Performance of Server Selection Algorithms for Content Replication Networks*

David Starobinski[1] and Tao Wu[2]

[1] ECE Dept., Boston University
staro@bu.edu
[2] Nokia Research Center Boston
tao.wu@nokia.com

Abstract. In this paper, we investigate the problem of optimal server selection in "content replication networks," such as peer-to-peer (P2P) and content delivery networks (CDNs). While a number of server selection policies have been proposed or implemented, understanding of the theoretical performance limits of server selection and the relative performance of existing policies remains limited. In this paper, we introduce a mathematical framework, based on the $M/G/1$ Processor Sharing queueing model, and derive closed-form expressions for the optimal server access probabilities and the optimal average delay. We also analyze the performance of two general server selection policies, referred to as EQ_DELAY and EQ_LOAD, that characterize a wide range of existing algorithms. We prove that the average delay achieved by these policies can theoretically be as much as N times larger than the optimal delay, where N is the total number of servers in the system. Furthermore, simulation results obtained using our $M/G/1$-PS workload model and the ProWGen Web workload generator show that the optimal policy can reduce the average delay of requests by as much as 30% as compared to EQ_LOAD and EQ_DELAY, in realistic scenarios. They also show that the optimal policy compares favorably to the other policies in terms of fairness and sensitivity to traffic parameters.

1 Introduction

Content replication has emerged as one of the most useful paradigms for the provision of scalable and reliable Internet services [1]. With content replication, the same data (e.g., Web pages, MP3 files, etc.) is stored at multiple geographically distant servers. Request by clients are then forwarded to one of these servers, usually the one that minimizes the delay perceived by the client. Because of its inherent scalability and fault-tolerance, content replication has become the cornerstone of most modern networking architectures, including content delivery networks (CDNs) and peer-to-peer (P2P) networks [2, 3].

One of the key issues arising with content replication is that of server selection. In most of the existing network architectures, a number of *server-selection nodes* (e.g.,

* This work was supported in part by NSF CAREER award ANI-0132802 and DOE Early Career Principal Investigator award DE-FG02-04ER25605

request redirection routers in CDNs [2] or supernodes in P2P networks [3]) are responsible for aggregating incoming client requests and forwarding them to one of the servers. The main question within this context is to determine the optimal server selection policy that minimizes the overall average delay of requests in the network. We note that this problem is fundamentally different from traditional load-balancing problems, which usually assume that all the servers are co-located [4].

Obviously, because server response time is an increasing function of the load, the best solution to the server-selection problem is usually not the one that directs all the requests to a single server (e.g., the fastest), which would slow down or even crash the server [5]. Thus, a number of server selection policies have been proposed in the literature or implemented in commercial products to better balance the load over the servers. Generally speaking, these policies fall within one of the following two categories: 1) Equal load (EQ_LOAD), where the access probabilities to the servers are set so that all servers have the same utilization; round-robin, or weighted-round-robin for heterogeneous servers [4], and certain adaptive algorithms such as Least Loaded [6] and WebSeAl [7], are examples of policies following this approach. 2) Equal delay (EQ_DELAY), where the access probabilities are set so that the average delay at all the selected servers is equal or at least on the same order. The intuition behind this approach is to avoid forwarding request to very slow servers, as may happen with EQ_LOAD. SPAND [8] and the application layer anycast architecture of [5] implement variations of this approach.

In the absence of any analytical benchmark, it is unclear whether the aforementioned policies perform optimally and, if not, how well they perform compared to the optimal policy. Our goal in this paper is to address these fundamental questions and explore the strength and limitations of existing server selection policies. To this end, we model the behavior of the servers using an $M/G/1$-PS (Processor Sharing) queuing framework [9]. Using this framework, we formulate our problem as a constrained non-linear optimization problem. We show that the problem at hand has enough special structure for allowing the derivation of *closed-form* expressions for the optimal server access probabilities and the optimal average delay. Using a similar analytical procedure, we also derive expressions for the access probabilities and average delay of the EQ_LOAD and EQ_DELAY policies.

Our major theoretical findings in this paper are as follows. First, our analysis reveals that both EQ_LOAD and EQ_DELAY perform sub-optimally. Second, assuming that the system consists of a total of N servers, we prove that the average delay achieved with these policies can be as much as N times larger than the optimal average delay. Finally, although EQ_DELAY may achieve a much lower average delay than EQ_LOAD at low load, we prove that these policies perform identically at high load.

We illustrate our analytical results with an extensive number of numerical examples and simulations. Specifically, numerical results obtained using our $M/G/1$-PS workload model show that, in realistic network configurations, the optimal policy can reduce the average delay by as much as 30% as compared to EQ_LOAD and EQ_DELAY. Moreover, simulation results obtained using the ProWGen Web workload generator [10] show similar performance gain for synthetic traces exhibiting temporal locality (i.e., temporal correlation) in the popularity of files. Other simulation results show that the

fairness properties of the optimal policy are comparable or even better than those of the two other policies.

The remainder of the paper is organized as follows. In Section 2, we introduce our model and notations. In Section 3, we derive the optimal server selection policy and obtain a closed-form expression for the optimal average delay. In Section 4, we analyze the EQ_DELAY and EQ_LOAD policies and compare their performance with that of the optimal policy. In Section 5, we present our simulation results and we conclude the paper in Section 6.

2 Model and Problem

2.1 Model and Notations

We consider a network consisting of M server-selection nodes and N servers. We assume that each server-selection node j generates requests following a Poisson process with rate λ_j. Further, we assume that the requests generated by different server-selection nodes are independent. Therefore, the aggregate request process is also Poisson with rate $\lambda = \sum_{i=1}^{M} \lambda_i$. Each server-selection node assigns request to server i with probability p_i, independently of other requests. Therefore, the arrival process to each server i, $i = 1, 2, \ldots, N$, is an independent Poisson process with rate $p_i\lambda$. We note that measurements of request arrivals to Web servers have been shown to match well a Poisson process, at least over small to moderate timescales [9]

We denote the service capacity of each server i by C_i. The file size distribution is arbitrary but identical at all the servers. We denote the mean file size by $\bar{x}$. The mean service rate of requests is denoted by $\mu_i = C_i/\bar{x}$. We assume that each server implements a *Processor Sharing* (PS) scheduling policy, where all the requests share the service capacity equally and continuously [11]. Processor Sharing is an idealization of the round-robin policy implemented by most existing Web servers, whereby each request is given an equal share of the service capacity [12]. Finally we assume that network delays are negligible compared to the delay that a request experienced at the server. This assumption is in line with recent measurement studies which have shown that, with over-provisioning, delays across backbones are now mostly dominated by speed-of-light delays with minimal queueing delay in the routers [13, 14].

Under the above assumptions, each server behaves as an $M/G/1$-PS queue. The average delay of a request forwarded to server i is therefore given by the following expression [11]

$$\overline{T}_i(p_i) = \frac{1}{\mu_i - p_i\lambda}. \tag{1}$$

We note that the average delay depends on the file size distribution only through its mean.

Let $\mathbf{p} = (p_1, p_2, \ldots, p_N)'$ denote the service access probability vector. Then, for a given vector $\mathbf{p}$, the average delay of a request in the system is

$$\overline{T}(\mathbf{p}) = \sum_{i=1}^{N} p_i \overline{T}_i(p_i) = \sum_{i=1}^{N} \frac{p_i}{\mu_i - p_i\lambda}. \tag{2}$$

Note that any feasible vector $\mathbf{p}$ must satisfy several conditions. First, to be a legitimate probability vector, the coordinates of $\mathbf{p}$ must be non-negative and sum to one, that is $p_i \geq 0$ for all i and $\sum_{i=1}^{N} p_i = 1$. In addition, the coordinates of $\mathbf{p}$ must satisfy the *individual stability conditions*, i.e., $p_i < \mu_i/\lambda$, to guarantee that the arrival rate of requests is smaller than the service rate at each server i. We denote by $\mathcal{P}$ the set of vectors $\mathbf{p}$ that satisfy all the above constraints.

The set $\mathcal{P}$ is non-empty if and only if the *aggregate stability condition* $\lambda < \sum_{i=1}^{N} \mu_i$ is satisfied. To see this, define $\mu = \sum_{i=1}^{N} \mu_i$. If the aggregate stability condition holds, then the probability vector $\mathbf{p} = (\mu_1/\mu, \mu_2/\mu, \ldots, \mu_N/\mu)'$ is always feasible. On the other hand, if the aggregate stability condition does not hold, then the individual stability can never all be simultaneously satisfied. Therefore, we will assume from now and on that the aggregate stability condition always holds.

3 Derivation of the Optimal Policy

In this section, we determine the optimal vector $\mathbf{p}^*$ that minimizes the average delay expression in Eq. (2). Specifically, our problem can be formally stated as follows:

Problem 1 (OPT). Find the optimal server access probability vector

$$\mathbf{p}^* = \arg\min_{\mathbf{p}\in\mathcal{P}} \overline{T}(\mathbf{p}).$$

Problem 1 has already been the subject of studies in the literature under the general context of load sharing in queueing networks [15, 16] as well as flow assignment [17]. However, our contribution here is to provide *closed-form* expressions for the optimal server access probabilities for the specific case of $M/G/1$-PS servers. These expressions will be used to prove our main result (Theorem 4 in Section 4).

As a first step to obtain the optimal server access probabilities, we note that there exists a unique solution to Problem 1, since $\overline{T}(\mathbf{p})$ is strictly convex over $\mathcal{P}$. Next, in order to solve the constrained optimization problem, we make use of Lagrange multiplier techniques [18]. We start by defining the Lagrangian function

$$\begin{aligned} L(\mathbf{p}, l, \mathbf{m}) &= \overline{T}(\mathbf{p}) + l(\sum_{i=1}^{N} p_i - 1) - \sum_{i=1}^{N} m_i p_i \\ &= \sum_{i=1}^{N} \frac{p_i}{\mu_i - p_i\lambda} + l(\sum_{i=1}^{N} p_i - 1) - \sum_{i=1}^{N} m_i p_i, \end{aligned} \tag{3}$$

where l and $\mathbf{m} = (m_1, m_2, \cdots, m_N)$ are the so-called Lagrange multipliers. The Lagrange multiplier l enforces the equality constraint $\sum_{i=1}^{N} p_i = 1$, while the multipliers m_i enforce the inequality constraints $p_i \geq 0$. In the sequel, we denote by $I(\mathbf{p})$ the set of *inactive* servers to which requests are never forwarded, that is, $I(\mathbf{p}) = \{i \mid p_i = 0\}$.

Since $\overline{T}(\mathbf{p})$ is strictly convex and the set of constraints is convex as well, the Karush-Kuhn-Tucker (KKT) conditions stated below are both necessary and sufficient for the existence of a global minimum $\mathbf{p}^*$, assuming that the individual stability conditions are satisfied (see Proposition 3.3.4 in [18]).

Theorem 1 (KKT) *Let $\mathbf{p}^*$ be the optimal solution of Problem 1. Then, there exist unique Lagrange multipliers l^* and $\mathbf{m}^* = (m_1^*, m_2^*, \cdots, m_N^*)$, such that*

1. $\nabla_{\mathbf{p}} L(\mathbf{p}^*, l^*, \mathbf{m}^*) = 0$,
2. $m_i^* = 0, \quad \forall i \notin I(\mathbf{p}^*)$,
3. $m_i^* \geq 0, \quad \forall i \in I(\mathbf{p}^*)$.

Without loss of generality, we can assign the server indices so that $\mu_1 \geq \mu_2 \geq \cdots \geq \mu_N$. We now observe that in the optimal solution, a faster server should always serve a larger fraction of requests than a slower server. Therefore, the access probabilities must satisfy the following ordering $p_1^* \geq p_2^* \geq \cdots \geq p_N^*$. As a result, the set of inactive servers $I(\mathbf{p}^*)$ must be one of the following: $\emptyset, \{N\}, \{N-1, N\}, \ldots, \{2, 3, \ldots, N\}$.

Let us denote the slowest active server in the optimal solution as N^*, that is, $I(\mathbf{p}^*) = \{N^*+1, N^*+2, \ldots, N\}$. The following theorem establishes the value of N^* and provides a closed-form expression for the optimal solution $\mathbf{p}^*$. This theorem is proven by showing that the optimal solution satisfies all the KKT conditions stated in Theorem 1 as well as the individual stability conditions. Due to space limitation, the proof is omitted here but can be found in our technical report [19].

Theorem 2 *The optimal solution $\mathbf{p}^*$ to Problem 1 can be obtained as follows. Define*

$$\alpha_i = \frac{\mu_i}{\lambda} - \frac{\left(\sum_{j=1}^{i} \mu_j - \lambda\right)\sqrt{\mu_i}}{\lambda \sum_{j=1}^{i} \sqrt{\mu_j}} \quad 0 \leq i \leq N. \tag{4}$$

Then,

1. *N^* is the maximum index i for which $\alpha_i > 0$.*
2. $I(\mathbf{p}^*) = \{N^*+1, N^*+2, \ldots, N\}$,
3. $p_i^* = \frac{\mu_i}{\lambda} - \frac{\left[\sum_{j=1}^{N^*} \mu_j - \lambda\right]\sqrt{\mu_i}}{\lambda \sum_{j=1}^{N^*} \sqrt{\mu_j}} \quad \forall i \notin I(\mathbf{p}^*)$,
4. $p_i^* = 0 \quad \forall i \in I(\mathbf{p}^*)$.

We have developed an efficient algorithm based on Theorem 2 to determine N^* and $\mathbf{p}^*$. This algorithm requires the computation of only N expressions. It starts by assuming $I(\mathbf{p}^*) = \emptyset$, that is, server N is the slowest active server. If $\alpha_N > 0$, then the assumption is valid and the coordinates of $\mathbf{p}^*$ are set using the expressions given in Theorem 2. However, if $\alpha_N \leq 0$, then the assumption is not valid and it must be the case that $p_N^* = 0$. The algorithm proceeds by repeating the same procedure with the sets $I(\mathbf{p}^*) = \{N-i+1, N-i+2, \ldots, N\}$, $i = 1, 2, \ldots$, until an index i is found such that $\alpha_i > 0$, at which point N^* is determined. The access probabilities p_i^*, $i \notin I(\mathbf{p}^*)$, are then computed using Theorem 2. The pseudo-code for this algorithm can be found in [19].

We can now determine the average delay achieved by the optimal policy, by substituting the derived expression of $\mathbf{p}^*$ into Eq. (2):

$$\overline{T}(\mathbf{p}^*) = \sum_{i=1}^{N^*} \frac{p_i^*}{\mu_i - p_i^* \lambda} = \frac{\left(\sum_{i=1}^{N^*} \sqrt{\mu_i}\right)^2}{\lambda\left(\sum_{i=1}^{N^*} \mu_i - \lambda\right)} - \frac{N^*}{\lambda}. \tag{5}$$

4 Performance Analysis of the EQ_DELAY and EQ_LOAD Policies

In this section, we evaluate the performance of the EQ_DELAY and EQ_LOAD policies. We derive the server access probabilities of each policy. Using these results, we show that at high load, when all the servers are active, EQ_DELAY and EQ_LOAD achieve the same average delay. We also prove our main result that the average delay of EQ_DELAY and EQ_LOAD can be as much as N times larger than the optimal average delay, where N denotes the total number of servers in the system.

4.1 General Solution for EQ_DELAY

The goal of the EQ_DELAY policy is to set the probability access vector such that the average delay at all the active servers is the same and minimal. To formalize the problem, define the set

$$\mathcal{S} = \{\mathbf{p} \mid \overline{T}_i(p_i) = \overline{T}_j(p_j) \quad \forall i, j \notin I(\mathbf{p})\}. \tag{6}$$

We can then formulate the optimization problem for the EQ_DELAY policy as follows:

Problem 2 (EQ_DELAY). Find the server access probability vector

$$\hat{\mathbf{p}} = \arg \min_{\mathbf{p} \in (\mathcal{P} \cap \mathcal{S})} \left(\overline{T}(\mathbf{p})\right).$$

As with Problem 1, it is fairly easy to verify that the set of inactive servers $I(\hat{\mathbf{p}})$ must be one of the following: $\emptyset, \{N\}, \{N-1, N\}, \ldots, \{2, 3, \ldots, N\}$. Denote the slowest active server in the minimal solution of Problem 2 as $\hat{N}$. The following theorem, which is the analog of Theorem 2, provides expressions for $\hat{N}$ and $\hat{\mathbf{p}}$:

Theorem 3 *The solution $\hat{\mathbf{p}}$ to Problem 2 can be obtained as follows. Define*

$$\beta_i = \frac{\lambda + i\mu_i - \sum_{j=1}^{i} \mu_j}{i\lambda}, \quad 0 \le i \le N. \tag{7}$$

Then,

1. *$\hat{N}$ is the maximum index i for which $\beta_i > 0$.*
2. *$I(\hat{\mathbf{p}}) = \{\hat{N}+1, \hat{N}+2, \ldots, N\}$,*
3. *$\hat{p}_i = \frac{\lambda + \hat{N}\mu_i - \sum_{j=1}^{\hat{N}} \mu_j}{\hat{N}\lambda} \quad \forall i \notin I(\hat{\mathbf{p}})$,*
4. *$\hat{p}_i = 0 \quad \forall i \in I(\hat{\mathbf{p}})$.*

We note that the algorithm for the OPT policy in Section 3 can easily be modified to compute $\hat{N}$ and $\hat{\mathbf{p}}$.

The average delay of requests for the EQ_DELAY policy is obtained by inserting the derived expression of $\hat{\mathbf{p}}$ into Eq. (2)

$$\overline{T}(\hat{\mathbf{p}}) = \sum_{i=1}^{\hat{N}} \frac{\hat{p}_i}{\mu_i - \hat{p}_i\lambda} = \frac{\hat{N}}{\sum_{i=1}^{\hat{N}} \mu_i - \lambda}. \tag{8}$$

4.2 General Solution for EQ_LOAD

The EQ_LOAD policy aims at achieving the same utilization at each of the servers in the system. The problem can be formally stated as follows:

Problem 3 (EQ_LOAD). Find a server access probability vector $\tilde{\mathbf{p}} \in \mathcal{P}$ such that

$$\frac{\tilde{p}_i}{\mu_i} = \frac{\tilde{p}_j}{\mu_j} \quad \forall i, j = 1, 2, \ldots, N.$$

The solution to this problem is straightforward and is given by

$$\tilde{p}_i = \frac{\mu_i}{\sum_{j=1}^{N} \mu_j} \quad \forall i = 1, 2, \ldots, N. \tag{9}$$

The average delay for EQ_LOAD satisfies the following expression

$$\overline{T}(\tilde{\mathbf{p}}) = \frac{N}{\sum_{i=1}^{N} \mu_i - \lambda}. \tag{10}$$

4.3 Comparison

In this section, we compare the performance of the OPT, EQ_DELAY and EQ_LOAD policies. Our first observation is that EQ_DELAY and EQ_LOAD have the same average delay when $\hat{N} = N$, because Eq. (8) becomes identical to Eq. (10) in this case. Therefore, EQ_DELAY and EQ_LOAD always perform comparably at high load, since EQ_DELAY must use all the servers to ensure stability in this regime. This result is summarized by the following lemma.

Lemma 1. *If $\hat{N} = N$, then $\overline{T}(\hat{\mathbf{p}}) = \overline{T}(\tilde{\mathbf{p}})$.*

We next show, that for a given number of servers N, there exist arrival rate and service rate parameters such that the ratio of the average delay of EQ_DELAY to that of OPT approaches N. This situation occurs at high load, when $\hat{N} = N$. Therefore, from Lemma 1 the same result applies for the ratio of the average delay of EQ_LOAD to that of OPT.

Theorem 4 *For any given N, there exist parameters λ and μ_i, $i = 1, 2, \ldots, N$, such that $\overline{T}(\tilde{\mathbf{p}})/\overline{T}(\mathbf{p}^*) = \overline{T}(\hat{\mathbf{p}})/\overline{T}(\mathbf{p}^*) \to N$.*

Proof. For any N, we derive an example with a particular configuration of arrival rate and service rates parameters such that the average delay ratio tends to N. Assume the sum of service rates of all the servers is 1. Furthermore, assume μ_1 is close to 1 and $\mu_1 \gg \mu_2 = \mu_3 = \cdots = \mu_N$. Thus, $\mu_i = \frac{1-\mu_1}{N-1}$ for $1 < i \leq N$. In addition, assume $\sum_{i=1}^{N-1} \mu_i < \lambda < 1$. Then $N^* = \hat{N} = N$ and $\overline{T}(\tilde{\mathbf{p}}) = \overline{T}(\hat{\mathbf{p}})$. The ratio of Eq. (8) to Eq. (5) yields

$$\begin{aligned} \frac{\overline{T}(\hat{\mathbf{p}})}{\overline{T}(\mathbf{p}^*)} &= \frac{N\lambda}{\left(\sum_{i=1}^{N^*} \sqrt{\mu_i}\right)^2 - N\left(\sum_{i=1}^{N} \mu_i - \lambda\right)} \geq \frac{N\lambda}{\left(\sum_{i=1}^{N} \sqrt{\mu_i}\right)^2} \\ &= \frac{N\lambda}{\left(\sqrt{\mu_1} + \sqrt{(N-1)(1-\mu_1)}\right)^2}. \end{aligned} \tag{11}$$

Note that for any given N, the expression $(N-1)(1-\mu_1)$ appearing in the denominator of Eq. (11) becomes arbitrarily small as $\mu_1 \to 1$. Since $\mu_1 < \lambda < 1$, we also have that $\lambda \to 1$ as $\mu_1 \to 1$. Therefore, Eq. (11) becomes arbitrarily close to N as $\mu_1 \to 1$ and the theorem is proven. □

5 Simulations

In this section, we perform extensive simulations to validate and compare the OPT policy versus EQ_DELAY and EQ_LOAD. We first consider the case where the workload conforms to our $M/G/1$-PS model. We assume that the file size (and hence the service time) follows a heavy-tailed Pareto distribution with cumulative distribution function

$$F(x) = 1 - \frac{1}{(1+ax)^b} \quad x \geq 0,$$

where b is the *Pareto tail index*. This choice is justified by the large number of experimental studies which have shown that Web file size distributions follow a Pareto distribution [20, 21]. Using this workload model, we compare the average delay and standard deviation of the delay (which relates to fairness) of the three policies.

Next, we evaluate the performance of the three policies using synthetic traces generated by a Web workload generator, called ProWGen [10]. These traces exhibit temporal locality in file popularity and, therefore, differ from the $M/G/1$-PS model. Our simulations shows that OPT outperforms substantially the two other policies in this case as well.

5.1 Average Delay

We first compare the average delay of requests with OPT, EQ_DELAY and EQ_LOAD as a function of the aggregate load ρ, in the system which is defined as $\rho = \lambda / \sum_{j=1}^{N} \mu_j$. We fix the sum of the server rates to be 100 KB/s. We assume that the system consists of $N = 5$ servers, with one of them eight times faster than the others. Thus, the fastest server has service rate 66.67 KB/s, while the others have rate 8.33 KB/s. For the file size distribution, we assume a Pareto distribution with tail index $b = 2.2$. The mean file size is 11 KB. Each simulation point represents the average over 50 runs. A simulation run consists of 0.5×10^6 requests.

Figure 1 depicts the analytical results and simulation results for the average delay using the OPT, EQ_DELAY, and EQ_LOAD policies. The analytical expressions are obtained respectively from Eqs. (5), (8), and (10). As expected the simulation results match the analytical expressions. The figure also displays 95% confidence intervals which turn out to be very small (for this reason, we do not show them in other figures).

From Fig. 1, we observe that both OPT and EQ_DELAY perform much better than EQ_LOAD at low load. The reason is that, in this regime, OPT and EQ_DELAY forward requests only to the fastest server, while EQ_LOAD always sends $1/3$ of the requests to the slower servers. At high load, EQ_LOAD and EQ_DELAY achieve the same average delay, as predicted by Lemma 1, and OPT performs significantly better than these two policies. For instance, at load $\rho = 0.9$, the average delay of EQ_DELAY and EQ_LOAD is approximately 30% higher than the optimal average delay.

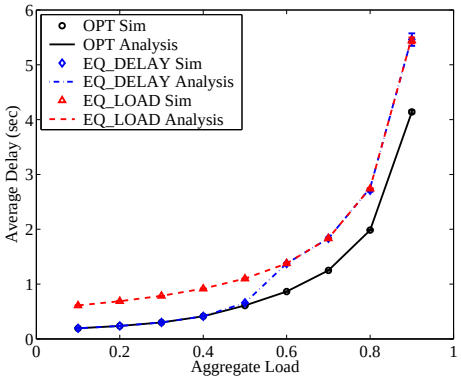

Fig. 1. Average delay: simulation and analysis

5.2 Standard Deviation of Delay and Fairness

An apparent advantage of EQ_DELAY is to serve each request with the same average delay (at the cost of higher overall average delay). This might lead to the belief that it exhibits better fairness properties than the other server selection policies. However, we show that this is actually not the case, at least with respect to OPT.

In the following set of simulations, we evaluate the standard deviation of the delay obtained with each policy. This metric has been recognized as one of the best ways to measure fairness in queues [22]. We consider a system consisting of $N = 10$ servers with service rate vector (188.2 65.18 40.71 25.13 20.72 19.53 15.87 14.63 8.67 6.41) KB/s. The Pareto tail index is $b = 2.2$ and the mean file size is 40 KB.

Figure 2 depicts the results. We observe that OPT and EQ_DELAY perform very similarly, while the standard deviation of EQ_LOAD is noticeably higher at all utilization values. A few comments are in order here. First, even though the average delay at each server is the same for EQ_DELAY, the actual delay of a request is still a random variable. In particular, this delay depends on the number of other requests concurrently being served, which obviously varies over time. Therefore, the standard deviation of EQ_DELAY is non-zero and turns out to be on the same order as that of OPT. Second, the standard deviation of EQ_LOAD is higher than that of EQ_DELAY, even at high load. This result is somewhat surprising since the average delay of these two policies is identical in this regime. This discrepancy is resolved by noting that the average delay of requests at different servers is not the same for EQ_LOAD. Therefore, EQ_LOAD and EQ_DELAY are not statistically identical even at high load. Finally, we observe that the standard deviation is quite high with all the policies. This is due to the extremely high variability of the file sizes generated by the Pareto distribution.

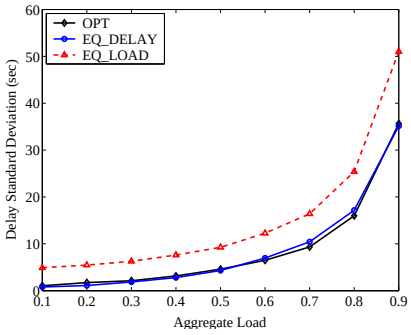

Fig. 2. Standard deviation of delay

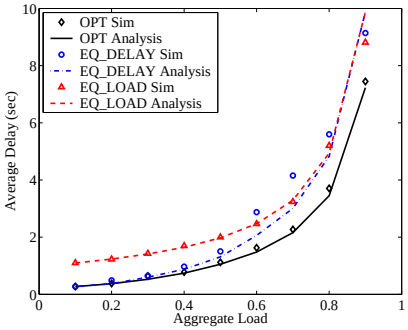

Fig. 3. Average delay with ProWGen traces

5.3 ProWGen Simulations

Our analytic $M/G/1$ Processor Sharing model and solutions are based on the assumption that the service time of different requests is independent. On the other hand, some experimental studies have shown that Web requests exhibits temporal locality, see e.g. [23]. In this section, we examine how OPT, EQ_DELAY and EQ_LOAD perform when request streams have short-term temporal correlation.

We use the Web workload generator ProWGen [10] to produce synthetic Web workload that exhibits temporal locality in file popularity. ProWGen, originally developed for Web cache performance evaluation, models the file popularity using Zipf distribution, and file size distribution using a combination of lognormal body and a Pareto tail. It can also model positive or negative correlation between file size and popularity if needed. Furthermore, it can use an LRU stack to model request temporal locality.

In our simulations, we use default values for most ProWGen parameters, with the Zipf slope of 0.75, the Pareto Tail index of 1.2, the lognormal mean of 7KB and the standard deviation of 11KB, and the tail cutoff size of 10KB. We also have a "dynamic" stack with a depth of 1000 requests to introduce temporal locality. Finally, the correlation between file size and popularity is set to zero, which is consistent with literature findings [24].

We find that a peculiarity of ProWGen is that popular files tend to be generated at the end of the trace, and "one-timers" (files accessed only once) are more likely to be generated earlier in the trace. To mitigate this artifact while still maintaining most of the short-term temporal locality, we divide the trace generated by ProWGen into segments each consisting of 4,000 requests and reshuffle the segments to obtain the trace used in our simulations. We scale the mean file size to 40 KB, and use the same service rate array as in Section 5.2. Each run contains about 550,000 requests, and the results are averaged over 50 runs.

The simulation results are shown in Figure 3 together with analytical values based on our $M/G/1$-PS model.

We make the following observations from the results obtained in these experiments. First, OPT still achieves the minimal average delay, which is substantially smaller than that of EQ_LOAD or EQ_DELAY at high load. Second, simulation results match the

analysis fairly well for OPT and EQ_LOAD, but poorly for EQ_DELAY. This is probably because of the temporal locality of ProWGen-generated traces. Even after the reshuffling, the moving average file size can significantly deviate from the long-term average. Since EQ_DELAY is very sensitive to load estimation errors [19], its performance suffered the most in this dynamic environment.

6 Conclusions

In this work, we have investigated the problem of optimal server selection in content replication networks. This problem has gained significant importance in recent years with the deployment of large-scale content delivery and peer-to-peer network architectures over the Internet. For this purpose, we have introduced a mathematical framework, based on the $M/G/1$ Processor Sharing queueing model, which allowed us to provide quantitative, yet non-trivial, insight into the performance server selection policies. We have also provided justification to our modeling assumptions based on measurement studies reported in the literature.

Based on our modeling assumptions, we have derived closed-form expressions for the optimal server access probabilities and the optimal average delay. We have also proposed a simple algorithm of linear computational complexity $O(N)$ to compute these probabilities, where N denotes the total number of servers in the system.

We have also evaluated the performance of two widely deployed server selection policies, generically referred to as EQ_DELAY and EQ_LOAD, that are representative of a large class of existing algorithms. In particular, we have analytically proved that the average delay of EQ_DELAY or EQ_LOAD can be as much as N times larger than the optimal delay. Thus, an important theoretical contribution of this work has been to show that the performance difference between EQ_DELAY (or EQ_LOAD) and OPT is unbounded as N grows. This radical difference in performance can be observed at high load and when the service capacity across servers is highly heterogeneous. Another interesting finding from our analysis is that EQ_DELAY and EQ_LOAD have identical average delay at high load, although their delay variance is different.

We have also conducted extensive simulations that demonstrated the significant superiority of OPT over EQ_LOAD and EQ_DELAY, especially at high load. In particular, we have shown that OPT has the potential of reducing the average delay in this regime by as much as 30%. We have also evaluated the degree of fairness of the three policies, that was quantified by computing the standard deviation of the delay, and shown that OPT performs comparably to EQ_DELAY and much better than EQ_LOAD.

The actual implementation of the OPT policy into real network settings is an important area of research left for future work. However, our initial results in this paper are extremely promising with this regard. In particular, we have shown that the performance of OPT is not too sensitive to the workload model, as illustrated by our simulations results obtained with the ProWGen workload generator.

References

1. Obraczka, K., Danzig, P., DeLucia, D.: Massively replicating services in autonomously managed, wide-area internetworks (1993) University of Southern California - Technical Report

No. 93-541.
2. Vakali, A., Pallis, G.: Content delivery networks: Status and trends. IEEE Internet Computing **7** (2003) 68–74
3. Yang, B., Molina, H.G.: Designing a super-peer network. In: Proceedings of the 19th International Conference on Data Engineering (ICDE), Bangalore, India (2003)
4. Cardellini, V., Casalicchio, E., Colajanni, M., Yu, P.: The state of the art in locally distributed web-server systems. ACM Computing Surveys (CSUR) **34** (2002) 263–311
5. Zegura, E., Ammar, M., Fei, Z., Bhattacharjee, S.: Application-layer anycasting: a server selection architecture and use in a replicated web service. IEEE/ACM Transactions on Networking **8** (2000)
6. Cisco: The global server load balancing primer. (http://www.cisco.com/en/US/products/hw/contnetw/ps4162/products_white_paper09186a00801b7725.shtml)
7. Korilis, Y., Lazar, A., Orda, A.: Architecting noncooperative networks. IEEE Journal of Selected Areas in Communications **13** (1995) 1241–1251
8. Stemm, M., Katz, R., Seshan, S.: A network measurement architecture for adaptive applications. In: Proceedings of IEEE INFOCOM, Tel-Aviv,Israel (2000)
9. Villela, D., Pradhan, P., Rubenstein, D.: Provisioning servers in the application tier for e-commerce systems. In: Proceedings of IWQoS '04. (2004)
10. Busari, M., Williamson, C.: ProWGen: a synthetic workload generation tool for simulation evaluation of Web proxy caches. Computer Networks **38** (2002)
11. Kleinrock, L.: Queueing systems. Volume 2. Wiley (1976)
12. Apache: (http://www.apache.org/)
13. Fraleigh, C., Moon, S., Lyles, B., Cotton, C., Khan, M., Moll, D., Rockell, R., Seely, T., Diot, C.: Packet-level traffic measurements from the Sprint IP backbone. IEEE Network Magazine **17** (2003) 6–16
14. Ranjan, S., Karrer, R., Knightly, E.: Wide area redirection of dynamic content by Internet data centers. In: Proceedings of IEEE INFOCOM, Hong Kong, China (2004)
15. Tantawi, A., Towsley, D.: Optimal static load balancing in distributed computer systems. Journal of the ACM **32** (1985) 445–465
16. Kim, C., Kameda, H.: An algorithm for optimal static load balancing in distributed computer systems. IEEE Transactions on Computers **41** (1992) 381–384
17. Gerla, M., Kleinrock, L.: On the topological design of distributed computer networks. IEEE Transactions on Communications **25** (1977) 48–60
18. Bertsekas, D.: Nonlinear Programming. second edn. Athena Scientific (1999)
19. Starobinski, D., Wu, T.: Server selection for scalable internet services: Algorithms and analysis. http://www.bu.edu/systems/research/publications/2004/2004-IR-0020.pdf (2004) Boston University - Technical Report No. 2004-IR-0020.
20. Crovella, M., Bestavros, A.: Self-similarity in world wide web traffic: Evidence and possible causes. IEEE/ACM Trans. on Networking **5** (1997) 835–846
21. Starobinski, D., Sidi, M.: Modeling and analysis of power-tail distributions via classical teletraffic methods. Queueing Systems (QUESTA) **36** (2000) 243–267
22. Avi-Itzhak, B., Levy, H.: On measuring fairness in queues. Advances of Applied Probability (2004)
23. Mahanti, A., Eager, D., Williamson, C.: Temporal locality and its impact on Web proxy cache performance. Performance Evaluation **42** (2000) 187–203
24. Breslau, L., Cao, P., Fan, L., Phillips, G., Shenker, S.: Web caching and zipf-life distributions: evidence and implications. In: Proceedings of IEEE INFOCOM '99, New York, NY (1999) 126–134