Data Freshness in Leader-Based Replicated Storage

Amir Behrouzi-Far, Emina Soljanin and Roy D. Yates
Department of Electrical and Computer Engineering, Rutgers University
{amir.behrouzifar,emina.soljanin,ryates}@rutgers.edu

Abstract—Leader-based data replication improves consistency in highly available distributed storage systems via sequential writes to the "leader" nodes. After a write has been committed by the leaders, "follower" nodes are written by a multicast mechanism and are only guaranteed to be eventually consistent. With Age of Information (AoI) as the freshness metric, we characterize how the number of leaders affects the freshness of the data retrieved by an instantaneous read query. In particular, we derive the average age of a read query for a deterministic model for the leader writing time and a probabilistic model for the follower writing time. We obtain a closed-form expression for the average age for exponentially distributed follower writing time. Our numerical results show that, depending on the relative speed of the write operation to the two groups of nodes, there exists an optimal number of leaders which minimizes the average age of the retrieved data, and that this number increases as the relative speed of writing on leaders increases.

I. introduction

Databases are a central part of online services [1], [2]. They constantly execute write and read operations from different clients/services. A large fraction of write operations are updates to the currently stored data [3]. Each update increments the version of the data and sets its timestamp to the current time. Read operations, on the other hand, ask for the latest version of the data, which also has the most recent timestamp. A write/read operation is consistent if it writes to or reads from the latest version of the data.

Data availability is a major objective in database systems [4], [5]. To achieve high availability, data is partitioned into multiple chunks, and each chunk gets replicated in multiple storage nodes [6], [7]. These replicas can possibly be located in different geographic locations to further improve data availability and storage robustness. However, according to the CAP theorem [8], availability and consistency are not achievable at the same time in a distributed system. Therefore, availability-consistency trade-offs are inevitable in a distributed database.

Leader-based (replicated) databases have been proposed to increase data consistency in large scale distributed storage [9], [10]. In these databases, some of the storage nodes assigned to a given chunk of data are elected as leaders. Every write operation to the data chunk is first written to the leaders, where sequential writes from one leader to the next guarantee that they always have the latest version of the data. Thus, write consistency is guaranteed at the leaders. After sequential writes to the leaders, the update is committed and gets transmitted to a set of follower nodes via a multicast mechanism. After an update is committed, it becomes available to read and further updates may be initiated. Due to the guaranteed leaders' consistency and high availability (by the existence of the replicas), leader-based systems are widely deployed in practice. For example, Google Spanner [9], Amazon DynamoDB [11], Apple FoundationDB [12] have used it as a part of their storage ecosystem, where they employ the Paxos algorithm [13] for leader election among the storage nodes. The Raft algorithm [14], an alternative for Paxos, has been used by MongoDB and InfluxDB as a leader election algorithm. These databases are *eventually* consistent in that the data is guaranteed to be consistent among the leader nodes, and the data on each follower eventually will become consistent, in the absence of subsequent writes.

Freshness of retrieved data in eventually consistent database systems have been studied in [15]–[18]. With a probabilistic definition of consistency, [15] studies the delay-consistency trade-off in a distributed database system. Authors in [17] establish a version-based staleness metric and investigate the delay benefits of eventually consistent databases and shed light on why they are good enough in practice. Data staleness in dynamo-style quorum-based replicated storage systems is studied in [19]. In this work, using AoI as the freshness metric, the average age of the retrieved data is studied for different types of quorum consensus. AoI has been recently used to characterize data freshness in various types of data transmission systems [20]–[32].

Despite the existing work on eventually consistent databases, we are far from understanding the dynamics governing data freshness in such systems. In particular, in a leader-based model, we know very little about the effect of the number of leaders on the freshness of the data retrieved by a read operation. Having more leaders improves the consistency of the database since leaders are guaranteed to have the latest version of the data. On the other hand, due to the sequential, and thus time-costly writes, having many leaders can prolong the commit time of an update, which makes the update more stale when it becomes available to read. Furthermore, the relative speed of writing an update to the leaders, compared to the multicast writing to the follower nodes, is a critical factor in optimizing system parameters for increasing the freshness of data retrieved by a read operation.

Using the AoI metric [33]–[43], we study the average freshness of the data retrieved by a read query in a leader-based replicated database. We develop two different models for write operations to 1) the leaders, which are written by sequential writes, and 2) the followers, which are written through a multicast. We then derive the dependence of the average age of the data retrieved by a read query on our model parameters. We assume the writing time to the leaders is deterministic and scales linearly with the number of leaders. However, it is a random variable for each follower. For the exponential write time distribution of the follower nodes, we drive a closed form for the average age of the retrieved data by a read operation. Our numerical results show that, depending on the relative speed of the write operation to the two groups of nodes, there exists an optimal number of leaders which minimizes the average age of the retrieved data. In addition, we observe that the average age monotonically increases or decreases with the number of leaders, depending the relative speed of the write operations. Furthermore, for a system with time-varying demands, we show that it is possible to increase the number of followers without increasing the average age, when the data demand gets high.

II. System model and Problem Formulation

Consider a leader-based database system, where each data chunk is replicated on n nodes. Out of the n nodes storing the same chunk one is elected as the leader. To ensure write consistency, any write query first has to update the leader. To further improve consistency, the leader may initiate a series of sequential writes some other nodes, which with the original leader form the leaders group. We refer to the rest of the nodes as followers. An update is initiated to the leaders and is committed once it is written to all leaders. Each update is written in an update round, which increments the version of the data and updates its timestamp to the update initiation time. A node is said to be consistent if it is written with the latest update. Thus all leaders get consistent in every update round, the followers may only be eventually consistent.

In our model, fresh updates are initiated to the leaders at time instances $0, c, 2c, \ldots$. The write operation requires time c to complete, and thus the kth update initiated at time kc is committed at time $(k+1)c$. We refer to the time interval of length c as a write slot. After a write is committed to all leaders, it is forwarded to the followers, and the next write is initiated by the system. An update becomes available to read only after it is committed. We assume that forwarding an update to the followers happens through a multicast mechanism so that a random number of follower nodes also receive the latest update in a given round. Specifically, we assume the write time to each follower node is described by the random variable T_w. Thus a follower receives the latest update only if $T_w < c$. We also assume that the T_w are iid across the followers, with CDF $F_w(t)$. We refer to a follower with the latest version of the data as consistent; otherwise a followers is non-consistent. Thus, the set of consistent followers may be different in each round.

When a read query is generated, it is sent to r nodes. The query size parameter r is fixed across the queries. After reading from r possibly different versions of data, the system responds to the query with the most recent version among the r accessed nodes. A read query is consistent if it retrieves the latest version of the data. The consistency of a read query is guaranteed if it is sent to at least one leader. However, sending every read query to a leader may overload the nodes in the leader group, which in turn reduces their availability [44]. In practical, implementations of a leader-based database, e.g. Amazon DynamoDB, consistency of the read queries is traded for availability of the leader nodes by sending each query to r randomly chosen nodes.

In our model, the system cancels the write of the kth update on a follower if it is not written by the time $(k+2)c$, i.e. the end of time slot $k+1$. In other words, the kth update gets preempted when update $k+1$ is initiated to the followers. Thus, the age process at the followers is statistically the same across both the followers and the time slots. The age process at the leaders is the same across the leaders and the time slots, as well. Therefore, to study the age of a read query it is sufficient to study the age process at the nodes in a time slot $[kc, (k+1)c)$, for any $k \in \{0, 1, 2, \ldots\}$. We assume the system has started running at $-\infty$ and the arrival time of a read query T_a is a uniform random variable in the range $[0, c)$. Arrivals in time slots $[kc, (k+1)c), k = 1, 2, \ldots$ observe statistically the same age instances at the nodes, as arrivals in the interval $[0, c)$. We further note that, practical evaluations in [45] show that the average latency of read operations is an order of magnitude smaller than the write operations. Therefore, we assume the read queries are instantaneous in this work.

We study the average age of the data retrieved by a read query, defined as follows. The age of information $\Delta_i(t)$ (or simply the age) at node i at time t is the time difference between time t and the timestamp $u_i(t)$ of the latest data update at node i:

$$\Delta_i(t) = t - u_i(t). \tag{1}$$

The age of a read query $\Delta(t)$ at time t is defined as the minimum age over the set $\mathcal{S}_r$ of nodes the query is sent to:

$$\Delta(t) = \min_{i \in \mathcal{S}_r} \Delta_i(t). \tag{2}$$

Leaders experience a different age process than the followers since, in each update round, leaders are guaranteed to get the most recent update of the data while the followers may or may not receive it. The age processes at a leader and a follower are shown in Fig. 1. Leaders are written in every update round and their data becomes available only after every leader is updated and the write

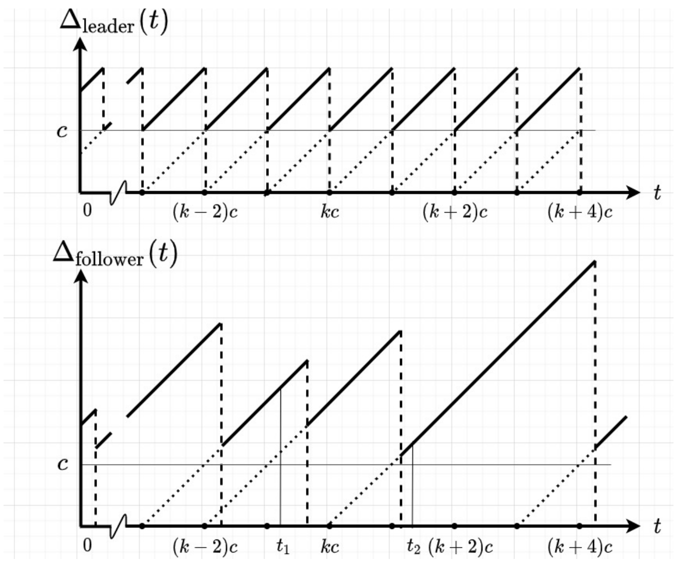

Fig. 1. The age process for a sample leader node and a sample follower node. The leader nodes experience the same age process, illustrated by $\Delta_{\text{leader}}(t)$. The follower nodes may experience different yet statistically identical age process, a sample path of which is illustrated by $\Delta_{\text{follower}}(t)$.

is committed, which is c units of time after the update initiation. Thus, the age process is the same for every leader node, as shown by Δ_{leader} in Fig. 1.

The age processes at the follower nodes are statistically identical, and illustrated by $\Delta_{\text{follower}}(t)$ in Fig. 1, since the write time to the follower nodes are iid random variables. (However, in each update round, a random subset of followers receives the data at random times.)

III. Age Analysis

For a read query, r instances of the age processes are sampled from the collective set of l leaders and $n-l$ followers. The age of the read is the minimum of the ages at the queried nodes. The age of the retrieved data depends on the the types of nodes a read is sent to. Specifically, we analyse the average age of a read query based on the following partition:

- B_1: At least one node is chosen from the leaders,
- B_2: No node is chosen from the leaders.

The probability of having at least one leader queried is

$$\Pr\{B_1\} = \begin{cases} 1 - \frac{\binom{n-l}{r}}{\binom{n}{r}} & r \le n-l, \\ 1 & r > n-l. \end{cases} \tag{3}$$

In what follows, we study the non-trivial case $r \le n-l$.

Since leaders are guaranteed to receive every update, they are always consistent, and with the freshest data. Therefore, the age of the retrieved data in event B_1 is the age at leaders, characterized by the following lemma.

Lemma 1. Given B_1, the average age of a read query is

$$\mathbb{E}[\Delta | B_1] = 3c/2. \tag{4}$$

Proof. When B_1 occurs, the instantaneous age of the read is $c + T_a$, where c is the commit time of a write operation and T_a is the arrival time of the read query, which is uniformly distributed in the interval $[0, c)$. Then the average age of a read can be calculated as

$$\begin{aligned}\mathbb{E}[\Delta|B_1] &= \int_0^c f_{T_a}(t)\mathbb{E}[\Delta|B_1, T_a = t]dt, \\ &= \frac{1}{c}\int_0^c (c+t)dt = 3c/2. \qquad \square\end{aligned}$$

When B_2 occurs, the age of the read query is the age at the most recently updated follower, among the r followers the read query is sent to. A follower is consistent at the arrival time of a read T_a if it is written with the most recent update by T_a. For instance, in Fig. 1, the follower would not return the most recent $(k-2)$th update to a read arriving at $T_a = t_1$. Whereas, it would return the most recent kth update to a read arriving at $T_a = t_2$. The following lemma gives the average age of a read query in event B_2.

Lemma 2. Given B_2, the average age of a read query is

$$\mathbb{E}[\Delta|B_2] = \frac{3c}{2} + \frac{\int_0^c [1 - F_w(t)]^r\, dt}{1 - [1 - F_w(c)]^r}. \tag{5}$$

Proof. Under event B_2, the data is read from one of the r selected followers, which we index $(1), (2), \ldots, (r)$. For a given update to be successful at a follower, it should be written at most c units of time past the commitment of that update. Thus, there is a fixed probability $p_c = F_w(c)$ (across the update rounds) that a follower successfully receives an update. Accordingly, we define the geometric random variable $Z_{(i)}$ as the number of update rounds follower (i) has missed after its latest update. Thus, $Z_{(i)} \sim \text{Geo}(1-p_c)$. At the time of arrival T_a, the age at the selected follower i is

$$\Delta_{(i)} = Z_{(i)}c + T_a. \tag{6}$$

The age of a read query is

$$\Delta = \min\{\Delta_{(1)}, \Delta_{(2)}, \ldots, \Delta_{(r)}\}. \tag{7}$$

Therefore, the average age of a read can be written as

$$\begin{aligned}\mathbb{E}[\Delta|B_2] &= \mathbb{E}[\min\{Z_{(1)}, \ldots, Z_{(r)}\}|B_2]c + \mathbb{E}[T_a|B_2], \\ &= \mathbb{E}[Z_{\min}|B_2]c + \mathbb{E}[T_a],\end{aligned} \tag{8}$$

where $Z_{\min}$ is the number of missed update rounds at the most recently updated follower, among the r followers the read query is sent to. Since the $Z_{(i)}$ are independent,

$$\begin{aligned}\mathbb{E}[Z_{\min}|B_2] &= \int_0^c f_{T_a}(t)\mathbb{E}[Z_{\min}|T_a = t, B_2]dt \\ &= \frac{1}{c}\int_0^c \sum_{z=0}^{\infty} \left(\Pr\{Z_{(i)} > z | T_a = t, B_2\}\right)^r dt\end{aligned} \tag{9}$$

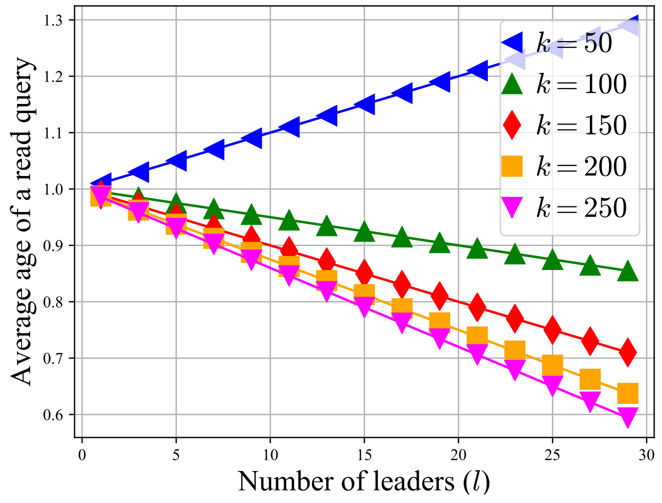

Fig. 2. The average age of a read query vs. the number of leaders, see (16), when $n = 50$, $r = 1$ and $\lambda = 1$. Increasing l either monotonically increases the average age ($k = 50$) or monotonically increases it ($k \geq 100$).

Defining $F_w(t) = p_t$,

$$\begin{aligned}
&\Pr\{Z_{(i)} = 1 | T_a = t, B_2\} = p_t,\\
&\Pr\{Z_{(i)} = 2 | T_a = t, B_2\} = (1-p_t)p_c,\\
&\Pr\{Z_{(i)} = 3 | T_a = t, B_2\} = (1-p_t)(1-p_c)p_c,\\
&\qquad\vdots\\
&\Pr\{Z_{(i)} = j | T_a = t, B_2\} = (1-p_t)(1-p_c)^{j-2}p_c. \qquad (10)
\end{aligned}$$

Therefore,

$$\begin{aligned}
&\Pr\{Z_{(i)} > 0 | T_a = t, B_2\} = 1,\\
&\Pr\{Z_{(i)} > 1 | T_a = t, B_2\} = 1 - p_t,\\
&\Pr\{Z_{(i)} > 2 | T_a = t, B_2\} = (1-p_t)(1-p_c),\\
&\qquad\vdots\\
&\Pr\{Z_{(i)} > z | T_a = t, B_2\} = (1-p_t)(1-p_c)^{z-1}. \qquad (11)
\end{aligned}$$

Substituting (11) in (9) gives

$$\begin{aligned}
\mathbb{E}[Z_{min}|B_2] &= \frac{1}{c}\int_0^c \sum_{z=0}^{\infty} \left(\Pr\{Z_{(i)} > z | T_a = t, B_2\}\right)^r dt\\
&= \frac{1}{c}\int_0^c \left[1 + \sum_{z=1}^{\infty}(1-p_t)^r(1-p_c)^{r(z-1)}\right] dt\\
&= 1 + \frac{1}{c}\sum_{z=1}^{\infty}(1-p_c)^{r(z-1)} \int_0^c [1-F_w(t)]^r\, dt\\
&= 1 + \frac{1}{c}\frac{\int_0^c [1-F_w(t)]^r\, dt}{1-(1-p_c)^r}. \qquad (12)
\end{aligned}$$

Finally, by substituting (12) in (8),

$$\mathbb{E}[\Delta|B_2] = \frac{3c}{2} + \frac{\int_0^c [1-F_w(t)]^r\, dt}{1-[1-F_w(c)]^r}. \qquad \square$$

Theorem 1. When the follower writing time T_w is exponential(λ), the average age of a read query is

$$\mathbb{E}[\Delta] = \frac{3c}{2} + \frac{\binom{n-l}{r}}{\binom{n}{r}}\frac{1}{\lambda r}. \qquad (13)$$

Proof. With $F_w(t) = 1 - e^{-\lambda t}$,

$$\int_0^c [1-F_w(t)]^r\, dt = \frac{1}{\lambda r}\left(1 - e^{-\lambda r c}\right),$$

and, $1 - [1-F_w(c)]^r = \left(1 - e^{-\lambda r c}\right)$, which by substituting in (5) completes the proof. $\square$

IV. Numerical Results

In this section, we present numerical results based on our average age analysis. First, we study the effect of the number of leaders on the average age of a read query. Since the writes to the leaders occur sequentially, the number of leaders should affect the commit time c. On the other hand, knowing that all the nodes are the members of the same ecosystem, there should be a connection between the writing time to a leader and that of a follower node. Accordingly, we assume c is related to the number of leaders l and the rate λ of the write process to the followers by

$$c = \frac{l}{k\lambda}, \qquad (14)$$

where k can be interpreted as the relative speed of writing to the leaders compared to writing to a follower. Note that this is not the only model to scale c with the other system parameters; there are other possibilities. We think of k as an inherent factor of the underlying system and show our results for a range of values. With this model and $T_w \sim \text{Exp}(\lambda)$, the average age of a read query is

$$\mathbb{E}[\Delta] = \frac{1}{\lambda}\left[\frac{3l}{2k} + \frac{1}{r}\frac{\binom{n-l}{r}}{\binom{n}{r}}\right]. \qquad (15)$$

We further assume that the time unit is chosen such that $\lambda = 1$ and the average write time to a follower is one time unit.

Fig. 2 shows the variation of the average age of a read query as the number of leaders change, when $n = 50$ and $r = 1$. The effect of increasing the number of leaders on the average age depends on the parameter k. With $k = 50$ the average age increases monotonically; while for $k \geq 100$, it decreases, also monotonically, with l. Intuitively, when writing on leaders is relatively slow, i.e. k is small, having more leaders within a fixed number of nodes increases the staleness of an update, since it is available to read only after it is committed. On the other hand, when writing to the leaders is fast (k is large), then having more leaders increases the chance of a read query to be sent to a consistent node. This also can be verified from (15) by substituting $\lambda = 1$, $r = 1$ and $n = 50$,

$$\mathbb{E}[\Delta] = 1 + \left[\frac{75}{k} - 1\right]\frac{l}{n}. \qquad (16)$$

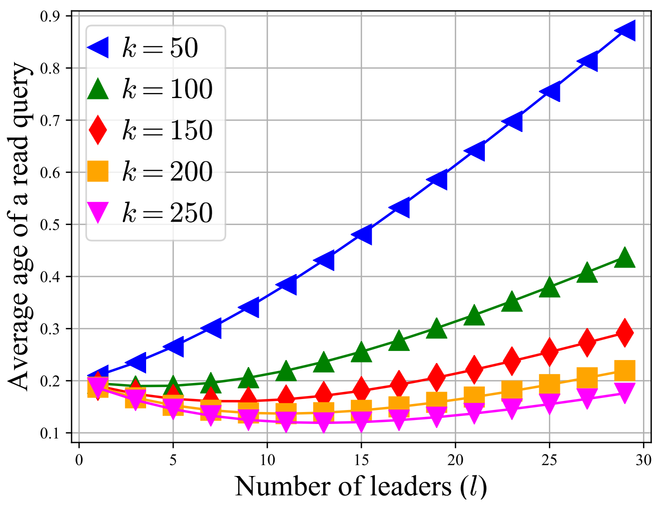

Fig. 3. The average age of a read query vs. the number of leaders, see (13), when $n = 50$, $r = 4$ and $\lambda = 1$. With small k, the average age increases monotonically with l. With larger k, it decreases initially, reaches an optimum point and then it increases.

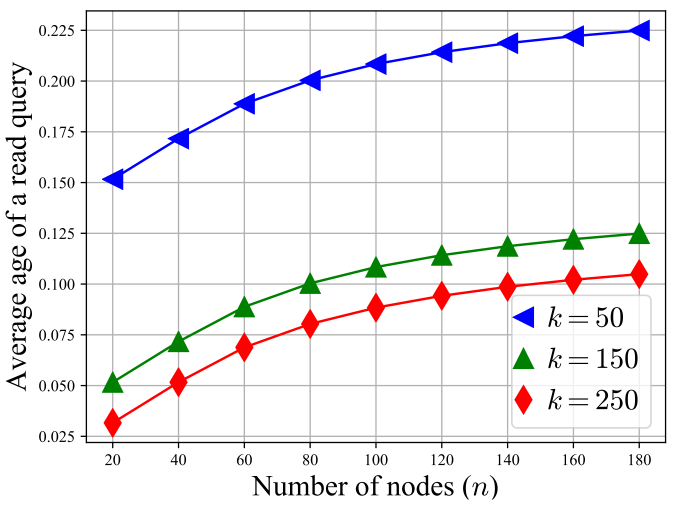

Fig. 4. Average age of a read query vs. the number of nodes, see (13), when $r = 10$, $l = 5$ and $\lambda = 1$. The average age increases monotonically with the number of nodes.

In Fig. 3, the effect of number of leaders on average age is illustrated when $n = 50$ and $r = 5$. With a larger size of read query, the probability that a read query is sent to a consistent node is higher. For that reason the average age is dropped in Fig. 3 compared to Fig. 2. On the other hand, the monotonic changes in the average age do not occur for every value of k with $r = 5$. For $k \geq 100$, the average age decreases initially with the number of leaders and after reaching an optimum point it starts increasing. The reason for this behaviour can be explained as follows. Increasing the number of leaders has two competing effects on the average age. First, by increasing the commit time it increases the staleness of an update. Second, by increasing the number of leaders and, at the same time, reducing the number of followers it increases the probability of a read query to be sent to a consistent and receive (relatively) less stale data. Now, when r is large, the probability that a read query is sent to a consistent node is already high. Therefore, the second effect has smaller contribution to the average age. Thus, increasing the number of leaders, even when they are fast to write, does not decrease the average age monotonically. From (15), the range of k for which the average age initially decreases is $k \geq \lceil 3n(n-1)/2(n-r) \rceil$. With $n = 50$ and $r = 5$, the average age initially decreases with $k \geq 82$.

It is clear that increasing the size of a read query r reduces the average age of its retrieved data. Furthermore, with fixed l, increasing the number of nodes reduces the probability of a read query to be sent to a consistent node and thus it increases the average age, as it is shown in Fig. 4. Nevertheless, the effect of changing both n and r, at the same time, is more complex. Consider the following scenario. A leader-based database has stored a data chunk, with parameters n, r and l. Due to a growth in the demand, the system administrator decides to increase the number of replicas n, but without deteriorating the performance. To that end, she could increase the size of read queries r and therefore may ask the following question: how much increment in r is needed when n increases to maintain the same average age? Our numerical results show that, with establishing a relationship between n and r, one can possibly avoid the increase in the average age as n increases. For instance, in Fig. 5, the relationship between the two parameters is $r = 10 + n/20$. With this dependence, we can see that increasing the number of nodes beyond $n = 100$ does not increase the average age. This result could be especially useful for systems with time-varying demands. Finding the exact dependence, however, is out of the scope of this paper and is left as an open problem for future study.

V. Conclusion

The average age of the retrieved data by an instantaneous read query in a leader-based database was studied. With a deterministic model for the writing time on the leaders and a probabilistic model for the writing time on a follower, the average age of read query was analyzed. Closed-form average age under exponential distribution of the follower writing time was derived. It was shown numerically that, when the writing time on the leaders group scales linearly with the number of leaders, the average age could monotonically increase, monotonically decrease or behave non-monotonically, depending on the size of the read query and the relative speed of writing on a leader to writing on a follower. Furthermore, for a data with dynamic demands, it was shown that it is possible to keep the average age fixed when the number of follower nodes increases, with a slight increase in the size of the read query.

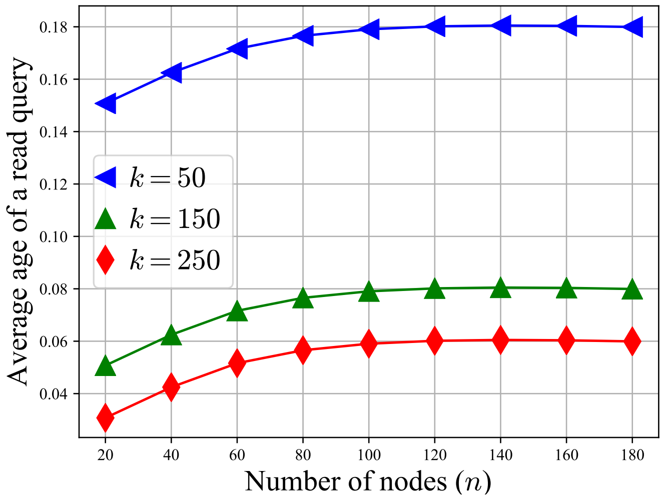

Fig. 5. Average age of a read query vs. the number of nodes, see (13), when $l = 5$ and $\lambda = 1$. The size of the read query is dependent to the number of nodes, $r = 10 + n/20$. Beyond $n = 100$ the average age does not increase with n.

Acknowledgement

This research was supported in part by the NSF awards No. CIF-1717041, CIF-1717314 and CCF-1559855.

References

[1] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, “Bigtable: A distributed storage system for structured data,” ACM Transactions on Computer Systems (TOCS), vol. 26, no. 2, p. 4, 2008.

[2] 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,” in ACM SIGOPS operating systems review, vol. 41, no. 6. ACM, 2007, pp. 205–220.

[3] T. G. Armstrong, V. Ponnekanti, D. Borthakur, and M. Callaghan, “Linkbench: a database benchmark based on the facebook social graph,” in Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, 2013, pp. 1185–1196.

[4] A. Lakshman and P. Malik, “Cassandra: a decentralized structured storage system,” ACM SIGOPS Operating Systems Review, vol. 44, no. 2, pp. 35–40, 2010.

[5] D. Ford, F. Labelle, F. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, “Availability in globally distributed storage systems,” 2010.

[6] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso, “Understanding replication in databases and distributed systems,” in Proceedings 20th IEEE International Conference on Distributed Computing Systems. IEEE, 2000, pp. 464–474.

[7] M. F. Aktas, A. Behrouzi-Far, E. Soljanin, and P. Whiting, “Load balancing performance in distributed storage with regular balanced redundancy,” arXiv preprint arXiv:1910.05791, 2019.

[8] S. Gilbert and N. Lynch, “Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services,” Acm Sigact News, vol. 33, no. 2, pp. 51–59, 2002.

[9] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild et al., “Spanner: Google’s globally distributed database,” ACM Transactions on Computer Systems (TOCS), vol. 31, no. 3, p. 8, 2013.

[10] A. Sharov, A. Shraer, A. Merchant, and M. Stokely, “Take me to your leader!: online optimization of distributed storage configurations,” Proc. of the VLDB Endowment, vol. 8, no. 12, pp. 1490–1501, 2015.

[11] S. Mathew and J. Varia, “Overview of amazon web services,” Amazon Whitepapers, 2014.

[12] C. Chrysafis, B. Collins, S. Dugas, J. Dunkelberger, M. Ehsan, S. Gray, A. Grieser, O. Herrnstadt, K. Lev-Ari, T. Lin et al., “Foundationdb record layer: A multi-tenant structured datastore,” arXiv preprint arXiv:1901.04452, 2019.

[13] T. D. Chandra, R. Griesemer, and J. Redstone, “Paxos made live: an engineering perspective,” in Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing. ACM, 2007, pp. 398–407.

[14] D. Ongaro and J. Ousterhout, “In search of an understandable consensus algorithm,” in 2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14), 2014, pp. 305–319.

[15] M. R. Rahman, L. Tseng, S. Nguyen, I. Gupta, and N. Vaidya, “Characterizing and adapting the consistency-latency tradeoff in distributed key-value stores,” ACM Transactions on Autonomous and Adaptive Systems (TAAS), vol. 11, no. 4, p. 20, 2017.

[16] W. Golab, X. Li, and M. A. Shah, “Analyzing consistency properties for fun and profit,” in Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing. ACM, 2011, pp. 197–206.

[17] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica, “Quantifying eventual consistency with pbs,” The VLDB Journal, vol. 23, no. 2, pp. 279–302, 2014.

[18] F. Bannour, S. Souihi, and A. Mellouk, “Adaptive quorum-inspired sla-aware consistency for distributed sdn controllers.”

[19] J. Zhong, R. D. Yates, and E. Soljanin, “Minimizing content staleness in dynamo-style replicated storage systems,” in IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). IEEE, 2018, pp. 361–366.

[20] M. Bastopcu and S. Ulukus, “Age of information for updates with distortion,” arXiv preprint arXiv:1904.10444, 2019.

[21] A. Arafa, J. Yang, S. Ulukus, and H. V. Poor, “Age-minimal transmission for energy harvesting sensors with finite batteries: Online policies,” CoRR, vol. abs/1806.07271, 2018. [Online]. Available: http://arxiv.org/abs/1806.07271

[22] B. T. Bacinoglu, Y. Sun, E. Uysal-Biyikoglu, and V. Mutlu, “Achieving the age-energy tradeoff with a finite-battery energy harvesting source,” in Proc. IEEE Int’l. Symp. Info. Theory (ISIT), Jun. 2018, pp. 876–880.

[23] I. Kadota, A. Sinha, E. Uysal-Biyikoglu, R. Singh, and E. Modiano, “Scheduling policies for minimizing age of information in broadcast wireless networks,” CoRR, vol. abs/1801.01803, 2018. [Online]. Available: http://arxiv.org/abs/1801.01803

[24] H. B. Beytur and E. Uysal-Bıyıkoglu, “Minimizing age of information on multi-flow networks,” in 2018 26th Signal Processing and Communications Applications Conference (SIU), May 2018, pp. 1–4.

[25] E. Sert, C. Sönmez, S. Baghaee, and E. Uysal-Biyikoglu, “Optimizing age of information on real-life tcp/ip connections through reinforcement learning,” in 2018 26th Signal Processing and Communications Applications Conference (SIU), May 2018, pp. 1–4.

[26] A. Arafa and S. Ulukus, “Age minimization in energy harvesting communications: Energy-controlled delays,” in Asilomar Conference on Signals, Systems, and Computers, Oct 2017, pp. 1801–1805.

[27] A. Baknina and S. Ulukus, “Coded status updates in an energy harvesting erasure channel,” CoRR, vol. abs/1802.00431, 2018. [Online]. Available: http://arxiv.org/abs/1802.00431

[28] M. Bastopcu and S. Ulukus, “Age of information for updates with distortion: Constant and age-dependent distortion constraints,” arXiv preprint arXiv:1912.13493, 2019.

[29] I. Kadota, E. Uysal-Biyikoglu, R. Singh, and E. Modiano, “Minimizing the age of information in broadcast wireless networks,” in 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton), Sept 2016, pp. 844–851.

[30] Q. He, D. Yuan, and A. Ephremides, “Optimizing freshness of information: On minimum age link scheduling in wireless systems,” in 2016 14th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), May 2016, pp. 1–8.

[31] J. Zhong, E. Soljanin, and R. Yates, “Status updates through multicast networks,” in Proc. Allerton Conf. on Commun., Control and Computing, Oct. 2017, pp. 463–469.

[32] J. Zhong, R. Yates, and E. Soljanin, “Two freshness metrics for local cache refresh,” in Proc. IEEE Int’l. Symp. Info. Theory (ISIT), Jun. 2018, pp. 1924–1928.

[33] S. Kaul, R. Yates, and M. Gruteser, “Real-time status: How often should one update?” in 2012 Proceedings IEEE INFOCOM. IEEE, 2012, pp. 2731–2735.

[34] S. Kaul and R. Yates, “Age of information: Updates with priority,” in Proc. IEEE Int’l. Symp. Info. Theory (ISIT), Jun. 2018, pp. 2644–2648.

[35] S. Kaul, R. Yates, and M. Gruteser, “Real-time status: How often should one update?” in Proc. IEEE INFOCOM, March 2012, pp. 2731–2735.

[36] Y. Sun and B. Cyr, “Information aging through queues: A mutual information perspective,” in Signal Processing Advance for Wireless Communications (SPAWC), 2018, pp. 481–484.

[37] A. Kosta, N. Pappas, A. Ephremides, and V. Angelakis, “Age and value of information: Non-linear age case,” in 2017 IEEE International Symposium on Information Theory (ISIT), June 2017, pp. 326–330.

[38] R. Yates, E. Najm, E. Soljanin, and J. Zhong, “Timely updates over an erasure channel,” in Proc. IEEE Int’l. Symp. Info. Theory (ISIT), Jun. 2017, pp. 316–320.

[39] J. Zhong, R. Yates, and E. Soljanin, “Backlog-adaptive compression: Age of information,” in Proc. IEEE Int’l. Symp. Info. Theory (ISIT), Jun. 2017, pp. 566–570.

[40] S. K. Kaul and R. Yates, “Status updates over unreliable multiaccess channels,” in Proc. IEEE Int’l. Symp. Info. Theory (ISIT), Jun. 2017, pp. 331–335.

[41] R. D. Yates, “The age of information in networks: Moments, distributions, and sampling,” arXiv preprint arXiv:1806.03487, vol. abs/1806.03487, 2018.

[42] Y. Sun, E. Uysal-Biyikoglu, R. D. Yates, C. E. Koksal, and N. B. Shroff, “Update or wait: How to keep your data fresh,” IEEE Trans. Info. Theory, vol. 63, no. 11, pp. 7492–7508, Nov. 2017.

[43] J. Zhong, R. Yates, and E. Soljanin, “Multicast with prioritized delivery: How fresh is your data?” in Signal Processing Advance for Wireless Communications (SPAWC), Jun. 2018, pp. 476–480.

[44] W. Vogels, “Eventually consistent,” Communications of the ACM, vol. 52, no. 1, pp. 40–44, 2009.

[45] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica, “Probabilistically bounded staleness for practical partial quorums,” Proceedings of the VLDB Endowment, vol. 5, no. 8, pp. 776–787, 2012.