# Synergy via Redundancy: Adaptive Replication Strategies and Fundamental Limits

Gauri Joshi, *Member, IEEE,* and Dhruva Kaushal, *Student Member, IEEE*

***Abstract*—The maximum possible throughput (or the rate of job completion) of a multi-server system is typically the sum of the service rates of individual servers. Recent work shows that launching multiple replicas of a job and canceling them as soon as one copy finishes can boost the throughput, especially when the service time distribution has high variability. This means that redundancy can, in fact, be used to create synergy among servers such that their overall throughput is greater than the sum of individual servers. This work seeks to find the fundamental limit of the throughput boost achieved by job replication and the optimal replication policy to achieve it. While most previous works consider upfront replication policies, we expand the set of possible policies to delayed launch of replicas. The search for the optimal adaptive replication policy can be formulated as a Markov Decision Process, using which we propose two myopic replication policies, MaxRate and AdaRep, to adaptively replicate jobs. In order to quantify the optimality gap of these and other policies, we derive upper bounds on the service capacity, which provide fundamental limits on the throughput of queueing systems with redundancy.**

## I. Introduction

THE emergence of cloud computing services allows users who rent servers from service providers such as Amazon, Microsoft, and Google to seamlessly scale up or scale down their computational resource usage as per user demand. In order to offer this scalability and flexibility at extremely low cost, cloud service providers employs large-scale sharing of resources. Each server is shared by multiple users as well as background processes in both time and computing bandwidth. Such resource sharing is not centrally coordinated but rather the result of several schedulers operating independently. An adverse effect of large-scale resource sharing in cloud computing systems is that the response time of individual servers can be large and unpredictable. This inherent variability in response time is the norm and not an exception [2]. A simple yet powerful solution to combat service time variability is to replicate computing jobs at multiple servers and wait for any one copy to finish. This idea was first used at a large-scale in MapReduce [3] and further developed in several other systems works including [4], [5]. A similar idea has been previously studied in [6] to route packets in networks and in [7] in the context of DNS queries.

Although job replication is used in practical systems, only a few theoretical works provide an understanding of when redundancy is most beneficial in reducing latency. Works such as [8]–[13] analyze multi-server queueing systems with redundancy. In these works, incoming jobs are replicated upfront and join queues at multiple servers simultaneously. As soon as any one replica finishes, all its copies are canceled immediately. Job replication affects response time (waiting time in queue plus service time) in two opposing ways:

G. Joshi (gaurij@andrew.cmu.edu) and D. Kaushal (dkaushal@alumni.cmu.edu) are with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh PA 15213

A short 3-page abstract of this paper appeared in the 2017 ACM Sigmetrics MAMA Workshop, and a 6-page version appeared in IFIP WG 7.3 Performance 2018 [1].

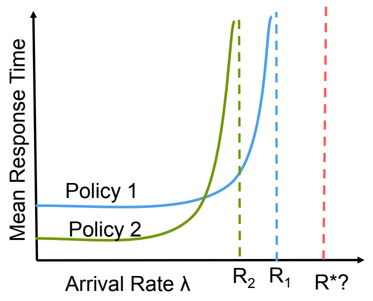


Fig. 1. Contrary to intuition, when service times have high variability, job replication can enable the maximum possible throughput to be greater than the sum of the service rates of the individual servers. Our goal is to find the fundamental limits of this throughput-boost and the replication policy that achieves it.

1) *Queue Diversity*: Replicas provide diversity by help finding the shortest among the queues that they join, thus reducing the overall waiting time. Unlike the join-the-shortest queue or power-of-choice scheduling policies which consider the queue lengths without accounting for the service time realizations of queued jobs, replication allow us to find queues that will be the first to become idle. This effect was studied in [12], [14].
2) *Load due to Redundant Service*: A downside of launching more replicas is that more than one replicas may enter service at different servers, potentially adding load to the system and increasing the waiting time for subsequent jobs. However, [8], [10], [15] identify surprising scenarios where replicating jobs (and canceling the copies as soon as any one finishes) can in fact reduce the system load, and result in the overall system throughput being higher than the sum of service rates of individual servers as illustrated in Fig. 1.

This paper seeks to dive deeper into the second factor, the effect of redundancy on the system load and throughput, and find replication policies that achieve the maximum achievable throughput, which is also referred to as the *service capacity*. To the best of our knowledge, this is the first paper to attempt finding the service capacity with adaptive (rather than just upfront) replication of jobs. Our system model accounts for server heterogeneity, job size variability as well as delays in cancellation of replicas.

### *A. Related Work*

**Job Dispatching Policies in Multi-server systems.** Job dispatching and scheduling policies for multi-server systems have been studied for many decades in queueing theory [16]. Traditionally metrics such as throughput and mean response time were studied in the context of operations and manufacturing systems. Queueing theory re-emerged in the early nineties as a rigorous way to design and analyze scheduling algorithms for computer systems [17]. However, *queues with redundancy were not considered in queueing theory until recently because in operations research and early computer systems, the variability in the service time was largely due to randomness in job sizes*. Thus, replicating jobs offered no benefit in terms of reducing latency and thus the maximum possible throughput or the service capacity of such multi-server systems was simply the sum of the service rates of its servers. There is a rich line of literature on designing and analyzing the mean response time of job dispatching policies such as join-the-shortest-queue, power-of-$d$ choices [18], least-work-left policy etc., proving their throughput-optimality and analyzing their mean response times. While the choice of the dispatch policy affects the mean response time, it does not change the maximum achievable throughput or the service capacity of the system.

**Redundancy to Overcome Delay Variability.** In the early 2000's, computing began to shift from local servers to the cloud, where computing resources are shared at a massive scale with limited central co-ordination. Although such loosely coordinated resource-sharing provides tremendous benefits in terms of cost, flexibility, and scalability, it causes *random fluctuations in the server response times*. This service time variability is often referred to as "tail latency". Due to tail latency, the same job can take vastly execution times at two different servers [2]. The adverse effect of tail latency is further magnified in jobs with many parallel tasks because the probability of at least one of the tasks being a straggler increases exponentially. To overcome stragglers, several heuristic redundancy approaches such as back-up tasks, clones or hedged requests [3], [19] began to be employed in computer systems. Although frequently used in systems, the addition of redundancy to overcome service time variability is a new paradigm in queueing theory with little understanding its fundamental limits.

**Fundamental Limits of the "Free Lunch" Offered by Redundancy.** Only a few theoretical works [8], [10], [12], [14], [20]–[24] have rigorously studied the latency of queues with redundancy and proposed redundancy scheduling policies. These works demonstrate that redundant jobs are extremely effective in finding the shorter queues in the system. However, intuition suggests that this benefit comes at the cost of additional load to system. This is because, when two or more replicas enter serve, they use additional and redundant computing time of the servers and cause subsequent jobs to wait longer in queue. Contrary to this intuition, recent works [1], [8], [10], [14], [25]–[27] identify regimes (when service times have high delay variability) where *redundancy can not only reduce overall latency but boost the throughput of queueing systems*. In [28] the authors analyze the increase in the service capapity (or the stability region) with different redundancy dispatch policies. [29] analyzes the service capacity for scaled Bernoulli service times whereas [30] analyzes the service capacity of processor sharing systems with heterogeneous service rates. However, these works only consider *upfront replication policies* where all the replicas at launched at the same time – delayed launch of replicas depending on the elapsed time of the original job has only been studied in [25] for parallel computing tasks without considering the effect of queueing of jobs. Understanding the fundamental limits of the "free lunch" offered by redundancy is a unique and unexplored problem in queueing theory. And designing optimal redundancy strategies to take full advantage of this free lunch is of critical importance since it can help boost the efficiency of data centers and reduce their energy consumption.

**Replication and Erasure Coding in Jobs with many parallel tasks.** Erasure codes, originally designed for error-correction and reliable transmission of information over a lossy communication channel, are a generalization of replication. Beyond their error-correction application, erasure codes can also be used to reduce delay and overcome stragglers in jobs with a large number of parallel tasks. For example, [9], [10] considered the problem of reducing the download time of a content file that is divided into $k$ chunks and coded into $n$ chunks using a maximum-distance-separable (MDS) code. Erasure coding allows us to recover the file from any $k$ out of $n$ chunks. Recently, erasure codes have also been shown to be effective in mitigating stragglers in parallel computing tasks such as matrix computations and distributed inference [31]–[35].

Analyzing the mean response time experienced by such jobs with $n$ parallel tasks where it is sufficient to complete any $k$ tasks is equivalent to an $(n, k)$ fork-join system. It is a generalization of the fork-join queueing system [36]–[38], which is a notoriously hard problem even for exponential service times. Papers such as [9], [10], [39]–[41] give bounds on the latency of the $(n, k)$ fork-join system while others such as [42] use mean-field analysis to compare replication and erasure coding. Instead of latency, in this paper, we focus on the maximum achievable throughput or the service capacity with job replication. Going beyond replication, characterizing the service capacity of erasure-coded storage and computing systems is an open future problem and has been considered in only a few recent works [26].

**Main Differences from Previous Works.** To summarize, the problem formulation of this paper differs from prior works on redundancy in queueing systems in the three key ways: 1) considering non-exponential service times for which the service capacity can potentially be increased using job replication, 2) instead of upfront replication, we consider gradual launch of additional replicas in order to preserve high throughput, and 3) the first attempt (to the best of our knowledge) to determine the service capacity of a multi-server system with job replication under these general conditions.

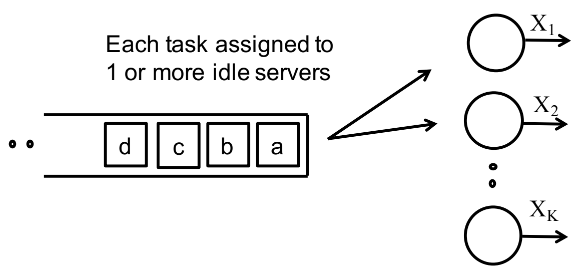


Fig. 2. System of $K$ servers where a job replicated at two idle servers 1 and 2 takes time $\min(X_1, X_2)$ to finish, where the random variable $X_i$ captures the service time variability.

## II. Problem Formulation

Consider a system of $K$ servers with a central queue containing jobs, illustrated in Fig. 2. These jobs are served in a first-come-first-served manner, and each job can be assigned to one or more idle servers. We do not explicitly define a job arrival process and instead assume that the central queue is never idle. Since our objective is to maximize the throughput, or the rate of job completion, there is no loss of generality in this assumption. Only in Section V-A, we consider Poisson job arrivals with rate $\lambda$ into the central queue – this is for the purpose of simuations that demonstrate the mean response times of the proposed replication policies in low or moderate traffic regimes.

### A. Job Service Times

The system consists of $K$ heterogeneous servers, where server $i$ takes time $X_i$ to finish a job assigned to it, where the probability distribution of $X_i \sim F_{X_i}$. The random variable $X_i$ captures the variability in job service time due to server slowdown, assumed to be i.i.d. across jobs assigned to that server. [1] We also consider that when a job is replicated, each server running it (including the server running the original copy of the job) reserves a cancellation window of length $\Delta$. As soon as one replica is served, the scheduler sends a cancellation signal to the other replicas, triggering their cancellation. All these events occur in time $\Delta$, after which the servers are available to serve subsequent jobs.

### B. Scheduling Policy

The policy $\pi$ used to schedule replicas can be based on the service time distributions of $X_1, \ldots, X_K$. The scheduler only knows these distributions, but does not know their realizations for currently running jobs. As soon as a server becomes idle, the scheduler can take one of two possible actions:

- ***new***: assign a new job to that server
- ***rep***: launch a replica of a job currently running on one of the other servers.

The space of scheduling policies with these actions is denoted by $\Pi_{n,r}$ and we aim to find the policy $\pi^*_{n,r}$ that maximizes the throughput. This space of policies can be expanded by allowing additional actions such as pausing a currently running job, or killing and relaunching it to another server. We only focus on the ***new*** and ***rep*** actions in this paper. Only in Section VI we use job pausing to find an upper bound on the service capacity.

Note that all job replication policies in $\Pi_{n,r}$ are work-conserving, that is, they do not allow any server to be idle for a non-zero time interval. Claim 1 below shows that there is no loss of generality in restricting our attention to work-conserving policies.

**Claim 1.** *The throughput-optimal scheduling policy $\pi^*$ is work-conserving, that is, it does not allow any server to be idle for a non-zero time interval.*

The proof is given in the Appendix.

### C. Performance Metrics

Let us formally define the throughput of policy $\pi$.

**Definition 1** (Throughput $R$). *Let $T_1(\pi) \leq T_2(\pi) \leq \cdots \leq T_n(\pi)$ be the departure times of jobs $1, 2, \ldots n$ from the system, when the scheduler follows a policy $\pi$. Then the throughput is defined as*

$$R(\pi) \triangleq \lim_{n \to \infty} \frac{n}{T_n(\pi)}. \tag{1}$$

**Definition 2** (Service Capacity $R^*_{n,r}$). *The service capacity $R^*_{n,r} = \max_{\pi \in \Pi_{n,r}} R(\pi)$, the maximum achievable throughput over all scheduling policies in $\Pi_{n,r}$. The policy $\pi^*_{n,r}$ that achieves $R^*_{n,r}$ is called the throughput-optimal policy.*

An alternate interpretation of $R(\pi)$ is that if jobs are arriving in the central queue at rate $\lambda$, then if $\lambda < R(\pi)$ the system is stable, that is, the mean response time (waiting time plus service time) experienced by jobs is finite. Thus by using the throughput-optimal policy $\pi$ that maximizes $R(\pi)$, we can support the maximum possible job arrival rate.

Next we define another performance metric, the computing time $C$ per job. In we will show how throughput $R$ can be expressed in terms of $\mathbb{E}[C]$.

**Definition 3** (Computing Time $C$). *The computing time $C$ is the total time collectively spent by the servers per job.*

The expected computing time $\mathbb{E}[C]$ is proportional to the cost of running a job on a system of servers, for instance, servers rented from Amazon Web Services (AWS), which are charged by the hour. In our system model, if a job is assigned to only to server $i$ then $\mathbb{E}[C] = \mathbb{E}[X_i]$. Instead, if it is assigned to two servers $i$ and $j$, and the replica is canceled when any one copy finishes then $\mathbb{E}[C] = 2(\mathbb{E}[\min(X_i, X_j)] + \Delta)$ where $\Delta$ is the cancellation window at each of the servers. Depending upon $X_i$ and $\Delta$, $\mathbb{E}[C]$ with replication may be greater or less than that without replication.

[1]We assume that the service time variability comes only from the server and not the size of the job. However, it is possible to account for job size variability via another random variable $Y$, which is independent of $X_i$ for all $i$. This method of multiplying the randomness from the two sources of variability was introduced in [21]. The value of $Y$ is same across replicas of a job. Thus, if a job is replicated at two idle servers $i$ and $j$, the time taken to complete any one replica is $Y \cdot \min(X_i, X_j)$. For simplicity and brevity we assume $Y = 1$ (deterministic job size) in this paper, but most of the results can be extended to random $Y$ by adding a scaling factor $\mathbb{E}[Y]$ multiplying $\mathbb{E}[\min(X_i, X_j)]$.

**Claim 2.** *For any work-conserving scheduling policy,*

$$R = \frac{K}{\mathbb{E}[C]}. \tag{2}$$

*Proof.* Consider jobs $1, 2, \ldots n$ run on the system of servers. If the scheduling policy is work-conserving, the total busy time of each server is exactly equal to $T_n$, the departure time of the last job. Since $\mathbb{E}[C]$ is defined as the total expected time spent at servers per job, by law of large numbers we have

$$\mathbb{E}[C] = \lim_{n\to\infty} \frac{KT_n}{n} = \frac{K}{R}, \tag{3}$$

where the second equality follows from Definition 1. $\square$

Thus, minimizing $\mathbb{E}[C]$ is equivalent to maximizing $R$.

## D. Main Contributions and Organization

To illustrate the main contributions and organization of this paper, let us consider some replication policies for a simple two-server example. The upcoming sections will develop each of the replication policies considered in this example in greater detail and rigor.

**Example 1.** Consider a sytem of two servers with service time distributions

$$X_1 = 2 \tag{4}$$

$$X_2 = \begin{cases} 1 & \text{w.p.} \quad (1-p) = 0.9 \\ 20 & \text{w.p.} \quad p = 0.1 \end{cases} \tag{5}$$

The cancellation delay $\Delta = 0$. The throughput or the rate of job completion with full replication and no replication respectively are

$$R_{NoRep} = \frac{1}{\mathbb{E}[X_1]} + \frac{1}{\mathbb{E}[X_2]} = 0.8448, \tag{6}$$

$$R_{FullRep} = \frac{1}{\mathbb{E}[\min(X_1, X_2)]} = 0.909. \tag{7}$$

In Section III we analyze more general 'upfront' replication policies that launch $r < K$ replicas of a job at the same time and cancel the outstanding replicas as soon as any one copy is served. An alternative to upfront replication is to add replicas gradually, only if the original copy of the job does not finish in reasonable time. One such policy is the adaptive replication (AdaRep) policy $\pi_{AdaRep}$, which launches a replica of a job assigned to server 2 only if it has spent more than 1 second in service. To evaluate the throughput of this policy, we consider time instants called renewals when both servers become idle. There are three types of intervals between successive renewal instants as illustrated in Fig. 3. The throughput is the expected number of jobs completed in an interval, divided by the expected interval length.

$$R_{AdaRep} = \frac{\sum_{i=1}^{3} \Pr(\text{Type i interval}) \cdot (\#\text{jobs completed})}{\text{Expected length of a renewal interval}} \tag{8}$$

$$= \frac{0.9 \times 0.9 \times 3 + 0.9 \times 0.1 \times 3 + 0.1 \times 2}{0.9 \times 0.9 \times 2 + 0.9 \times 0.1 \times 4 + 0.1 \times 4} \tag{9}$$

$$\approx 1.2185, \tag{10}$$

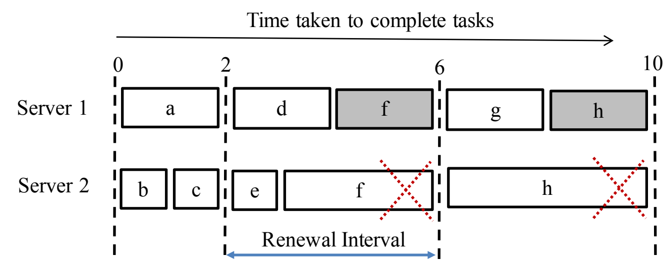


Fig. 3. Illustration of renewal instants of the system of 2 servers with the adaptive replication policy ($\pi_{AdaRep}$) described in Section II-D.

which clearly outperforms the two extreme policies.

For systems with more general service distributions, finding the optimal adaptive policy involves solving a Markov Decision Process (MDP). We formulate this MDP in Section IV. This MDP can have a large state space and we need to resort to myopic policies. We propose two such policies – MaxRate and AdaRep in Section V. The MaxRate policy which launches replicas so as to maximize the instantaneous job departure rate from the system. The AdaRep policy that launches replicas when the elapsed time currently running copies crosses a pre-specified threshold. In order to quantify the optimality gap of these policies, in Section VI we obtain upper bounds on the service capacity for the two heterogeneous servers case, and a more general upper bound for $K$ homogeneous servers. Finally Section VII presents major implications and future directions.

## III. Upfront Replication

In this section we explore 'upfront' replication policies that simultaneously launch a job and its replicas. The number of replicas and the servers where they are launched governs the overall throughput.

## A. No Replication and Full Replication

First let us compare the throughput achieved by two extreme policies: no replication and full replication. This analysis demonstrates how replication can create synergy and boost the throughput of a server cluster.

**Lemma 1** (Throughput with No Replication)**.** *If each job is assigned to the first available idle server in a system of $K$ servers, the throughput is,*

$$R_{NoRep} = \sum_{i=1}^{K} \frac{1}{\mathbb{E}[X_i]} \tag{11}$$

*Proof.* This policy is work-conserving and thus keeps all servers busy all the time. Thus, if we look at server $i$, the departure time of the $n^{th}$ job assigned to that server is $T_n^{(i)}$ is the sum of $n$ i.i.d. realizations of the service time $X_i$. Thus, the rate of departure of jobs from server $i$ is,

$$R_i = \lim_{n\to\infty} \frac{n}{T_n^{(i)}} = \frac{1}{\mathbb{E}[X_i]}. \tag{12}$$

Adding the rates of departure from all the servers yields overall throughput as given by (11). $\square$

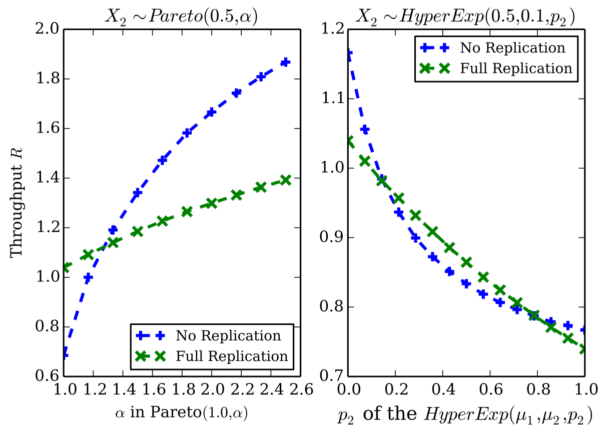


Fig. 4. Comparison of the no replication and full replication policies for $X_1 \sim 0.5 + Exp(1)$ and different $X_2$. When $X_2 \sim Pareto(0.5, \alpha)$, full replication is better for smaller $\alpha$ (heavier tail). When $X_2 \sim HyperExp(\mu_1 = 0.5, \mu_2 = 0.1, p_2)$, full replication is better for intermediate $p_2$.

**Lemma 2** (Throughput with Full Replication)**.** *Suppose each job is assigned to all servers, and as soon as one replica finishes, the others are canceled. The throughput achieved by this full replication policy is,*

$$R_{FullRep} = \frac{1}{\Delta + \mathbb{E}\left[\min(X_1, X_2, \dots X_K)\right]} \tag{13}$$

*Proof.* With the full replication policy, all $K$ servers are working on the same job at any time instant. The total time spent by them on each job is,

$$\mathbb{E}\left[C\right] = K(\Delta + \mathbb{E}\left[\min(X_1, X_2, \dots X_K)\right]) \tag{14}$$

Then (13) follows from the result in Claim 2. □

Using Lemma 1 and Lemma 2 we can compare the two policies for any given distributions $X_1, \dots, X_K$ and cancellation delay $\Delta$. In Fig. 4 we show a comparison of full replication and no replication for the two server case, with $\Delta = 0$. In both subplots, the service time $X_1 \sim 0.5 + Exp(1)$, a shifted exponential. We observe that full replication gives higher throughput when $X_2$ has higher variability. In the left subplot, $X_2 \sim Pareto(0.5, \alpha)$ and replication is better for smaller $\alpha$ (heavier tail). In the right subplot, $X_2$ is a hyper-exponential $HyperExp(\mu_1, \mu_2, p_2)$, that is, it is an exponential with rate $\mu_2$ with probability $p_2$ and otherwise it is exponential with rate $\mu_1$. In this case, replication is better for intermediate $p_2$ where $X_2$ has higher variability.

## B. General Upfront Replication

Instead of replicating job at all servers, or not replicating at all we can replicate jobs at a subset of the servers. Each subset is treated as a 'super-server' such that jobs are replicated at all servers in a super-server. We refer to this class of policies as upfront replication policies, defined formally below.

**Definition 4** (Upfront Replication)**.** *For $h \in \mathcal{N}$, consider a partition of set $[K] = 1, 2, 3, \dots K$. The partition is a collection of non-empty subsets $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_h$ of $[K]$, such that $\mathcal{S}_i \cap \mathcal{S}_j = 0$, and $\cup_j S_j = [K]$. When the servers in a set $\mathcal{S}_j$ become idle (they will always become idle simultaneously), assign the next job in the central queue to them.*

The no replication policy is a special case with $\mathcal{S}_j = j$ for all $j \in [K]$. Full replication is also a special case with $\mathcal{S}_1 = [K]$.

**Theorem 1** (Throughput with Upfront Replication)**.** *The throughput $R_{UpFr}$ achieved by upfront replication at server sets $\mathcal{S}_1, \dots, \mathcal{S}_h$ is*

$$R_{UpFr}(\mathcal{S}_1, \dots, \mathcal{S}_h) = \sum_{j=1}^{h} \frac{1}{\mathbb{E}\left[X_{\mathcal{S}_j}\right] + \Delta}, \tag{15}$$

$$\text{where } X_{\mathcal{S}_j} = \min_{l \in \mathcal{S}_j} X_l \tag{16}$$

The proof is given in the Appendix. To maximize the throughput, we need to find the partition $\{\mathcal{S}_1, \dots, \mathcal{S}_h\}$ that maximizes (41). The number of possible partitions of a set of size $K$ is given by the Bell number $B_k$. It can be computed using the recursion

$$B_K = \sum_{i=0}^{K-1} \binom{K-1}{i} B_i, \tag{17}$$

with base $B_0 = 1$. This number is exponential in $K$. Thus, when the number of servers $K$, searching over all possible partitions to find the partition that maximizes the throughput can be computationally intractable.

However, most practical multi-server systems consist of only a few types of servers, such that servers of the same type have the same service time distribution. Finding the best partition of the servers can be tractable in such systems. For example, for $K$ homogeneous servers with service time distribution $F_X$, the throughput of the optimal upfront replication policy is given by the following result.

**Theorem 2** (Upfront Replication Throughput Bound for $K$ Homogeneous Servers)**.** *For a system of $K$ homogeneous servers with i.i.d. service times $X \sim F_X$, let $r^*$ be the positive integer that minimizes $r(\mathbb{E}\left[X_{1:r}\right] + \Delta)$. The throughput achieved with upfront replication of jobs satisfies*

$$R_{UpFr} \leq \frac{K}{r^*(\mathbb{E}\left[X_{1:r}\right] + \Delta)}. \tag{18}$$

*Equality holds in (18) if $r^*$ divides the number of servers $K$.*

The proof is given in the Appendix. For $\Delta = 0$, $r^*$ is the $r$ that minimizes $r\mathbb{E}\left[X_{1:r}\right]$. Fig. 5 illustrates the throughput of a system of $K$ servers, which is equal to $K\mathbb{E}\left[X\right]/r\mathbb{E}\left[X_{1:r}\right]$ versus $r$ for four different service distributions: shifted exponential $0.1 + Exp(1.0)$, hyper-exponential $HyperExp(0.6, 0.2, 0.4)$, shifted hyper-exponential $0.1 + HyperExp(1.0, 0.2, 0.4)$, and Pareto $Pareto(0.5, 1.2)$. When the tail distribution $\Pr(X > x)$ of $X$ is log-concave (for example shifted-exponential), the optimal $r$ is $r = 1$, whereas for log-convex $X$ (for example hyper-exponential), $r^* = K$ is optimal. This property of log-concave (log-convex) distributions was proved in [14]. For other distributions such as shifted hyperexponential or Pareto, intermediate $r$ can be optimal.

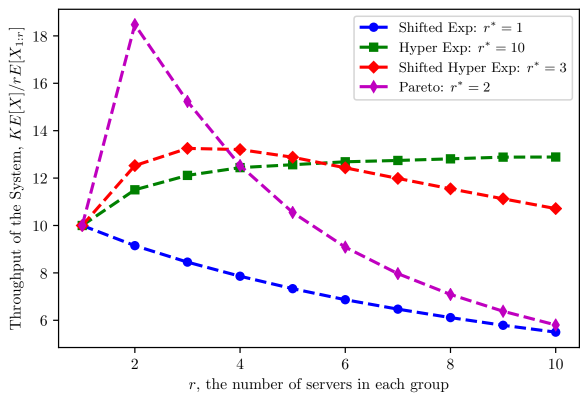


Fig. 5. The throughput of a system of $K = 10$ servers versus the number of replicas $r$ per job for different service distributions, with $K = 10$. When $X$ is shifted-exponential (log-concave), the optimal $r$ is $r = 1$, whereas for hyper-exponential $X$, $r^* = K$ is optimal.

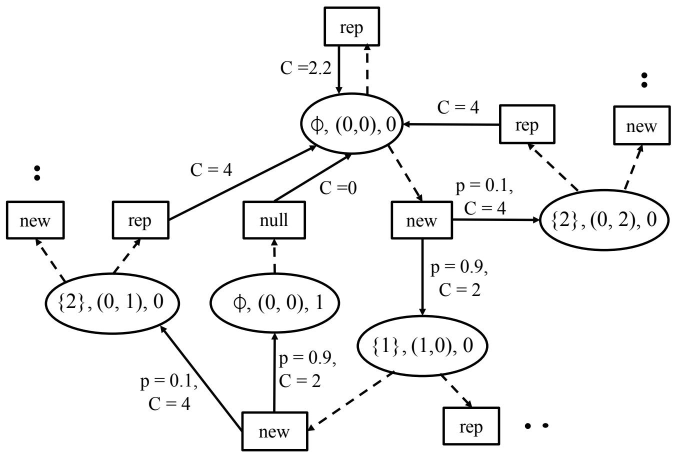


Fig. 6. Illustration of the MDP for the service distributions in Section II-D. Dotted arrows correspond to the actions taken from a state and solid arrows lead to the new state resulting from the action. Parts of the MDP resulting from sub-optimal actions are omitted in this figure.

## IV. MDP Formulation of the Throughput-Optimal Replication Policy

Instead of launching replicas upfront, they could be added conditionally, only if the original job does not finish in some given time. We propose a Markov Decision Process (MDP) framework to search for the throughput-optimal policy the achieves service capacity. Finding the optimal policy directly using this framework is an intractable problem, but it provides valuable insights into the design of myopic policies in Section V. We describe the state-space, actions, and cost per transition below. Observe that state-space and actions satisfy the Markov property, that is, the transition from state $s$ to $s'$ only depends on the action $\pi(s)$, and is conditionally independent of all previous states and actions.

### A. State-space

We denote the state evolution by $s_0, s_1, \dots s_i, \dots$ such that the system transitions to state $s_i$ as soon as the $i^{th}$ job departs. The state-space can be collapsed into states $[\mathcal{B}, \mathbf{t}, D_r]$ where $\mathcal{B}$ contains disjoint sets of server indices that are running the unfinished jobs in the system. For example, if $\mathcal{B} = \{\{1\}, \{2, 3\}\}$ there are two unfinished jobs in the system, one running on server 1 and another on servers 2 and 3. The vector $\mathbf{t} = (t_1, t_2, \dots t_K)$ where $t_k$ is the time spent by server $k$ on its current job. Since we observe the system immediately after a job departure, at least one of the elapsed times $t_1, t_2, \dots t_K$ is zero. The purpose of the $D_r$ term is to ensure that each state transition corresponds to a single job departure. It is the number of jobs that have finished, but are still to depart. If $h > 1$ jobs exit the system simultaneously and result in the job assignment set $\mathcal{B}$ and elapsed-time vector $\mathbf{t}$, then the system goes through states $[\mathcal{B}, \mathbf{t}, h-1] \rightarrow [\mathcal{B}, \mathbf{t}, h-2] \rightarrow \dots \rightarrow [\mathcal{B}, \mathbf{t}, 0]$.

### B. Actions

In each state $s$, denote the set of possible actions is $\mathcal{A}_s$. The scheduling policy $\pi$ determines the action $a = \pi(s)$ that is taken from state $s$. First note that no jobs are assigned in the exit states $s = [\mathcal{B}, \mathbf{t}, D_r]$ with $D_r > 0$. Thus, for these states, the action space $\mathcal{A}_s$ contains a single placeholder ***null*** action. The system directly transitions to $[\mathcal{B}, \mathbf{t}, D_r - 1]$.

In states $s = [\mathcal{B}, \mathbf{t}, 0]$, the scheduler can assign new jobs to idle servers (***new***), or replicate existing jobs (***rep***). For example, consider a system of 2 servers (illustrated in Fig. 6 for the service time distributions in Example 1). In states $[\{2\}, (0, t), 0]$ or $[\{1\}, (t, 0), 0]$ with $t > 0$, one server is idle while the other has spent $t$ seconds on its current job. From the state $s = [\emptyset, (0, 0), 0]$ where both servers are idle, the ***new*** action assigns two new jobs, one to each server, and the ***rep*** action replicates a new job at both servers.

### C. Cost

The cost $C(s, s', a)$ associated with a transition from state $s$ to $s'$ when action $a$ is taken in state $s$ is defined as the total time spent by the servers in that interval. Thus, the throughput-optimal policy $\pi^*_{n,r}$ is the solution to the following cost minimization problem,

$$\pi^*_{n,r} = \arg \min_{\pi \in \Pi_{n,r}} \sum_{j=0}^{\infty} C(s_j, s_{j+1}, \pi(s_j)). \tag{19}$$

As illustrated in Fig. 6, we observe that this MDP can have a large state-space even for simple service distributions. And more generally, if $X_i$ for any $i$ is a continuous random variable for which the MDP will have a continuous state-space, which becomes even harder to solve.

## V. The MaxRate and AdaRep Replication Policies

As an alternative to solving the MDP, we propose two replication policies, MaxRate and AdaRep. The MaxRate policy is a greedy myopic policy directly based on the MDP and it is defined as follows.

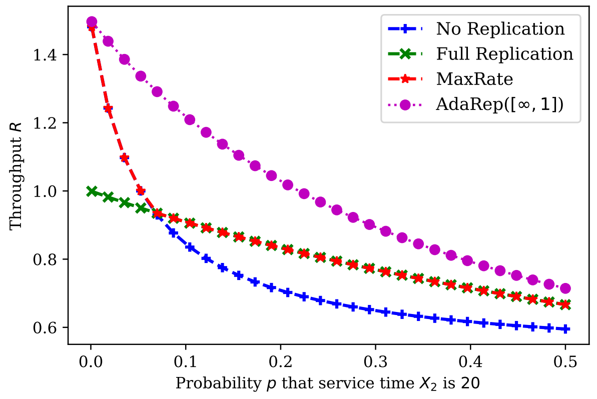


Fig. 7. For the service distributions in Example 1 with different values of $p$, the throughput with the MaxRate policy is a maximum of the throughputs with the FullRep and NoRep policies. The AdaRep policy with a replication threshold of 1 for jobs originally launched on server 2 achieves the best throughput.

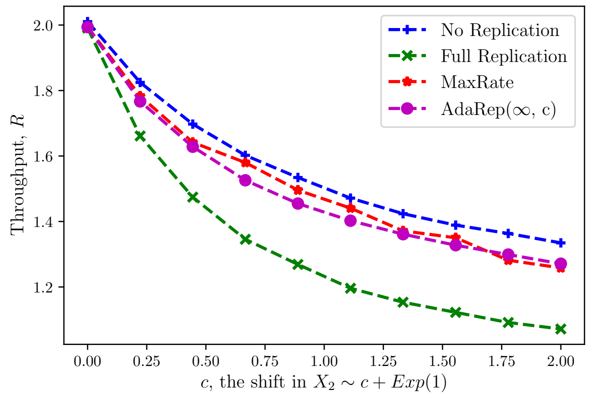


Fig. 8. For the two-server case with service time distributions $X_1 \sim Exp(1)$ and $X_2 \sim c + Exp(1)$, we plot the throughputs of MaxRate, AdaRep($[t_{1\to2}, t_{2\to1}] = [\infty, c]$), Full Replication and No replication poilicies versus the initial delay $c$ in the service time $X_2 \sim c + Exp(1)$. The MaxRate and AdaRep policies are close to the no replication policy which yields the best throughput.

**Definition 5** (MaxRate Policy). *From state $s$, the MaxRate policy chooses the action $a^*$ that maximizes the instantaneous service rate $\hat{R}(a)$ which is defined as,*

$$\hat{R}(a) \triangleq \sum_{m=1}^{M(a)} \frac{1}{\mathbb{E}[D_m(a)]}. \tag{20}$$

*where $M(a)$ is the number of unfinished jobs after taking action $a$, and $\mathbb{E}[D_m]$ is the expected remaining time until the departure of job $j$, assuming it is not replicated further.*

**Corollary 1.** *Consider a two server system, with cancellation delay $\Delta = 0$. Suppose server 1 becomes idle, and the job assigned to server 2 has spent time $t_2 > 0$ in service. Let $X_2^{rs} = (X_2 - t_2)|X_2 > t_2$ be the residual computing time. The MaxRate policy launches a replica at server 1 if*

$$\frac{1}{\mathbb{E}[\min(X_1, X_2^{rs})]} > \frac{1}{\mathbb{E}[X_1]} + \frac{1}{\mathbb{E}[X_2^{rs}]}. \tag{21}$$

*and otherwise it assigns a new job to server 1.*

The MaxRate policy implicits finds replication thresholds $t_{i\to j}$ such that a job running on server $i$ is replicated at server $j$ if it does not finish in $t_{i\to j}$ seconds. Based on this idea we propose another class of policies called AdaRep($\mathbf{t}$), which is explicitly parametrized by a replication threshold vector $\mathbf{t}$.

**Definition 6** (AdaRep Policy). *Consider a vector of server indices $\mathbf{u} = (j_1, j_2, \dots j_k)$ for $k < K$ such that a job first launched on server $j_1$ was later replicated on $j_2$, $j_3$ and so on. This job is replicated at server $i$ if the job has spent at least $t_{\mathbf{u}\to i}$ time in service from the time its original copy was launched. Otherwise it assigns a new job to the idle server.*

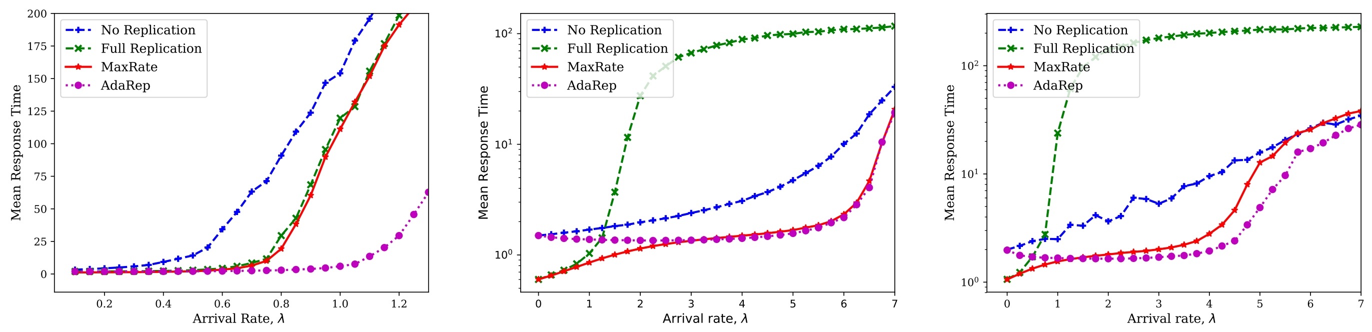
For example for $K = 2$ servers, the vector $\mathbf{t} = [t_{1\to2}, t_{2\to1}]$. In general, choosing the best replication thresholds is a non-trivial problem. In the next section we propose a method to choose $\mathbf{t}$ for the two-server case.

Fig. 7 illustrates the MaxRate and AdaRep policies in comparison with the FullRep and NoRep policies for the service distributions in Example 1. Observe that the throughput of the MaxRate policy is the maximum of the throughputs of the NoRep and FullRep policies. The AdaRep($[\infty, 1]$) policy proposed in Example 1 gives a throughput $R = 1.2185$ when $p = 0.1$, which is significantly higher than $R = 0.909$ achieved by the greedy myopic MaxRate policy. However, unlike MaxRate where the replication thresholds are implicitly found by maximizing the instantaneous job departure rate, the throughput of the AdaRep policy is highly sensitive to the choice of the replication thresholds.

Fig. 8 illustrates the MaxRate and AdaRep policies in comparison with the FullRep and NoRep policies for two servers with exponential $X_1 \sim Exp(1)$ and shifted exponential $X_2 \sim c + Exp(1)$ service time distributions (for a constant $c \geq 0$) respectively. Due to the memoryless property of the exponential distribution, when $c = 0$, the no replication and full replication policies give the same throughput. When $c > 0$, full replication gives strictly lower throughput than no replication. Observe that the MaxRate policies tries to dynamically emulate NoRep. AdaRep with a replication threshold $t_{2\to1} = c$ for jobs originally launched in server 2 gives lower throughput than NoRep because the optimal replication thresholds are $[t_{1\to2}, t_{2\to1}] = [\infty, \infty]$ in this case.

## A. Mean Response Time in Low and Moderate Traffic Regimes

Although the focus of this paper is to find throughput-optimal replication policies, the MaxRate and AdaRep policies proposed above work very well in the low and moderate traffic regimes as we show via simulations below. The simulation setting is as follows. We consider Poisson job arrivals with rate $\lambda$ into the central queue shown in Fig. 2 instead of assuming that the queue is saturated with jobs. Unlike the saturated central queue case considered so far, where a server is assigned a new job (or a replica of an existing job) as soon as it becomes idle, servers now remain idle when there are no jobs in the central queue. We then record the mean response time (waiting



(a) $K = 2$ servers with service distributions as given in Example 1 with $p = 0.1$. The replication thresholds used in the AdaRep policy are $(t_{1\to 2}, t_{2\to 1}) = (\infty, 1)$.

(b) $K = 10$ homogeneous servers with shifted exponential service times $X \sim 0.5 + Exp(1)$. The AdaRep replication thresholds for launching additional replicas of a job are $(0.1, 0.2, \ldots, 0.9)$, that is, we launch the $i^{th}$ additional replica if the original copy of the job spends at least $0.1 \times i$ units of time in service.

(c) $K = 10$ homogeneous servers with Pareto $X \sim Pareto(1, 2)$. The AdaRep replication thresholds for launching additional replicas of a job are $(0.1, 0.2, \ldots, 0.9)$, that is, we launch the $i^{th}$ additional replica if the original copy of the job spends at least $0.1 \times i$ units of time in service.

Fig. 9. Mean response times versus arrival rate $\lambda$ for the No Replication, Full Replication, MaxRate and AdaRep policies.

time in queue plus service time) experienced by jobs averaged over 100 simulation runs with 1000 jobs each.

Fig. 9(a) shows a comparison of the mean response time experienced by jobs for the No Replication, Full Replication, MaxRate and AdaRep($[\infty, 1]$) policies for the two server case with service distributions as given in Example 1 with $p = 0.1$. The cancellation delay is assumed to be $\Delta = 0$. In addition to boosting the throughput in high-traffic, MaxRate and AdaRep also result in faster response times in the low traffic regime. As $\lambda$ increases, observe that MaxRate transitions from full replication to no replication. The AdaRep policy gives the lowest response time in all traffic regimes.

Fig. 9(b) shows a comparison of the mean response time experienced by jobs for the No Replication, Full Replication, MaxRate and AdaRep policies for $K = 10$ homogeneous servers with shifted exponential service times $X \sim 0.5 + Exp(1)$. The AdaRep replication thresholds for launching additional replicas of a job are $(0.1, 0.2, \ldots, 0.9)$, that is, we launch the $i^{th}$ additional replica if the original copy of the job spends at least $0.1 \times i$ units of time in service. The cancellation delay is assumed to be $\Delta = 0$. In addition to boosting the throughput in high-traffic, MaxRate and AdaRep also result in faster response times in the low traffic regime. In very low traffic, MaxRate launches replicas at all the idle servers in order to greedily maximize the instantaneous job departure rate, and thus, its throughput resembles that of the full replication policy. As $\lambda$ increases, both MaxRate and AdaRep replicate fewer times and come closer to no replication, which is throughput-optimal in heavy traffic. A similar trend is observed in Fig. 9(c) for $K = 10$ homogeneous servers with Pareto service times $X \sim Pareto(1, 2)$. Here, the AdaRep policy performs better than MaxRate in the moderate traffic regime.

## VI. Upper Bounds on the Service Capacity

In order to quantify the optimality gap of the MaxRate and AdaRep policies proposed above and understand the limits of the service capacity of a multi-server system with job replication, we now provide two fundamental throughput upper bounds. The first bound is for a system of $K = 2$ heterogeneous servers, and the second is for a system of $K > 2$ homogeneous servers. The derivations of these bounds use two different techniques to construct a genie system whose throughput is always better than the system under consideration.

### A. Upper Bound for Two Heterogeneous Servers

Recall that in our problem formulation, jobs can be replicated only at time instants when one or more servers become idle. To find the upper bound on $R^*_{n,r}$, we consider a system where the scheduler is also allowed to pause ongoing jobs.

**Definition 7** (The Pause-and-Replicate System)**.** *A job can be replicated at any server where it is not already running by pausing the ongoing job on that server. The paused job is resumed when the replica is either served or canceled.*

For the example shown in Fig. 3, the pause-and-replicate system can pause job $g$ at time 7 to run a replica of job $h$, and resume job $g$ afterwards. Both $g$ and $h$ will then finish at time 9, which is 1 second faster than with the AdaRep policy without job pausing.

**Claim 3.** *The service capacity $R^*_{p,r}$ of the pause-and-replicate system is an upper bound on the service capacity $R^*_{n,r}$ of the original system.*

*Proof.* The set of feasible policies $\Pi_{n,r}$ is a subset of $\Pi_{p,r}$, the set of policies in the pause-and-replicate framework. Thus,

$$R^*_{p,r} = \max_{\pi \in \Pi_{p,r}} R(\pi) \geq \max_{\pi \in \Pi_{n,r}} R(\pi) = R^*_{n,r}. \tag{22}$$

□

In the pause-and-replicate framework, the AdaRep($\mathbf{t}$) policy can replicate a job exactly after $t_{\mathbf{u}\to i}$, instead of waiting for server $i$ to become idle. In Theorem 3 below, we obtain

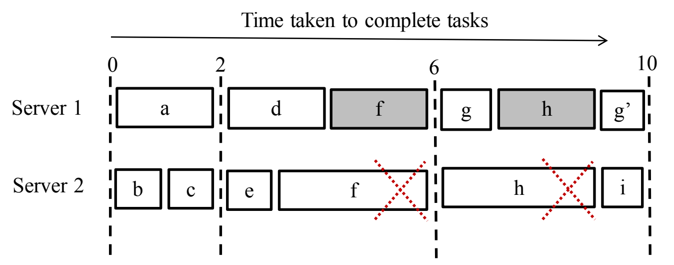


Fig. 10. Illustration of the optimal replication policy in the pause-and-replicate framework. The service distributions are as given in Example 1. Due to the ability to pause jobs this policy completes more jobs than the AdaRep policy in Fig. 3.

a closed-form expression for the throughput $R_{p,r}(\mathbf{t})$ of the AdaRep policy for $K = 2$ servers. We show that there is no loss of generality in focusing on AdaRep policies. Thus, in the two-server case, the converse bound $R^*_{p,r} = R_{p,r}(\mathbf{t}^*)$, the throughput of the best AdaRep policy.

**Theorem 3.** *The throughput $R_{p,r}(\mathbf{t})$ of AdaRep($\mathbf{t} = [t_{1\to 2},\, t_{2\to 1}]$) in the pause-and-replicate framework can be expressed as follows. For $t_{1\to 2} > 0$ and $t_{2\to 1} > 0$,*

$$R_{p,r}(\mathbf{t}) = \frac{\mathbb{E}\left[X_1^{tr}(t_{1\to 2})\right] + \mathbb{E}\left[X_2^{tr}(t_{2\to 1})\right]}{\mathbb{E}\left[X_1^{tr}(t_{1\to 2})\right]\mathbb{E}\left[X_2^{tr}(t_{2\to 1})\right](1 + \gamma_{1\to 2} + \gamma_{2\to 1})} \tag{23}$$

*where,*

$$\gamma_{t_{1\to 2}} \triangleq \frac{\Pr(X_1 > t_{1\to 2})(\Delta + \mathbb{E}\left[\min(X_1^{rs}(t_{1\to 2}), X_2)\right])}{\mathbb{E}\left[X_1^{tr}(t_{1\to 2})\right]} \tag{24}$$

$$\gamma_{t_{2\to 1}} \triangleq \frac{\Pr(X_2 > t_{2\to 1})(\Delta + \mathbb{E}\left[\min(X_1, X_2^{rs}(t_{2\to 1}))\right])}{\mathbb{E}\left[X_2^{tr}(t_{2\to 1})\right]}, \tag{25}$$

*and $X_i^{tr}(\tau) = \min(X_i, \tau)$, the truncated part of $X_i$, and $X_i^{rs}(\tau) = (X_i|(X_i > \tau) - \tau)$, the residual service time after $\tau$ seconds of service.*

*If $t_{1\to 2} = 0$ or $t_{2\to 1} = 0$,*

$$R_{p,r}(\mathbf{t}) = \frac{1}{\Delta + \mathbb{E}\left[\min(X_1, X_2)\right]}. \tag{26}$$

The proof is given in the Appendix. In Corollary 2 below we give the throughput expression for the special case where $t_{1\to 2}$ set to infinity.

**Corollary 2.** *The throughput $R_{p,r}(\mathbf{t} = [\infty,\, t_{2\to 1}])$ of the two-server pause-and-replicate system is*

$$R_{p,r}(t_{2\to 1}) = \frac{\mathbb{E}\left[X_2^{tr}\right]}{\mathbb{E}\left[X_2^{ac}\right]}\left(\frac{1}{\mathbb{E}\left[X_1\right]}\right) + \frac{1}{\mathbb{E}\left[X_2^{ac}\right]} \tag{27}$$

*where, $\mathbb{E}\left[X_2^{tr}\right] = \min(X_2, t_{2\to 1})$, is the truncated part of $X_2$, and $\mathbb{E}\left[X_2^{ac}\right]$ is the effective service time of jobs launched on server 2.*

$$\mathbb{E}\left[X_2^{ac}\right] = \mathbb{E}\left[X_2^{tr}\right] + \Pr(X_2 > t_{2\to 1})(\Delta + \mathbb{E}\left[\min(X_1, X_2^{rs}\right]), \tag{28}$$

*where $X_2^{rs} = (X_2|(X_2 > t_{2\to 1}) - t_{2\to 1})$, the residual service time after time $t_{2\to 1}$ of service.*

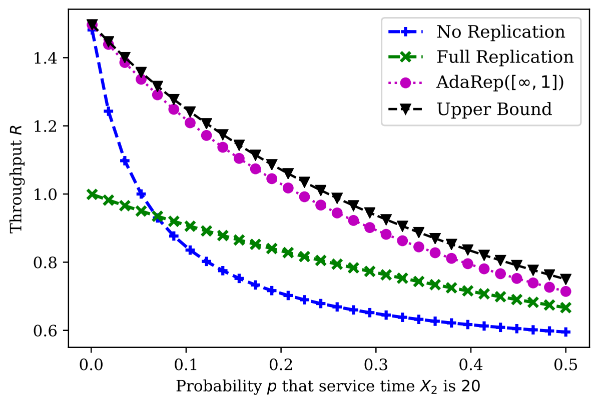


Fig. 11. Illustration of the upper bound on the service capacity $R^*_{n,r}$, along with the NoRep, FullRep and AdaRep policies. The service distributions are as defined in Example 1, with $p$ varying along the x-axis. The AdaRep policy with carefully chosen replication thresholds gives throughput that is close to the upper bound.

Here is an intuitive explanation of the throughput in (27). Since server 2 is never paused, its throughput of server 2 is $1/\mathbb{E}\left[X_2^{ac}\right]$, where $\mathbb{E}\left[X_2^{ac}\right]$ accounts for the reduction in service time due to replication of jobs. For server 1, the throughput is $\zeta/\mathbb{E}\left[X_1\right]$, where $\zeta = \mathbb{E}\left[X_2^{tr}\right]/\mathbb{E}\left[X_2^{ac}\right]$, the fraction of time server 1 is not paused.

To maximize the throughut we can find $t_{2\to 1}$ that maximizes (27). For example, for the service distributions in Example 1, $t^*_{2\to 1} = 1$. Thus, if a job does not finish in 1 seconds on server 2, we launch a replica on server 1 by pausing its ongoing job. This policy is illustrated in Fig. 10.

**Lemma 3.** *For $K = 2$ servers, there is no loss of generality in focusing on AdaRep policies to find the optimal throughput $R^*_{p,r}$ in the pause-and-replicate framework. That is, $R^*_{p,r} = \max_{\mathbf{t}} R_{p,r}(\mathbf{t})$.*

The proof is given in the appendix. Based on this throughput upper bound, we propose an adaptive replication policy $\pi_{AdaRep}$. This policy tries to emulate the best AdaRep thresholds $\mathbf{t}^*$, under the limitation that it cannot pause ongoing jobs to launch replicas. Recall that we already saw this policy in Example 1. Now we define it more generally for any service distributions. In Fig. 11 we plot the throughput achieved by the AdaRep policy, alongwise the upper bound given by maximizing (27) over $t_{2\to 1}$. The service distributions are same as Example 1 with $p$ varying along the x-axis. We observe that the AdaRep policy comes closest to the upper bound, with the gap resulting from its inability to pause jobs.

Generalizing this pause-and-replicate system based bounding technique to more than two servers is a difficult and non-trivial problem. This is because there can be deadlock situations where a job being run on a server $a$ cannot be replicated at server $b$ because server $c$ had already paused server $b$ to replicate its own current job. This makes it difficult to find a closed-form expression of the throughput as we did in the two server case in Theorem 3. Below, we present a different upper bounding technique for the case of $K$ homogeneous servers.

## B. Upper Bound for $K$ homogeneous servers

Going beyond the case of two heterogeneous servers, we now present an upper bound on the service capacity for a system of $K$ homogeneous servers. This bound holds for any service time distribution $X \sim F_X$ and cancellation delay $\Delta$.

**Theorem 4** (Throughput Upper Bound for $K$ Homogeneous Servers)**.** *For a system of $K$ homogeneous servers with service times $X \sim F_X$ that are i.i.d. across jobs and servers, and cancellation delay $\Delta$, the service capacity $R^*_{n,r}$ is bounded as follows*

$$R^*_{n,r} \leq \frac{K}{\min_{0 \leq t_2 \leq \dots \leq t_K} \mathbb{E}_X[C(t_2, \dots, t_K)]} \tag{29}$$

*where*

$$\mathbb{E}_X[C(t_2, \dots, t_K)] = \sum_{k=1}^{K} \mathbb{E}_X[(S - t_k)^+] + \Delta \left( \sum_{k=2}^{K} \mathbb{E}\left[\mathbb{1}(t_k < S)\right] + \mathbb{E}\left[\mathbb{1}(t_2 < S)\right] \right) \tag{30}$$

*where $t_2 \leq t_3, \leq \dots t_K$ are the start times of the replicas of a job relative to the first copy which starts at time $t_1 = 0$, and $(x)^+ = \max(0, x)$. The service time $S$ of a job is the time from the start of the earliest copy until any one of the replicas is served, that is,*

$$S = \min(X^{(1)}, X^{(2)} + t_2, \dots, X^{(K)} + t_K),$$

*where $X^{(k)} \sim F_X$.*

The proof is given in the Appendix. Now we look at some special cases to give an intuitive understanding of this upper bound. When the service time of each server is deterministic, that is, $X = c$, observe that the throughput upper bound is $K/c$, which is achieved by the no replication policy. On the other hand, if we have two homogeneous servers with exponential service times $X^{(1)}, X^{(2)} \sim Exp(\mu)$ and cancellation delay $\Delta > 0$, then we have

$$S = \begin{cases} X^{(1)} | X^{(1)} < t_2 & X^{(1)} \leq t_2 \\ t_2 + Exp(2\mu) & X^{(1)} > t_2 \end{cases} \tag{31}$$

$$C(t_2) = \begin{cases} X^{(1)} | X^{(1)} < t_2 & X^{(1)} \leq t_2 \\ t_2 + 2Exp(2\mu) + 2\Delta & X^{(1)} > t_2 \end{cases} \tag{32}$$

$$\mathbb{E}\left[C(t_2)\right] = \frac{1}{\mu} + 2\Delta e^{-t_2 \mu} \tag{33}$$

To minimize $\mathbb{E}\left[C(t_2)\right]$ we need to set $t_2$ to $\infty$, and thus we get $R^*_{n,r} \leq 2\mu$. This implies that the no replication policy is the best for exponentially distributed service times when there is a non-zero cancellation delay.

## VII. Concluding Remarks

The traditional view of a multi-server system is that its service capacity or maximum possible throughput is equal to the sum of the service rates. However, when we employ job replication to overcome variability in service time, there is a paradigm shift in this notion of service capacity – redundancy, when used effectively, can lead to a synergistic combination of servers such that the overall throughput is greater than the sum of the service rates of individual servers. Motivated by the idea that job replication can boost the throughput of multi-server system, this paper aims to find the maximum possible throughput. We first tackle the simpler case of upfront replication and determine the optimal number of replicas that maximize the throughput. Next, we consider gradual launch of additional replicas and propose two replication policies: MaxRate, a myopic policy based on an MDP formulation of the problem, and AdaRep, a tunable threshold-based replication policy. These policies are effective even in the low and moderate traffic regimes as demonstrated by simulation results presented in this paper. We also obtain upper bounds on the service capacity to understand the fundamental limits of the achievable throughput.

The main contribution of this paper is to demonstrate how replication can not only cope with service variability, but also make more efficient use of computing resources. Generalizations of the system model include allowing job killing and accounting for data locality constraints. Another future direction is analyze the mean response time of delayed job replication policies and finding the optimal policy in the low and moderate traffic regime. Designing online learning-based replication policies that can function in a system where the service time distributions are unknown or time-varying is also an interesting open problem.

## Acknowledgments

The authors are grateful to Emina Soljanin, Devavrat Shah, Gregory Wornell, Isaac Grosof, Mor Harchol-Balter and Samarth Gupta for fruitful discussions that helped improve this work.

## Appendix

*Proof of Claim 1.* Consider a non-work-conserving scheduling policy $\pi_{nwc}$ which results in job departure times $T_1(\pi_{nwc}) \leq \dots \leq T_n(\pi_{nwc})$. Construct a work-conserving $\pi_{wc}$ that follows all the actions of $\pi_{nwc}$, except the idling of servers. For example, consider a set of $r \geq 1$ servers that become idle at times $h_1, h_2, \dots, h_r$ respectively. If $\pi_{nwc}$ launches replicas of a job $i$ on these servers at times $h_1 + \epsilon_1, h_2 + \epsilon_2, \dots, h_r + \epsilon_r$, where $\epsilon_j \geq 0$ are the idle times, then $\pi_{wc}$ starts the replicas at times $h_1, h_2, \dots, h_r$ instead.

We use induction to prove that $T_i(\pi_{nwc}) \geq T_i(\pi_{wc})$ for all $1 \leq i \leq n$. In both policies, all servers are available for job assignment at time 0. The departure time of the first job is

$$\begin{aligned} T_1(\pi_{nwc}) &= \min(X_1 + \epsilon_1, X_2 + \epsilon_2, \dots X_r + \epsilon_r) && (34) \\ &\geq \min(X_1, X_2, \dots X_r) && (35) \\ &= T_1(\pi_{wc}). && (36) \end{aligned}$$

This is the induction base case. For the induction hypothesis, assume that for all $i \leq n-1$, $T_i(\pi_{nwc}) \geq T_i(\pi_{wc})$. We now prove that $T_n(\pi_{nwc}) \geq T_n(\pi_{wc})$. Suppose $\pi_{nwc}$ assigns job $n$ to $r \geq 1$ servers. The times $h_1, h_2, \dots, h_r$ when these servers become idle belong to the set $\{0, T_1(\pi_{nwc}), \dots T_{n-1}(\pi_{nwc})\}$,

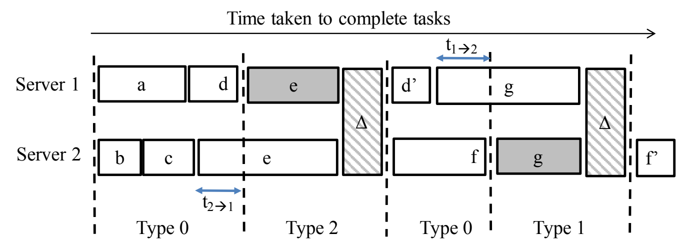


Fig. 12. Illustration of different types of intervals used to evaluate the throughput in Theorem 3. jobs $d$ and $f$ are paused to launch the replicas of $e$ and $g$ respectively, and they are resumed when the replicas are served or canceled.

the departure times of previous jobs. By the induction hypothesis, with $\pi_{wc}$ the servers become idle earlier at times $g_1, g_2, \dots, g_r$ where $g_j \le h_j$ for all $1 \le j \le r$. Thus,

$$\begin{aligned} T_n(\pi_{nwc}) &= \min(X_1 + h_1 + \epsilon_1, X_2 + h_2 + \epsilon_2, \dots, \\ &\qquad X_r + h_r + \epsilon_r) && (37)\\ &\ge \min(X_1 + h_1, X_2 + h_2, \dots X_r + h_r) && (38)\\ &\ge \min(X_1 + g_1, X_2 + g_2, \dots X_r + g_r) && (39)\\ &= T_{i+1}(\pi_{wc}) && (40)\end{aligned}$$

Thus, by induction, $T_n(\pi_{nwc}) \ge T_n(\pi_{wc})$ for any $n \in \mathcal{N}$. Hence by (1), $R(\pi_{nwc}) < R(\pi_{wc})$. □

*Proof of Theorem 1.* Incoming jobs are replicated at any one super-server, and the replicas are canceled as soon as one copy is served. Thus, the total time spent by each server in super-server $\mathcal{S}_j$ on a job is $\min_{l \in \mathcal{S}_j} X_l + \Delta$. The throughput of that super-server is

$$R_{\mathcal{S}_j} = \frac{1}{\mathbb{E}\left[\min_{l\in\mathcal{S}_j} X_l\right] + \Delta}. \qquad (41)$$

The overall throughput is the sum of the throughputs of the super-servers $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_h$, and is given by (41). □

*Proof of Theorem 2.* Let the number of servers in server $i's$ group be denoted by $r_i$. For example if $K = 5$ are divided into two groups of 3 and 2, then $r_1 = r_2 = r_3 = 3$ and $r_4 = r_5 = 2$. The throughput of a group with $r_i$ servers is $1/(\mathbb{E}[X_{1:r_i}] + \Delta)$. If we normalize by the number of servers, the throughput per server is $1/r_i(\mathbb{E}[X_{1:r_i}] + \Delta)$. Summing this over all servers we have,

$$\begin{aligned} R_{UpFr} &= \sum_{i=1}^{K} \frac{1}{r_i(\mathbb{E}[X_{1:r_i}] + \Delta)} && (42)\\ &\le \sum_{i=1}^{K} \frac{1}{r^*(\mathbb{E}[X_{1:r^*}] + \Delta)} && (43)\\ &= \frac{K}{r^*(\mathbb{E}[X_{1:r^*}] + \Delta)} && (44)\end{aligned}$$

If $r^*$ divides $K$, then dividing servers into groups of $r^*$ servers each gives equality in (43) above. □

*Proof of Theorem 3.* When $t_{1\to 2} = 0$ or $t_{2\to 1} = 0$, all jobs are replicated at both servers. Thus by Lemma 2 we get (26).

Now consider the case where $t_{1\to 2} > 0$ and $t_{2\to 1} > 0$. Time can be divided into three types of intervals as illustrated in Fig. 12. In Type 0 intervals, no jobs are replicated. In a Type 1 interval, both servers are serving a job that was originally launched on server 1. As soon as any one copy finishes, its replica is canceled. Then server 2 can resume its paused job, and we go back to a Type 0 interval. Similarly, in a Type 2 interval, both servers are serving a job that was originally run on server 2.

One job departs the system at the end of each Type 1 or Type 2 interval. Consider that this departure time is shifted to the end of the Type 0 preceding this Type 1 or Type 2 interval. This shift does not affect the overall throughput. Further, we rearrange the intervals to concatenate all Type 0 intervals together at the beginning of the time horizon, followed by all Type 1 and Type 2 intervals. Now the concatenated Type 0 interval can be viewed as a system of two servers running jobs according to the no replication policy, with service times $X_1^{tr}(t_{1\to 2}) = \min(X_1, t_{1\to 2})$ and $X_2^{tr}(t_{2\to 1}) = \min(X_2, t_{2\to 1})$, which are truncated versions of the original service times. Thus the rate of job completion in the concatenated Type 0 interval is

$$R_0 = \frac{1}{\mathbb{E}[X_1^{tr}(t_{1\to 2})]} + \frac{1}{\mathbb{E}[X_2^{tr}(t_{2\to 1})]}. \qquad (45)$$

Since all job departures are shifted to the end of Type 0 intervals, the rate of job completion in Type 1 and Type 2 intervals is zero, that is, $R_1 = R_2 = 0$. The overall throughput can be expressed as

$$\begin{aligned} R_{p,r} &= \mu_0 R_0 + \mu_1 R_1 + \mu_2 R_2 && (46)\\ &= \mu_0 R_0, && (47)\end{aligned}$$

where $R_i$ is the rate of job completion in concatenated interval of Type $i$. The weight $\mu_i$ is the fraction of total time spent in a Type $i$ interval. The ratios $\mu_1/\mu_0$ and $\mu_2/\mu_0$ can be expressed in terms of $t_{1\to 2}$ and $t_{2\to 1}$ as follows.

$$\begin{aligned} \frac{\mu_1}{\mu_0} &= \frac{\Pr(X_1 > t_{1\to 2})(\Delta + \mathbb{E}[\min(X_1^{rs}(t_{1\to 2}), X_2)])}{\mathbb{E}[X_1^{tr}(t_{1\to 2})]} && (48)\\ \frac{\mu_2}{\mu_0} &= \frac{\Pr(X_2 > t_{2\to 1})(\Delta + \mathbb{E}[\min(X_1, X_2^{rs}(t_{2\to 1})])}{\mathbb{E}[X_2^{tr}(t_{2\to 1})]} && (49)\end{aligned}$$

Every job originally run on server 1 spends $\mathbb{E}[X_1^{tr}(t_{1\to 2})]$ expected time in a Type 0 interval, and $\Pr(X_1 > t_{1\to 2})(\Delta + \mathbb{E}[\min(X_1^{rs}(t_{1\to 2}), X_2)])$ expected time in a Type 1 interval. Thus, the ratio $\mu_1/\mu_0$ is given by (48). Similarly we get (49).

Using (48) and (49) along with the fact that $\mu_0 + \mu_1 + \mu_2 = 1$, we can solve for $\mu_i$. Substituting $\mu_0$ in (47), we get the result in (27).

□

*Proof of Lemma 3.* AdaRep policies replicate a job run on server 1 (or server 2) after a fixed elapsed time $t_{1\to 2}$ (or respectively $t_{2\to 1}$). Instead of fixed $\mathbf{t}$, the replication thresholds could be chosen randomly such that the threshold vector $\mathbf{t}^{(i)}$ for some $i \in [1, 2, \dots I]$ is chosen with probability $Pr(\mathbf{t} = \mathbf{t}^{(i)})$. First let us show that this does not improve the throughput.

We can divide time into $I$ types of intervals, such that in the Type $i$ interval, replicas are launched according to the threshold vector $\mathbf{t}^{(i)}$. We can concatenate all intervals of Type $i$ together. Each type $i$ interval can be further divided into three types sub-intervals as given in the proof of Theorem 3 to compute the rate of job completion in that interval. The overall throughput can be expressed as a linear combination of rates of job completion in each of these interval types,

$$R_{p,r} = \sum_{i=0}^{I} Pr(\mathbf{t} = \mathbf{t}^{(i)}) R_{p,r}(\mathbf{t}_{(i)}) \tag{50}$$

$$\leq \sum_{i=0}^{I} Pr(\mathbf{t} = \mathbf{t}^{(i)}) \max_{\mathbf{t}} R_{p,r}(\mathbf{t}) \tag{51}$$

$$= \max_{\mathbf{t}} R_{p,r}(\mathbf{t}) \tag{52}$$

where $Pr(\mathbf{t} = \mathbf{t}^{(i)})$ is the fraction of time spent in the Type $i$ interval. The throughput of the best fixed threshold policy upper bounds each term in (50).

At any time instant the scheduler has two elapsed times available to it. AdaRep policies only consider the elapsed time of the job to be replicated. We now show that considering the elapsed time of the job that will be paused does not improve the throughput. To prove this we show that the throughput of any scheduling policy is independent of the elapsed time of the paused job. For any scheduling policy, the time horizon can be divided into three types of intervals as shown Fig. 12. Consider that the departures at the end of Type 1 and 2 are shifted to the end of the preceding Type 0 intervals. From the throughput analysis in the proof of Theorem 3 we can see that the rate of job completion in the concatenated Type 0 interval, and the fraction of time $\mu_0$ only depend on the elapsed times $t_{1\to 2}$ and $t_{2\to 1}$. Thus, considering the elapsed times of the job to be paused does not improve the throughput.

$\square$

*Proof of Theorem 4.* Consider a job that enters the central queue and is served by the system of $K$ homogeneous servers by launching copies at one or more servers. Without loss of generality, suppose that the first copy of a job starts at time $t_1 = 0$. Relative to this time, up to $K-1$ additional replicas start at times $t_2 \leq t_3 \leq \cdots \leq t_K$ respectively. If only $r < K$ copies of the job are launched, then $t_{r+1}, \ldots t_K$ are $\infty$. As soon as any one of these replicas is served, the others are canceled.

We first express the computation cost $C(t_2, t_3, \ldots, t_K)$, that is, the total time collectively spent by the $K$ servers on this job, in terms of these relative start times of the replicas $t_2 \leq t_3 \leq \cdots \leq t_K$. First, observe that the service time $S$ of the job, that is, the time from when the original copy of the job starts until the earliest replica finishes can be expressed in terms of the task start times as $S = \min(X_1, X_2 + t_2, \ldots, X_K + t_K)$. Since the $k^{th}$ server starts executing a replica of the job at time $t_i$ and the replicas are canceled at time $S$, the $k^{th}$ server spends $(S - t_k)^+$ on the job. Therefore, the total computation time (not including cancellation delay) collectively spent by the $K$ servers on the job is $\sum_{k=1}^{K} (S - t_k)^+$. A cancellation delay $\Delta$ is incurred at all the servers where the job has started service (including the server that finishes first). But if the job is launched at only one server and it finishes before any replica starts ($t_2 < S$) then there is no cancellation delay. Therefore, the expected computation time (including cancellation delay) is given by the expression (30).

The relative start times of the replicas $t_2 \leq t_3 \leq \cdots \leq t_K$ depend on the choice of the replication policy (for example MaxRate, AdaRep, upfront replication etc.) as well as service times of previous jobs which determine when the servers become available to serve current job. Thus, the throughput of any replication policy is

$$R_{n,r} = \frac{K}{\mathbb{E}_{t_2,\ldots,t_K} \mathbb{E}_X [C(t_2, t_3, \ldots, t_K)]} \tag{53}$$

where the joint distribution of $t_2, \ldots, t_K$ depends on the choice of the replication policy. Since expectation is lower-bounded by the minimum, we get the upper bound

$$R^*_{n,r} \leq \frac{K}{\min_{0 \leq t_2 \leq \cdots \leq t_K} \mathbb{E}_X [C(t_2, t_3, \ldots, t_K)]}. \tag{54}$$

$\square$

## References

[1] G. Joshi, "Synergy via redundancy: Boosting service capacity with adaptive replication," *SIGMETRICS Performance Evaluation Review*, vol. 45, pp. 21–28, Mar. 2018.

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

[3] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *ACM Commun. Mag.*, vol. 51, pp. 107–113, Jan. 2008.

[4] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, vol. 10, p. 10, 2010.

[5] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling," in *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pp. 69–84, 2013.

[6] N. F. Maxemchuk, "Dispersity routing," *Proceedings of the International Conference on Communications (ICC)*, pp. 10–13, Jun. 1975.

[7] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low latency via redundancy," in *Proceedings of the ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pp. 283–294, 2013.

[8] G. Koole and R. Righter, "Resource allocation in grid computing," *Journal of Scheduling*, vol. 11, pp. 163–173, June 2008.

[9] G. Joshi, Y. Liu, and E. Soljanin, "On the Delay-storage Trade-off in Content Download from Coded Distributed Storage," *IEEE Journal on Selected Areas on Communications*, May 2014.

[10] N. B. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?," *IEEE Transactions on Communications*, vol. 64, no. 2, pp. 715–722, 2016.

[11] Y. Sun, Z. Zheng, C. E. Koksal, K. Kim, and N. B. Shroff, "Provably delay efficient data retrieving in storage clouds," in *Proceedings of IEEE INFOCOM*, Apr. 2015.

[12] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytiä, and A. Scheller-Wolf, "Reducing latency via redundant requests: Exact analysis," in *Proceedings of the ACM SIGMETRICS*, Jun. 2015.

[13] G. Joshi, E. Soljanin, and G. Wornell, "Efficient replication of queued tasks for latency reduction in cloud systems," in *Proceedings of the Allerton Conference*, Oct. 2015.

[14] G. Joshi, E. Soljanin, and G. Wornell, "Efficient redundancy techniques for latency reduction in cloud systems," *ACM Transactions on Performance Evaluation of Computer Systems*, vol. 2, pp. 1–30, Apr. 2017.

[15] G. Joshi, Y. Liu, and E. Soljanin, "Coding for fast content download," in *Proceedings of the Allerton Conference on Comm., Control and Computing*, pp. 326–333, Oct. 2012.

[16] L. Kleinrock, *Theory, Volume 1, Queueing Systems*. New York, NY, USA: Wiley-Interscience, 1975.

[17] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, 2013.

[18] M. Mitzenmacher, *The power of two choices in randomized load balancing*. PhD thesis, University of California Berkeley, CA, 1996.

[19] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, “Effective straggler mitigation: Attack of the clones,” in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, pp. 185–198, Apr. 2013.

[20] K. Gardner, S. Zbarsky, M. Harchol-Balter, and A. Scheller-Wolf, “The power of d choices for redundancy,” in *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*, (New York, NY, USA), p. 409410, Association for Computing Machinery, 2016.

[21] K. Gardner, M. Harchol-Balter, and A. Scheller-Wolf, “A better model for job redundancy: Decoupling server slowdown and job size,” in *Proceedings of IEEE MASCOTS*, Sept. 2016.

[22] G. Joshi, E. Soljanin, and G. Wornell, “Queues with redundancy: Latency-cost analysis,” in *Proceedings of the ACM SIGMETRICS Workshop on Mathematical Modeling and Analysis*, June 2015.

[23] U. Ayesta, T. Bodas, and I. M. Verloop, “On a unifying product form framework for redundancy models,” *Performance Evaluation*, vol. 127-128, pp. 93–119, Nov. 2018.

[24] Y. Raaijmakers, S. Borst, and O. Boxma, “Delta probing policies for redundancy,” *SIGMETRICS Performance Evaluation Review*, vol. 46, p. 7273, Jan. 2019.

[25] D. Wang, G. Joshi, and G. W. Wornell, “Efficient straggler replication in large-scale parallel computing,” *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 4, Apr. 2019.

[26] M. Aktas, S. E. Anderson, A. Johnston, G. Joshi, S. Kadhe, G. L. Matthews, C. Mayer, and E. Soljanin, “On the service capacity of accessing erasure coded content,” in *Proc. Allerton Conf. Commun., Control and Computing*, Oct. 2017.

[27] F. Poloczek and F. Ciucu, “Contrasting effects of replication in parallel systems: From overload to underload and back,” *arXiv:1602.07978*, Feb. 2016.

[28] E. Anton, U. Ayesta, M. Jonckheere, and I. M. Verloop, “On the stability of redundancy models,” 2019.

[29] Y. Raaijmakers, S. Borst, and O. Boxma, “Redundancy scheduling with scaled bernoulli service requirements,” *Queueing Systems*, vol. 93, no. 1, pp. 67–82, 2019.

[30] E. Anton, U. Ayesta, M. Jonckheere, and I. Verloop, “Improving the performance of heterogeneous data centers through redundancy,” 2020.

[31] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, “Speeding up distributed machine learning using codes,” *IEEE Transactions on Information Theory*, 2017.

[32] Q. Yu, M. Maddah-Ali, and S. Avestimehr, “Polynomial codes: an optimal design for high-dimensional coded matrix multiplication,” in *Advances in Neural Information Processing Systems*, pp. 4406–4416, 2017.

[33] S. Dutta, V. Cadambe, and P. Grover, “Short-dot: Computing large linear transforms distributedly using coded short dot products,” in *Advances In Neural Information Processing Systems*, pp. 2100–2108, 2016.

[34] J. Kosaian, K. V. Rashmi, and S. Venkataraman, “Parity models: A general framework for coding-based resilience in ML inference,” *CoRR*, vol. abs/1905.00863, 2019.

[35] A. Mallick, U. Sheth, G. Palanikumar, M. Chaudhari, and G. Joshi, “Rateless Codes for Near-Perfect Load Balancing in Distributed Matrix-Vector Multiplication,” in *ACM Sigmetrics 2020*, May 2020.

[36] E. Varki, A. Merchant, and H. Chen, “The M/M/1 fork-join queue with variable sub-tasks,” *unpublished, available online*, 2008.

[37] R. Nelson and A. Tantawi, “Approximate analysis of fork/join synchronization in parallel queues,” *IEEE Transactions on Computers*, vol. 37, pp. 739–743, Jun. 1988.

[38] A. Rizk, F. Poloczek, and F. Ciucu, “Computable bounds in fork-join queueing systems,” in *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS 15, (New York, NY, USA), p. 335346, Association for Computing Machinery, 2015.

[39] Y. Xiang, T. Lan, V. Aggarwal, and Y. R. Chen, “Joint latency and cost optimization for erasure-coded data center storage,” *IEEE/ACM Transactions on Networking*, vol. 24, no. 4, pp. 2443–2457, 2016.

[40] P. Parag, A. Bura, and J.-F. Chamberland, “Latency analysis for distributed storage,” in *IEEE International Conference on Computer Communications (INFOCOM)*, May 2017.

[41] A. Badita, P. Parag, and J. Chamberland, “Latency analysis for distributed coded storage systems,” *IEEE Transactions on Information Theory*, vol. 65, pp. 4683–4698, Aug 2019.

[42] B. Li, A. Ramamoorthy, and R. Srikant, “Mean-field-analysis of coding versus replication in cloud storage systems,” in *IEEE International Conference on Computer Communications (INFOCOM)*, pp. 1–9, April 2016.