Latency-Redundancy Tradeoff in Distributed Read-Write Systems

Saraswathy Ramanathan, Gaurav Gautam, Vikram Srinivasan, and Parimal Parag
Dept. of Electrical Communication Engineering, *Indian Institute of Science*, Bangalore, India, 560012
Email: {saraswathyr, gauravgautam, vikramsriniv, parimal}@iisc.ac.in

2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS) | 978-1-6654-2104-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/COMSNETS53615.2022.9668414

***Abstract*—Data is replicated and stored redundantly over multiple servers for availability in distributed databases. We focus on databases with frequent reads and writes, where both read and write latencies are important. This is in contrast to databases designed primarily for either read or write applications. Redundancy has contrasting effects on read and write latency. Read latency can be reduced by potential parallel access from multiple servers, whereas write latency increases as a larger number of replicas have to be updated. We quantify this tradeoff between read and write latency as a function of redundancy, and provide a closed-form approximation when the request arrival is Poisson and the service is memoryless under prioritized reads. We empirically show that this approximation is tight across all ranges of system parameters. Thus, we provide guidelines for redundancy selection in distributed databases.**

I. Introduction

Storage systems are designed with specific applications in mind. In this article, we focus on the systems where read and write are both frequent, and we would refer to these as *read-write systems*. Some examples of cloud systems with frequent reads and writes are banking, personal storage, e-commerce, etc. Cloud storage systems like Dropbox, GitHub, OneDrive, Google Drive etc. have frequent updates (writes) to the same file and benefits from study of systems with frequent reads and writes. In a personal storage cloud like Dropbox, the daily average of uploaded files is 1.2 billion. Dropbox receives 1.67 billion API calls in a day, of which 345.6 million ($\sim 21\%$) are edits to files [1]. State Bank of India receives around 131.16 million transactions per month[2] from 296.82 million page visits[3]. Out of the total number of queries sent to the bank cloud servers, 44% are write requests. We will focus on the latency performance of these systems, which is an important user requirement that has monetary impact.

Distributed read-write systems are employed in many modern storage and computing architectures, for graceful scaling up. There are several important considerations in the design and implementation of such distributed systems, such as consistency, latency, availability, storage cost, among others. Availability in the event of failures, is ensured by redundant storage of data over multiple servers, in these systems.

This work is supported in part by the Centre for Networked Intelligence (a Cisco CSR initiative) of the Indian Institute of Science, Bangalore.

[1]https://expandedramblings.com/index.php/Dropbox-statistics/

[2]https://www.business-standard.com/company/st-bk-of-india-1375/annual-report/director-report

[3]https://www.similarweb.com/website/retail.onlinesbi.com/\#overview

Most commercial distributed database systems provision for eventual consistency [1], [2], where read requests can access an older version of data. We consider the read and write latencies for eventually consistent systems. In addition, we adopt the primary-secondary architecture with redundant replication for distributed read-write systems as shown in Fig. 1. This architecture is employed by popular databases such as MySQL, DynamoDB, MongoDB, PostgreSQL, etc. As shown in Fig. 1, write requests arrive at the designated *primary* server, and the remaining *secondary* servers copy the written data from the primary. In contrast, multiple read requests to a file can be served simultaneously. Data read requests can be directed to any server in the cluster holding a copy of the data. Often read and write requests are stored in separate queues, and depending on the application, one of them is prioritized over the other. We consider distributed read-write systems for read priority.

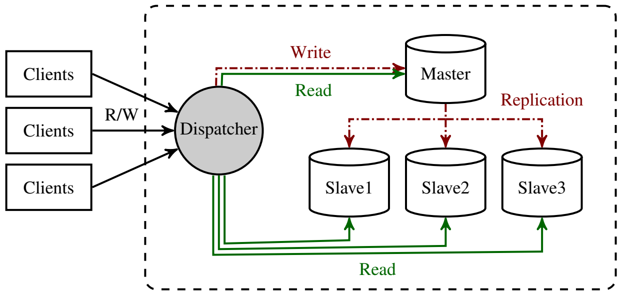


Fig. 1: Distributed read-write system with primary-secondary architecture.

In this work, we are interested in I/O bound distributed database systems with geographically co-located data, where all the requests suffer from a similar negligible network delay. Accordingly, our focus is on the queueing delay suffered by the read and write requests. This is the scenario for many SaaS companies hosted on cloud service providers, with the database situated in one or two availability zones. Our analysis framework is also suitable for the cases when network latency does not scale with redundancy, and can be accounted for by an additional network latency. Redundant storage of data is advantageous for read latency, as it allows for parallel access. It can be shown[4] that in many practical situations, the read latency decreases with increase in redundancy. Contrastingly,

[4]https://docs.gitlab.com/ee/administration/database_load_balancing.html

the write latency increases with redundancy [3], [4], as a write should be completed at all redundant copies of the data. This alludes to a tradeoff between read and write latency with increased redundancy, as illustrated in Fig. 2, where we observe that there exists an optimal redundancy that minimizes the average request latency (averaged over all read and write requests) for a single file. A quantitative characterization of read and write latency is the main objective of this article. We note that, the read and write latencies can be weighted depending on the application. For simplicity, we consider the case when they are equally weighted.

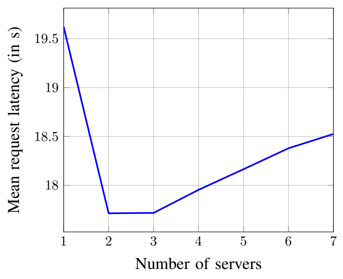


Fig. 2: Empirical mean latency of requests in a distributed read-write system, with increasing number of servers.

A. Related Work

Redundancy schemes and request scheduling for read latency reduction in distributed storage systems has been studied in [5]–[13]. Latency for a single class of requests with coded redundancy in distributed storage and compute systems has been studied in [5], [10], [11], [14]–[17]. Incoming requests are forked to all redundant servers, and a request is considered completed when a subset of servers finish completion. These systems are called *fork-join* queues and have been studied for memoryless arrivals and service in [7], [14], [18]–[20]. For an eventually consistent read-write system with replication redundancy and instantaneous reads, staleness of reads is characterized in [4]. All the above mentioned works focus on a single class of requests (either read or write), assuming either an instantaneous service for the other request class or immutability of data. In practical storage systems, however, the data is written and read with non-negligible workload from both. We focus on distributed systems where the servers are deployed within the same availability zone and thus have similar network latency for all requests. Thus, the network latency does not scale with redundancy and can be ignored, unlike [21] that study latency in geo-distributed systems.

B. Main contribution

We analytically compute the read and write latency for a single file in a distributed read-write system, and obtain the optimal redundancy to minimize the aggregate latency.

1) Unlike previous works on latency reduction, we characterize the read and write processes separately, while accounting for their joint presence. This allows the optimization problem to be tailored for different applications based on read and write latency constraints.
2) Further, we provide approximations with closed-form expressions for latency redundancy trade-off, that can be used for large-scale system design. We remark that the Markov chain considered is more complex than previously studied fork-join queues. These queues have not been analytically studied in the literature to the best of our knowledge. We empirically show that proposed approximations remain tight over the entire range of system parameters in the system stability region.
3) As a consequence of our analysis, we show that the optimal number of servers depends on the traffic pattern in the system. Hence, from the system design perspective, it is not always beneficial to set redundancy factor to the typical value of two.
4) We conducted numerical experiments for read-write systems with non-memoryless service distributions. In addition, we performed empirical experiments on real world read-write systems. We observed the existence of optimal redundancy in both of these situations, which confirms that the insights obtained from our theoretical studies continue to hold in general.

II. System model

We consider a distributed read-write system with a primary-secondary architecture. In such systems, write occurs at the primary first and then replicated at the secondary servers, whereas reads can occur from any server. For simplicity, we focus on a single file stored at the primary and n secondary servers. However, the framework can be extended to study systems with multiple files as well.

A. Arrivals of read/write requests

We assume that the read and the write requests for the file arrive as Poisson processes with rates λ_r and λ_w respectively. This is a widely accepted model for arrivals in distributed storage [11], [14], [22] and caching systems [23]–[25]. This assumption is motivated by analytical tractability, and the fact that this is a good approximation for the arrivals [26]when a large number of independent clients are reading from and writing to the system. In the following subsections, we discuss the modeling assumption on read and write processes separately.

B. Dispatch of read/write requests

We can distinguish read and write request arrivals as two separate classes of arrivals. Note that all $(n+1)$ servers receive both read and write requests.

1) Read requests: We assume that incoming read requests are dispatched to one of the $(n+1)$ servers uniformly at random, independent of all other decisions. This is thinning of the Poisson process [27],and hence the read request arrival at each server is a Poisson process with rate $\frac{\lambda_r}{(n+1)}$. We note that in a typical distributed read-write system with primary-secondary architecture, incoming read requests are directed to one of the $n+1$ servers in a round-robin fashion [28].We remark that even though optimal routing scheme would be to join the shortest queue or a variant, these schemes have communication overhead. Therefore, many practical systems employ round robin scheduling for simplicity. This results in an effective arrival rate of $\lambda_r/(n+1)$ at each server, identical to the Poisson arrival rate achieved by the random splitting of Poisson process. Since the two arrival process only differ in higher order moments, we assume the random splitting for analytical tractability.

2) Write requests: In a primary-secondary architecture, the write request joins the write queue at the primary. After the write is completed at the primary, the request is forked to all n secondary servers. A write is considered completed, if write request is completed at all n secondary servers.

C. Scheduling

We assume there is a priority order between read and write classes, and requests within a class are served in a first come first serve (FCFS) manner at each server.

1) Priority between classes: In many distributed read-write systems, one class has priority over the other[5] [28].In practical systems, priorities are non-preemptive [29].That is, if a request of higher priority class arrives when the server is serving a request of low priority class, then the higher priority request has to wait until the request in service is completed. For simplicity, we consider preemptive resume priority [30], [31], where a high priority arrival replaces a low priority request from service, and the low priority request resumes its service once there are no more high priority requests. This reduces the state space, and makes analysis tractable. We will consider a system with read priority.

2) FCFS within a class: In practical systems, write requests are served in an FCFS manner [32],while the read requests are served by processor sharing[6] [28].According to the insensitivity property [33],all work conserving policies that do not depend on service time of requests have the same mean waiting time. Since we are only interested in the expected behavior of the system for our analysis, we assume that both read and write requests are served in FCFS manner within their class.

D. Execution of read/write requests

The uncertainty in the execution time of the request at each server occurs due to independent background processes at individual servers, and hence the execution time can be modeled by independent random variables, both across the servers and the requests. We also assume that homogeneous servers such that each execution time has identical distribution, depending on the request class.

1) Read requests: Real traces from Amazon S3 show that the empirical distribution of time to retrieve a fixed size chunk of data can be well approximated by an exponential distribution [8]. This assumption makes the analysis tractable, and is a popular assumption for content download time in literature [5], [7], [14], [18], [34]. Accordingly, we assume that the read times are *i.i.d.* across read requests and servers, distributed exponentially with rate μ_r.

[5]https://mariadb.com/kb/en/high_priority-and-low_priority/
[6]https://www.sqlshack.com/locking-sql-server/

2) Write requests: It has been shown that the random write latency in distributed storage systems, can be well modeled by shifted exponential distribution [4], [9], [10], [35], [36]. The shifted exponential distribution can be approximated to an exponential distribution when the constant shift is much smaller than the mean of the distribution. Since exponential distributions offer analytical tractability, we consider the case when the constant shift is negligible in empirical write distributions. Hence, we assume that the write times are *i.i.d.* across write requests and servers, distributed exponentially with rate μ_w.

E. Performance Metric

For an $(n+1)$ server system, with arrival and service rate pairs (λ_r, μ_r) for read requests and (λ_w, μ_w) for write requests, we will measure the system performance by the limiting average of number of requests in the system. We denote aggregate read and write load on the system by $\rho_r \triangleq \lambda_r/\mu_r$ and $\rho_w \triangleq \lambda_w/\mu_w$ respectively. Since write requests join all $(n+1)$ servers, and read requests join one of the $(n+1)$ servers, the system is stable if $\rho_w + \rho_r/(n+1) < 1$. Let $M_n(t)$ denote the number of requests in the $(n+1)$-server system at time t, then the limiting average number of requests is $\bar{M}_n = \lim_{t\to\infty} \frac{1}{t}\int_{s=0}^{t} M_n(s)ds$.

From Little's law [37],we know that the limiting mean number of requests is directly proportional to the limiting mean sojourn time of requests in the system. Performance metric of choice here is the number of requests in the system, which is sum of the number of requests of individual classes. Implicitly, we have assumed that the read and write requests are of equal importance in our work. However, since we separately compute the number of read and write requests in the system, our framework can be used for any performance metric that is a function of the two numbers.

We will show that the number of secondary servers n is an important system parameter that controls the system performance. Specifically, we will show that under certain traffic and service parameters, there exists an optimal choice of number of secondary servers n, that minimizes the limiting average of number of requests in the system. Formally, we solve the following problem.

Problem 1. For the distributed read-write system described above, find the optimal number of servers n^* such that $n^* = \arg\min_n \bar{M}_n$. In particular, we will find the optimal number of servers for read-first priority. We will assume that the system is stable for all $n \in \mathbb{Z}_+$, i.e. $\rho_r + \rho_w < 1$. Further, we assume finite write load on the system, i.e. $\rho_w > 0$.

III. Background

For our system model, read requests are easier to understand. Since read arrivals get routed to one of $(n+1)$ servers uniformly at random, the read requests queues would have remained independent if there were no write requests in the system. The write requests arrive at the primary, and are sent to all n secondary servers at the instant of write completion at the primary. A request is considered completed, if it is completed at all n servers. That is, a write request is forked to all n secondary servers, and joins after service from all n of them. This is precisely the setting of (n, n) fork-join queues [38].

In this section, we will just focus on the evolution of an (n, n) fork-join system for a single class of requests. We study the impact of preceding primary server in Section IV. Specifically, we consider an n server system, where request arrival is a Poisson process with rate λ, and the service time of each request at all servers is assumed to be *i.i.d.* exponential with rate μ. An arriving request is forked to all n servers, and it leaves a server after service. Customers are served in an FCFS manner at each server.

Let $X_j(t)$ be the number of requests in the queue j, then we make two observations. The evolution of each queue follows an $M/M/1$ queue and the total number of requests in the system at time t is given by $\max\{X_j(t) : j \in [n]\}$. The limiting distribution of number of requests in each of the n queues can be computed easily. If the queues were independent, then this also gives us the distribution of total number of requests in the system. However, since all queues get new requests at the identical arrival instant, they are not independent. Nevertheless, we do have bounds on the limiting mean number of requests. A simple lower bound is derived from the application of Jensen's inequality to convex function $\max$, and an upper bound is derived from the fact that $\max$ is upper bounded by the sum. That is, we have

$$\max_{j\in[n]} \lim_{t\to\infty} \mathbb{E}X_j(t) \leqslant \lim_{t\to\infty} \mathbb{E} \max_{j\in[n]} X_j(t) \leqslant \sum_{j\in[n]} \lim_{t\to\infty} \mathbb{E}X_j(t).$$

We observe that these bounds can be very loose in high load settings. Therefore, we consider the alternative way of viewing fork-join system as tandem queues with pooled service as proposed in [18]. We adapt this approach to study the read priority systems under consideration. We propose a new approximation for read priority system, under which the computed approximate mean number of write requests in the system is shown to be close to the empirical mean in the original system.

A. Tandem queue approach and approximation

An alternative way of state representation of fork-join queues at any time t, is the sequence of set of servers that have served each request in the system [18]. Each incoming request is served in FCFS fashion at each server and is forked instantaneously to all n servers. Therefore, the set of servers serving the newer requests are the ones that have already served the older requests. From the homogeneity in the system, it follows that the identities of servers do not matter, and the system state is sufficiently represented by the number of servers that have served each request in the system. From the FCFS service discipline, it follows that older requests are served by the number of servers no less than the newer requests. Therefore, one can partition all requests in the system by the number of servers that have served it. Accordingly, let $Y_i(t)$ denote the number of requests in the system that have been served by i servers at time t, and denote $Y(t) \triangleq (Y_0(t), \dots, Y_{n-1}(t))$. It turns out that $(Y(t), t \geqslant 0)$ is a sufficient state representation for a fork-join queue. We call $Y_i(t)$ to be the number of requests in level i. We observe that each incoming arrival is served by 0 servers, and hence this increases $Y_0(t) \to Y_0(t) + 1$ at the arrival instant t. Further, when a request is served by all n servers, it departs the system and decreases $Y_{n-1}(t) \to Y_{n-1}(t) - 1$ at the departure instant t. When a request is served by $i < n$ servers, then it becomes a request with i server completions from $(i-1)$ server completions. Correspondingly, we have $Y_{i-1}(t) \to Y_{i-1}(t) - 1$ and $Y_i(t) \to Y_i(t) + 1$ at this service completion instant t. That is, $Y(t)$ has a tandem queue interpretation with queues 0 to $(n-1)$ from left to right, with external arrivals to queue 0 and external departures from queue $(n-1)$. A departure from queue $(i-1)$ leads to an arrival to queue i.

For each partition of requests that have been served by i servers, can only be served by remaining $(n-i)$ servers. At these $(n-1)$ server queues, there maybe requests that have been served by $(i+1)$ or more servers. The requests with $(i+1)$ or more completions are older than the requests with i completions, and hence are served first due to FCFS service discipline. We let $N_i(t)$ denote the number of servers at time t, whose head request has been served by exactly i servers. That is, the requests in level i of the tandem queue, are served by $N_i(t)$ servers in parallel at time t. We have illustrated the tandem queue interpretation of the system in Figure 3, where a request served by i servers moves to level $i+1$ once it receives service from one of the $N_i(t)$ servers.

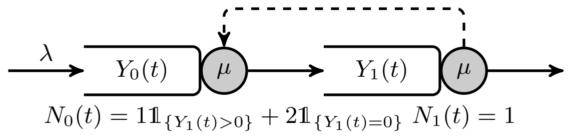

λ
$Y_0(t)$
μ
$Y_1(t)$
μ
$N_0(t) = 1\mathbb{1}_{\{Y_1(t)>0\}} + 2\mathbb{1}_{\{Y_1(t)=0\}}$
$N_1(t) = 1$

Fig. 3: Write requests in the system represented as a tandem queue with pooled servers.

Proposition 1. *For the (n, n) fork-join queuing system under consideration, the vector $Y(t)$ represents the occupancy of an n-tandem queue at time t. If $Y(t) = y$, then the number of servers serving the head request at ith tandem queue is denoted by $N_i(t) = N_i(y)$ such that*

$$N_i(y) = \begin{cases} 1, & i = n-1, \\ 1 + N_{i+1}(y)\mathbb{1}_{\{y_{i+1}=0\}}, & i < n-1. \end{cases}$$

Proof. We just provide a proof sketch here, which is adapted from [18].Each request joins all n queues, gets served in an FCFS manner at each of them, and leaves the system when it has been served by all n servers. When a request gets serviced at a server, we say that the corresponding server has been observed by the request. Due to FCFS service policy, the subset of observed servers for each request in the system forms a chain [18],i.e. the set of observed servers for an older request in the system contains the set of observed servers of newer request. Thus, we can aggregate all the requests with identical set of observed servers. Then the number of requests with identical set of i observed servers is denoted by $Y_i(t)$ at time t. We note that because of FCFS service policy, only the oldest of the Y_i requests is in service. Since requests leave after observing n servers, we have $i \in \{0,\dots,n-1\}$ and the number of observed servers i is referred to as level i. One can see that Y is the number of requests in a tandem queue with n levels. Incoming requests have not observed any server and hence join level 0. When a request with i observed servers, gets served by another useful server, it leaves level i, and joins level $i+1$. A request departs the system from level $n-1$.

For an (n,n) fork-join system, the set of useful servers for a request is the set of all servers without the observed servers. The number of useful servers for the requests in level i is $n-i$. However, some of these servers are useful to the requests ahead of them as well, due to chain property of observed servers. The number of available servers for requests with i observed servers is denoted by $N_i(t)$ at time t. For $i=n-1$, the number of useful and available servers remain same and equal to 1, since requests with n observed servers leave. A request with i observed servers has one available server not useful to requests with $(i+1)$ observed servers, and it can use the available servers from the $(i+1)$th level, if there are no requests with $(i+1)$ observed servers. Since, this relation only depends on time t through the occupancy vector Y, we get the result. ■

Proposition 2. *For the (n,n) fork-join queuing system described above, the occupancy vector $(Y(t):t\in\mathbb{R}_+)$ forms a continuous-time Markov chain, with possible transitions*

$$a_i(y) \triangleq \begin{cases} y+e_0, & i=0,\\ y+e_i-e_{i-1}, & i\in[n-1], y_{i-1}>0,\\ y-e_{n-1}, & i=n, y_{n-1}>0,\end{cases}$$

and the corresponding transition rates are

$$Q(y,a_i(y)) = \lambda \mathbb{1}_{\{i=0\}} + N_{i-1}(y)\mu \mathbb{1}_{\{y_{i-1}>0\}}\mathbb{1}_{\{i\in[n]\}}.$$

Proof. In the proof of Proposition 1, we observed that the occupancy vector $Y(t)$ has a tandem queue interpretation. We will show that the random time to next transition depends only on the current state, and has an exponential distribution. Further, conditioned on the current state, the jump probability of next transition is independent of the random transition time, and previous jump transitions. This shows that process Y is a continuous-time Markov chain [39].

Recall that there are three types of possible transitions from the current state y, namely an external Poisson arrival to level 0, a service for request with $(i-1)$ observed servers that leads to departure of this request from level $(i-1)$ and arrival to level i for $i<n$, and a service for request with $(n-1)$ observed servers that leads to an external departure from the system. Recall that the number of servers available to level $(i-1)$ is $N_{i-1}(y)$, each server has an *i.i.d.* exponential service time with rate μ, and the inter-arrival times for external arrivals are *i.i.d.* exponential with rate λ. Therefore, the residual times are independent and exponentially distributed.

The time for next transition is minimum of all these random times. Conditioned on the current state, next inter-transition time is an exponential random variable independent of the past, with rate given by the sum $\lambda + \mu\sum_{i=1}^{n} N_{i-1}(y)\mathbb{1}_{\{y_{i-1}>0\}}$. Further, the probability that one of these transitions take place is given by the ratio of the transition rate and the sum-rate. This probability is independent of the inter-transition time and past transitions, given the current state y. ■

This Markov process is interpreted as a tandem queue, as shown in Figure 3, where each level i has its own dedicated service rate μ, which gets pooled in the levels below when level i is empty. To compute the mean number of write requests, we need to find the invariant distribution of the Markov process Y. However, this problem remains intractable since it is equivalent to finding an eigenvector of an n-dimensional operator with eigenvalue unity. Further, the Markov process Y is not reversible, and hence there are no known techniques to find the invariant distribution of this process. However, a tight reversible approximation of this process was proposed in [18], which we quote here for reference.

Approximation 1. *The pooled tandem queue Y with dedicated rates $(\mu,\dots,\mu)$ is approximated by a continuous-time Markov process $\bar{Y} \triangleq (\bar{Y}(t)\in\mathbb{Z}_+^n, t\in\mathbb{R}_+)$, which is an unpooled tandem queue with Poisson arrival rate λ and exponential service rates $(\bar{\gamma}_0,\dots,\bar{\gamma}_{n-1})$, where the service rate for level i in unpooled tandem queue is*

$$\bar{\gamma}_i \triangleq (n-i)\mu - (n-i-1)\lambda. \tag{1}$$

IV. Read priority system

For systems where consumers are not sensitive to the timeliness of data, but sensitive to the read latency, read requests are prioritized. Examples of such systems are video streaming, cataloging, data mining, content management systems, etc. In this section, we analyze the distributed read-write systems with priority for read requests. Recall that incoming read requests are routed to one of the $(n+1)$ servers, uniformly at random. At any server in a read priority system, any write request in service is preempted by an incoming read request. Let $R_j(t)$ be the number of read requests at server j at time t. The evolution of $R_j(t)$ is equivalent to an $M/M/1$ queue with Poisson arrivals of rate $\lambda_r/(n+1)$ and exponential service of rate μ_r. The read load on any server j is $\rho_r/n+1$.

Remark 1. To show the explicit dependence of mean number of read requests on the number of redundant secondary servers, we denote this mean by $q(n)$ in the read priority system. Due to linearity of expectation and the evolution of $R_j(t)$ at all servers as identical $M/M/1$ queues, we can write the mean number of read requests in the read priority system as

$$q(n) \triangleq \sum_{j=0}^{n} \mathbb{E}R_j = (n+1)\mathbb{E}R_0 = \frac{(n+1)\rho_r}{n+1-\rho_r}.$$

We observe that the mean number of read requests in the read priority system is decreasing in number of redundant servers n, and saturates to the read load ρ_r.

We next focus on the number of unique write requests in the system, denoted by $W(t)$ at time t. Recall that write requests initially join the primary server, and upon service completion join all the n secondary servers instantaneously. A write request leaves the system, when it has been served by all the servers. Since a write request is preempted when an incoming read request arrives at that server, we cannot use Approximation 1 for the write process as it is coupled with number of read requests in the system.

Let $W_0(t)$ denote the number of write requests in the primary server at time t, then $W(t) - W_0(t)$ denotes the number of write requests forked at n secondary servers. Service for forked write requests at server j depends on the number of read requests $R_j(t)$ at this server as well. For each write request $k \in [W(t) - W_0(t)]$ ordered by their arrival time, we define $S_k(t) \subset [n]$ to be the set of secondary servers that has already served this write request at time t. Note that the write request exits the system after being served by all n secondary servers. Since the system follows FCFS policy, the later arrivals only get served by the servers that have already served the previous arrivals. In particular, we observe that the $(k+1)$th request cannot be served by a server before it has served the kth request. Therefore,

$$\emptyset \subseteq S_{k+1}(t) \subseteq S_k(t) \subset [n], \quad k, k+1 \in [W(t)-W_0(t)]. \quad (2)$$

We write the sequence of set of observed servers for all forked write requests in the system as $S(t) \triangleq (S_k(t) : k \in [W(t) - W_0(t)])$, and we denote the number of read requests at $(n+1)$ servers by $R(t) \triangleq (R_j(t) : 0 \leqslant j \leqslant n)$. Then, we can denote the state of the system at time $t \in \mathbb{R}_+$ by $Z(t) \triangleq (W_0(t), S(t), R(t)) \in \mathcal{Z} \triangleq \mathbb{Z}_+ \times [n]^* \times \mathbb{Z}_+^{\{0,\dots,n\}}$.

Theorem 3. *For a distributed read-write system with priority for read requests, the random process $(Z(t), t \in \mathbb{R}_+)$ forms a continuous-time Markov chain. For a state $z = (w_0, s, r) \in \mathcal{Z}$, defining $s_k^j \triangleq (s_1, \dots, s_k \cup \{j\}, \dots)$ and $s' \triangleq (s_2, \dots, s_{|s|})$, the associated generator matrix Q is given by*

$$\begin{aligned} Q(z, z') &= \frac{\lambda_r}{n+1} \sum_{j=0}^{n} \mathbb{1}_{\{z'=(w_0,s,r+e_j)\}} \\ &+ \mu_r \sum_{j=0}^{n} \mathbb{1}_{\{z'=(w_0,s,r-e_j)\}} \mathbb{1}_{\{r_j \geqslant 1\}} + \lambda_w \mathbb{1}_{\{z'=(w_0+1,s,r)\}} \\ &+ \mu_w \mathbb{1}_{\{z'=(w_0-1,(s,\emptyset),r)\}} \mathbb{1}_{\{r_0=0\}} \mathbb{1}_{\{w_0 \geqslant 1\}} \\ &+ \mu_w \Bigg(\sum_{j:r_j=0} \sum_{k>1} \mathbb{1}_{\left\{z'=(w,s_k^j,r)\right\}} \mathbb{1}_{\{j \in s_{k-1} \setminus s_k\}} + \mathbb{1}_{\{j \notin s_1\}} \\ &\Big(\mathbb{1}_{\left\{z'=(w,s_1^j,r)\right\}} \mathbb{1}_{\{|s_1|<n-1\}} + \mathbb{1}_{\{z'=(w,s',r)\}} \mathbb{1}_{\{|s_1|=n-1\}} \Big) \Bigg). \end{aligned}$$

Proof. We observe that system state can change if there is (a) an external arrival of read request, (b) a read request gets serviced, (c) an external arrival of write request, and (d) a write request gets serviced. Since all distributions are continuous and arrival and service times are independent, only one of these events take place in an infinitesimal time. We will first focus on state transitions due to read requests, since their evolution is easier to understand when they are prioritized.

(a) External arrival of a read request: External read arrival at each server is an independent Poisson process of rate $\frac{\lambda_r}{n+1}$. Therefore, an external arrival of a read request to server $j \in [n]$ changes the state $(w_0, s, r) \to (w_0, s, r + e_j)$, and the inter-transition time for such transitions are independent for all $j \in [n]$ and distributed exponentially with rate $\frac{\lambda_r}{(n+1)}$.

(b) Service of a read request: Service time for read requests at each server is independent and memoryless with rate μ_r, and hence the inter-transition time for transitions $(w_0, s, r) \to (w_0, s, r - e_j)$ are independent and memoryless with rate μ_r for all $j \in [n]$ such that $r_j \geqslant 1$.

We next focus on the evolution of write requests in the system. Specifically, we look at the arrival and service of write requests.

(c) External arrival of a write request: External arrival of write requests is Poisson with rate λ_w, and the incoming write requests initially join the primary server. This leads to an increase in number of write requests w_0 at the primary. Thus, the inter-transition time for transitions $(w_0, s, r) \to (w_0 + 1, s, r)$ are independent and memoryless with rate λ_w.

(d) Service of a write request: We first focus on service of an existing write request at primary server 0, which can only happen when there are no read requests at the primary server. This request departs from the primary server upon service completion, and is forked to all n secondary servers. This request has not been served by any secondary servers at this instant. Since the service time for write requests are *i.i.d.* and memoryless with rate μ_w, the inter-transition time for transitions $(w_0, s, r) \to (w_0 - 1, (s, \emptyset), r)$ are independent and memoryless with rate μ_w, and they occur when $r_0 = 0$ and $w_0 \geqslant 1$.

We next focus on servers $j \in [n]$ without any read requests, that can serve write requests. Recall that service time for write requests at each server is *i.i.d.* and memoryless with rate μ_w, and the sequence of set of secondary servers that have finished serving existing $|s|$ write requests are $(s_1, \dots, s_{|s|})$. From the monotonicity of these sets in Eq. (2) due to FCFS scheduling, server j can serve request $k \geqslant 2$, only if it has already served first $(k-1)$ requests, and hence $j \in s_{k-1} \setminus s_k$. This service leads to kth request getting served by server j, and it follows that the inter-transition time for transitions $(w_0, s, r) \to (w_0, s_k^j, r)$ are independent and memoryless with rate μ_w, for all $j \in [n]$ such that $r_j = 0, j \in s_{k-1} \setminus s_k$, and $k \geqslant 2$. A server j can serve first existing request in the system, if it has not served it already and this service can lead to an external departure if $s_1 \cup \{j\} = [n]$. It follows that the inter-transition time for transitions $z \to (w_0, s_1^j, r)$ and $z \to (w_0, (s_2, \dots, s_{|s|}), r)$ are independent and memoryless with rates $\mu_w \mathbb{1}_{\{|s_1|<n-1\}}$ and $\mu_w \mathbb{1}_{\{|s_1|=n-1\}}$ respectively, for servers $j \in [n]$ such that $r_j = 0$ and $j \notin s_1$.

Since each of these transitions are independent and memoryless, it follows that the process $(Z(t), t \in \mathbb{R}_+)$ is a continuous-time Markov chain [39],and we have obtained the generator matrix for this Markov process. ■

A. Approximate Markov Chain

For the distributed read-write system with prioritized reads with service preemption, the number of read requests in the system $(R(t) : t \in \mathbb{R}_+)$ forms a continuous-time Markov chain. Recall that the evolution of $(R_j(t) : t \in \mathbb{R}_+)$ is governed by an $M/M/1$ queue for all $j \in \{0, 1, \dots, n\}$, and the read load on each server is $\frac{\rho_r}{n+1}$. Hence, the probability of a server j not having any read request is given by $1 - \frac{\rho_r}{n+1}$. Writing the number of write requests that have been served by i secondary servers as $Y_i(t)$, we observe that

$$Y_i(t) = |\{k \in [W(t) - W_0(t)] : |S_k(t)| = i\}|.$$

As seen in Eq. (2), the sequence of observed servers $(S_k(t) : k \in [W(t) - W_0(t)])$ for all write requests in the system are ordered by set inclusion. Hence, the write requests with i service completions are served by the same set of servers. From Proposition 1, it follows that $Y(t) \triangleq (Y_0(t), \dots, Y_{n-1}(t))$ is a pooled tandem queue, and evolves as a continuous-time Markov chain if there were no read requests in the system. This pooled tandem queue is approximated by an uncoupled tandem queue in Approximation 1, where the service rate of ith tandem queue is $\bar{\gamma}_i$. In read priority system, the evolution of $Y(t)$ also depends on the number of read requests in the system at each individual server, and the set of servers which are serving ith tandem queue.

Remark 2. In read priority systems, a secondary server can only serve a write request if there are no read requests at this server. Since the probability of having zero read requests at a server is same for all servers, we can approximate the expected rate at which the ith stage of write tandem queue is being served as $P\{R_j = 0\}\bar{\gamma}_i = (1 - \frac{\rho_r}{n+1})\bar{\gamma}_i$, where $\bar{\gamma}_i$ is defined in Eq. (1). Similarly, the average service rate at the primary queue is $(1 - \frac{\rho_r}{n+1})\mu_w$.

That is, we will approximate the process $(Y(t) : t \in \mathbb{R}_+)$ with process $(\bar{Y}(t) : t \in \mathbb{R}_+)$, where the $\bar{Y}(t)$ is an uncoupled tandem queue with no read requests, and the service rates of each stage is multiplied by the probability of no read request at any server.

Approximation 2. *For the distributed read-write system with prioritized preemptive reads, the number of write requests in the system can be modeled by a sequence of uncoupled tandem queues $(W_0(t), \bar{Y}_0(t), \dots, \bar{Y}_{n-1}(t) : t \in \mathbb{R}_+)$ that are served at memoryless service rates $(\mu_0, \beta_0, \dots, \beta_{n-1})$. The first queue W_0 is served at rate $\mu_0 \triangleq \mu_w(1 - \frac{\rho_r}{n+1})$ and the ith stage of tandem queue Y_i is served at rate $\beta_i \triangleq \bar{\gamma}_i(1 - \frac{\rho_r}{n+1})$, where $\bar{\gamma}_i$ is defined in Eq. (1) for $\lambda = \lambda_w, \mu = \mu_w$.*

Theorem 4. *The mean number of write requests in the approximate system is*

$$\mathbb{E}\bar{W} = \frac{\lambda_w}{\mu_0 - \lambda_w} + \sum_{i=1}^{n} \frac{\lambda_w}{\beta_{n-i} - \lambda_w}.$$

Proof. Each of the n unpooled tandem queues is an $M/M/1$ queue with arrival rate λ and service rate β_i. Therefore, the mean number of request in ith queue is $\frac{\lambda}{\beta_i - \lambda}$. Since the number of write requests in the system is the sum of requests in the primary queue and the n tandem queues, the result follows. ■

Remark 3. As in previous section, to show the explicit dependence of mean number of write requests on the number of redundant secondary servers, we denote $p(n) \triangleq \mathbb{E}\bar{W}$, in the approximate read-write system with read priority. The expression for the mean number of write requests in the approximate read priority system can be written in terms of the write load parameter $\nu \triangleq \rho_w/(1 - \rho_w)$, the read load parameter $\Delta_n \triangleq 1 - \frac{\rho_r \nu}{(n+1-\rho_r)}$, and the digamma function[7] $\psi : \mathbb{C} \to \mathbb{R}$ as

$$p(n) = \frac{(n+1)\nu}{n+1-\rho_r(1+\nu)} + \frac{(n+1)\nu(\psi(\Delta_n + n) - \psi(\Delta_n))}{(n+1-\rho_r)}.$$

Remark 4. We observe that as the number of redundant servers n increases, the mean number of read requests reaches a finite limit ρ_r, and the mean number of write requests grows to infinity.

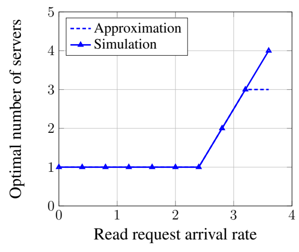
Remark 5. Since the mean number of requests $p + q$, has terms with digamma function, we numerically find the minimum x^* in its domain $\mathbb{R}_+$. Further, the optimal number of redundant servers n^* can be found by comparing mean number of requests $p+q$ at integer values $\lceil x^* \rceil$ and $\lfloor x^* \rfloor$, i.e. $n^* \triangleq \arg\min\{(p+q)(\lfloor x^* \rfloor), (p+q)(\lceil x^* \rceil)\}$.

Design Principle : We observe that $p(n)$ is lower bounded by $\nu \ln e(n+1)$ for all $n \in \mathbb{Z}_+$, and hence the asymptotic growth of the mean number of write requests is at least logarithmic in n. Further, the number of read requests decreases in the order of $\frac{1}{n}$ with n. Hence, this opposing behaviour of $p(n)$ and $q(n)$ should be taken into account while designing the system.

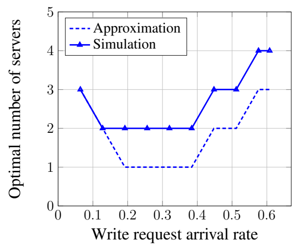
V. Numerical Studies

We numerically simulate a distributed read-write system with primary secondary architecture in this section. The system is simulated with non-preemptive scheduling and round-robin routing of read requests, as opposed to the simplifying assumption of preemptive scheduling and random routing, which we used for analysis. We compare the optimal number of servers obtained analytically using our proposed approximation, to the one observed empirically in the system simulation. In our simulation studies, we select the read service rate $\mu_r = 10$ an order higher than the write service rate $\mu_w = 1$. This is motivated by the observation that reads are typically faster than writes in practical systems [41]. We have plotted the optimal number of servers for the read priority system in Fig. 4. We fix arrival rate of the read or write requests, and vary the other within 95% of the stability region. The dashed curve shows the optimal number obtained analytically for the approximate system, while the solid curve denotes the optimal number obtained empirically under the system simulation.

[7]Digamma function $\psi(.)$ is the derivative of the logarithm of gamma function, and is continuous and differentiable in $\mathbb{R}_+$. It has the property $\psi(z+1) = \psi(z) + \frac{1}{z}$ [40].



(a) $\lambda_w = 0.6$



(b) $\lambda_r = 3.6$

Fig. 4: The optimal number of servers as a function of request arrival rate in a read priority system.

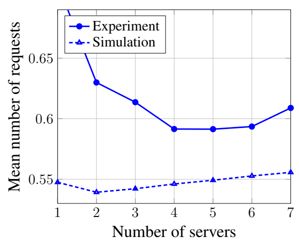
In most practical distributed databases, the number of redundant servers n is typically taken as two[8]. However, as we have observed for read priority system in Fig. 4a and Fig. 4b, the optimal number of secondary servers n can be different than two depending on the request arrival rate and service rate. Hence, the typically chosen value of two redundant servers is not always optimal from latency perspective.

A. Experimental Results

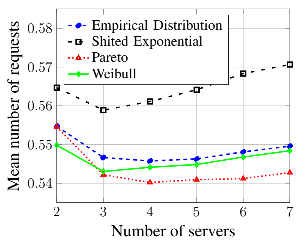
We perform experiments on a 7-server distributed storage system with primary-secondary architecture and a single client server. Most database systems employ a disk cache for serving client requests, while using a backend process to update the data into hard-disk periodically. This is due to the fact that read and write to cache is roughly 10 times faster than read and write to storage disk. Accordingly, we read the file from the cache of the storage system and wrote back to the cache as well. While serving the read or write request to the specific file of interest, each of these servers had other background processes running (e.g. reads and writes to other files).

The client server generates read and write request under Poisson arrival processes with rate λ_r and λ_w respectively. We empirically measure the service time distribution for read and write request to the cache, by averaging it over all the seven servers. For read update for 150MB and write update of 200MB, we obtained the mean service time to read and write in cache to be 0.021s and 0.183s respectively. We took the sum of number of requests in the system with service in progress or waiting for service, and obtained the mean number of requests by averaging them over time. We plot the mean number of requests in the read priority experiment and system simulation with identical distributions Figure 5a, with read arrival rate of $\lambda_r = 20$ and write arrival rate of $\lambda_w = 0.7$. Due to heterogeneity among servers, the read and write times at different servers do not have the same distribution. We observe in Figure 5 that the latency curves obtained from experiment and the simulated system do not coincide, due to heterogeneity among servers and the additional system delays including network delays which were unaccounted in the simulations. However, even though the simulation and experiment do not match due to non-idealities in the system, we observe that there exists an optimal redundancy that minimizes the number of requests in the system.

[8]https://docs.mongodb.com/manual/core/replica-set-architectures/



(a) Experiment on 7-server storage system with $\lambda_r = 20$ and $\lambda_w = 0.7$



(b) System with $\lambda_r = 15$ and $\lambda_w = 0.3$ under different distribution

Fig. 5: The mean number of requests as a function of number of servers in the read priority system.

We plot the mean number of requests in a read-write system for the empirical distributions obtained from experiments, and for the closest fitting distributions from some parametrized families of non-memoryless distributions. Each of the read and write time distributions have a mean of 0.021s and 0.183s respectively. We simulate the system under three different distributions: Shifted exponential, Pareto, Weibull, for read arrival rate $\lambda_r = 15$ and write arrival rate $\lambda_w = 0.3$ under read priority. We further compare the results obtained with the Empirical distribution from the experiment. Plotting the mean number of requests with respect to the number of servers in Figure 5b, we observe that there exists an optimal number of servers that minimises the mean number of requests under non-memoryless service as well.

VI. Conclusion and Future Directions

We studied the latency-redundancy tradeoff in a distributed read-write system with Poisson arrivals and exponential service distribution. We provided novel closed-form approximations for mean number of write requests under read priority. Under the proposed approximation, we characterized the optimal redundancy that minimizes the average request latency under read priority. We empirically showed that the optimal choice of redundancy under the proposed approximation closely follows the simulated result. We performed real world experiments and extensive numerical studies to demonstrate that the insights obtained from our theoretical study continue to hold true even in the real world settings and under non-memoryless service distributions.

Our analysis framework can be extended to the study of multiple-file systems where each file is written to a subset of the servers. In such systems, read and write queues would themselves be multi-class queues where the request for different files can be considered as a separate class. Further, we are interested in finding the optimal redundancy for alternative system architectures and different consistency guarantees. Another interesting future direction would be to characterize the latency-redundancy tradeoff with general distribution for arrival and service processes.

REFERENCES

[1] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," *ACM Spec. Interest Grp. Oper. Syst. Rev. (SIGOPS)*, vol. 41, no. 6, pp. 205—-220, Oct. 2007.

[2] P. Bailis and A. Ghodsi, "Eventual consistency today: Limitations, extensions, and beyond," *ACM Queue*, vol. 11, no. 3, pp. 20—-32, Mar. 2013.

[3] D. Abadi, "Consistency tradeoffs in modern distributed database system design: Cap is only part of the story," *Computer*, vol. 45, no. 2, pp. 37–42, Feb. 2012.

[4] J. Zhong, R. D. Yates, and E. Soljanin, "Minimizing content staleness in dynamo-style replicated storage systems," in *IEEE Inter. Conf. Comp. Commun. (INFOCOM)*, Apr. 2018, pp. 361–366.

[5] L. Huang, S. Pawar, H. Zhang, and K. Ramchandran, "Codes can reduce queueing delay in data centers," in *IEEE Inter. Symp. Inf. Theory (ISIT)*, Jul. 2012, pp. 2766–2770.

[6] N. B. Shah, K. Lee, and K. Ramchandran, "The mds queue: Analysing the latency performance of erasure codes," 2012.

[7] G. Joshi, Y. Liu, and E. Soljanin, "Coding for fast content download," in *Ann. Allerton Conf. Commun., Control, Comp. (Allerton)*, Oct. 2012, pp. 326–333.

[8] S. Chen, Y. Sun, U. C. Kozat, L. Huang, P. Sinha, G. Liang, X. Liu, and N. B. Shroff, "When queueing meets coding: Optimal-latency data retrieving scheme in storage clouds," in *IEEE Inter. Conf. Comp. Commun. (INFOCOM)*, Apr. 2014, pp. 1042–1050.

[9] G. Liang and U. C. Kozat, "Fast cloud: Pushing the envelope on delay performance of cloud storage with coding," *IEEE/ACM Trans. Netw.*, vol. 22, no. 6, pp. 2012–2025, Dec. 2014.

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

[11] Y. Xiang, T. Lan, V. Aggarwal, and Y.-F. R. Chen, "Joint latency and cost optimization for erasure-coded data center storage," *IEEE/ACM Trans. Netw.*, vol. 24, no. 4, pp. 2443–2457, Aug. 2016.

[12] P. Parag, A. Bura, and J.-F. Chamberland, "Latency analysis for distributed storage," in *IEEE Inter. Conf. Comp. Commun. (INFOCOM)*, May 2017, pp. 1–9.

[13] A. O. Al-Abbasi and V. Aggarwal, "Mean latency optimization in erasure-coded distributed storage systems," in *IEEE Inter. Conf. Comp. Commun. (INFOCOM)*, Apr. 2018, pp. 432–437.

[14] G. Joshi, Y. Liu, and E. Soljanin, "On the delay-storage trade-off in content download from coded distributed storage systems," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 989–997, May 2014.

[15] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, and E. Hyytia, "Reducing latency via redundant requests: Exact analysis," *ACM Perf. Eval. Rev. (SIGMETRICS)*, vol. 43, no. 1, pp. 347–360, Jun. 2015.

[16] A. O. Al-Abbasi and V. Aggarwal, "Video streaming in distributed erasure-coded storage systems: Stall duration analysis," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1921–1932, Aug. 2018.

[17] A. O. Al-Abbasi, V. Aggarwal, and T. Lan, "Ttloc: Taming tail latency for erasure-coded cloud storage systems," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 4, pp. 1609–1623, Dec. 2019.

[18] A. Badita, P. Parag, and J.-F. Chamberland, "Latency analysis for distributed coded storage systems," *IEEE Trans. Inf. Theory*, vol. 65, no. 8, pp. 4683–4698, Aug. 2019.

[19] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytiä, and A. Scheller-Wolf, "Queueing with redundant requests: exact analysis," *Queueing Syst. Theory Appl. (QUESTA)*, vol. 83, no. 3, pp. 227–259, Aug. 2016.

[20] W. Wang, M. Harchol-Balter, H. Jiang, A. Scheller-Wolf, and R. Srikant, "Delay asymptotics and bounds for multitask parallel jobs," *Queueing Syst. Theory Appl. (QUESTA)*, vol. 91, no. 3, pp. 207–239, Jan. 2019.

[21] M. Uluyol, A. Huang, A. Goel, M. Chowdhury, and H. V. Madhyastha, "Near-optimal latency versus cost tradeoffs in geo-distributed storage," in *USENIX Symp. Net. Sys. Desgn. Impl. (NSDI)*, Feb. 2020, pp. 157–180.

[22] B. Li, A. Ramamoorthy, and R. Srikant, "Mean-field analysis of coding versus replication in large data storage systems," *ACM Trans. Model. Perform. Eval. Comput. Syst. (PECS)*, vol. 3, no. 1, Feb. 2018.

[23] J. Li, T. K. Phan, W. K. Chai, D. Tuncer, G. Pavlou, D. Griffin, and M. Rio, "Dr-cache: Distributed resilient caching with latency guarantees," in *IEEE Inter. Conf. Comp. Commun. (INFOCOM)*, Apr. 2018, pp. 441–449.

[24] G. Quan, J. Tan, and A. Eryilmaz, "Counterintuitive characteristics of optimal distributed lru caching over unreliable channels," in *IEEE Inter. Conf. Comp. Commun. (INFOCOM)*, Apr. 2019, pp. 694–702.

[25] S. Zhang, L. Wang, H. Luo, X. Ma, and S. Zhou, "Aoi-delay tradeoff in mobile edge caching with freshness-aware content refreshing," 2020.

[26] V. B. Iversen, *Teletraffic engineering and network planning*. DTU Fotonik, 2015.

[27] S. N. Chiu, D. Stoyan, W. S. Kendall, and J. Mecke, *Stochastic Geometry and Its Applications*, 3rd ed. Wiley, Aug. 2013.

[28] D. Axmark, M. Widenius, and theMySQL Documentation Team, *MySQL 8.0 Reference Manual Including MySQL NDB Cluster 8.0*, Oracle.

[29] U. Shanker and S. Pandey, *Handling Priority Inversion in Time-Constrained Distributed Databases*. IGI Global, Jan. 2020.

[30] C. A. Phillips, C. Stein, and J. Wein, "Minimizing average completion time in the presence of release dates," *Math. Prog. Math. Opt. Soc. (MPMOS)*, vol. 82, pp. 199–223, Jun. 1998.

[31] R. Atar, A. Mandelbaum, and M. I. Reiman, "Scheduling a multi class queue with many exponential servers: Asymptotic optimality in heavy traffic," *Annals Appl. Prob.*, vol. 14, no. 3, pp. 1084–1134, Aug. 2004.

[32] K. Fraser, "Practical lock-freedom," Ph.D. dissertation, University of Cambridge, UK, 2004.

[33] G. Giambene, *Queuing Theory and Telecommunications: Networks and Applications*, 2nd ed. Springer US, Apr. 2014.

[34] L. Huang, S. Pawar, H. Zhang, and K. Ramchandran, "Codes can reduce queueing delay in data centers," 2012.

[35] G. Liang and U. C. Kozat, "Tofec: Achieving optimal throughput-delay trade-off of cloud storage using erasure codes," in *IEEE Inter. Conf. Comp. Commun. (INFOCOM)*, Apr. 2014, pp. 826–834.

[36] A. Behrouzi-Far and E. Soljanin, "Scheduling in the presence of data intensive compute jobs," in *IEEE Inter. Conf. Big Data (ICBD)*, Dec. 2019, pp. 5989–5991.

[37] J. D. C. Little, "A proof for the queuing formula: L = λw," *Inter. J. Oper. Res. (IJOR)*, vol. 9, no. 3, pp. 383–387, Jun. 1961.

[38] F. Baccelli and A. M. Makowski, "Simple computable bounds for the fork-join queue," *Proc. Conf. Inform. Sci. (PCIS)*, no. RR-0394, Apr. 1985.

[39] S. M. Ross, *Stochastic processes*, 2nd ed. Wiley India Pvt. Limited, 2008.

[40] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables*. Dover Publications, 1965.

[41] A. T. Kabakus and R. Kara, "A performance evaluation of in-memory databases," *J. King Saud Univ. Comput. Inf. Sci. (CIS)*, vol. 29, no. 4, pp. 520—-525, Oct. 2017.