job starts service in phase $\boldsymbol{n}$ is

$$\boldsymbol{\alpha}_{\text{ser}}(\boldsymbol{n}) = N! \prod_{i=1}^{m_{\text{task}}} \frac{\boldsymbol{\alpha}_{\text{task}}(i)^{n_i}}{n_i!}, \quad \boldsymbol{n} \in S^{\text{PH}},$$

where $\boldsymbol{\alpha}_{\text{task}}(i)$ is the probability that a task starts service in phase i. The transition rates of the sub-generator S_{ser} are summarized in Table 9. The first row considers the case where one task in service phase i jumps to phase j without completing service. The second row covers the case where a task in phase i completes service.

C.2 The Job Queuing-time Distribution

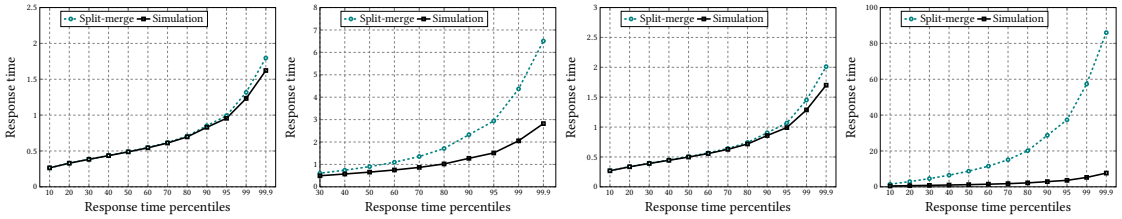
To determine the queuing-time distribution, we define a bivariate Markov process $\{(\mathcal{X}(t), \mathcal{J}^{\text{PH}}(t)|t{\geq}0\}$, where the *age* $\mathcal{X}(t)$ is the total time-in-system of the job in service, and the *phase* $\mathcal{J}^{\text{PH}}(t){=}\mathcal{S}^{\text{PH}}(t)$ is the same as the one defined for the service process. We thus need to define matrices S and $A^{(\text{jump})}$ that describe the evolution of the service phase process. The off-diagonal entries of S hold the service transition rates without a job service completion; thus it is given by S_{ser}. Similarly, $A^{(\text{jump})}{=}{-}(S_{\text{ser}}\mathbf{1})\boldsymbol{\alpha}_{\text{ser}}$, as the vector $-S_{\text{ser}}\mathbf{1}$ holds the rates at which a job completes service, and a new job selects its initial phase according to $\boldsymbol{\alpha}_{\text{ser}}$. We now follow the steps of Section 2 to derive the job queuing-time distribution PH ($\boldsymbol{\alpha}_{\text{wait}}, S_{\text{wait}}$).

After obtaining the PH representations for the queuing-time and service-time distributions, we can obtain the PH representation of the response-time distribution as

$$\boldsymbol{\alpha}_{\text{res}} = [\boldsymbol{\alpha}_{\text{wait}} \quad (1 - \boldsymbol{\alpha}_{\text{wait}}\mathbf{1})\boldsymbol{\alpha}_{\text{ser}}], \quad S_{\text{res}} = \begin{bmatrix} S_{\text{wait}} & (-S_{\text{wait}}\mathbf{1})\boldsymbol{\alpha}_{\text{ser}} \\ \mathbf{0} & S_{\text{ser}} \end{bmatrix},$$

given the independence between service and waiting times. This representation captures the observation that, for some jobs, the response time is made up of a period of waiting followed by a period of service, while others start service directly.

We show in Figure 18(a)-(b) and Figure 18(c)-(d) the response time percentiles obtained by the split-merge model with Poisson arrivals and MAP arrivals (SCV = 10, decay-rate = 0.5), respectively, alongside simulation results, at utilization levels 0.9 and 0.1. For the task processing-time, we used a mixture of two random variables: one negative exponential and one Erlang-4, with respective means 0.5 and 50.5 units and weights 99% and 1%, giving an overall mean service time of 1 – exactly as in Section B. As expected, we observe tight bounds when the utilization is low, as the response time comprises mostly service times and the impact of server-blocking due to spin-waits is small. However, the bounds become looser with increasing utilization, as the queuing time becomes significant at high loads due to the server blocking, thus providing a very conservative bound at utilization 0.9. Further, it is worth noting that the bound is



(a) Utilization: 0.1, Poisson (b) Utilization: 0.9, Poisson (c) Utilization: 0.1, MAP (d) Utilization: 0.9, MAP

Fig. 18. Upper bound of the split-merge model on the canceling-at-finish model ($(N, K) = (9, 6)$, C=10).

looser on the tail than on the lower percentiles. Also, with MAP arrivals, we observe a large difference between the bound and the simulation results at almost all percentiles, as bursty arrivals lead to longer queuing times, which are amplified due to the blocking effect in the split-merge model.

D Proof of Lemma 5.1

At low loads, the response times are mostly composed of service times as queuing times are negligible. If we focus on the service time only, under canceling-at-finish this time comprises K stages, where in stage $1 \leq i \leq K$ there are $N-i+1$ parallel tasks in service. In each stage the mean service time at each node is $1/(K\mu)$, assuming a uniform division of a job into tasks that are each K times shorter, on average, than in the single processor case. The mean service time for a job can therefore be written as

$$E[S_C(K,N)] = \frac{1}{K\mu}\sum_{i=1}^{K}\frac{1}{N-i+1},$$

where $S_C(K,N)$ is the service time random variable for a job under canceling-at-finish with (N,K) coding. This can be rewritten as, after some rearrangement,

$$E[S_C(K{+}1,N)] + \frac{1}{(K+1)\mu}\left(\frac{1}{K}\sum_{i=1}^{K}\frac{1}{N-i+1} - \frac{1}{N-K}\right).$$

Each term in the sum is strictly less than $1/(N-K)$ so that $E[S_C(K,N)] < E[S_C(K{+}1,N)]$ for all K. Hence the minimum mean service time is obtained when K=1.

E Proof of Lemma 5.2

As in the canceling-at-finish case, with canceling-at-start, the service time S_E comprises K stages, where in stage i there are $K-i+1$ tasks in service. Again, a task has mean service time $1/(K\mu)$ at each node so that the mean job service time is

$$E[S_E(K,N)] = \frac{1}{K\mu}\sum_{i=1}^{K}\frac{1}{K-i+1} = \frac{1}{K\mu}\sum_{i=1}^{K}\frac{1}{i},$$

This can be written as

$$E[S_E(K{+}1,N)] + \frac{1}{(K+1)\mu}\left(\frac{1}{K}\sum_{i=1}^{K}\frac{1}{i} - \frac{1}{K+1}\right).$$

Each term in the sum is greater than $1/K$ so that the sum is greater than 1. Hence $E[S_C(K,N)] > E[S_C(K{+}1,N)]$ for all K and so the minimum mean service time is obtained when K=N.

F Theorem 1 in [28]

Here we present the result [28, Theorem 1], which we exploit to obtain the PH representation of the response times. The result requires the PH representation of the service times $(\boldsymbol{\alpha}_{\text{service}}, S_{\text{service}})$, and that of the queueing times $(\boldsymbol{\alpha}_{\text{wait}}, S_{\text{wait}})$, of sizes m_{ser} and m, respectively. Also, the matrices T and S^{MAP} are as in (2) and the vector $\boldsymbol{\pi}(0)$ is given by (3). Recall that m_a is the number of phases in the arrival process and γ is the probability that a job has to wait. We also define n_f as the number of *not-full* phases, i.e., the size of the matrix $S_{\text{not-full}}$ defined in (4) and Table 3. From these, the PH representation of the response times is obtained as follows.

Theorem F.1. *The job response-time follows a PH distribution with parameters $(\boldsymbol{\alpha}_{res}, S_{res})$, where*

$$\boldsymbol{\alpha}_{res} = \begin{bmatrix} (1-\gamma)\boldsymbol{\pi}(0) & \tilde{\boldsymbol{\alpha}}_{busy} & \mathbf{0}_{1\times m} \end{bmatrix}, \quad S_{res} = \begin{bmatrix} S_{service} & 0_{m_{ser}\times m_{ser}} & 0_{m_{ser}\times m} \\ 0_{m_{ser}\times m_{ser}} & \tilde{S}_{service} & (-\tilde{S}_{service}\mathbf{1})P_{s,w} \\ 0_{m\times m_{ser}} & 0_{m\times m_{ser}} & S_{wait} \end{bmatrix}, \tag{10}$$

where $\tilde{S}_{service} = \Delta^{-1}S'_{service}\Delta$, Δ is a diagonal matrix such that $\Delta\mathbf{1} = \boldsymbol{\eta}'$, and $\boldsymbol{\eta} = -\boldsymbol{\alpha}_{service}S^{-1}_{service}$ is the stationary distribution of the phase during service. Also, $\tilde{\boldsymbol{\alpha}}_{busy} = (-S_{service}\mathbf{1})'\Delta$. Finally, $P_{s,w}$ is an $m_{ser}\times m$ matrix given by

$$P_{s,w} = \begin{bmatrix} \tilde{P}_{s,w} \\ 0_{m_a m_{nf}\times m} \end{bmatrix},$$

where $\tilde{P}_{s,w} = \Gamma^{-1}(T - S^{MAP})'\Lambda$, and Γ and Λ are diagonal matrices such that $\Gamma\mathbf{1} = (T - S^{MAP})'\Lambda\mathbf{1}$ and $\Lambda\mathbf{1} = \boldsymbol{\alpha}'_{busy}$, and $\boldsymbol{\alpha}_{busy} = -\boldsymbol{\pi}(0)T^{-1}$.