Towards the Tradeoff Between Service Performance and Information Freshness

Zhongdong Liu and Bo Ji

***Abstract*—The last decade has witnessed an unprecedented growth in the demand for data-driven real-time services. These services are fueled by emerging applications that require rapidly injecting data streams and computing updated analytics results in real-time (or near-real-time). In many of such applications, the computing resources are often shared for processing both updates from information sources and queries from end users. This requires joint scheduling of updates and queries because the service provider needs to make a critical decision upon receiving a user query: either it responds immediately with currently available but possibly stale information, or it first processes new updates and then responds with fresher information. Hence, the tradeoff between *service performance* (e.g., response time) and *information freshness* naturally arises in this context. To that end, we propose a simple *single-server two-queue model* that captures the coupled scheduling of updates and queries and aim to design scheduling policies that can properly address the important tradeoff between performance and freshness. Specifically, we consider the *response time* as a performance metric and the *Age of Information (AoI)* as a freshness metric. After demonstrating the limitations of the simplest First-Come-First-Served (FCFS) policy, we propose two *threshold-based policies*: the Query-k policy that prioritizes queries and the Update-k policy that prioritizes updates. Then, we rigorously analyze both the response time and the *Peak AoI (PAoI)* of the threshold-based policies. Further, we propose the Joint-(M, N) policy, which allows flexibly prioritizing updates or queries through choosing different values of two thresholds M and N. Finally, we conduct simulations to evaluate the response time and the PAoI of the proposed policies. The results show that our proposed threshold-based policies can effectively control the balance between performance and freshness.**

I. Introduction

The last decade has witnessed an unprecedented growth in the demand for data-driven real-time services (built on frameworks such as Apache Storm [1]). These services are fueled by emerging applications that require rapidly injecting data streams and computing updated analytics results in *real-time* (or *near-real-time*). For such applications, *service performance* (e.g., response time) perceived by end users is typically a primary concern and has been extensively studied in the literature. Yet, *freshness* of the information received by end users, another equally or even more important concern, has not received enough attention. Unilaterally optimizing service performance without accounting for information freshness could render users receive stale information, which is potentially of much less value or even useless. For example, upon receiving a user query, in order to minimize the response time the service provider may respond immediately with currently available but possibly outdated information. On the other hand, it may choose to first process new updates and then responds with fresher information if the goal is to optimize freshness. Hence, *there exists a natural tradeoff between service performance (e.g., response time) and information freshness*.

In this paper, we consider the *response time* as a performance metric and the *Age of Information (AoI)* [2] as a freshness metric. While the response time has been shared as a standard performance metric, the AoI, which is defined as the time elapsed since the generation of the freshest update among those that have been delivered to the receiver (see Section III for the formal definition), is a recently proposed freshness/timeliness metric [2]. Note that there is a limited body of existing work (see, e.g., [3]–[5]) that investigates the important tradeoff between performance and freshness as we do. However, all of these studies provide heuristic solutions only and fall short of theoretical results with rigorous analysis.

To that end, in this paper we aim to fill this important gap and design efficient policies that can properly address the critical tradeoff between performance and freshness. We summarize the main contributions of this paper as follows.

First, we propose a simple *single-server two-queue model* that captures the coupled scheduling of updates and queries. Second, after demonstrating the limitations of the First-Come-First-Served (FCFS) policy, we propose two *threshold-based policies*: the Query-k policy that prioritizes queries and the Update-k policy that prioritizes updates. Then, we rigorously analyze the response time and the Peak AoI (PAoI) (i.e., the maximum value of the AoI at the server immediately before a new update is processed) [6] of these two policies. To the best of knowledge, *this is the first analytical work that systematically studies the tradeoff between performance and freshness in a rigorous manner*. Further, we propose the Joint-(M, N) policy, which allows flexibly prioritizing updates or queries through choosing different values of two thresholds M and N. Finally, we conduct simulations to evaluate the response time and the PAoI of the proposed policies. The results show that our proposed threshold-based policies can effectively control the balance between performance and freshness.

The rest of this paper is organized as follows. We first discuss related work in Section II. Then, we describe our proposed model in Section III. In Section IV, we analyze the response time and the PAoI of our proposed threshold-based policies, followed by a discussion on the simulation results in Section V. Finally, we make concluding remarks in Section VI.

This work was supported in part by the NSF under Grants CCF-1657162 and CNS-1651947.

Zhongdong Liu (zhongdong.liu@temple.edu) and Bo Ji (boji@temple.edu) are with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA.

Due to space limitations, all the proofs are omitted here and provided in our online technical report [7].

II. Related Work

Research on the performance (e.g., response time) started very early. In [8], it studies under what condition such that the performance in FCFS policy is better than that in *Process-Sharing (PS)* policy. The authors conclude that a special task assignment which has the ability to inspect incoming tasks and assign them to hosts for service can achieve this goal. Further, the performance comparison between the *Shortest-Remaining-Processing-Time (SRPT)* policy and the PS policy is studied in [9], it shows SRPT has better performance than PS when the service load is high. Based on the SRPT policy, the work of [10] improves the performance by giving preference to the queries whose remaining size or original size is small. The simulation results show that even the queries for large files suffer little in this SRPT-based scheduling. The first analytical study of performance and robustness in threshold-based resource allocation policies appears in [11], where the authors conclude that using multiple thresholds does not always provide benefits to the performance and robustness. However, there is still no analytical work considering the freshness in these studies.

The notion of AoI is formally introduced in [2], where the authors analyze the time average AoI in M/M/1, M/D/1, and D/M/1 systems under the FCFS policy. Since this seminal work, the study on the AoI has attracted a lot of research interests. There is a large body of work that focuses on the analysis of the AoI under a number of queueing model. For example, the work of [6], [12], [13] focuses on the model where the updates arrive according to the Poisson process and are served by a single server. There is another body of work that considers how to minimize the AoI by carefully designing scheduling policies in different scenarios (e.g., wireless networks [14], [15] and energy harvesting networks [16], [17]). In [18], the authors propose the Pull model for investigating the expected AoI at the user's side and discover a new tradeoff between different levels of information freshness and different response times across the servers. Besides the above work that focuses on the analysis and optimization of the AoI, several other work also considers applications where the AoI is highly relevant (see, e.g., [19], [20]).

Despite the aforementioned studies on service performance and information freshness, the tradeoff between them has often been neglected in the literature (partially due to the nature of the considered applications), except for the following limited work. In [3], the tradeoff of performance and freshness has been considered for database-driven web servers, where the goal is to optimize performance under the freshness constraint. The work of [4] proposes to combine performance and freshness into a single compound metric and addresses the tradeoff between them through optimizing the compound metric. Further, the work of [4] has been extended to account for user preference for performance and freshness [5]. In stark contrast to these studies that provide heuristic solutions only, in this paper we aim to systematically understand this tradeoff by providing theoretical results with rigorous analysis.

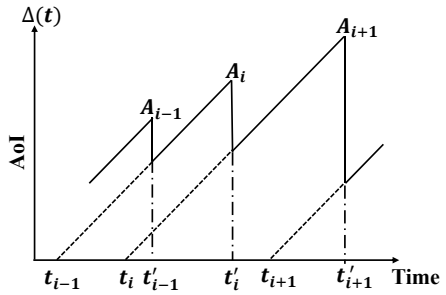


Fig. 1: An example of the AoI evolution

III. System Model

In this section, we describe the single-server two-queue model and give the formal definition of the AoI and the PAoI.

We consider a queueing system where a single server is shared by two M/M/1 queues. One is the update queue that buffers updates coming from the information source, and the other is the query queue that buffers queries from the user. We assume that the arrival processes of the updates and the queries are both Poisson with rate λ_u and λ_q, respectively. Also, we assume that the service times of the updates and the queries are both exponentially distributed with mean $1/\mu_u$ and $1/\mu_q$, respectively. Therefore, the loads of the update queue and query queue can be denoted by $\rho_u = \lambda_u/\mu_u$ and $\rho_q = \lambda_q/\mu_q$, respectively. In addition, we assume that the server does not remain at an empty queue if the other queue is nonempty. Further, let $X_{u,i}$ be the inter-arrival time between the i-th update and the $(i-1)$-th update, let $S_{u,i}$ be the service time of the i-th update, let $T_{u,i}$ be the system time of the i-th update, and let $N^u_{u,i}$ (resp., $N^q_{u,i}$) be the number of updates (resp., queries) seen by the i-th update upon its arrival. More generally, we drop subscript i and use X_u, S_u, and T_u to denote the corresponding quantities for an ordinary update. For example, X_u denotes the inter-arrival time of an update. Similarly, we define $X_{q,i}$, $S_{q,i}$, $T_{q,i}$, $N^u_{q,i}$, $N^q_{q,i}$, X_q, S_q, and T_q for queries. Also, we use N_u (resp., N_q) to denote the number of updates (resp., queries) in the system.

Next, we give the formal definition of the AoI and the PAoI. Let $U(t)$ denote the generation time of the freshest update among those that have been processed by the server. We use $\Delta(t)$ to denote the AoI at time t, which is defined as the time elapsed since the generation of this freshest update, i.e., $\Delta(t) \triangleq t - U(t)$. An example of the AoI evolution is shown in Fig. 1. The AoI increases linearly as time goes until a new update is completely processed. For example, consider the i-th update, which is generated at time t_i and finishes processing at time t'_i. When the server finishes processing the i-th update, the AoI drops to the value of $t'_i - t_i$, i.e., the system time of the i-th update. Then, the average AoI can be defined as

$$\Delta = \lim_{\tau\to\infty} \frac{1}{\tau} \int_0^\tau \Delta(t) dt. \tag{1}$$

TABLE I: Summary of Key Notations

Symbol	Meaning
λ_u	Arrival rate of the updates
μ_u	Service rate of the updates
ρ_u	Load of the update queue (i.e., $\rho_u = \lambda_u/\mu_u$)
$X_{u,i}$	Inter-arrival time between the i-th and $(i-1)$-th updates
$S_{u,i}$	Service time of the i-th update
$T_{u,i}$	System time of the i-th update
$N^u_{u,i}$	Number of updates seen by the i-th update upon arrival
$N^q_{u,i}$	Number of queries seen by the i-th update upon arrival
N_u	Number of updates in the system
A_i	The i-th PAoI
λ_q	Arrival rate of the queries
μ_q	Service rate of the queries
ρ_q	Load of the query queue (i.e., $\rho_q = \lambda_q/\mu_q$)
$X_{q,i}$	Inter-arrival time between the i-th and $(i-1)$-th queries
$S_{q,i}$	Service time of the i-th query
$T_{q,i}$	System time (or response time) of the i-th query
$N^u_{q,i}$	Number of updates seen by the i-th query upon arrival
$N^q_{q,i}$	Number of queries seen by the i-th query upon arrival
N_q	Number of queries in the system
ρ	Total load (i.e., $\rho = \rho_u + \rho_q$)

Analyzing the average AoI involves two important quantities: the inter-arrival time and the system time of the updates. The fact that the latter is dependent on the former often renders the analysis of the average AoI quite challenging except for some simple settings (e.g., M/M/1 queue) [2]. On the other hand, the analysis of the average PAoI is usually more tractable. The PAoI is the maximum value of the AoI achieved immediately before a new update is processed. Let A_i be the i-th PAoI. From Fig. 1, we can see $A_i = t'_i - t_{i-1}$. This can be rewritten as the sum of the inter-arrival time between the i-th update and the previous update (i.e., $t_i - t_{i-1}$) and the system time of the i-th update (i.e., $t'_i - t_i$). Therefore, the expected PAoI can be expressed as

$$\mathbb{E}[A] = \mathbb{E}[X_u] + \mathbb{E}[T_u], \tag{2}$$

where $\mathbb{E}[\cdot]$ is the expectation operator and A is the PAoI corresponding to an update. While computing the first term of the right hand side (RHS) of Eq. (2) is trivial, i.e., $\mathbb{E}[X_u] = 1/\lambda_u$, computing the second term $\mathbb{E}[T_u]$ is more involved as it depends on the underlying scheduling policy.

To measure the service performance, we consider the average response time, i.e., the system time of the queries T_q.

For quick reference, we provide a summary of the key notations of this paper in Table I.

IV. Scheduling Policies

In this section, we first consider a simple scheduling policy, the FCFS policy, and explain its limitation in balancing the service performance and information freshness. Then, we propose two threshold-based policies: the Query-k policy that prioritizes queries and the Update-k policy that prioritizes updates, and rigorously analyze the response time and the PAoI under these policies. Further, we propose the Joint-(M, N) policy, where we jointly set thresholds M and N for the updates and the queries, respectively. The Joint-(M, N) policy generalizes the Query-k policy and the Update-k policy and allows flexibly prioritizing updates or queries through choosing different values of M and N.

A. The FCFS Policy

We first consider the FCFS policy, a simple policy that serves updates and queries according to the order of their arrivals. Preemption is not allowed during the service. The main results for the FCFS policy are stated in Proposition 1.

Proposition 1: Under the FCFS policy, the expected response time is

$$\mathbb{E}[T_q] = \frac{\rho_u/\mu_u + (1-\rho_u)/\mu_q}{1-\rho_u-\rho_q}, \tag{3}$$

and the expected PAoI is

$$\mathbb{E}[A] = \frac{1}{\lambda_u} + \frac{\rho_q/\mu_q + (1-\rho_q)/\mu_u}{1-\rho_u-\rho_q}. \tag{4}$$

The FCFS policy is a simple algorithm and is easy to implement in practice. However, a key limitation is that the FCFS policy does not provide a knob for prioritizing either updates or queries and thus cannot achieve a desired balance between service performance and information freshness. To that end, in the following subsections we will propose threshold-based policies that can prioritize either queries or updates and thus control the tradeoff between the response time and the PAoI.

B. The Query-k Policy

In this subsection, we propose the Query-k policy that sets a threshold k for the query queue and prioritizes the queries whenever the length of the query queue reaches k. We will analyze the response time and the PAoI under this policy.

Specifically, the Query-k policy functions in the following manner: (i) there is one single threshold k for the query queue; (ii) when the server is currently serving the update queue, the server has to switch from the update queue to the query queue instantly either if the number of queries reaches the threshold k (thus preemption is allowed in this policy) or the update queue becomes empty; (iii) no work of updates is lost due to the switches; (iv) once the server switches to the query queue, it needs to empty all queries waiting in the queue before it switches back to the update queue; (v) within each queue, FCFS is applied.

In the following, we will discuss three cases of the threshold value: 1) $k = 1$, 2) $1 < k < \infty$, and 3) $k = \infty$.

1) Threshold k=1: In this case, the server processes queries as long as the query queue is non-empty. Hence, the query queue always has a higher priority than the update queue. This model is equivalent to a preemptive priority queue with two classes of jobs [21, Ch. 32].

Proposition 2: Under the Query-1 policy, the expected response time is

$$\mathbb{E}[T_q] = \frac{1}{\mu_q} + \frac{\rho_q/\mu_q}{1-\rho_q}, \tag{5}$$

and the expected PAoI is

$$\mathbb{E}[A] = \frac{1}{\lambda_u} + \frac{1/\mu_u}{1-\rho_q} + \frac{\rho_q/\mu_q + \rho_u/\mu_u}{(1-\rho_q)(1-\rho_q-\rho_u)}. \tag{6}$$

2) Threshold $1 < k < \infty$: Since the threshold k is now larger than 1, the query queue has a higher priority than the update queue only when the threshold $k > 1$ is reached. In other words, the query queue no longer has an absolute priority over the update queue. Hence, the analysis techniques used for the case of $k = 1$ is not applicable here. Instead, we will analyze the response time and the PAoI by use of the techniques developed in [22]. In our online technical report [7], we explain how their techniques can be applied to derive $\mathbb{E}[N_q]$ (i.e., the expected number of queries in the system), which will be needed for computing the expected response time (i.e., (7)) and the expected PAoI (i.e., (8)). We state the main results for the case of $1 < k < \infty$ in Proposition 3.

Proposition 3: Under the Query-k policy with $1 < k < \infty$, the expected response time is

$$\mathbb{E}[T_q] = \mathbb{E}[N_q]/\lambda_q, \tag{7}$$

and the expected PAoI is

$$\mathbb{E}[A] = \frac{1}{\lambda_u} + \frac{\mu_{\rm u}}{\lambda_u} \cdot \left(\frac{\lambda_{\rm q}/\mu_{\rm q}^2 + \lambda_{\rm u}/\mu_{\rm u}^2}{1-\rho} - \frac{\mathbb{E}[N_q]}{\mu_{\rm q}} \right), \tag{8}$$

where $\mathbb{E}[N_q]$ is the expected number of queries in the system.

3) Threshold $k = \infty$: In this case, since the threshold of the query queue is infinity, the server switches to serving the queries only when the update queue becomes empty. Then, it keeps serving the queries. Only when the query queue becomes empty, the server switches to serving the updates. Therefore, the system reduces to the classical two-queue model with exhaustive service at both queues (i.e., all jobs waiting in the current queue will be served before the server turns to the other queue) [23]. The work of [23] presents a method for deriving the distribution of waiting time for updates and queries. By using their method, we can obtain the expected system time for updates and queries, respectively, which can further be used to analyze the response time and the PAoI.

C. The Update-k Policy

Similar to the Query-k policy that priorities the queries, we propose another threshold-based policy, called the the Update-k policy, which prioritizes the updates. Similarly, we will discuss three cases: 1) $k = 1$, 2) $1 < k < \infty$, and 3) $k = \infty$.

1) Threshold k=1: In this case, the server always gives a higher priority to the update queue. Hence, the updates now belong to Class 1, and the queries belong to Class 2. We state the following proposition and omit the proof as it is similar to that of Proposition 2.

Proposition 4: Under the Update-1 policy, the expected response time is

$$\mathbb{E}[T_q] = \mathbb{E}[T(2)] = \frac{1/\mu_q}{1-\rho_u} + \frac{\rho_q/\mu_q + \rho_u/\mu_u}{(1-\rho_u)(1-\rho_q-\rho_u)}, \tag{9}$$

and the expected PAoI is

$$\mathbb{E}[A] = \mathbb{E}[X_u] + \mathbb{E}[T(1)] = \frac{1}{\lambda_u} + \frac{1}{\mu_u} + \frac{\rho_u/\mu_u}{1-\rho_u}. \tag{10}$$

2) Threshold $1 < k < \infty$: This case is similar to the case of the Query-k policy with $1 < k < \infty$. Following the same line of analysis as that in the proof of Proposition 3, we can compute $\mathbb{E}[N_u]$ using the techniques developed in [22] and analyze the expected response time and the PAoI. We state the main results in Proposition 5 and omit the detailed proof.

Proposition 5: Under the Update-k policy with $1 < k < \infty$, the expected response time is

$$\mathbb{E}[T_q] = \frac{\mu_q}{\lambda_q} \cdot \left(\frac{\lambda_{\rm q}/\mu_{\rm q}^2 + \lambda_{\rm u}/\mu_{\rm u}^2}{1-\rho} - \frac{\mathbb{E}[N_u]}{\mu_u} \right), \tag{11}$$

and the expected PAoI is

$$\mathbb{E}[A] = 1/\lambda_u + \mathbb{E}[N_u]/\lambda_u. \tag{12}$$

3) Threshold $k = \infty$: Same as the Query-k policy, the system reduces to the classical two-queue model with exhaustive service at both queues. The analysis will be exactly the same as that of the Query-k policy with $k = \infty$.

D. The Joint-(M, N) Policy

In the previous two subsections, we have been focused on threshold-based policies that prioritize either queries or updates. The analyses reveal the following insights: the priority is given to the queue with a threshold; the lower the threshold, the higher the degree of priority. Take the Query-k policy for example. When $k = 1$, the query queue always has a higher priority; when $k = \infty$, the query queue no longer has a higher priority, because the system reduces to the classical two-queue model with exhaustive service at both queues. Hence, one limitation of the single-threshold-based policies is that the priority is given to one queue only.

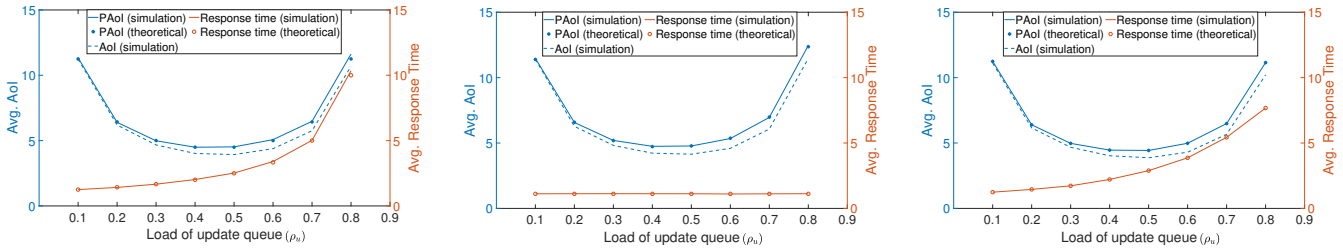
Next, we introduce the Joint-(M, N) policy, where we jointly set thresholds M and N for the updates and the queries, respectively. This policy generalizes the Query-k policy and the Update-k policy and allows flexibly prioritizing updates or queries through choosing different values of M and N.

Specifically, the Joint-(M, N) policy functions in the following manner: (i) the update queue has a threshold M, and the query queue has a threshold N; (ii) the server immediately switches to the queue whose queue length reaches its threshold and continues to serve this queue as long as the threshold of the other queue is not reached; (iii) if both thresholds are reached, the server will serve the queue with a new arrival.

The Query-k policy and the Update-k policy are two special cases of the Joint-(M, N) policy, where $(M = \infty, N = k)$ and $(M = k, N = \infty)$, respectively. When $1 < M < \infty$ and $1 < N < \infty$, the Joint-(M, N) policy becomes more flexible in prioritizing updates and queries. We leave the analyses of the general Joint-(M, N) policy as our future work. However, in Section V we provide simulation results to demonstrate its advantages compared to the one-threshold-based policies.

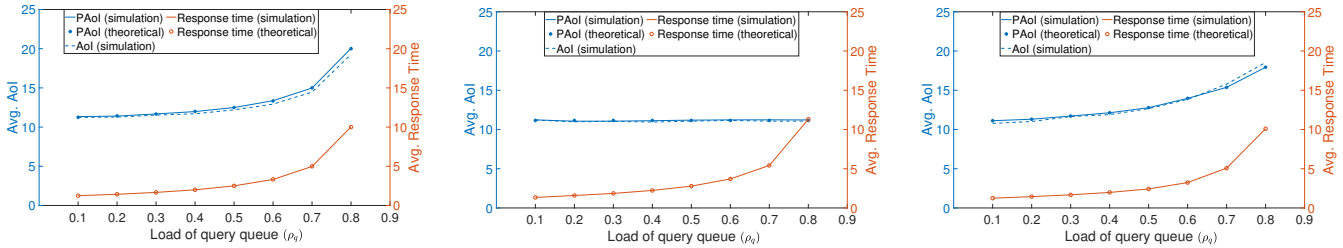
V. Numerical Results

In this section, we conduct simulations to evaluate the response time and the PAoI of the proposed policies. We first consider the FCFS policy and demonstrate its limitations.



(a) FCFS (b) Query-1 (c) Query-3

Fig. 2: Performance comparisons of different policies with varying update load ($\lambda_q = 0.1$ and $\mu_q = \mu_u = 1$)



(a) FCFS (b) Update-1 (c) Update-3

Fig. 3: Performance comparisons of different policies with varying query load ($\lambda_u = 0.1$ and $\mu_q = \mu_u = 1$)

Then, we show that the single-threshold-based policies (i.e., the Query-k policy and the Update-k policy) have the ability to effectively control the tradeoff between the response time and the PAoI. Finally, we demonstrate the flexibility of the Joint-(M, N) policy. We implement and simulate these policies in Java. In the simulation results, each data point is the average of 10 runs, and each run lasts 20,000 time units. We also include our analytical results computed using Wolfram Mathematica for the purpose of comparison.

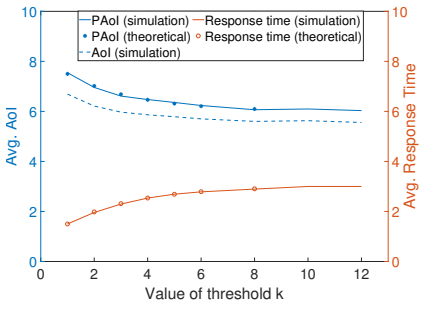
We first simulate the FCFS policy and assume $\lambda_q = 0.1$ and $\mu_q = \mu_u = 1$. The results are presented in Fig. 2(a). The results show that both the average PAoI and the average AoI decrease first and then increase as the update load increases. When the update load is low, the PAoI and the AoI are large due to large inter-arrival times of the updates; when the update load is high, the PAoI and the AoI are also large due to large queueing delays. On the other hand, the response time keeps increasing as a larger update load can only worsen the congestion condition for queries. Hence, when the update load is high, the response time and the PAoI can both be poor since the FCFS policy does not prioritize either queries or updates.

Next, we consider the Query-k policy and assume $\lambda_q = 0.1$ and $\mu_q = \mu_u = 1$. The results are presented in Figs. 2(b) and 2(c). We can observe from Fig. 2(b) that the average response time remains unchanged under the Query-1 policy since the queries are always given a higher priority than the updates, while the PAoI is only slightly larger than that under the FCFS policy (e.g., 12.46 vs. 11.62 when $\rho_u = 0.8$). Fig. 2(c) shows that under the Query-3 policy, the response time keeps increasing as the update load increases, but it is still better than that under the FCFS policy. Compared to the Query-1 policy, while the PAoI is a little smaller (e.g., 11.15 vs. 12.46 when $\rho_u = 0.8$), the response time becomes much worse. Therefore, one need to carefully choose the value of the threshold so as to effectively control the tradeoff between the response time and the PAoI. Note that the PAoI does not vary much under different policies due to a small query rate of 0.1.

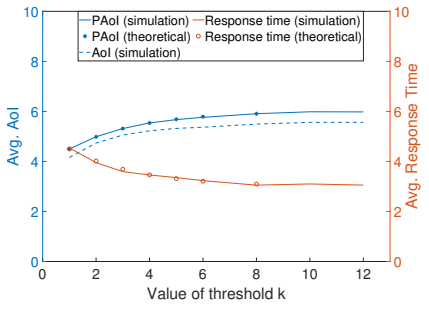
Similarly, we compare the FCFS policy with the Update-k policy with different values of k, by assuming $\lambda_u = 0.1$ and $\mu_q = \mu_u = 1$ and varying the query load. The results are presented in Fig. 3, where similar observations can be made.

Further, we investigate the impact of different values of the threshold under the threshold-based policies and present the results in Fig. 4. We assume $\lambda_u = \lambda_q = 1/3$ and $\mu_u = \mu_q = 1$. Fig. 4(a) shows that under the Query-k policy, as the threshold k increases, while the PAoI and the AoI decrease, the response time increases. This is because the degree of priority given to queries becomes lower as k increases. Such behavior saturates when k reaches a certain value (e.g., around $k = 8$ in Fig. 4(a)). Similar observations can be made in Fig. 4(b), which shows the results for the Update-k policy.

Finally, we also simulate the Joint-(M, N) policy. Assuming $\lambda_u = \lambda_q = 1/3$ and $\mu_u = \mu_q = 1$, we investigate the impact of different values of the thresholds M and N on the response time and the PAoI. The results are presented in Fig. 5. We observe that the larger (resp., smaller) the value of M (resp., N), the higher the PAoI and the lower the response time. Therefore, the Joint-(M, N) policy allows more flexibly prioritizing updates or queries through choosing different values of the two thresholds (i.e., M and N).



(a) Query-k



(b) Update-k

Fig. 4: Impact of the threshold on the single-threshold-based policies ($\lambda_u = \lambda_q = 1/3$ and $\mu_u = \mu_q = 1$)

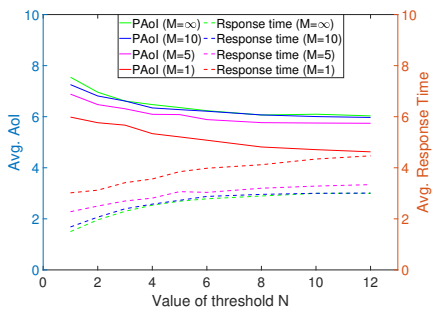


Fig. 5: Impact of different values of the thresholds M and N on the Joint-(M, N) policy ($\lambda_u = \lambda_q = 1/3$ and $\mu_u = \mu_q = 1$)

VI. Conclusion

In this paper, we proposed a simple single-server two-queue model that captures the coupled scheduling between updates and queries for data-driven real-time applications. Aiming to address the natural tradeoff between service performance and information freshness in such applications, we proposed threshold-based scheduling policies that prioritize updates or queries and analyzed the response time and the PAoI in a rigorous manner. The simulation results further demonstrated that by properly choosing the values of the thresholds, the proposed policies can achieve the desired balance between service performance and information freshness.

Although this paper provides useful insights towards the tradeoff between the response time and the PAoI, there remain some open questions, which will be investigated in our future work. For example, it would be interesting to rigorously analyze the average AoI under the threshold-based policies and to systematically study the Joint-(M, N) policy. In addition, we have implicitly assumed that there was a negligible overhead for the server to switch back and forth between the query queue and the update queue. It would be interesting to investigate and characterize the impact of switching cost.

References

[1] "Apache storm," http://storm.apache.org.

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

[3] A. Labrinidis and N. Roussopoulos, "Exploring the tradeoff between performance and data freshness in database-driven web servers," *The VLDB JournalThe International Journal on Very Large Data Bases*, vol. 13, no. 3, pp. 240–255, 2004.

[4] H. Qu, A. Labrinidis, and D. Mosse, "Unit: User-centric transaction management in web-database systems," in *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*. IEEE, 2006, pp. 33–33.

[5] H. Qu and A. Labrinidis, "Preference-aware query and update scheduling in web-databases," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. IEEE, 2007, pp. 356–365.

[6] M. Costa, M. Codreanu, and A. Ephremides, "Age of information with packet management," in *Information Theory (ISIT), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 1583–1587.

[7] Z. Liu and B. Ji, "Towards the Tradeoff Between Service Performance and Information Freshness," *arXiv e-prints*, p. arXiv:1901.00826, Jan 2019.

[8] M. Harchol-Balter, M. Crovella, and C. Murta, "To queue or not to queue?: When fcfs is better than ps in a distributed system," MIT, Tech. Rep., 1997.

[9] M. Harchol-Balter, M. Crovella, and S. Park, "The case for srpt scheduling in web servers," MIT, Tech. Rep., 1998.

[10] M. Harchol-Balter, B. Schroeder, N. Bansal, and M. Agrawal, "Size-based scheduling to improve web performance," *ACM Transactions on Computer Systems (TOCS)*, vol. 21, no. 2, pp. 207–233, 2003.

[11] T. Osogami, M. Harchol-Balter, and A. Scheller-Wolf, "Robustness and performance of threshold-based resource allocation policies," Working paper, Tech. Rep., 2005.

[12] C. Kam, S. Kompella, and A. Ephremides, "Age of information under random updates," in *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 66–70.

[13] R. D. Yates and S. Kaul, "Real-time status updating: Multiple sources," in *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*. IEEE, 2012, pp. 2666–2670.

[14] I. Kadota, A. Sinha, E. Uysal-Biyikoglu, R. Singh, and E. Modiano, "Scheduling policies for minimizing age of information in broadcast wireless networks," *arXiv preprint arXiv:1801.01803*, 2018.

[15] N. Lu, B. Ji, and B. Li, "Age-based scheduling: Improving data freshness for wireless real-time traffic," in *Proceedings of the Eighteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*. ACM, 2018, pp. 191–200.

[16] R. D. Yates, "Lazy is timely: Status updates by an energy harvesting source," in *Information Theory (ISIT), 2015 IEEE International Symposium on*. IEEE, 2015, pp. 3008–3012.

[17] 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 Transactions on Information Theory*, vol. 63, no. 11, pp. 7492–7508, 2017.

[18] Y. Sang, B. Li, and B. Ji, "The power of waiting for more than one response in minimizing the age-of-information," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–6.

[19] M. Patra, A. Sengupta, and C. S. R. Murthy, "On minimizing the system information age in vehicular ad-hoc networks via efficient scheduling and piggybacking," *Wireless Networks*, vol. 22, no. 5, 2016.

[20] X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. Bowers *et al.*, "Practical lessons from predicting clicks on ads at facebook," in *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*. ACM, 2014, pp. 1–9.

[21] M. Harchol-Balter, *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[22] O. J. Boxma, G. Koole, and I. Mitrani, "A two-queue polling model with a threshold service policy," in *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 1995. MASCOTS'95., Proceedings of the Third International Workshop on*. IEEE, 1995.

[23] L. Takács, "Two queues attended by a single server," *Operations Research*, vol. 16, no. 3, pp. 639–650, 1968.