Received: 3 January 2018 | Revised: 29 May 2018 | Accepted: 3 January 2019
DOI: 10.1002/cpe.5171

WILEY

RESEARCH ARTICLE

TAP: Timeliness-aware predication-based replica selection algorithm for key-value stores

Xianqian Zhou | Liyuan Fang | Haiming Xie | Wanchun Jiang

School of Computer Science and Engineering, Central South University, Changsha, China

Correspondence
Wanchun Jiang, School of Computer Science and Engineering, Central South University, 932 South Lushan Road, Changsha 410083, China.
Email: jiangwc@csu.edu.cn

Funding information
National Natural Science Foundation of China, Grant/Award Number: 61502539; China Postdoctoral Science Foundation, Grant/Award Number: 2015M582344 and 2016T90761; Hunan Province Science and Technology Plan project, Grant/Award Number: 2016JC2009; National Key Research and Development of China, Grant/Award Number: 2018YFB170043

Summary

In current large-scale distributed key-value stores, a single end-user request may lead to key-value access across tens or hundreds of servers. The tail latency of these key-value accesses is crucial to user experience and greatly impacts the revenue. To cut the tail latency, it is crucial for clients to choose the best replica server as much as possible for the service of each key-value access operation. Aware of the challenges on the time-varying performance across servers and the herd behaviors, an adaptive replica selection scheme C3 has been proposed recently. In C3, feedback from individual servers is brought into replica ranking to reflect the time-varying performance of servers, and the distributed rate control and backpressure mechanisms are invented. Despite C3's good performance, we reveal the timeliness issue of C3, which has large impacts on both the replica ranking and the rate control. To address this issue, we propose the TAP (timeliness-aware predication-based) replica selection algorithm, which predicts the queue size of replica servers under the poor timeliness condition, instead of utilizing the exponentially weighted moving average of the piggybacked queue sizes in history as in C3. Consequently, compared with C3, TAP can obtain more accurate queue-size estimation to guide the replica selection at clients. Simulation results also confirm the advantage of TAP over C3 in terms of cutting the tail latency.

KEYWORDS

key-value stores, prediction, tail latency, timeliness

1 | INTRODUCTION

In the current large-scale distributed key-value store system, data are partitioned into small pieces, replicated, and distributed across servers for parallel access and scalability. Consequently, a single end-user request may need key-value access from tens or hundreds of servers.[1-3] The tail latency of these key-value accesses decides the response time of the end-user request, which is directly associated with user experience and revenue.[4,5] Nevertheless, because the performance of servers is time varying,[6,7] the tail latency is hard to be guaranteed and may become long beyond expectation in a certain condition. A recent study shows that the 99th percentile latency can be one order of magnitude larger than the median latency,[6] indicating that there is a large space to cut the tail latency of key-value accesses. To cut the tail latency, the replica selection scheme, which chooses the best replica server for each key-value access as much as possible at clients, is crucial.[8] Other methods, including duplicate or reissue requests[2,6,9,10] for small tail latency, can also benefit from a good replica selection scheme.

However, the replica selection schemes of current classic key-value stores are very simple for efficiency. For example, the OpenStack Swift just reads from an arbitrary server and retries in case of failures.[11] HBase relies on HDFS, which chooses the physically closest replica server.[12] Riak uses an external load balancer such as Nginx,[13] which employs the least-outstanding requests (LOR) strategy. According to the LOR strategy, the client chooses the server to which it has sent the least number of outstanding requests. MongoDB selects the replica server with the smallest network latency.[14] Cassandra employs the dynamic snitching strategy, which considers the history of reading latencies and I/O load.[15] Obviously, all these methods never take the time-varying performance of servers into consideration. Hence, they are hard to ensure the choice of the fastest replica server.

In spite of the time-varying performance of servers, the design of the replica selection scheme still faces the following challenges. First, as all the clients independently choose the fastest server, they may concurrently access the fastest server, leading to great server performance degradation. The same behavior will subsequently be repeated on a different fast server. Therefore, this kind of herd behavior should be avoided by the replica selection algorithm. Second, the replica selection scheme should be simple enough in terms of both computation and coordination, as it is needed for every lightweight key-value access operation.

Aware of these challenges, an adaptive replica selection scheme C3 has been proposed in NSDI recently.[8] C3 piggybacks the queue size of keys waiting at servers and the service time from the servers to guide the replica ranking at clients and introduce both the cubic rate control algorithm[16] and the backpressure mechanism to adapt the sending rate of keys at the clients to the observed receipt capacity of servers. In this way, C3 can adapt to the time-varying service rate across servers and avoid the herd behavior.[8] The great benefit of the innovations in introducing the feedback is confirmed by both the experiments with Amazon EC2 and the at-scale simulations.

In this paper, we reveal the timeliness issue of feedback in C3, which has large impacts on both the replica ranking and the rate control. Specifically, in the replica ranking of C3, when the network delay is ignored, the server with minimal Q_s/μ_s is the best candidate to cut the tail latency, where Q_s denotes the queue size of keys waiting at servers and μ_s stands for the service rate of that server. However, our reproduced simulation shows the estimation accuracy of Q_s, ie, the exponentially weighted moving averages (EWMAs) of the piggybacked queue size in history, maybe poor in C3, especially when a concurrency compensation term $n * OS_s$ takes effect. Actually, the EWMAs of the piggybacked queue size in history is still used for replica ranking hundreds of microseconds later in C3, during which the waiting queue size of the corresponding server may have changed greatly. Consequently, this poor timeliness of the feedback information leads to the poor estimation accuracy of Q_s. Moreover, as the rate control mechanism of C3 also heavily relies on the feedback information, the poor timeliness of feedback also has great impacts on the rate adjustment result. It makes the rate control of C3 helpless to cut the tail latency of key-value access operations.

Motivated by these observations, we propose the timeliness-aware predication-based (TAP) replica selection algorithm in this paper. TAP is aware of the timeliness of the feedback information. When the feedback information is fresh, TAP just selects replica servers the same as C3. When the feedback information is of poor timeliness, the following queue-size prediction method is developed. Specifically, TAP observes that the trend of the queue-size variation would not change fast, and the variation trend in the near future can be inferred by the variation of queue size (Δq_s) and the difference of Δq_s piggybacked from the replica server. Accordingly, three states of queue-size variation trends are defined by TAP: stable, increase, and decrease. Moreover, TAP distinguishes these states on receiving each feedback information. Although mistakes may be made in states distinction, TAP can revise it as soon as the next feedback packet is received and guarantee the correctness in most cases. When the state transforms, TAP drops all the feedback information in history. When the state keeps unchanged, TAP collects all feedback queue sizes received recently for the following queue-size prediction, rather than all the data in history. In details, under the states of increase and decrease, TAP fits a line of queue size against time with the feedback queue sizes according to the least squares method and computes the predicted queue size in this fitted line. Under the stable state, TAP takes the EWMAs of the feedback queue sizes as the predicted queue size. In this way, the predicted queue size is strictly in accordance with the current state of the queue-size variation trend and, thus, has a large probability to be closer to the real queue size, compared to the EWMAs of all the feedback queue sizes in history utilized in C3. In other words, TAP obtains more accurate queue-size estimation than C3 for replica ranking and, thus, outperforms C3 in terms of cutting the tail latency.

In sum, we make the following contributions in this paper.

- We reveal the timeliness issue of the framework developed by C3 and the impacts of this timeliness issue on both the replica ranking and the rate control in C3.
- To address these issues, we propose the TAP algorithm, which predicts the queue size of replica servers under the poor timeliness condition. Simulation confirms the outperformance of TAP over C3 on cutting the tail latency.

The rest of this paper is organized as follows. Section 2 is the background, and Section 3 is the motivation behind this work. Next, Section 4 describes the design of the TAP algorithm, and Section 5 is the evaluation of TAP. Finally, Section 6 concludes this paper.

2 | BACKGROUND

In the key-value store, when a web server receives an end-user request, it typically generates tens or hundreds of keys and needs to access the corresponding values from different servers. The web server is also the client in the following key-value store, as shown in Figure 1.

For each key, the corresponding value is typically replicated and distributed across different servers. When there is a key to send, the client can find the corresponding replica servers via consistent hashing and select a replica server to send the key for the key-value access. Obviously, to cut the tail latency of key-value accesses, the fastest server is expected in the replica selection of each key at a client. On the other hand, a server can receive keys from different clients, and its service rates for keys are time varying. When the server is busy, the newcomer key will be put into the waiting queue. After a key is served, the corresponding value is returned to the client.

It is hard to ensure the choice of the fastest server for every key such that the corresponding value is returned as soon as possible. One reason is that the service time of keys is time varying, as the performance of the server is influenced by many factors.[6,7] The other reason is that the size of the waiting keys at the server is unknown, due to the large degree of concurrency in key-value access. In other words, to know which

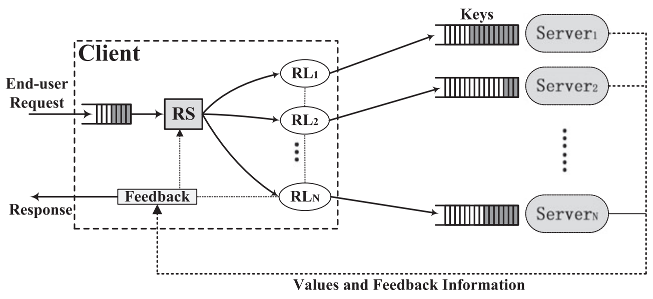


FIGURE 1 Framework of C3. RS, replica selection; RL, rate limiter

server is the fastest, we not only need to obtain the network latency but also have to capture the waiting time and the service time of keys at the server. Furthermore, the herd behavior, where the fast servers are preferred by most of the clients and get great performance degradation due to accompanying concurrent key-value accesses, should be avoided.

Aware of these challenges, C3 suggests the server to monitor the queue size of the waiting keys and its service time and piggyback this information to the client when the value is returned, as shown in Figure 1. The feedback information is utilized for both the replica ranking and the rate control in C3. Briefly, on the reception of a returned value, the client reads the feedback information and adjusts the rate limiter (RL) based on it via the rate control algorithm.

When there is a key to be sent, the client computes the scores of each replica, ranks the replicas based on the scores via the RS scheduler, and then subsequently inquires the states of RLs corresponding to these replicas. If the current sending rate is within an RL, the corresponding replica is chosen to send the key, and the inquiry stops. Otherwise, the following RL corresponding to a higher-score replica is inquired. If the current sending rate is not within all the RLs, the backpressure mechanism is triggered, and the key is put into the backlog queue until there is at least one server within the RL again.

In the replica ranking of C3, the replica server with the smallest expected waiting time $\bar{q}_s * \bar{T}_s$ is preferred, where $\bar{T}_s$ is the EWMAs of the feedback service time T_s of a key and $\bar{q}_s$ is the estimated queue size of waiting keys. Moreover, $\bar{q}_s$ is defined as follows:

$$\bar{q}_s \triangleq 1 + q_s + n * os_s. \tag{1}$$

Here, q_s is the EWMAs of the feedback queue size Q_s^f, n is the number of clients, and os_s is the number of outstanding keys whose values are not yet to be returned. In Equation (1), the term $n * os_s$ is considered as the concurrency compensation.[8] The specifical scoring function used for the replica ranking of C3 is as follows:

$$\Psi_s = \bar{R}_s - \bar{T}_s + \bar{q}_s{}^3 * \bar{T}_s, \tag{2}$$

where $\bar{R}_s$ is the EWMAs of the response times witnessed by the client, and thus, $\bar{R}_s - \bar{T}_s$ is considered as the delay. Moreover, the term $\bar{q}_s{}^3 * \bar{T}_s$ is the replacement of $\bar{q}_s * \bar{T}_s$ in order to penalize long queues in Equation (2), and the mechanism is named as cubic replica selection in C3. The replica server with the smallest Ψ_s is selected by the RS scheduler finally.

The rate control and backpressure mechanisms are as follows. As shown in Figure 1, a client maintains an RL for each server to limit the number of keys sent to the server within a specified time interval δ, named $sRate_s$. The key will not be sent to a server when the corresponding rate is limited. If the sending rates to all the replica servers of a key are limited, the key will be put into a backlog queue until the rate limitation of a replica server is released. The detailed rate control algorithm is borrowed from CUBIC.[16] Let $rRate_s$ be the number of values received from a server in a δ interval. $sRate_s$ is increased according to the following cubic function when $sRate_s < rRate_s$:

$$sRate \rightarrow \gamma * \left(\Delta T - \sqrt[3]{\frac{\beta * R_0}{\gamma}} \right)^3 + R_0, \tag{3}$$

wherein R_0 is the recorded $sRate_s$ before the previous rate decrease, ΔT is the elapsed time since the previous rate-decrease event, and γ is the constant coefficient. When $sRate_s > rRate_s$, and a $2 * \delta$ hysteresis period after the rate increase, $sRate_s$ is decreased to $\beta * sRate_s$, where β is a positive constant smaller than 1. The hysteresis period $2 * \delta$ is enforced for the measurement of $rRate_s$ after a rate increase. The rate adjustment is done on the receipt of each returned value, aiming to adapt $sRate_s$ to $rRate_s$, but the rate adjustment result takes effect when there are keys to be sent.

With the cooperation of the replica ranking method and the rate control and backpressure mechanisms, C3 can adapt to the time-varying service time across servers and avoid the herd behavior and accordingly improve the tail latency a lot, as confirmed by experiments and simulations in the work of Suresh et al.[8]

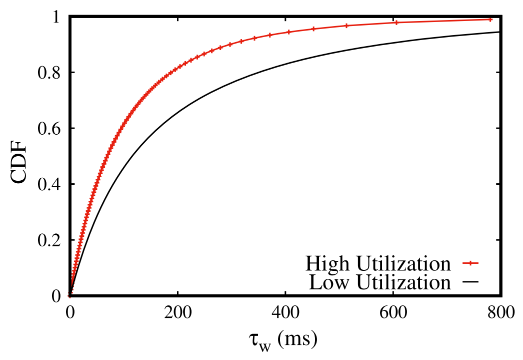


FIGURE 2 CDF of τ_w in C3

3 | MOTIVATION

Although C3 has great innovations on bringing feedback into the replica ranking and developing the rate control and backpressure mechanisms, the detailed replica ranking method can be further improved. Specifically, we find the timeliness issue of C3, which greatly influences the queue-size estimation for the replica ranking.

3.1 | Timeliness of feedback information

The feedback information plays a key role in the above framework of replica selection developed by C3. However, we find that the timeliness of the feedback information may be poor frequently. More specifically, the feedback information would be delayed for a propagation time τ_d before it arrives at the client, and there is also a time interval τ_w during the reception of the feedback information and the utilization of it for replica selection. To exhibit the timeliness of the feedback information, we reproduce simulations in C3 (see part A of section V for a detailed simulation configuration) and collect the values of τ_w before the sending of each key.[*] The cumulative distribution function of τ_w is shown in Figure 2. There is a very large probability that τ_w becomes as large as hundreds of milliseconds, especially when the server utilization is low. In contrast, the network latency τ_d is only on the order of several milliseconds. Therefore, the impacts of network latency can be ignored, whereas τ_w is the dominated factor.

As a step further, analyzing the simulation results, we find that the value of τ_w, ie, the timeliness of feedback information, is mainly influenced by the following factors. The first one is the traffic pattern, ie, the arrival pattern of keys at clients. For each pair of client and replica server, the client will not have the information of the replica server unless it receives the feedback information piggybacked by the previous key-value access operation. Therefore, when there is no key to be sent from a client to a replica server for a long time, the timeliness of feedback information about this replica server will be poor. The second factor is the impact of the replica selection result itself. If a replica server has not been selected for a long time, the feedback information about this replica server will be stale. Consequently, even if this replica server becomes good now, the client is unaware of it and, accordingly, would not select it. The third factor is the load and performance of the replica server. A client will receive the feedback information piggybacked in the value after a long time since it sends out the corresponding key to a replica server, when this replica has a large load or poor performance. It indicates that clients hardly update the information about the bad replica server on time. Obviously, in this case, the value of τ_w is larger than the latency of the key-value accesses from the bad replica servers. As the 99th percentile latency of key-value accesses can be one order of magnitude larger than the median latency,[6] the value of τ_w could also change in a very large range.

Therefore, the timeliness of feedback information in the above framework is poor. This may be also the reason why the replica selection algorithms in current classic key-value stores all do not heavily rely on the feedback information. Subsequently, we will focus on the impacts of the timeliness of feedback information on the replica ranking and the rate control of C3.

3.2 | Replica ranking

Due to the poor timeliness of feedback information, the estimation accuracy of the queue size of the waiting keys and the service time, both of which are crucial for the replica ranking of C3, is poor.

Specifically, as shown in Figure 3, we randomly choose a server s to show its queue size of waiting keys Q_s at each time when the scoring is executed at clients, as well as all of the feedback Q_s^f received from server s at clients, the os_s, and its estimation $\bar{q}_s$ on the queue size of server s in a random simulation time interval. There is a large difference among the piggybacked queue size Q_s^f, the estimation $\bar{q}_s$, and the real queue size Q_s.

[*] 600 000 values are collected. After the CDF is computed, we sample and present 5% of data to reduce the size of the figure, without changing the sharp of curves.

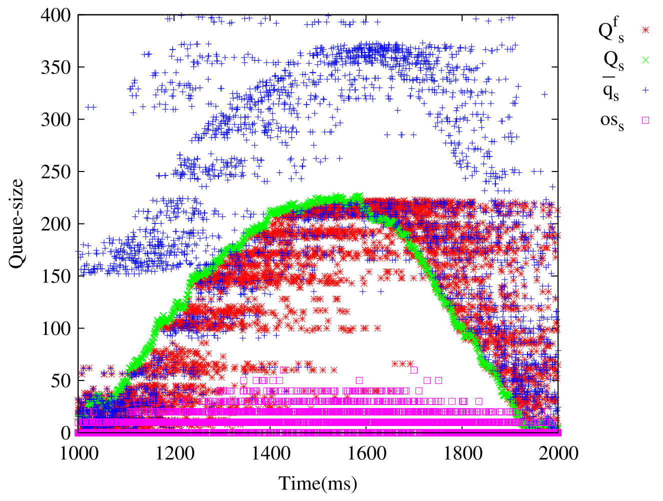


FIGURE 3 Queue size and its estimation in C3. Values exceeding 400 are not shown, and os_s is multiplied by 10 for the convenience of observation

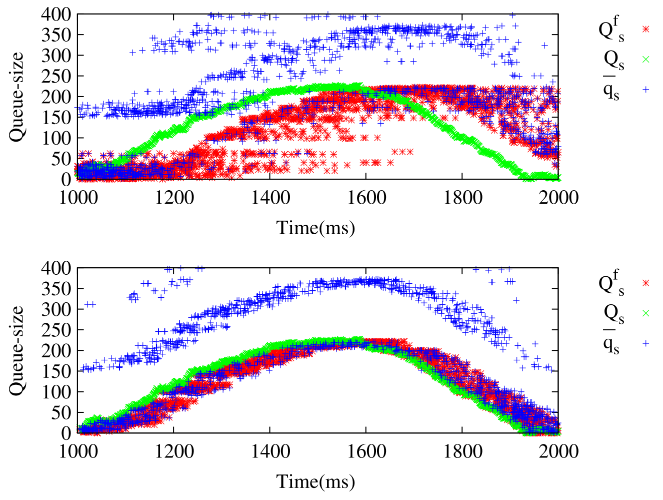


FIGURE 4 Queue size and its estimation in C3 with different values of τ_w. (Top) $\tau_w \geq 100$ ms; (Bottom) $\tau_w < 100$ ms

The large degree of concurrency in key-value access is considered as one of the main reason for this phenomenon, and accordingly, the term $n * os_s$ is utilized as the concurrency compensation in the computation of the estimated queue size $\bar{q}_s$, as represented in C3.[8] However, the term $n * os_s$ has not helped to improve the estimation accuracy of the queue size, as illustrated in Figure 3. In fact, dividing the data in Figure 3 into two subfigures with a threshold of 100 ms on τ_w, we show that *it is the poor timeliness of feedback information that leads to the poor estimation accuracy of the queue size*. Specifically, as shown in Figure 4, the difference among the real queue size Q_s, its estimation $\bar{q}_s$, and the feedback queue size Q^f_s is small when $\tau_w < 100\ ms$, excepting the condition that os_s is nonzero. When the value of τ_w becomes on the order of hundreds of milliseconds, the real queue size Q_s can change greatly during such a large time interval and, thus, cannot be estimated based on the stale feedback information. Therefore, when τ_w is large, replica selection methods independent of feedback information are needed. Similarly, the timeliness of the feedback service rate of servers may also become poor frequently.

Besides, the term $n * os_s$ cannot properly represent the degree of concurrency, as the degree of concurrency will be constrained by the rate control algorithm. Hence, it is not reasonable to assign weight n to os_s. In fact, we find that the term $n * os_s$ is helpful in simulation, not because it compensates the impact of concurrency and makes the queue-size estimation better but because the corresponding server should not be chosen before the outstanding keys are served. It should wait until the feedback information is piggybacked and renewed.

3.3 | Rate control

We also find that the timeliness of feedback information has great impacts on the rate control of C3. Although the rate adjustment is executed immediately after the feedback information is received from a server, this rate adjustment does not make sense if the client does not send any key to this server for a relatively long time interval. This is much different from the congestion control of the Internet, which assumes that there are always data to send. Even if the client sends keys to this server right after the rate adjustment, ie, the rate adjustment results take effect on time, the rate control algorithms face a forward time delay τ_w, which denotes the timeliness of feedback information. Note that the value of τ_w

can change in a very large range, ie, from several milliseconds to hundreds of milliseconds. This kind of delay would have great impacts on the stability of rate control algorithms. It will make the rate control of C3 helpless to cut the tail latency of key-value access operations. Consequently, we remove the rate control mechanism out of TAP.

In a word, we reveal the timeliness issue of C3 and its great impacts on the estimation of queue size in the replica ranking and the rate control.

4 | DESIGN OF TAP

In this section, we introduce the design of the TAP algorithm. Generally, the TAP algorithm follows the same framework as C3, but improves the scoring method in the face of the poor timeliness of feedback information.

4.1 | Basic idea

The procedure of the replica ranking of TAP is the same as that in C3, as illustrated in Figure 1. The main difference is that TAP is timeliness aware and employs prediction-based queue-size estimation for the scoring in replica selection in the face of the poor timeliness of feedback information. In addition, TAP does not employ rate limiters to each replica server at clients due to the ineffectiveness of rate control in C3.

Algorithm 1 Scoring on server *s* in TAP (when there is a key to send)

```
1:  Compute $\tau_w$
2:  if $\tau_w \geq 100ms$ then
3:      if $os_s == 0$ and $f_s == 0$ then
4:          $\bar{q}_s \leftarrow 0$    // no key to send for a long time
5:      else
6:          $\bar{q}_s \leftarrow 1 + os_s * n + prediction()$
7:      end if
8:  else
9:      $\bar{q}_s \leftarrow 1 + os_s * n + q_s$
10: end if
11: $Score \leftarrow R_s + \bar{q}_s{}^3 / \mu_s$
```

The detailed description of TAP is shown in Algorithm 1. When a key arrives at a client, the TAP algorithm selects between replica servers for each key in the following ways. First of all, TAP observes the value of τ_w, which represents the timeliness of the feedback information. Referring to the dynamic snitch mechanism of Cassandra, 100 ms is chosen to be the boundary. Specifically, if $\tau_w < 100$ ms, the timeliness of feedback information is considered as good, and the scoring method of TAP is the same as that of the C3 algorithm.

Otherwise, the feedback information is considered as out of date, and TAP will improve the scoring method. In other words, when $\tau_w \geq 100$ ms, the feedback information becomes useless with the time elapsing. It is unreasonable to make a replica selection decision depending on the outdated feedback information. Thus, TAP develops the following prediction-based scoring method for this condition. On one hand, TAP considers a special case. Let f_s be the number of times that the replica server *s* is not selected during the time interval τ_w recorded by the client. When $os_s = 0$ and $f_s = 0$, there is no key to be sent to the group of replica servers, which server *s* belongs to, for a long time τ_w due to the traffic pattern. The client tends to send the current key to server *s* to renew the information of replica server *s* in this condition.

On the other hand, except for the above special case, TAP utilizes the predicted queue size rather than the EWMA of the stale feedback queue size as C3 when the timeliness of feedback information is poor. The idea originates from the observation of the simulation results of C3 that the waiting queue size of servers is predictable under the poor timeliness condition. For example, Figure 5 shows the variation of the real queue size at a replica server chosen randomly and the feedback queue size Q_s^f received by a random client. From Figure 5, we have the following observations.

- The queue size changes frequently at servers, but the trends of queue-size variation can keep unchanged for a long period, from hundreds to thousands of milliseconds.
- Although the poor timeliness feedback cannot represent the present state of servers, it can still reflect the trends of queue-size variation.
- Although the feedback between one server and one client is relatively sparse, the feedback queue size can represent the trends of the queue size in the server roughly.

Therefore, it is probable to obtain a more accurate predicted queue-size estimation than the EWMAs of all historical data according to the trends of queue-size variation. Note that the prediction of queue size may not be absolutely accurate, but the prediction based on the right trends of queue-size variation can make sure that the predicted queue size has a large probability to be closer to the real queue size. In the following subsection, we will introduce the specified prediction method of TAP.

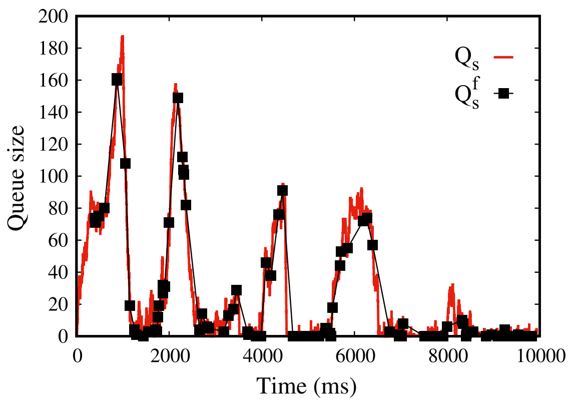


FIGURE 5 Real queue size and feedback queue size in C3

4.2 | Prediction of queue size

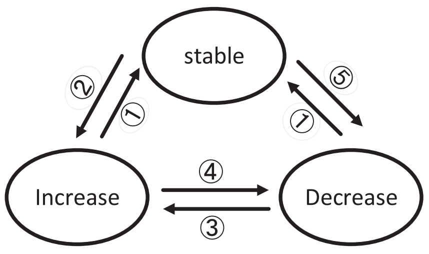
The prediction of queue size is composed of three parts. First, TAP judges the trends of queue-size variation as soon as receiving the feedback information. Second, TAP selects the useful feedback information according to the trends of queue-size variation. Finally, when a key arrives at client, the trends of queue-size variation and the effective feedback information are used for the queue-size prediction.

Feedback information. Feedback information is still important in the prediction part of the TAP algorithm. Specifically, the queue size Q_s^f and the service rate of the respective key μ_s are still piggybacked as in C3. Both $\bar{q}_s$ and $\bar{\mu}_s$ are still the EWMAs of Q_s^f and μ_s at clients. Excepting Q_s^f and μ_s, the variation of the waiting queue size Δq_s, which is collected every 20 ms, is also piggybacked to clients in the TAP algorithm.

Trends judgment. The prediction of queue size is based on the right trends of queue-size variation. Therefore, we first introduce how TAP judges the trends of queue-size variation. In the TAP algorithm, three different trends of the queue-size variation are defined, as represented by the three states shown in Figure 6. Once clients receive feedback information, TAP determines which state the current trends are in. The TAP algorithm distinguishes these states according to Q_s^f and Δq_s. Specifically, the difference between Δq_s and the last feedback Δq_s (denoted by Δq_{slast}) is computed at first. The sign of Δq_s can represent the trends of the queue-size variation. However, the change of the sign of Δq_s is always slower than the change of the trends of the queue-size variation, as shown in Figure 9. Combining Δq_s and $\Delta q_s - \Delta q_{slast}$, TAP can know the trends of queue-size variation earlier. Moreover, when the feedback queue size is small enough, TAP treats it to be stable, ie, the queue size would vary just a little. To smooth the small oscillation in a stable state, Q_s^f is the dominated value used to distinguish the stable state from other states.

In the following, we introduce the state transition in more detail according to Figure 6. The initial state of the queue-size variation is the stable state in TAP. The states transition conditions are listed in Figure 6. Any time when condition ①, ie, $Q_s^f \leq 30$, is satisfied, the future queue-size variation is considered to be stable, ie, TAP enters into the stable state. Conversely, $Q_s^f \leq 30$ is one of the necessary conditions that TAP deviates from the stable state and transfers to either the increase state or the decrease state.

② and ③ are the conditions that TAP enters into the increase state. If TAP enters into the increase state from the stable state, the queue size increases rapidly, and the sign of Δq_s changes rapidly. Thus, condition ② includes both inequalities $\Delta q_s > 0$ and $\Delta q_s - \Delta q_{slast} > 0$. If TAP enters into the increase state from the decrease state, either $\Delta q_s > 0$ or the decrease rate of the queue size gets slower indicates that the trend is going to change. Therefore, condition ③ involves either $\Delta q_s > 0$ or $\Delta q_s - \Delta q_{slast} > 0$. In both cases, inequality $Q_s^f > 30$ should be satisfied.



①: $Q_s^f \leq 30$;
②: $\Delta q_s > 0$ and $\Delta q_s - \Delta q_{slast} > 0$ and $Q_s^f > 30$;
③: $\Delta q_s > 0$ or $\Delta q_s - \Delta q_{slast} > 0$ and $Q_s^f > 30$;
④: $\Delta q_s < 0$ or $\Delta q_s - \Delta q_{slast} < 0$ and $Q_s^f > 30$;
⑤: $\Delta q_s < 0$ and $\Delta q_s - \Delta q_{slast} < 0$ and $Q_s^f > 30$;

FIGURE 6 State transition for the trends of queue-size variation

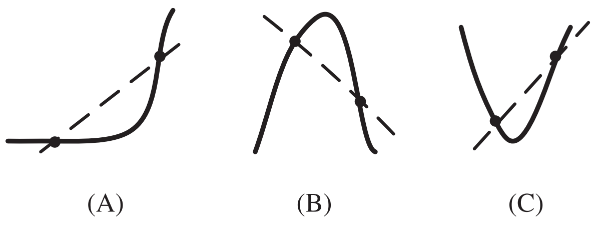


FIGURE 7 The cases where points need to be remained in the next trend state

④ and ⑤ are the conditions that TAP enters into the decrease state. These conditions are designed in a similar way with conditions ② and ③. Note that TAP cannot directly transfer from the stable state to the decrease state in fact. However, there may be no feedback information received during the increase state, and thus, the state transformation with condition ⑤ must be added.

In some cases, the TAP algorithm may fail to capture the trends correctly in time, but the mistake can be revised once the subsequent feedback information is received.

When the state is not changed, TAP stores all the feedback information and the time of receipt of this feedback information and uses this feedback information for the prediction of the queue size. When the state is changed, some old feedback queue size will be dropped.

Feedback information selection. After the judgment of trends, TAP performs a crucial operation, selects the effective feedback information. Obviously, only the feedback queue size in the same trend takes effect to the prediction, and the feedback queue size belonging to the different state is useless or even counterproductive. However, the feedback at the boundary of state transition, which is also the last feedback, is hard to define. It cannot be dropped roughly, as more positive feedback can increase prediction accuracy. Therefore, when the state is unchanged, TAP just simply puts the newcomer feedback into the feedback sets. When the state changes, the operation is a bit complex; TAP decides whether the boundary feedback information can be kept first and drops all useless feedback to ensure the right direction of prediction. TAP only stores the useful feedback queue size and the time receiving them.

Whether the boundary feedback information can be kept or not depends on whether it can be the starting point for the new trends. Actually, there are only three conditions that the feedback information at the boundary should remain. Specifically, when TAP transfers from the stable state to the increase state, the feedback queue size at this boundary will be involved in the prediction of queue size under the following increase trend, as shown in Figure 7A. When TAP transfers from the increase state to the decrease state, if the border feedback queue size is greater than the new feedback queue size, it can be the start point of the new state. As shown in Figure 7B, this border feedback information can be kept. When TAP transfers from the decrease state to the increase state, as shown in Figure 7C, the left point is smaller than the new feedback, and the increase state can be considered start over here. Excepting the above three conditions, all feedback in the last state is dropped. With these operations, most of the usable feedback points are obtained.

Queue-size prediction. The effective feedback information and the queue-size variation is used to make predictions when clients select a replica. Once there is a new key coming, TAP provides two different predicted methods according to the state of queue-size variation. Specifically, if the state is stable, TAP just uses the EWMAs of all the feedback queue sizes collected under this trend as the predicted queue size. On the other hand, if the state is increase or decrease, TAP fits a line with the least squares method by using these stored feedback queue sizes and computes the queue size under the current time based on this fitted line.

In general, for each key, we can make a prediction for the queue size of each replica. Following the trends, the predicted queue size will be closer to reality in a high probability.

4.3 | Discussion

Compared to C3, TAP utilizes the same framework and has a similar replica ranking method, but excludes the rate control and backpressure methods. Hence, the same as C3, TAP is also simple and implementable, can avoid the herd behavior, and is adaptive to the time-varying performance across servers.

The design of TAP is motivated by the observation of the fundamental timeliness issue of C3. Because replica selection is needed for each lightweight key-value access operation, it is impossible to obtain better information of replica servers without greatly increasing the overhead. Based on existing information, C3 always conducts good replica selection when the timeliness of feedback information is good. Thus, TAP focuses on only the circumstance of poor timeliness of feedback information. Other alternative approaches such as revising the framework of replica selection to avoid the timeliness issue may also work. However, we left these reforming approaches as the future work.

The performance of TAP will be better than C3 due to the following reasons. First, TAP's scoring method distinguishes the timeliness of feedback information. In TAP, the poor timeliness feedback information is not directly used for scoring when $\tau_w \geq 100$ ms. In reality, because of the large degree of concurrency and the poor timeliness of feedback, it is hard to accurately estimate the queue size of waiting keys of servers, especially when $\tau_w > 100$ ms. Second, when the timeliness of feedback information is poor due to the traffic pattern, ie, a client does not have keys to a replica server for a long time, the client will send the current key to this replica server with TAP, rather than still scoring this replica server with the stale feedback information for replica selection as in C3. In this way, TAP will remit the influence of the traffic pattern on timeliness. Finally, excepting the special case where $\tau_w \geq 100$ ms, $os == 0$, and $f_s == 0$, TAP utilizes the predicted queue size rather than the EWMA of the stale feedback queue size as C3 when the timeliness of feedback information is poor. Obviously, when TAP successfully captures the trend of queue-size variation, its queue-size prediction based on the data collected within the recent trend of queue-size variation is probably better than EWMAs of all the piggybacked queue-size data in history. In other words, compared to C3, TAP rules out the interference of invalid historical feedback information for queue-size estimation. Therefore, TAP has a large probability of selecting better replica servers than C3 under the circumstance of poor feedback information timeliness.

As a step further, there are corner cases where many of the replica selection decisions of TAP are worse than those of C3, but the number of worse replica selection in TAP would be small compared to the large total number of replica selections for all keys. For example, with the increased number of clients, the communication density between each pair of client and replica server becomes sparse, and the queue-size prediction accuracy decreases in TAP. In this condition, TAP may also incur suboptimal replica selection for certain keys. However, the timeliness of feedback information would also become worse in C3. In total, TAP would not render things worse compared with C3, although its performance may be decreased in certain conditions. Similar phenomena may occur when the distribution of feedback information among clients and servers is changed with the skewed demands.

5 | EVALUATION

5.1 | Implementation and setup

Setup. We implement TAP based on the open-source code of C3.[8] As in C3, the workload generators create keys at a set of clients according to a Poisson arrival process to mimic the arrival of user requests at web servers.[3] These keys are sent to a set of servers, each of which is chosen according to the replica selection algorithm at the client from three replica servers. The server maintains a FIFO queue for waiting keys but can serve a tunable number (four, by default) of requests in parallel. The service time of each key is drawn from an exponential distribution with a mean service time of $T_s = 4$ ms as in the work of Vulimiri et al.[9] The time-varying performance fluctuations of servers are simulated by a bimodal distribution[17] as follows: each server sets its mean service rate either to T_s^{-1} or to $D * T_s^{-1}$ with uniform probability for every fluctuation interval T ms, where D is a range parameter with a default value of 3. The arrival rate of keys corresponds to 70% of the average service rate of the system by default.

Configuration. The configurations and metrics are the same as those for C3. Moreover, 200 workload generators, 50 servers, and 150 clients are used by default. The one-way network latency is 250 μs. The 99th percentile latency is computed by taking the average and standard deviation of five repeated simulations, where different random seeds and 600 000 keys are used in each run.

5.2 | Simulation results

Impacts of time-varying service rate. As both C3 and TAP are designed to be adaptive with the time-varying performance across servers, we evaluate TAP with the time-varying service rate at first. With the fluctuation interval T of the average service time of servers changing from 10 to 500 ms, the 99th percentile latencies are shown in Figure 8. The 99th percentile latencies of C3 are almost the same as those in Figure 14 in the work of Suresh et al.[8] It can serve as evidence for the correctness of our simulator implementation. Moreover, the 99th percentile latencies of schemes satisfy $ORA < TAP < C3$. It indicates that the tail latency can be cut greatly with perfect knowledge of the queue size and the service time, and TAP outperforms C3 due to its better queue-size estimation based on prediction. Note that the difference among the 99th percentile latencies becomes significant, when the time interval T is a large value like 500 ms, ie, the average service time of servers is not changed frequently. When $T = 10$ ms, ie, the average service time of servers changes frequently, the feedback information becomes stale very

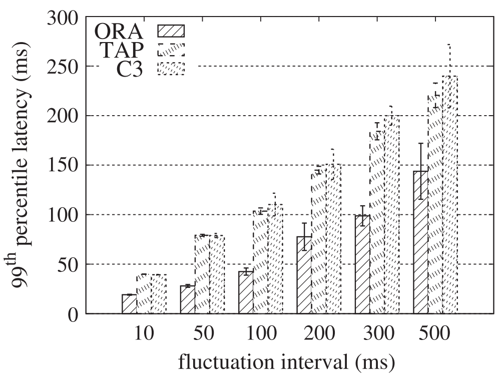


FIGURE 8 The impacts of different fluctuation intervals T

fast. Correspondingly, the replica ranking based on feedback information becomes poor in both TAP and C3. Therefore, the difference between TAP and C3 is small with $T = 10$ ms.

Queue-size estimation. Subsequently, in the simulation of TAP, one client and one server are randomly selected, respectively, and their queue-size information is presented in Figure 9 to show the accuracy of queue-size estimation in TAP. In Figure 9, Q_s is the real queue size of the server, and Q_s^f is the feedback queue size. When $\tau_w \geq 100$ ms, we record q_e, which is the EWMAs of all the piggybacked queue size in history as in C3, and q_p, which is the predicted queue size of TAP. Obviously, the queue-size estimation of TAP is more accurate than that of C3, especially when the trend of queue-size variation is not changed. For example, in the increase state between 7500 and 8000 ms, q_e just stays unchanged without new feedback information, whereas q_p increases over time according to the prediction of TAP.

Obviously, the correctness of the trend judgment on queue-size variation definitively influences the accuracy of prediction in TAP. Figure 9 also shows our foresight of the trend of queue-size variation. The three different lines represent the corresponding state transition. In most conditions, the correctness of the trend judgment can be guaranteed. Sometimes, TAP makes a wrong decision, eg, when the increase state changes to the decrease state at about 8000 ms. However, once new feedback information is returned, the trend judgment is corrected.

To be more convincing, we count up the difference between the real queue size and queue-size prediction result of TAP or the EWMAs of the queue size of C3 throughout the whole simulation, respectively. When the difference of the queue size obtained from two methods is smaller than or equal to 1, we consider both methods similar to each other. We repeat this simulation five times. As a result, TAP is better than C3 at **57%** of all points, these two algorithms are similar at **13%** of all points, and TAP is worse than C3 at the remaining **30%** of all points. In total, TAP has more accurate queue-size estimation than the EWMAs of C3.

Latency. To compare the performance of C3 and TAP in detail, we also illustrate the 50th percentile latencies, the 95th percentile latencies, and the 99.9th percentile latencies in Figure 10, when $T = 500$ ms. Under all of these metrics, TAP outperforms C3, and the advantage of TAP becomes the most significant with the metric of 99.9th percentile latency. In fact, the CDF of the latencies of all key-value accesses can illustrate the advantage of TAP over C3 better, as shown in Figure 11.

Impacts of utilization. Subsequently, we compare the performance of TAP and C3 under different utilizations of the whole key-value system. In detail, we gradually increase the utilization from 40% to 80%. As shown in Figure 12, the 99th percentile tail latency of TAP and C3 is almost the same when the utilization is less than or equal to 65%. However, when the utilization is larger than 65%, the 99th percentile tail latency of TAP is smaller than that of C3, and the advantage of TAP over C3 becomes significant with the increase of utilization. This is because the timeliness of feedback becomes poor in C3 when the average load of replica servers is high. In contrast, TAP can predict the queue size of replica

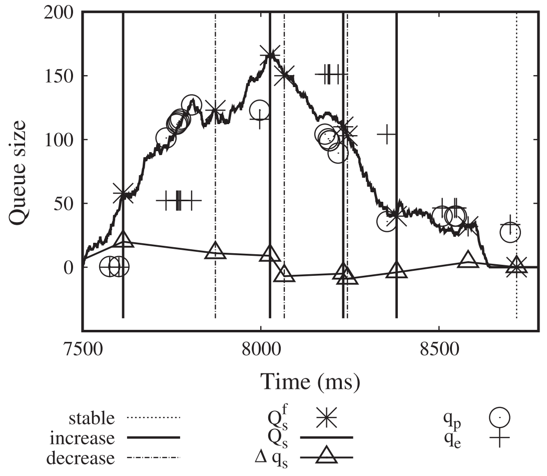


FIGURE 9 The effect of predicted queue size and the trends judgment

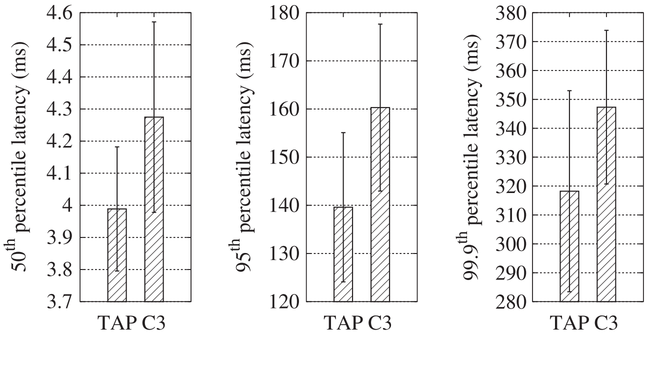


FIGURE 10 Latencies of C3 and TAP

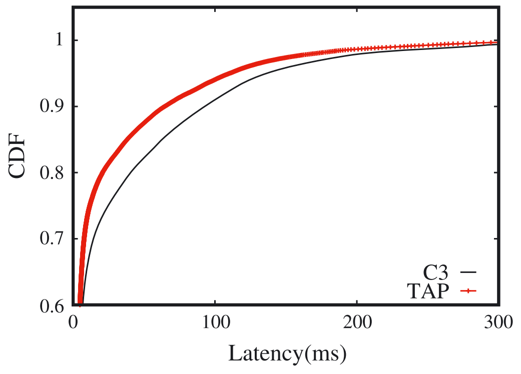


FIGURE 11 CDF of the latency of C3 and TAP

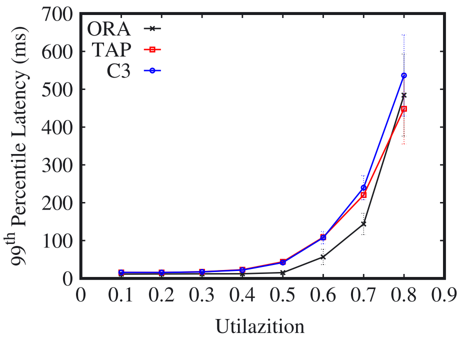


FIGURE 12 The impact of utilization

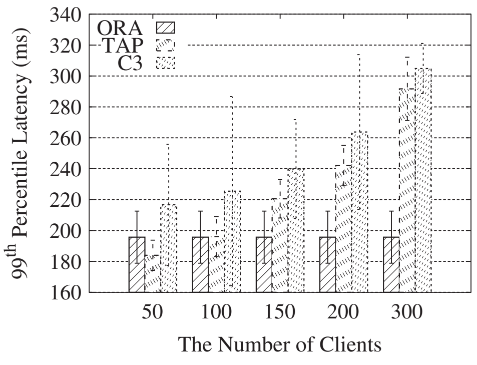


FIGURE 13 Impacts of the number of clients

servers to remit the impacts of the timeliness issue incurred by the high load. Moreover, the intensive feedback information with high load is also in favor of the prediction in TAP.

Impacts of the number of clients. Next, we compare the performance of TAP and C3 under the different numbers of clients. Figure 13 shows the 99th percentile tail latency when the number of clients is varied from 50 to 300. Obviously, the smaller the number of clients, the smaller the tail latency. This is because the number of clients directly influences the degree of concurrency. The smaller proportion of the number of clients over the number of servers, the lighter the degree of concurrency. Accordingly, the queue size changes slowly, and both C3 and TAP have better performances in terms of cutting the tail latency. As a step further, when the number of clients is small, the feedback information received at each client is intensive. Therefore, TAP has a more accurate queue-size prediction and a more significant advantage on cutting the tail latency compared to C3. Conversely, in the face of spare feedback information, TAP is similar to C3.

Impacts of the skewed demands. As many realistic workloads are skewed in practice,[18] we also study the effect of heavy demand skews. In detail, 80% of total keys are generated by 20% clients and 50% clients, respectively. Figures 14 and 15 show the simulation results, respectively. Obviously, the tail latency of TAP is much smaller than that of C3 in the face of heavily skewed traffic. This is because the feedback information is more intensive at a certain number of clients under the heavily skewed traffic, and accordingly, the queue-size estimation of TAP is more accurate. It confirms the advantage of TAP over C3 as a step further.

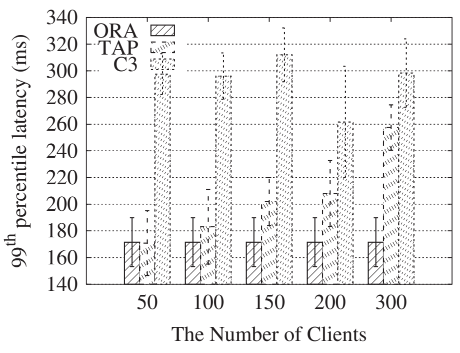


FIGURE 14 Impact of the skewed client demands: 20% clients generate 80% of total keys

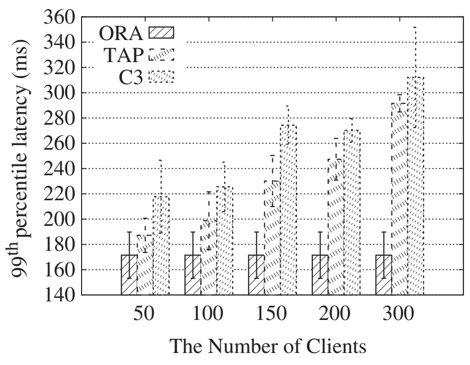


FIGURE 15 Impact of the skewed client demands: 50% clients generate 80% of total keys

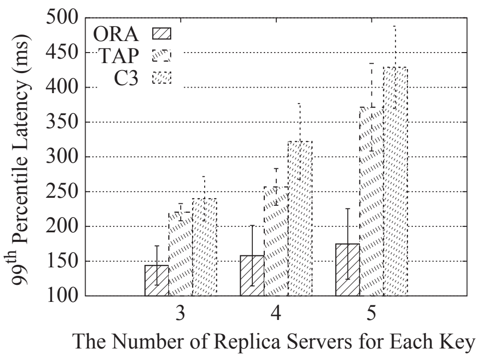


FIGURE 16 Impact of the number of replica servers for each key

Impacts of the replica factor. Finally, Figure 16 shows the impact of the number of replica servers for each key. Here, we vary the replica factor from 3 to 5. With the increase of the number of replica servers for each key, it becomes more difficult to select the fastest replica server. Therefore, the 99th tail latency of both TAP and C3 increases with the increase of the number of replica servers for each key. Moreover, TAP still outperforms C3.

6 | CONCLUSION

Nowadays, the replica selection scheme is crucial to reduce the tail latency of key-value accesses. To address the challenges of time-varying performance across servers and the herd behavior, C3 has been proposed recently. In this paper, we find drawbacks of C3 in terms of poor queue-size estimation and an ineffective rate control mechanism and reveal the timeliness issue of the framework developed by C3. These insights motivate us to further develop the TAP scheme, improving the replica ranking by predicting the queue size at replica servers under the poor timeliness condition. Evaluation results based on the open-source code of C3 confirm the good performance of TAP against C3. Further work can be, but not limited to, the evaluation of TAP with real experiments and totally addressing the timeliness issue of the framework developed by C3.

ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China (NSFC) under grant 61502539, the China Postdoctoral Science Foundation under grants 2015M582344 and 2016T90761, the Hunan Province Science and Technology Plan project under grant 2016JC2009, and the National Key Research and Development of China under grant 2018YFB170043. This work is the extension of our conference papers.[19,20]

ORCID

Wanchun Jiang https://orcid.org/0000-0001-5067-321X

REFERENCES

1. DeCandia G, Hastorun D, Jampani M, et al. Dynamo: Amazon's highly available key-value store. In: Proceedings of the Twenty-First ACM SIGOPS Symposium on Operating Systems Principles (SOSP); 2007; Stevenson, WA.
2. Jalaparti V, Bodik P, Kandula S, Menache I, Rybalkin M, Yan C. Speeding up distributed request-response workflows. In: Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM; 2013; Hong Kong.
3. Nishtala R, Fugal H, Grimm S, et al. Scaling memcache at Facebook. In: Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI); 2013; Lombard, IL.
4. Rumble SM, Ongaro D, Stutsman R, Rosenblum M, Ousterhout JK. It's time for low latency. Paper presented at: 13th Workshop on Hot Topics in Operating Systems (HotOS); 2011; Napa, CA.
5. Brutlag J. Speed matters. 2009. http://googleresearch.blogspot.com/2009/06/speed-matters.html
6. Dean J, Barroso LA. The tail at scale. *Commun ACM*. 2013;56(2):74-80.
7. Kambadur M, Moseley T, Hank R, Kim MA. Measuring interference between live datacenter applications. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC); 2012; Salt Lake City, UT.
8. Suresh L, Canini M, Schmid S, Feldmann A. C3: cutting tail latency in cloud data stores via adaptive replica selection. In: Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI); 2015; Oakland, CA.
9. Vulimiri A, Godfrey PB, Mittal R, Sherry J, Ratnasamy S, Shenker S. Low latency via redundancy. In: Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT); 2013; Santa Barbara, CA.
10. Wu Z, Yu C, Madhyastha HV. CosTLO: cost-effective redundancy for lower latency variance on cloud storage services. In: Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI); 2015; Oakland, CA.
11. OpenStack. Swift documentation. http://docs.openstack.org/developer/swift/
12. Borthakur D. The Hadoop distributed file system: architecture and design. *Hadoop Proj Website*. 2007;11(11):1-10.
13. Riak: load balancing and proxy configuration. 2014. http://docs.basho.com/riak/1.4.0/cookbooks/Load-Balancing-and-Proxy-Configuration/
14. Bogdanov K, Peón-Quirós M, Maguire Jr GQ, Kostić D. The nearest replica can be farther than you think. In: Proceedings of the Sixth ACM Symposium on Cloud Computing (SoCC); 2015; Kohala Coast, HI.
15. Apache Cassandra 2.0 Documentation. 2014. http://www.datastax.com/documentation/cassandra/2.0
16. Ha S, Rhee I, Xu L. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS Oper Syst Rev*. 2008;42(5):64-74.
17. Schad J, Dittrich J, Quiané-Ruiz JA. Runtime measurements in the cloud: observing, analyzing, and reducing variance. *Proc VLDB Endow*. 2010;3(1-2)460-471.
18. Atikoglu B, Xu Y, Frachtenberg E, Jiang S, Paleczny M. Workload analysis of a large-scale key-value store. In: Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS); 2012; London, UK.
19. Jiang W, Fang L, Xie H, Wang J. Tars: timeliness-aware adaptive replica selection for key-value stores. Paper presented at: 2017 26th International Conference on Computer Communication and Networks (ICCCN); 2017; Vancouver, Canada.
20. Fang L, Xie H, Zhou X, Jiang W. PRS: Predication-based replica selection algorithm for key-value stores. Paper presented at: Third International Conference of Pioneering Computer Scientists, Engineers and Educators; 2017; Changsha, China.

How to cite this article: Zhou X, Fang L, Xie H, Jiang W. TAP: Timeliness-aware predication-based replica selection algorithm for key-value stores. *Concurrency Computat Pract Exper*. 2019;31:e5171. https://doi.org/10.1002/cpe.5171