Boosting Service Capacity via Adaptive Task Replication

Gauri Joshi
Carnegie Mellon University
Pittsburgh PA 15213
gaurij@andrew.cmu.edu

1. INTRODUCTION

The service capacity (maximum rate of task completion) of a multi-server system is typically the sum of the service rates of individual servers. Several recent works study queueing systems with task replication, where the copies are canceled when any one replica is served. Replication affects the system in two ways: 1) replicas provide load-balancing by finding the shortest among the queues that they join, and 2) redundant time spent by multiple servers on the same task can add load to the system. Contrary to intution, [1–4] identify cases where replication can in fact reduce the system load, and thus increase service capacity. We seek to find the fundamental limit of this capacity boost, which is an open problem. We present a Markov Decision Process (MDP) framework to find the throughput-optimal replication policy. The MDP is hard to solve in general, and we have to resort to myopic replication policies. To help quantify the gap from optimality, we present an upper bound on the service capacity for the two server case.

2. PROBLEM FORMULATION

Consider a system of K servers with a central queue of tasks, as shown in Fig. 1. Since our objective is to maximize service capacity, we do not explicitly define an arrival process and assume that the queue is never idle.

2.1 Task Service Times

Server i takes time $S = YX_i$ to finish a task assigned to it. The random variable X_i captures the variability in task service time due to server slowdown. It is independent across servers, and i.i.d. across tasks assigned to any one server. The dependence of the service time on the size of the task is captured by 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 [5]. We also consider a cancellation delay Δ at all servers running a replicated task, after which they can serve subsequent tasks.

2.2 Scheduling Policy

When a server becomes idle, the scheduler can take one of two possible actions:

- ***new***: assign a new task to that server, or
- ***rep***: replicate a task that is currently running at one or more servers.

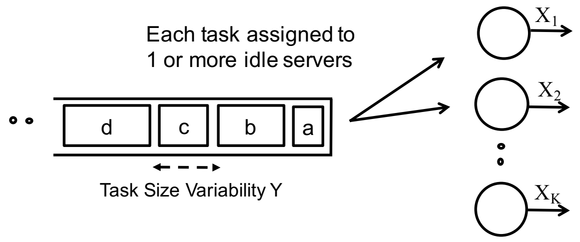

Fig. 1: A task replicated at two idle servers 1 and 2 takes time $Y \min(X_1, X_2)$ to finish, where Y captures the task-size variability and X_i captures the server slowdown.

The space of scheduling policies with the ***new*** and ***rep*** actions is denoted by $\Pi_{n,r}$. The scheduling policy can based on the distributions of Y, X_1, ..., X_K, but the scheduler does not know their realizations for currently running tasks.

Note that all policies in $\Pi_{n,r}$ are work-conserving, that is, they never allow a server to be idle for a non-zero time interval. In the extended version [6] we show that there is no loss of generality in focusing on work-conserving policies.

2.3 Throughput Metric

Our objective is to determine the policy $\pi^*_{n,r}$ that maximizes the throughput, which is defined as follows.

Definition 1 (Throughput R). *Let $T_1(\pi) \leq T_2(\pi) \leq \cdots \leq T_n(\pi)$ be the departure times of tasks $1, 2, \ldots n$, scheduled using policy π. The throughput is defined as*

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

We denote the maximum achievable throughput over all policies in $\Pi_{n,r}$ by $R^*_{n,r} = \max_{\pi \in \Pi_{n,r}} R(\pi)$. The policy $\pi^*_{n,r}$ that achieves $R^*_{n,r}$ is called the throughput-optimal policy.

Claim 1. *For any work-conserving policy, $R = K/\mathbb{E}[C]$, where C is the total time spent by the servers per task.*

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

3. FINDING THE OPTIMAL POLICY

3.1 No Replication and Full Replication

First let us compare the throughput of two extreme policies: no replication and full replication.

Lemma 1 (No Replication). *If each task is assigned to the first available idle server, the throughput is,*

$$R_{NoRep} = \sum_{i=1}^{K} \frac{1}{\mathbb{E}[Y]\,\mathbb{E}[X_i]} \tag{2}$$

Lemma 2 (Full Replication). *Suppose each task is assigned to all servers, and as soon as one replica finishes, the others are canceled. The resulting throughput is,*

$$R_{FullRep} = \frac{1}{\Delta + \mathbb{E}[Y]\,\mathbb{E}[\min(X_1, X_2, \dots X_K)]} \tag{3}$$

The proofs are given in [6]. Using Lemma 1 and Lemma 2 we can compare the two policies for any given $X_1, \dots, X_K, Y$ and cancellation delay Δ.

Example 1. Consider a system with two servers, and assume that the task size variability $Y = 1$ and the cancellation delay $\Delta = 0$. The service times of the two servers are

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

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

Fig. 2 compares the throughputs with full replication and no replication as p varies from 0 to 0.5. For $p > 0.068$, the FullRep policy outperforms NoRep.

3.2 MDP Formulation of the Optimal Policy

Instead of replicating tasks upfront, replicas could be added conditionally if the original task does not finish in some given time. We now propose a Markov Decision Process (MDP) framework to search for the best replication policy.

3.2.1 State-space

Let us 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} task departs. A state $s = [\mathcal{B}, \mathbf{t}, d]$ where $\mathcal{B}$ contains disjoint sets of server indices that are running the unfinished tasks in the system. For example, if $\mathcal{B} = \{\{1\}, \{2, 3\}\}$ there are two unfinished tasks 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 task. If a server is idle, its $t_i = 0$. The d term ensures that each state transition corresponds to a single task departure. If $d + 1$ tasks exit the system simultaneously and result in the task assignment set $\mathcal{B}$ and elapsed-time vector $\mathbf{t}$, then the system goes through states $[\mathcal{B}, \mathbf{t}, d] \to [\mathcal{B}, \mathbf{t}, d-1] \to \dots \to [\mathcal{B}, \mathbf{t}, 0]$.

3.2.2 Actions

In states $s = [\mathcal{B}, \mathbf{t}, 0]$, the scheduler can assign new tasks to idle servers, or replicate existing tasks. No tasks are assigned in the exit states $s = [\mathcal{B}, \mathbf{t}, d]$ with $d > 0$. Thus, for these states, the action space $\mathcal{A}_s$ contains a single placeholder ***null*** action. The system directly transitions to $[\mathcal{B}, \mathbf{t}, d-1]$.

3.2.3 Cost

The cost $C(s, s', a)$ of taking action a in state s and going to state s' is defined as the total time spent by the servers in that interval. Thus, the throughput-optimal policy is

$$\pi^*_{n,r} = \arg \min_{\pi \in \Pi_{n,r}} \sum_{j=0}^{\infty} C(s_j, s_{j+1}, \pi(s_j)). \tag{6}$$

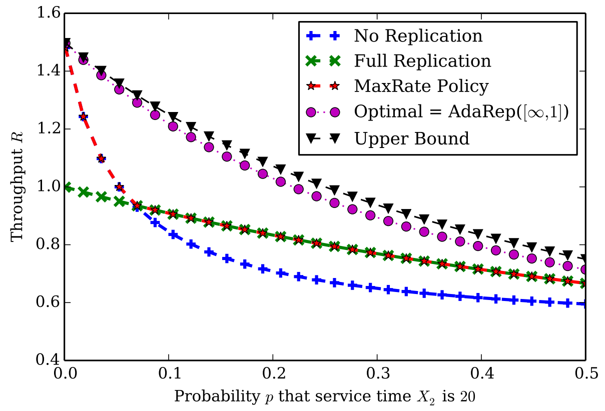

Fig. 2: Comparison of the throughputs of different replication policies, for the service times defined in Example 1. The derivation of the upper bound on $R^*_{n,r}$ is given in Section 4.

For the service distributions in Example 1, we can solve the MDP. The optimal policy (illustrated in Fig. 2) is to replicate a server 2's task at server 1 only if it does not finish in 1 second. In general the MDP can have a large state-space even for simple service distributions. And if X_i for any i or Y is a continuous random variable, then the MDP will have a continuous state-space and it is even harder to solve.

3.3 Proposed Replication Policies

As an alternative to solving the MDP, we propose a myopic policy called the MaxRate policy.

Definition 2 (MaxRate Policy). *When one or more servers become idle, the MaxRate policy chooses the action a that maximizes the instantaneous service rate $\hat{R}(a)$ which is defined as,*

$$\hat{R}(a) \triangleq \sum_{m=1}^{M(a)} \frac{1}{\mathbb{E}[D_m(a)]}, \tag{7}$$

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

Corollary 1. *Consider a two server system, with deterministic task size ($Y = 1$) and no cancellation delay ($\Delta = 0$). Suppose server 1 becomes idle, and the task running on server 2 has spent time $t_2 > 0$ in service. Then MaxRate launches a replica at server 1 if*

$$\frac{1}{\mathbb{E}[\min(X_1, X_2^{rs})]} > \frac{1}{\mathbb{E}[X_1]} + \frac{1}{\mathbb{E}[X_2^{rs}]}. \tag{8}$$

where $X_2^{rs} = (X_2 - t_2) | X_2 > t_2$, the residual computing time. Otherwise it assigns a new task to server 1.

Fig. 2 illustrates the MaxRate policy for the service distributions in Example 1. In this case the throughput of the MaxRate policy is the maximum of the throughputs of the NoRep and FullRep policies.

In general, (8) can help find replication thresholds $t_{i \to j}$ such that a task 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 policy called AdaRep($\mathbf{t}$), which is directly parametrized by a replication threshold vector $\mathbf{t}$.

DEFINITION 3 (ADAREP POLICY). *Consider a vector* $\mathbf{u} = (j_1, j_2, \dots j_k)$ *for* $k < K$ *such that a task first launched on server* j_1 *was later replicated on* j_2*,* j_3 *and so on. Replicate this task at server* i *if server* j_k *has spent at least* $t_{\mathbf{u}\to i}$ *time on it. Otherwise assign a new task to the idle server. If more than one tasks satisfy the replication condition, choose the task whose elapsed time is closest to its* $t_{\mathbf{u}\to i}$*.*

For example for $K = 2$ servers, the vector $\mathbf{t} = [t_{1\to 2},\, t_{2\to 1}]$. The optimal policy shown in Fig. 2 obtained by solving the MDP is AdaRep($[\infty, 1]$). In the next section we propose a method to choose $\mathbf{t}$ for the two-server case.

4. UPPER BOUND ON CAPACITY

To quantify the optimality gap of a policy without solving the MDP, we need an upper bound on $R^*_{n,r}$. Recall that in our problem formulation, tasks can be replicated only at time instants when one or more servers become idle. To find an upper bound, we consider that the scheduler is also allowed to pause ongoing tasks.

DEFINITION 4 (THE PAUSE-AND-REPLICATE SYSTEM). *A task can be replicated at any server where it is not already running by pausing the ongoing task on that server. The paused task is resumed after the replica is served or canceled.*

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

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

4.1 Evaluating the Upper Bound

In the pause-and-replicate framework, AdaRep($\mathbf{t}$) can replicate a task exactly after time $t_{\mathbf{u}\to i}$, instead of waiting for server i to become idle. In Theorem 1 below, we obtain a closed-form expression for the throughput $R_{p,r}(\mathbf{t})$ of the AdaRep policy for $K = 2$ servers and $Y = 1$. In [6] we show that there is no loss of generality in focusing on AdaRep policies. Thus, the upper bound $R^*_{p,r} = R_{p,r}(\mathbf{t}^*)$, the throughput of the best AdaRep policy.

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

$$R_{p,r}(\mathbf{t}) = \frac{1}{1 + \gamma_{1\to 2} + \gamma_{2\to 1}} \left(\frac{1}{\mathbb{E}\left[X_1^{tr}(t_{1\to 2})\right]} + \frac{1}{\mathbb{E}\left[X_2^{tr}(t_{2\to 1})\right]} \right) \tag{9}$$

where,

$$\gamma_{i\to j} \triangleq \frac{\Pr(X_i > t_{i\to j})(\Delta + \mathbb{E}\left[\min(X_i^{rs}(t_{i\to j}), X_j)\right])}{\mathbb{E}\left[X_i^{tr}(t_{i\to j})\right]}, \tag{10}$$

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* τ *seconds of service. If* $t_{1\to 2} = 0$ *or* $t_{2\to 1} = 0$*,* $R_{p,r}(\mathbf{t}) = 1/(\Delta + \mathbb{E}\left[\min(X_1, X_2)\right])$*.*

PROOF SKETCH. Consider the case $t_{1\to 2} > 0$ and $t_{2\to 1} > 0$. Time can be divided into intervals as illustrated in Fig. 3. In Type 0 intervals, no tasks are replicated. In a Type i interval, both servers are serving a task that was originally launched on server i. The overall throughput is

$$R_{p,r} = \mu_0 R_0 + \mu_1 R_1 + \mu_2 R_2. \tag{11}$$

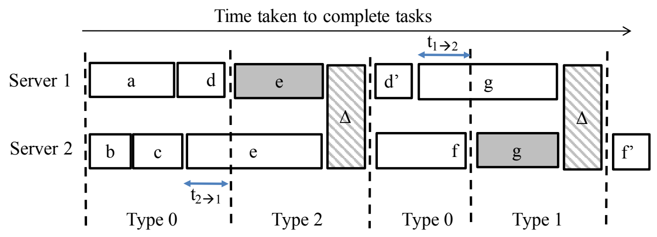

Fig. 3: Illustration of the types of intervals used to evaluate the throughput in Theorem 1. Tasks d and f are paused to launch the replicas of e and g, and are resumed afterwards.

where μ_i is the fraction of time spent in a Type i interval. One task departs the system at the end of each Type 1 or Type 2 interval. Without affecting overall throughput, let us shift these departure instants to the end of the preceding Type 0 interval. As a result,

$$R_0 = \frac{1}{\mathbb{E}\left[X_1^{tr}(t_{1\to 2})\right]} + \frac{1}{\mathbb{E}\left[X_2^{tr}(t_{2\to 1})\right]}, \tag{12}$$

and $R_1 = R_2 = 0$. To determine μ_0, we can find ratios μ_1/μ_0 and μ_2/μ_0 in terms of $t_{1\to 2}$ and $t_{2\to 1}$, and use the fact that $\mu_0 + \mu_1 + \mu_2 = 1$. □

The upper bound $R^*_{p,r}$ can be obtained by maximizing (9) over the parameters $[t_{1\to 2}, t_{2\to 1}]$. For example, for the service distributions in Example 1, $\mathbf{t}^* = [\infty, 1]$. The upper bound is shown in Fig. 2.

4.2 Choosing AdaRep replication thresholds

We propose using the optimal $\mathbf{t}^*$ that maximizes $R_{p,r}(\mathbf{t})$ as the replication threshold vector for the AdaRep policy in the original system. This policy tries to emulate the optimal pause-and-replicate policy, under the limitation that it cannot pause ongoing tasks. The AdaRep($\mathbf{t}^*$) policy is throughput-optimal for the example in Fig. 2, but checking its optimality in general is an open question. We are also exploring the idea of choosing $\mathbf{t}$ to $K > 2$ servers by recursively combining servers, one pair at time.

5. REFERENCES

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

[2] N. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?," in *Proceedings of the Allerton Conference*, Oct. 2013.

[3] 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.

[4] 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.

[5] 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.

[6] G. Joshi, "Synergy via Redundancy: Boosting Service Capacity with Adaptive Replication." https://goo.gl/549f8G, July 2017.