Optimal Reissue Policies for Reducing Tail Latency

The MIT Faculty has made this article openly available. Please share how this access benefits you. Your story matters.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>As Published</td>
<td><a href="https://doi.org/10.1145/3087556.3087566">https://doi.org/10.1145/3087556.3087566</a></td>
</tr>
<tr>
<td>Publisher</td>
<td></td>
</tr>
<tr>
<td>Version</td>
<td>Author’s final manuscript</td>
</tr>
<tr>
<td>Accessed</td>
<td>Thu Jan 03 13:10:39 EST 2019</td>
</tr>
<tr>
<td>Citable Link</td>
<td><a href="http://hdl.handle.net/1721.1/116708">http://hdl.handle.net/1721.1/116708</a></td>
</tr>
<tr>
<td>Terms of Use</td>
<td>Article is made available in accordance with the publisher's policy and may be subject to US copyright law. Please refer to the publisher's site for terms of use.</td>
</tr>
<tr>
<td>Detailed Terms</td>
<td></td>
</tr>
</tbody>
</table>
Optimal Reissue Policies for Reducing Tail Latency

Tim Kaler
MIT CSAIL
tfk@mit.edu

Yuxiong He
Microsoft Research
yuxhe@microsoft.com

Sameh Elnikety
Microsoft Research
sameh@microsoft.com

ABSTRACT
Interactive services send redundant requests to multiple different replicas to meet stringent tail latency requirements. These additional (reissue) requests mitigate the impact of non-deterministic delays within the system and thus increase the probability of receiving an on-time response. There are two existing approaches of using reissue requests to reduce tail latency. (1) Reissue requests immediately to one or more replicas, which multiplies the load and runs the risk of overloading the system. (2) Reissue requests if not completed after a fixed delay. The delay helps to bound the number of extra reissue requests, but it also reduces the chance for those requests to respond before a tail latency target.

We introduce a new family of reissue policies, Single-Time / Random (SingleR), that reissue requests after a delay $d$ with probability $q$. SingleR employs randomness to bound the reissue rate, while allowing requests to be reissued early enough so they have sufficient time to respond, exploiting the benefits of both immediate and delayed reissue of prior work. We formally prove, within a simplified analytical model, that SingleR is optimal even when compared to more complex policies that reissue multiple times.

To use SingleR for interactive services, we provide efficient algorithms for calculating optimal reissue delay and probability from response time logs through data-driven approach. We apply iterative adaptation for systems with load-dependent queuing delays. The key advantage of this data-driven approach is its wide applicability and effectiveness to systems with various design choices and workload properties.

We evaluated SingleR policies thoroughly. We use simulation to illustrate its internals and demonstrate its robustness to a wide range of workloads. We conduct system experiments on the Redis key-value store and Lucene search server. The results show that for utilizations ranging from 40%-60%, SingleR reduces the 90th-percentile latency of Redis by 30-70% by reissuing only 2% of requests, and the 99th-percentile latency of Lucene by 15-25% by reissuing 1% only.

1 INTRODUCTION
Interactive online services, such as web search, financial trading, and games require consistently low response time to attract and retain users [13, 24]. The service providers therefore define strict targets for tail latencies — 95th percentile, 99th percentile or higher response times [6, 7, 14, 31] — to deliver consistently fast responses to user requests. For many distributed and layered services, a request could span several servers and the responses are aggregated, in which case the slower servers typically dominate the response time [18]. As a result, tail latencies are more suitable performance metrics than averages in latency-sensitive applications that employ concurrency.

Variability in a service’s response-time can lead to tail-latencies that are several orders of magnitude larger than the average or median. Rare work-intensive requests can have a disproportionate impact on tail-latency by causing other requests to be delayed. Other, often nondeterministic, factors also play a significant role: random load-balancing can lead to short-term skew between machines; background tasks on servers can lead to temporary shortages in CPU cycles, memory, and disk bandwidth; network congestion can increase latency and reduce throughput of communication channels. Reducing tail latency, influenced by all of these contributing factors, is challenging.

The judicious use of redundant computation is often a highly effective technique for reducing tail-latency in interactive services. The basic idea is to exploit inter-machine parallelism by sending multiple copies of a request to replicated servers in order to boost the probability of receiving at least one timely response. This technique is widely used by interactive services, yet despite its prevalence there has been little guidance on optimizing its usage.

We develop a methodology for designing reissue policies that is composed of 3 steps. First, we define several families of reissue policies of varied complexity. These reissue policies are parametrized by variables such as: a) if to reissue a request, b) when to reissue a request, and c) how many times to reissue a request. We choose an optimal family of policies among the candidates guided by a theoretical analysis under a simplified model where the system’s response-time distributions are static. Second, we provide an algorithm to find the optimal values for the policy’s parameters using response-time logs, solving the constrained optimization problem efficiently. Third, we provide iterative algorithms for refining a policy’s parameters in response to changes in system load, and for adjusting the total fraction of requests that are reissued to minimize tail-latency.

Related work and challenges. This technique of reissuing latency-sensitive requests is not new. It has been employed by a wide variety of systems such as key-value stores [4, 19, 26, 29], distributed request-response workloads [15], DNS lookup [2, 27], TCP flows [8, 30], and web search [6]. Existing systems that reissue requests to reduce tail-latency predominantly employ one of two strategies. For systems that run at low utilization, the common approach is to perform immediate reissue of requests — i.e. dispatch multiple copies of all requests. The effectiveness of immediate reissue has been investigated in previous studies [8, 26, 27, 30]. The primary advantage of the immediate reissue approach is that all copies of a request have an equal chance to respond before a tail-latency deadline since they are dispatched at the same time. This advantage is a motivation within RepFlow [30] for employing immediate reissue for the replication of short TCP flows (under 100KB). The disadvantage of immediate reissue, however, is that its impact on overall load renders it ineffective for systems with moderate and high utilization. A recent study in [27] on memcached, for example, shows that immediate reissue can degrade performance at utilizations as low as 10%.

For systems that run at higher utilization, an alternative approach is to perform delayed reissue of requests [5, 6, 15, 29] — i.e. dispatch a second copy of a request after a delay $d$, which we refer to as Single-Time / Deterministic policy or StmLeD. The StmLeD...
policy family corresponds to the scheme proposed in “The Tail at Scale” by Dean and Barroso [6], where, for example, the delay \(d\) could be decided using 95th-percentile latency of the workload. The advantage of delayed reissue is that we save the cost of reissuing the requests that would respond fast anyway. However, if the delay \(d\) is picked to be too large, then there may not be sufficient time for a reissue request to respond before the latency target.

Along the line of analytical work, prior work only studied immediate reissues for average latency under very specific arrival/service time distributions. Joshi, et.al. [16, 17] study the impact of immediate reissuing on log-concave and log-convex service-time distributions. Gardner, et.al. [10] present an exact analysis of immediate reissue for poisson arrivals and exponential service-times. Lee et.al. [20] consider minimizing average latency by reissuing requests with a known cancellation overhead. Shah et.al. [25] analyze the effectiveness of immediate reissuing in the MDS queue model.

When it comes to developing effective reissue policies for reducing tail-latency on a wide range of workloads and systems, many questions remain largely unanswered. The problem is challenging for multiple reasons: (1) The impact of reissuing is complex: one must weigh the odds of reducing tail latency by sending a duplicate request against the increase in system utilization caused by adding load. (2) There is a large search space with many different choices of which requests to reissue and when. (3) The complex and different workload properties of various interactive services, such as service-time distributions, arrival patterns, request correlations, and system settings make it difficult to derive general strategies for reducing tail latency. (4) Analytical work using queuing theory is challenging even when making strong assumptions about response-time distributions (e.g. drawn from exponential family), and conclusions drawn from such simple models are hard to generalize to more complex systems.

Methodology and Key Results. The goal of our work is to find a reissue policy that minimizes a workload’s 99th percentile tail latency by issuing a fixed percentage (or budget) of redundant requests. We explore the space and devise reissue policies in a principled manner — directed by theoretical analysis to identify the key insights of effective reissue policies, and driven by empirical data from actual systems for wide applicability.

We introduce a new family of reissue policies, Single-Time / Random (SingleR), that reissue requests after a delay \(d\) with probability \(q\). The use of randomness in SingleR provides an important degree of freedom that allows to bound the reissue budget while also ensuring that reissue requests have sufficient time to respond, exploiting the benefits of both immediate and delayed reissue of prior work.

Using a simplified analytical model, we formally prove that SingleR is the optimal trade-off between the immediate and delayed reissue strategies. More precisely, we define the Multiple-Time / Random (Multipler) policies which reissue requests multiple times with different delays and reissue probabilities. We prove that, surprisingly, the optimal policies in Multipler and SingleR are equivalent. It is a powerful result, restraining the complexity of reissue policies to one time reissue only while guaranteeing the effectiveness of SingleR.

Next, we present how to apply SingleR for interactive services through a data-driven approach to efficiently find the appropriate parameters, reissue time and probability, given sampled response times of the workloads. Our approach takes into account correlations between primary and reissue request response times. It is computationally efficient, finding optimal values of the parameters in close to linear time, with respect to the data size.

Moreover, we show how to devise reissue policies for systems which are sensitive to added load by adaptively refining a reissue policy in response to feedback from the system. This method remains oblivious to many system design details, relies on iterative adaptation to discover a system’s response-time distributions and its response to added load. This data-driven approach is performed in a principled manner: every refined policy is the solution to a well defined optimization problem based on updated response-time distributions, applicable to a wide range of workloads with varying properties.

Empirical evaluation. We illustrate the properties of SingleR using both simulation and system experiments. Through careful simulation, we illustrate two key points: 1) the use of randomization in SingleR is especially important for workloads with correlated service times and queuing delays, 2) the effectiveness of SingleR is robust to varied workload properties and design choices including: utilization, service-time distribution, target latency percentiles, service-time correlations, and load-balancing/request-prioritization strategies.

We also evaluate SingleR using two distributed systems based on Redis [32] and Lucene enterprise search [21]. We demonstrate that, on a wide range of utilizations from 20-60%, SingleR is able to reduce tail-latency significantly while reissuing only a small number of requests. Even at 40-60% utilization, which is high for interactive services, SingleR reduces the 99th-percentile latency of Redis by 30-70% while reissuing only 2% of requests, and the 99th-percentile latency of Lucene by 15-25% while reissuing just 1% of requests.

Summary of contributions.

(1) We introduce the SingleR reissue policy family that reissues requests after a delay \(d\) with probability \(q\). It exploits randomness to permit the timely reissue of requests with bounded budget, achieving the benefits of both immediate and delayed reissue (Section 2).

(2) We prove within a simplified analytical model that the optimal policies in Multipler and SingleR are equivalent. Reissuing more than once does not offer additional benefit — SingleR is simple and effective. (Section 3).

(3) We show how to apply SingleR for interactive services by providing efficient algorithms for obtaining reissue delay and probability parameters from response time logs. (Section 4).

(4) We evaluate SingleR using both simulation and system experiments on Redis key value store and Lucene search server (Section 5 and Section 6).

Note that our methodology for developing reissue policies utilizes multiple performance models of increasing complexity. This is a strategic choice that allows us to make definitive design choices that are guided by theoretical insights. The proof that SingleR is optimal relative to SingleD and Multipler operates in a simplified model in which policies reissue only a fixed fraction of requests, and where the service’s response-time distributions are static and uncorrelated. This simplified model allows us to address questions about the general structure of reissue policies that are otherwise intractable. Our algorithms for finding the optimal SingleR policy for a specific interactive service operates in a less constrained model where response-times may be correlated. Our techniques for adaptively refining SingleR policies are in a more general model in which a system may have load-dependent queuing delays — i.e. reissue requests perturb the response-time distribution. The sequence of decisions made with respect to performance model are not arbitrary. As shown in the empirical analysis of SingleR on simulated workloads in Section 5 and in real-world systems in Section 6 these steps lead to effective reissue policies and the insights made in simpler models are readily recognizable in our empirical results.

2. DETERMINISTIC VS RANDOM REISSUE

In this section, we introduce the Single-Time / Random (SingleR) policies, which reissues a request with probability \(q\) after a delay.
We shall, for the moment, operate within a simplified performance model while employing the Single-Time / Deterministic (SINGLED) policies which formalize the "delayed reissue" strategy of prior work. We present SINGLER policies in Section 2.3 and discuss their benefits over SINGLED in Section 2.4.

2.1 Model and Terminology

We shall, for the moment, operate within a simplified performance model in which there are no queueing delays and query response-times are independent and identically distributed. Later, in Section 4.2 we describe how these limitations can be overcome to adapt our techniques to workloads with correlated response-times and queueing delays.

Formally, we consider an interactive workload to be a collection of queries where each query is composed of exactly one primary request that is dispatched at time \( t = 0 \) and zero or more reissue requests dispatched at times \( d \geq t \).

The response-time of a query is based on the length of time between the dispatch of the primary request and the arrival of any reply from either a primary or reissue request.

The reissue rate of a workload consisting of \( N \) queries and \( M \) reissue requests is defined as the ratio \( M/N \).

We look for a reissue policy that minimizes a workload’s 5th percentile tail-latency with the reissue rate equal to a given reissue budget \( B \).

2.2 The SINGLED Policies

The Single-Time / Deterministic (SINGLED) policy family is a 1-parameter family of policies that is parametrized by a reissue delay \( d \). A SINGLED policy reissues a request if a response has not been received after \( d \) seconds.

Let the random variable \( X \) denote the response time of the primary request and \( Y \) denote the response time of the reissue request. A query \( Q \) completes before time \( t \) if its primary response-time \( X \) is less than \( t \), or if the reissue request response-time \( Y \) is less than \( t - d \). The probability that the query \( Q \) responds before time \( t \) is given by Equation (1):

\[
\Pr(Q \leq t) = \Pr(X \leq t) + \Pr(X > t)\Pr(Y \leq t - d)
\]

The expected number of reissue requests created by a SINGLED policy is equal to the number of primary requests that respond after time \( d \), i.e., the reissue budget is

\[
B = \Pr(X > d).
\]

Therefore, if a system can tolerate 10% additional requests, then the delay \( d \) is chosen for the SINGLED policy such that \( \Pr(X > d) = 1/10 \). The smaller the delay \( d \), more requests are reissued, and the higher the budget \( B \).

2.3 The SINGLER Policies

The Single-Time / Random (SINGLER) policy family is a 2-parameter family of policies that is parametrized by a reissue delay \( d \) and a reissue probability \( q \). A SINGLER policy reissues a request with probability \( q \) if a response has not been received after \( d \) seconds.

A query \( Q \) responds before time \( t \) if the primary request responds before time \( t \), or if a reissue request was created and its response time is less than \( t - d \). The probability that \( Q \) completes before time \( t \) while employing SINGLER is given by Equation (3):

\[
\Pr(Q \leq t) = \Pr(X \leq t) + q\cdot\Pr(X > t)\Pr(Y \leq t - d)
\]

The reissue budget is

\[
B = q\cdot\Pr(X > d) \tag{4}
\]

Given Equation (3) and (4), we write the constrained optimization problem which identifies the reissue delay and probability parameters of the optimal SINGLER policy given the primary and reissue response time distributions \( X \) and \( Y \).

Optimal Policy For SINGLER:

Given tail-latency percentile \( k \), a reissue budget \( B \), and policy family SINGLER

minimize \( d, q \)

subject to \( \Pr(X \leq t) + q\cdot\Pr(X > t)\Pr(Y \leq t - d) \geq k \), \( q\cdot\Pr(X \geq d) \leq B \)

2.4 Randomization Is Essential

The use of randomization in SINGLER allows the reissue budget, and thus the added resource and system load, to be bounded while also ensuring that requests can be reissued early enough so they have sufficient time to respond. This may not be allowed under SINGLED, which we illustrate in the following example.

Suppose, for example, that we want to minimize a workload’s 95th percentile tail-latency by reissuing no more than 5% of all queries. Clearly, this cannot be achieved using a SINGLED policy — its limited reissue budget forces it to reissue requests later than the original 95th percentile tail-latency.

In general, a SINGLED policy cannot reduce any workload’s 4th percentile latency with budget \( B < 1-k \). Randomization is an essential part of an effective reissue policy.

3 SINGLE VS MULTIPLE REISSUE

As we saw in Section 2, randomness provides SINGLER policies an important degree of freedom that enables a continuous trade-off between the advantages of immediate and delayed reissuing. A natural question arises: can we obtain an even better policy family by introducing additional degrees of freedom?

In this section, we address this question by introducing MULTIPLE policies that can reissue requests more than once, at multiple different times, and with different probabilities. We prove a surprising fact: for a given reissue budget \( B \) and tail-latency percentile \( k \), the optimal MULTIPLE and SINGLER policies achieve the same tail-latency reduction.

Note that we continue to operate in the simplified model described in Section 2.1 in which there are no queueing delays and query response-times are independent and identically distributed. These limitations will be lifted in Section 4.2 as we show how to adapt SINGLER policies to handle correlated response-times and queueing delays.

3.1 Multiple Time Policies

The Multiple-Time / Random (MULTIPLE) policy family contains policies that can reissue requests multiple times. A policy that reissues a request at-most \( n \) times consists of a sequence of \( n \) delays \( d_1, d_2, \ldots, d_n \) and \( n \) probabilities \( q_1, q_2, \ldots, q_n \). Like SINGLER, the MULTIPLE family explores the space between two extremes — the "immediate reissue" and "delayed reissue" strategies. Specifically, the reissue times \( d_i \) of a MULTIPLE policy lie between 0, the time of immediate reissue, and \( d' \), the time selected by a "delayed reissue" SINGLED policy, where \( \Pr(X > d') = B \). For any \( d_i \), since \( d_i \leq d' \), the following condition holds

\[
\Pr(X > d') \geq B. \tag{5}
\]

For the purpose of our later arguments, we also define the Double-Time / Random (DOUBLE) policy family. The DOUBLE
family is a subset of \textit{Multiplier} and contains policies that reissue requests at most twice.

### 3.2 Single Is Optimal

We prove the optimality of \textit{SingleR} in two steps: (1) We show in Theorem 3.1 that the optimal policies in the \textit{SingleR} and \textit{DoubleR} families achieve identical tail-latency reduction; (2) Finally, we prove a generalization in Theorem 3.2 for \textit{Multiplier} policies that have \( n > 2 \) reissue times.

**Theorem 3.1.** The optimal \textit{SingleR} and \textit{DoubleR} reissue policies achieve the same \( k \)th percentile tail-latency when given the same reissue budget \( B \).

**Proof.** Consider the optimal \textit{SingleR} policy with budget \( B \) that minimizes \( t \), the \( k \)th percentile tail-latency. Suppose that this policy reissues requests at time \( d' \). Then, the probability that a query using the optimal \textit{SingleR} policy responds before time \( t \) is given by Equation (6) below.

\[
\Pr(Q \leq t) = \Pr(X \leq t) + G^*_{SR}
\]

where,

\[
G^*_{SR} = B \frac{\Pr(X > d') - \Pr(X > t) \Pr(Y \leq t - d')}{\Pr(X > t) \Pr(Y \leq t - d')}.
\]

The first term \( \Pr(X \leq t) \) is the probability that the primary request returns before the tail-latency deadline. The term \( G^*_{SR} \) corresponds to the case for which the primary request misses the deadline, but the reissue request responds on-time.

Now consider a \textit{DoubleR} policy with reissue times \( d_1, d_2 \) and reissue probabilities \( q_1, q_2 \). The probability that a query using this policy responds before time \( t \) is given by Equation (8) below.

\[
\Pr(Q \leq t) = \Pr(X \leq t) + G_1 + G_2
\]

where,

\[
G_1 = q_1 \Pr(X > t) \Pr(Y \leq t - d_1)
\]
\[
G_2 = q_2 (1 - q_1) \Pr(Y \leq t - d_2) \Pr(X > t) \Pr(Y \leq t - d_2)
\]

The term \( G_1 \) corresponds to the case for which the primary request misses the deadline, but the first reissue request responds on-time. Lastly, the third term \( G_2 \) corresponds to the case where both the primary and first reissue request miss the deadline, but the second reissue request responds on-time.

We shall show that \( G_1 + G_2 \leq G^*_{SR} \). After this has been shown, it follows that no \textit{DoubleR} policy can achieve a lower tail-latency than a \textit{SingleR} policy with the same budget.

First, we provide a bound on \( G_1 \).

Consider a \textit{SingleR} policy that reissues requests at time \( d_1 \) with probability \( B \cdot \Pr(X > d_1) \). Using this policy, the probability that a query returns before time \( t \) is given by

\[
\Pr(Q \leq t) = \Pr(X \leq t) + G^*_{SR,1}
\]

where,

\[
G^*_{SR,1} = B \frac{\Pr(X > t) - \Pr(Y \leq t - d_1)}{\Pr(X > t) \Pr(Y \leq t - d_1)}.
\]

Since \( G^*_{SR} \) is the optimal policy for a budget \( B \), we have that \( G^*_{SR,1} \leq G^*_{SR} \).

Multiplying both sides of Inequality (13) by \( q_1 \Pr(X > d_1) B^{-1} \) gives us the upper bound on \( G_1 \) shown in Inequality (14).

\[
G_1 \leq q_1 \Pr(X > d_1) \frac{B}{B}
\]

Second, we provide an upper bound on \( G_2 \).

We begin by formulating an upper bound on \( G_2 \) that is a function of \( q_1 \). This requires a sequence of observations. We note that the budget constraint for the \textit{DoubleR} policy implies the following inequality:

\[
q_1 \Pr(X > d_1) + q_2 \Pr(X > d_2) (1 - q_1 \Pr(Y \leq d_2 - d_1)) \leq B
\]

Then, given \( q_1 \) Inequality (15) implies the following upper bound on \( q_2 \):

\[
q_2 \leq \frac{B - q_1 \Pr(X > d_1)}{\Pr(X > d_2) (1 - q_1 \Pr(Y \leq d_2 - d_1))}.
\]

Finally, we incorporate this bound on \( q_2 \) into the expression for \( G_2 \) given in Equation (10) to obtain an upper bound on \( G_2 \) as a function of \( q_1 \).

\[
G_2 \leq \frac{B - q_1 \Pr(X > d_1)}{\Pr(X > d_2)} Y \Pr(Y \leq t - d_2)
\]

where, \( Y = (1 - q_1 \Pr(Y \leq t - d_1)) / (1 - q_1 \Pr(Y \leq d_2 - d_1)) \). Note that \( Y \) is at most 1 since \( d_2 \) is less than \( t \) which allows us to omit \( Y \) in Inequality (17) to obtain a simpler (albeit weaker) upper bound on \( G_2 \).

Now consider a \textit{SingleR} policy that reissues at time \( d_2 \) with probability \( B \cdot \Pr(X > d_2) \). The probability that a query using this policy responds before time \( t \) is given by:

\[
\Pr(Q \leq t) = \Pr(X \leq t) + G_{SR,2}
\]

where,

\[
G_{SR,2} = \frac{B}{\Pr(X > t)} \Pr(Y \leq t - d_2).
\]

We have that for all positive \( a \) that \( a^*_{SR,2} \leq a^*_{SR} \). Let \( a = 1 - q_1 \Pr(X > d_1) B^{-1} \), which is strictly positive since the budget constraint on the \textit{DoubleR} policy implies the inequality \( q_1 \Pr(X > d_1) < B \).

Then, combining Equation (19) and Inequality (17) we have that

\[
G_2 \leq 1 - \frac{q_1 \Pr(X > d_1)}{B} G_{SR,2}
\]

\[
\leq 1 - \frac{q_1 \Pr(X > d_1)}{B} G^*_{SR}.
\]

Together the upper bounds on \( G_1 \) and \( G_2 \) imply that \( G_1 + G_2 \leq G^*_{SR} \) completing the proof.

**Theorem 3.2.** The optimal \textit{SingleR} and \textit{Multiplier} reissue policies achieve the same \( k \)th percentile tail-latency when given the same reissue budget \( B \).

**Proof.** Assume as an inductive hypothesis that the theorem holds for \( n \) and \((n+1)\)-time \textit{Multiplier} policies. The base cases for 1-time and 2-time \textit{Multiplier} policies follows from Theorem 3.1.

Consider an optimal \((n+2)\)-time \textit{Multiplier} policy \( P_{n+2} \) with reissue times \( d_1, ... d_{n+2} \). To complete the inductive argument, we will show that there exists an \((n+1)\)-time \textit{Multiplier} policy with reissue times \( d_1, ..., d_n, d' \) that achieves the same \( k \)th percentile tail-latency.

Let \( P_n \) be the \( n \)-time \textit{Multiplier} policy obtained by taking the first \( n \) reissue times and reissue probabilities in \( P_{n+2} \). The policy \( P_n \) consumes budget \( aB \leq B \), where \( a \leq 1 \).

Let \( Q[P_n] \) be a random variable representing the response-time distribution of a query reissued using policy \( P_n \).

Let’s now transform the original problem to a new but equivalent problem of minimizing the \( k \)th percentile tail-latency of a workload \( W' \) with primary response-time distribution \( Q[P_n] \) and reissue response-time distribution \( Y \).

We want to show that, for the workload \( W' \), a reissue policy with budget \((1-a)B\) that reissues at times \( d_1, ... d_n \) is a \textit{DoubleR} policy. In particular, we want to show that its budget and reissue times satisfy the condition of Inequality (5) under \textit{Multiplier} definition,
We use a data-driven approach to efficiently find the appropriate
This section presents how to use the reissue probability, minimizing tail latency. Our al-
gorithm is computationally efficient, taking O(NlogN) time where N is the number of
response time samples. (2) We extend the alg-
We present an algorithm that computes optimal reissue time and probability, minimizing tail latency. Our al-
gorithm is computationally efficient, taking O(NlogN) time where N is the number of
response time samples. (2) We extend the alg-
We present an algorithm that computes optimal reissue time and probability, minimizing tail latency. Our al-
gorithm is computationally efficient, taking O(NlogN) time where N is the number of
response time samples. (2) We extend the alg-
We present an algorithm that computes optimal reissue time and probability, minimizing tail latency. Our al-
gorithm is computationally efficient, taking O(NlogN) time where N is the number of
response time samples. (2) We extend the alg-

The impact of these reissue requests on the primary response-time. The response-time of a request can be divided into two components: timalSingleR by modifying line 19 of "[1,22] over pairs (x,y) where x and y are the primary and reissue response times.

Each range query performed within SingleRSuccessRate takes O(logN) time, and SingleRSuccessRate is invoked at most 2N times by ComputeOptimalSingleR. Therefore, the procedure ComputeOptimalSingleR which takes into account correlation computes the optimal SingleR policy in Θ(NlN) time.

4.3 Iterative Adaptation for Queue Delays

The queuing delay of requests in a workload depends on the arrival process to a service. The use of a reissue policy can perturb this arrival process and change the response-time distributions used by ComputeOptimalSingleR to find a SingleR policy.

The impact of added load on a workload’s response-time distributions can be significant. Consider the inverse CDFs illustrated in Figure 2a for Original and Primary requests. The Original curve illustrates the inverse CDF of the original primary response-time distribution of the system when no requests are reissued. The Primary curve illustrates the new inverse CDF of the primary response-time distribution when using a SingleR policy with a 30% reissue budget. The impact of these reissue requests on the primary response-time distribution is dramatic: the 85th percentile grows from 50 to 350.

We employ an adaptive approach to iteratively refine a SingleR policy in response to changes in the response-time distribution. First, we begin with a reissue policy P that reissues requests at time d = 0 with probability B. We then execute the system with the reissue policy and sample the response-time distributions of primary and reissue requests. The sampled response-time distributions are used within ComputeOptimalSingleR to compute the optimal SingleR policy Popt for these response-time distributions. Next, we obtain a new policy P’ that has reissue delay d’ = d + λ(dlocal − d) where λ is a learning rate. Finally, this process is repeated until the empirical 4th percentile tail-latency converges to the value predicted by ComputeOptimalSingleR and the empirical reissue rate converges to B.

This adaptive approach is based upon two observations: a) using the same budget, reissuing later tends to impact load more as it is more likely to reissue requests with more work and higher resource demands; and, b) small changes to the reissue delay result in only small changes to the response-time distributions. Observation (a) implies that the predicted 4th percentile tail-latency at each step of ComputeOptimalSingleR increases after each step of the algorithm. Observation (b) implies that for sufficiently small λ that the true optimal reissue time dopt lies between d’ and dlocal at each step of the algorithm.

Figure 2b shows the 95th percentile tail-latency achieved on each step of the adaptive algorithm using a learning rate of 0.2 for a SingleR policy with a reissue budget of 30%. Convergence can be detected by comparing the policy optimizer’s predicted tail-latency with the observed latency when using the policy. For this workload, convergence is achieved after ≈ 6 iterations.

4.4 Extended Scenarios

The tools and algorithms presented in the preceding sections can be applied to handle common scenarios that occur in practice. Since space limitations prohibit an exhaustive examination of each of these scenarios, we shall instead sketch a few strategies for addressing common use cases.

Varying load / response-time distributions. In practice a system’s response-time distribution can vary over time on both short (hourly, daily), and long (monthly, yearly) time scales. The iterative algorithm for adaptively refining a SingleR policy can be applied in an on-line fashion to address these temporal variations, but requires modifications which depend on specific application needs and the time-scale of interest to properly balance exploration and exploitation in its search.

Selecting optimal reissue budget. The adaptive algorithm described in this section assumes the use of a fixed reissue budget. As we learned in Section 2, SingleR policies are able to reduce tail-latency in a “smooth” fashion even with very small reissue budgets. As a consequence, the tail-latency reduction of SingleR as a function of the reissue budget tends to be a parabola whose extrema can be readily found through simple binary search techniques.

1The corresponding simulation setup for Figure 2a is discussed in Section 5.
To evaluate the practicality of this simple approach, we implemented a simple budget selection procedure that performs the following steps: 1) set $\delta = 1\%$ and set best-budget = 0; 2) for budget best-budget + $\delta$ run the adaptive SINGLER policy optimizer for 5 adaptive trials to produce reissue policy $P$; 3) collect response-time data from the system when using reissue policy $P$. 4) if the budget best-budget + $\delta$ has smaller 99th percentile tail-latency than best-budget, then set $\delta = 3\%$. Otherwise, set $\delta = 3\%$. An example of this binary search procedure is presented later in Figure 8 as part of our system experiments in Section 6.

Meeting tail-latency with minimal resources. Interactive services often formulate service-level agreements (SLA) that guarantee a fixed latency for $k\%$ of all requests. In such a scenario, a system designer may be interested in minimizing the resources required to satisfy the SLA. Given a particular tail-latency target $T$, the budget can be minimized using either a brute force search, starting at small reissue rates, or by using a variation of the binary search procedure for finding the optimal budget that transforms tail-latency values $L$ using the function $f(L) = \min(T,L)$.

5 SIMULATIONS
In this section we use a discrete-time event simulator to carefully evaluate the behavior and tail-latency impact of SINGLER policies. Simulation allows us to vary workload and system properties covering a wide range of scenarios.

First, we provide simulation results on three types of workloads: Independent, Correlated, and Queueing, corresponding to the three workload models in Section 4. This experiment demonstrates two points: a) Randomness in SINGLER is, in fact, especially important for workloads with correlated service-times and queueing delays; and, b) The optimal SINGLER policy takes workload characteristics into account in order to maximize the value of each reissued request.

Next, we conduct a sensitivity study that varies the Queueing workload along many dimensions: utilization, service time distribution, percentile targets, strength of service-time correlations, load balancing strategies, and request prioritization strategies. The results demonstrate SINGLER is effective and robust over varying workloads and system design properties.

5.1 Simulated Workload
Figure 3 provides simulation results on a set of three workloads: Independent, Correlated, and Queueing. The service-times in each workload are drawn from a PARETO distribution with shape parameter 1.1 and mode 2.0.

In the Independent workload, the service-times of primary and reissue requests are independent and have no queueing delays (i.e., there are an infinite number of servers). In the Correlated workload, the primary and reissue request service-times are correlated via the relationship $Y = rx + x$ where $x$ is the sampled primary request service-time, $Z$ is an independently drawn service-time, and $r = 0.5$ is a linear correlation ratio. In the Queueing workload, requests have correlated service-times and arrive according to a Poisson process. The request is dispatched to the FIFO queue of one of 10 servers selected uniformly at random. The arrival rate is chosen to achieve a system utilization of 30%.

Figure 3a compares the 95th percentile tail-latency reduction achieved by the optimal SINGLER and SINGLED policies for varied reissue budgets. For the Queueing workload, both the SINGLER and SINGLED policies are selected using adaptive policy refinement (for the SINGLED policy this adaptive refinement is needed to ensure the reissue budget is satisfied). Figure 3b illustrates the "remediation rate" of SINGLER and SINGLED policies. The remediation rate measures the average value of added (i.e., actually issued) reissue requests and is defined to be the probability that a primary request $X$ exceeds a tail-latency target $t$, but the reissued request $Y$ responds before time $t - d$, i.e., $Pr(X > t \cap Y < t-d)$. Figure 3c plots the reissue times and probabilities used by the optimal SINGLER policy for each budget.

5.2 Benefits of Randomization
The results of Figure 3a illustrates the benefits of randomization in reissue policies. For all three workloads, there exists a range of reissue budgets for which the SINGLED policy is ineffective at reducing the 95th percentile tail-latency. On the Independent workload a SINGLED policy is unable to achieve any tail-latency reduction when the reissue budget is less than 5%. On the Correlated workload, SINGLED policies are ineffective for reissue budgets less than 10%. Worst of all, SINGLED policies actually increase the 95th percentile latency of the Queueing workload with reissue budgets less than 10% — since these reissued requests increase system load.

In contrast, SINGLER is able to reduce the 95th percentile tail-latency for all reissue budgets on the Independent and Correlated workloads. On the Queueing workload, SINGLER begins to reduce tail-latency once the reissue budget is greater than 3%. For all three workloads, randomization allows for SINGLER to achieve better tail-latency reduction than SINGLED for budgets less than 15%.

5.3 Impact of Correlation and Queueing
The procedure outlined in Section 4 for finding an optimal SINGLER policy takes into account the properties of the primary and reissue response-time distributions, and adapts to queueing delays. By inspecting the three workloads in Figure 3, we can gain insight into how SINGLER reissue policies are able to outperform SINGLED.

The goal of ComputeOptimalSINGLER is to find a SINGLER policy that minimizes the workload’s $k$th percentile tail-latency with a reissue budget of $B$. One can think of ComputeOptimalSINGLER as searching over all policies that use budget $B$ in order to find the policy which maximizes the value of each added request. A convenient measure of the “value” of each reissue request is its remediation rate $- i.e., the probability that the redundancy provided by the reissue request was necessary for the query to meet its tail-latency target. Figure 3c illustrates the way in which SINGLER changes its choice of reissue delay and probability based upon the reissue budget and workload characteristics. We shall discuss the behavior of SINGLER policies for each of our three workloads in the case where the reissue budget is 10%.

On the Independent workload, the optimal SINGLER policy reissues requests with probability 0.7 at a time $d$ where approximately 15% primary requests remain outstanding — resulting in approximately 10% of all requests being reissued in total. On the Correlated workload, the optimal SINGLER policy chooses to reissue requests with probability 0.4 at a time $d$ where 25% of requests are outstanding.

On the Correlated workload, the optimal SINGLER policy must reissue requests earlier due to service-time correlations. When optimizing its success rate it takes into account the fact that if a query’s primary request exceeds a tail latency target, there is a higher chance of its reissue request responding slowly. By reissuing requests earlier in time, the probability that the reissued request will help tail latency (i.e., the remediation rate) increases. Therefore, on this workload the optimal policy reissues requests earlier at a time $d$ when 40% of requests are outstanding, and reissues with a smaller probability of 25%.

On the Queueing workload, the optimal SINGLER policy reissues requests with probability 0.8 at a time $d$ where approximately 13% of requests are outstanding. Although this workload’s service-times are correlated, the latency of requests in the tail of the response-time distribution is dominated by queueing delays which depends on the service process as well as on request arrival process and load balancing. Indeed, we can observe in Figure 4b that the addition of queueing delays dampens the strength of correlation between
Figure 3: Simulation results for SingleR and SingleD policies with varied reissue budgets on three simulated workloads: Independent, Correlated, and Queueing.

(a) Tail-latency reduction ratio.  
(b) Reissue remediation rate.  
(c) Reissue time and probability.

Figure 4: Response-time correlations between primary and reissue requests on the Correlated and Queueing workloads. The service-time $X$ of the primary request is drawn from a Pareto distribution with shape 1.1 and mode 2. The service-time $Y$ of a reissued request is drawn from $Y = rx + Z$, where $x$ is the observed service-time of the primary request, $Z$ is drawn from a Pareto distribution with shape 1.1 and mode 2, and $r = 0.5$. 

(a) Correlated  
(b) Queueing
We study the sensitivity of SingleR compared to the original tail-latencies when no requests are reissued. The results demonstrate: (1) Reissue obtains higher benefit under less loaded systems, but even at rather high load of 50% utilization, SingleR achieves latency reduction of up to 1.5 times. (2) The benefit of reissue tends to increase for higher target percentiles.

**Correlation.** We use the same default Pareto distribution to model service time, and progressively increase the service time correlation ratio \( r \) between the primary request and its corresponding reissued request (defined in Section 5.1). The P95 latency without reissue is 567, and is independent of the correlation ratio \( r \). Figure 5a reports P95 latency of SingleR using a fixed reissue rate of 25% as a function of the correlation ratio. As expected, the less service-times are correlated the larger benefit reissuing has on tail-latency. Even when primary and reissue requests are strongly correlated (e.g. \( r = 1 \)) SingleR is still able to reduce the response-times of queries delayed due to queueing delays.

**Load-balancing.** Figure 5b shows the impact of different load-balancing strategies on tail latency. We make two observations: (1) Using more sophisticated load-balancing strategies such as Min-of-2 (select the server with shorter queue among two randomly selected servers to dispatch a request) or Min-of-All (select the server with the shortest queue among all servers to dispatch a request) helps reduce the P95 tail-latency relative to the simpler Random strategy that picks a server uniformly at random. (2) In all cases, SingleR reduces the P95 latency by a factor of 2 or more.
We evaluate SingleR workload’s service-time distribution, but have a substantial impact. These rare queries do little to skew the aggregate statistics of the Redis workload consists of set-intersection queries performed over a synthetic collection of 1000 sets. Query traces consist of 40,000 intersections between randomly selected pairs of sets.

The service-time distribution for the Redis set-intersection workload is illustrated in Figure 9 discretized into 20 msec bins. Over 98% of set-intersection queries in this workload have a service-time less than 10 msec. Indeed, the workload’s mean service time $\mu_R = 2.366$ milliseconds and standard deviation $\sigma_R = 8.64$ may lead us to expect request latencies to be well-behaved, even in the tail.

A handful of queries (≈ 20), however, have service times greater than 150 msec. These queries correspond to the rare case in which an intersection is performed between two abnormally large sets. These rare queries do little to skew the aggregate statistics of the workload’s service-time distribution, but have a substantial impact on tail-latency. As shown in Figure 7b, the 99th percentile tail-latency for the set-intersection workload is 900 msec when one does not reissue requests.

These “queries of death” are a common problem in database applications, and their impact on tail-latency can be difficult to predict a priori. In particular, the influence of these requests depends to a large extent on the queueing mechanisms used in the system. In Redis, requests are serviced in a round-robin fashion from each active client connection in a batch. If even a single client issues a long-running request, then the requests of all other clients will be delayed until completion. Furthermore, in an open-loop system such delays lead to a backlog of requests that further extends the impact of the slow request for multiple rounds.

6 EXPERIMENTAL EVALUATION

We evaluate SingleR policies in two distributed systems based on Redis [32] which is a key-value store that supports stored procedures, and Lucene [21] which is an enterprise search engine. Our main target is reducing the P99 tail latency.

6.1 Experimental Setup and Workloads

We use a cluster of 16 servers to execute the workload. Each server has a dual-core 2.4 GHz Intel E5-2676 processor and 32GB of RAM. The data sets and its associated indices both for Redis and for Lucene fit in the main memory. To execute each query workload, we employ several machines emulating clients that send requests in an open loop with exponential inter-arrival times.

To enable request reissuing, we assign each primary request a timestamp, and add it to a FIFO queue so that the request can be reissued later. A reissue thread consumes the entries from the FIFO queue, and dispatches the request to a server after a policy-specified delay. Prior to sending a reissue request, the completion status of its associated query is checked using a client-local boolean array.

All reported system utilizations refer to CPU utilization on a single core as measured by the Linux sysstat [11] utility. We use 10 adaptive iterations (with learning rate $\lambda = 0.5$) to compute the SingleD and SingleL policies satisfying the reissue budget. The measured reissue rate and the target reissue budget tend to closely agree with the predictions of the reissue policy optimizer and thus we report only the empirical rate in all figures.

6.2 Redis Set-Intersection Workload

The Redis workload consists of set-intersection queries performed over a synthetic collection of 1000 sets. Each set stores a random subset of integers in the range $1$ and $10^6$, and set cardinalities are distributed according to a lognormal distribution. Query traces consist of 40,000 intersections between randomly selected pairs of sets.

The service-time distribution for the Redis set-intersection workload is illustrated in Figure 9 discretized into 20 msec bins. Over 98% of set-intersection queries in this workload have a service-time less than 10 msec. Indeed, the workload’s mean service time $\mu_R = 2.366$ milliseconds and standard deviation $\sigma_R = 8.64$ may lead us to expect request latencies to be well-behaved, even in the tail.

A handful of queries (≈ 20), however, have service times greater than 150 msec. These queries correspond to the rare case in which an intersection is performed between two abnormally large sets. These rare queries do little to skew the aggregate statistics of the workload’s service-time distribution, but have a substantial impact on tail-latency. As shown in Figure 7b, the 99th percentile tail-latency for the set-intersection workload is 900 msec when one does not reissue requests.

These “queries of death” are a common problem in database applications, and their impact on tail-latency can be difficult to predict a priori. In particular, the influence of these requests depends to a large extent on the queueing mechanisms used in the system. In Redis, requests are serviced in a round-robin fashion from each active client connection in a batch. If even a single client issues a long-running request, then the requests of all other clients will be delayed until completion. Furthermore, in an open-loop system such delays lead to a backlog of requests that further extends the impact of the slow request for multiple rounds.

Tail Latency Reduction under SingleL and SingleD. On the Redis workload, the SingleL and SingleD policies are able to reduce the P99 latency at 40% utilization from 900 milliseconds to 400 milliseconds. SingleL is able to meet this target P99 latency with a budget of just $3.5\%$, whereas SingleD requires a budget of at least $5\%$. For reissue budgets between 3 and 5%, the reissue probability of SingleL increases from 0.8 to 1.0 so that for budgets greater than $5\%$ SingleL and SingleD are equivalent.

Figure 7a shows the P99 latency in msec (Y-axis) against the reissue rate between 0 and 6% (X-axis) for SingleL and SingleD. Both Redis (top figure) and Lucene (bottom figure) running at 40% baseline utilization without any reissue.

We make two observations: First, both the SingleL and SingleD curves illustrate reduced tail latency relative to the baseline system without reissuing. Second, we note that the SingleL policy achieves strictly better tail-latencies than SingleD for small reissue rates. For example at 2% reissue rate in Redis, SingleL reduces the P99 latency to 405 msec, compared to 900 msec for the baseline system and 820 msec for SingleD.

Varied System Utilization. Next, we illustrate the performance of SingleL under three system utilization levels: 20%, 40%, and 60% in Figure 7b, which depicts the P99 latency against fixed reissue rate. For all utilizations between 20–60%, SingleL is able to reduce the 99th percentile tail latency.

In particular, at 60% utilization (which is very high for interactive services) SingleL with a 3% reissue rate reduces the Redis P99 latency from 1750 msec to 1000 msec, and the Lucene P99 latency from 1603 msec to 1157 msec.

The best latency reduction occurs when choosing the optimal reissue rate, which depends on the system utilization. For 20% utilization, we illustrate the process of finding the optimal reissue rate via binary search in Figure 8. At 20% utilization the best reissue budget is approximately 8%. At both 40% and 60% utilization, the best reissue rate is approximately 5%.

Figure 7c illustrates the best tail-latency achieved by a SingleL policy for the Redis workload for utilizations between 20% and 60%. The Best Reissue Rate curve illustrates the P99 latency achieved by the best SingleL policy (with reissue rate determined via a binary search procedure), and the No Reissue curve illustrates the P99 latency of the baseline system without reissuing.

6.3 Lucene Search Workload

The Lucene search workload consists of search queries over a corpus of 33 million articles from the English Wikipedia dataset [9]. Queries are drawn randomly from a set of 10,000 queries from the Lucene nightly regression tests [23].

The service-time distribution for queries in the search workload contrasts with set-intersection in that it is not as highly skewed towards very-low latencies. The distribution for search service-times is illustrated in Figure 9 discretized into 20 msec bins. Approximately 90% of all requests have service times between 1 and 70 msec, and the overall distribution has mean service-time $\mu_L = 39.73$ msec with standard deviation $\sigma_L = 21.88$.

Similar to the set-intersection workload, the search workload also has rare slow queries. Approximately 1% of search queries have service-times greater than 100 msec. The impact of these slow queries on tail-latency, however, is different in Lucene than in Redis. At 40% utilization, the search workload’s 99th percentile tail latency is reduced by 8% to 1000 msec. For 60% utilization, the reduction in latency is 40%, from 1750 msec to 1000 msec.

Prioritized LIFO is the same as Prioritized FIFO, however, does not reissue requests. A primary request is reissued if the reissue queue is empty: preventing multiple reissued requests from delaying a primary request. (3) Prioritized FIFO is the same as Prioritized FIFO but processes the reissue queue in LIFO order.

Figure 5c compares the three systems and shows that changing the priority scheme has a modest impact on the tail latency improvements of SingleR.

The overall results in this section show that SingleR and its adaptive policy optimizer is robust and reduces the tail latency for these different system design choices and workload characteristics.
Figure 7: System experiment results for the Redis and Lucene workloads. Figure 7a compares the P99 latency of SingleR and SingleD for reissue budgets between 0 and 6% at 40% utilization. Figure 7b shows the P99 latency for SingleR with varied reissue rates for 20%, 40%, and 60% utilization. Figure 7c shows the P99 latency achieved when using the best reissue budget and a SingleR policy for utilizations ranging from 20% to 60%.

Figure 8: Illustration of binary search for optimal budget for the Intersection Counting query to minimize 99th percentile tail-latency. 20% utilization.

Figure 9: Service time distributions for the Redis set-intersection and Lucene search workloads.
latency is $\approx 435$ msec when there are no reissuved requests. This is not entirely due to the differences in service-time distribution, although it certainly is an important influence. The Lucene search server also differs in how it manages concurrent requests. Requests from all open connections are placed into a single FIFO queue which results in relatively good tail-latency behavior — FIFO is, in fact, optimal for light-tailed service-time distributions [28].

Tail Latency Reduction under SingleR and SingleD. On the Lucene workload, we observe in Figure 7a that SingleR reduces Lucene’s P99 latency at 40% utilization from 433 milliseconds to 339 milliseconds, and the SingleD policy reduces P99 to 346 milliseconds. This gap, while small, is not merely measurement noise — all reported values reflect the median of multiple runs.

The improved performance of the SingleR policy is due to its use of randomization that allows it to reissue queries earlier than SingleD. At 40% utilization, the optimal reissue rate for SingleR is 4%, and the optimal policy reissues requests with probability approximately 0.75. As the reissue rate grows the achieved latency gap between SingleR and SingleD closes, and the reissue probability of the optimal SingleR policy converges to 1.0. Randomization is more valuable on the Lucene search workload than it was for Redis because of the much higher mean service time of requests.

Varied System Utilization. Next, we illustrate the performance of SingleR under three system utilization levels: 20%, 40%, and 60% in Figure 7b, which depicts the P99 latency against fixed reissue rate. SingleR reduces the tail-latency of Lucene search workload for all utilizations between 20–60%. At 60% utilization (high load), SingleR reduces the P99 latency from 1603 to 1157 msec. Figure 7c illustrates the best tail-latency achieved by a SingleR policy for the Lucene workload for utilizations between 20% and 60%. The Best Reissue Rate curve illustrates the P99 latency achieved by the best SingleR policy (with reissue rate determined via a binary search procedure), and the No Reissue curve illustrates the P99 latency of the baseline system without reissuing. We observe significant tail latency reduction due to SingleR over the baseline.

7 CONCLUSION

We have illustrated principled methods of generating reissue policies for interactive services. By operating within a simplified model, we were able to prove that SingleR is an optimal compromise between the commonly used immediate and delayed reissue strategies. There are a few general lessons that we think are useful to impart: a) there is little reason to choose a reissue policy more complex than SingleR if that additional complexity does not leverage application-specific insight; and, b) reissue policies that reduce tail-latency as a “smooth” function of their budget admit relatively simple strategies for adapting to load-dependent queueing delays and searching for optimal reissue budgets. As we have shown, we were able to adapt SingleR policies to systems and workloads with a wide range of properties through iterative adaptation. As we have seen, this leads to a simple process for finding effective reissue policies in real systems: SingleR is able to reduce tail-latency in simulated and real-world workloads even when reissuing a small fraction of requests.

REFERENCES

7. Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Plakhin, Swamnathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: amazon’s highly available key-value store. In SOSP.