Stochastic Fluid Model for P2P Caching Evaluation

Florence Clévenot-Perronnin, Philippe Nain
INRIA
B.P. 93
06902 Sophia Antipolis
France
{clevenot,nain}@sophia.inria.fr

Abstract

In this paper we propose a stochastic fluid model to analyze the performance of Squirrel, a P2P cooperative Web cache. This work provides a scalable and insightful extension of our previous analysis of Squirrel.

This new model provides a closed-form expression for the hit probability when documents are equally popular. Realistic object popularity is also addressed through a clustering approximation. The accuracy of this model is validated by a comparison with discrete-event simulations.

Our model allows us to study the impact of various parameters on the performance of the Squirrel system. In particular, we emphasize the importance of taking object popularity into account. We also investigate the impact of clients announcing their departure on the resulting hit probability.

1. Introduction

P2P systems capitalize on individual user resources to build up self-organizing, scalable file-sharing systems. The Squirrel system [7] was recently introduced by Iyer, Rowstron and Druschel as a cooperative P2P web cache. Squirrel leverages the individual storage capabilities of the members of an institutional network to build a shared, decentralized web cache. In Squirrel each document (or object) is mapped to a unique node in the network, called the *home node*, identified by a node-Id. The home node of an object is responsible for delivering cached copies of this object to any user in the Squirrel network.

Because of its peer-to-peer structure, Squirrel is expected to be scalable and cost-effective. While this promising system is still in the test phase it is important for dimensioning and optimization purposes to study its performance and scalability. In [2] we developed a quantitative analysis of Squirrel in which we derived the hit probability (i.e. the probability that a requested document is found in the Squirrel network). The analysis was based on the observation that Squirrel basically evolves on two different time-scales: a slow time-scale corresponding to the process at which nodes join and leave Squirrel, and a fast time-scale corresponding to the frequency at which documents are requested. In [2] the node dynamics were modeled by a Engset process, a N-state Markov process, with N the number of nodes in the Squirrel network. As to the request process, we assumed that each active Squirrel node generated requests at a constant rate. We then showed that the total number of available documents in the Squirrel network was accurately modeled by a piecewise deterministic fluid process.

The aim of the present work is to extend the analysis in [2] in two main directions. First, we replace the Engset model by an infinite-state Markov process (the M/M/∞ queuing model – see Section 2.3), which yields a dramatic decrease in the complexity of computing the hit probability. Indeed, the Engset model solution involves binomial coefficients and exponentials in the size of the network. This restricted the performance analysis to the order of 10,000 nodes. Our new M/M/∞ model allows us to easily handle real size networks (e.g., 100,000 nodes for a large corporate network, or even larger). Second, we relax the assumption made in [2] that all documents are equally popular, and provide an efficient method for computing the hit probability in realistic situations (i.e. with Zipf-like document popularity distribution). Numerical comparisons with discrete-event simulations validate these extensions.

In Section 2 we give an overview of Squirrel and related work, and we describe our fluid model. We then show in Section 3 how to compute the hit probability

under the assumption that all objects are equally popular. The latter assumption is relaxed in Section 4, where we incorporate a Zipf-like object popularity distribution. These models are used in Section 5 to investigate the impact of unequal document popularity and of announced/unannounced departures (see Section 5.3) on Squirrel performance. Section 6 is devoted to the experimental validation of our approach, and Section 7 concludes the paper.

2 Overview and model

2.1 Overview of Squirrel

Squirrel [7] is a peer-to-peer cooperative Web cache that relies on Pastry [9] as a location and routing protocol. When a client requests an object it first sends a request to the Squirrel proxy running on the client's machine. If the object is uncacheable then the proxy forwards the request directly to the origin Web server. Otherwise it checks the local cache, like every Web browser would do, in order to exploit locality and reuse. If a fresh copy of the object is not found in this cache, then Squirrel tries to locate one on some other node. To do so, it uses the distributed hash-table and the routing functionalities provided by Pastry. First, the URL of the object is hashed to give a 128-bit object identity (a number called *object-Id*) from a circular list; then the routing procedure of Pastry forwards the request to the node with the identity (called *node-Id*; this number is assigned randomly by Pastry to a participating node) the closest to *object-Id*. This node then becomes the *home node* for this object. Squirrel then proposes two schemes from this point on: *home-store* and *directory* schemes.

In the home-store scheme, objects are stored both at client caches and at its home node. The client cache may either have no copy of the requested object or a stale copy. In the former case the client issues a GET request to its home-node, and it issues a *conditional* GET (cGET) request in the latter case. If the home-node has a fresh copy of an object then it forwards it to the client or it sends the client a not-modified message depending on which action is appropriate. If the home-node has no copy of the object or has a stale copy in its cache, then it issues a GET or a cGET request, respectively, to the origin server. The origin server then either forwards a cacheable copy of the object or sends a not-modified message to the home-node. Then, the home-node takes the appropriate action with respect to the client (i.e. send a not-modified message or a copy of the object).

In the directory scheme the home-node for an object maintains a small directory of pointers to nodes that have recently accessed the object. A request for this object is sent randomly to one of these nodes. We will not go deeper into the description of this scheme since from now on we will only focus on the home-store scheme. We do so mainly because the latter scheme has been shown to be overall more attractive than the directory scheme [7]. In addition, the home-store scheme is more amenable to a fluid analysis than the directory scheme.

In a Squirrel network (a corporate network, a university network, etc.), like in any peer-to-peer system, clients arrive and depart the system at random times. There are two kinds of departures: abrupt and announced departures. Each departure has a different impact on the performance of Squirrel. An abrupt failure will result in a loss of objects. To see this, assume that node i is the home-node for object O. If node i fails, then a new home-node for object O has to be found by Pastry, as explained above, the next time object O is requested. Assume that the copy of object O was fresh when node i failed and consider the first GET request issued for O after the failure of node i. The GET request is therefore forwarded to the new home-node for object O (say node j); this request will result in a miss if j has no copy of O or if its copy is stale. In this case, the failure of node i will yield a degradation in the performance. If a node is able to announce its departure and to transfer its content to its immediate neighbors in the node-Id space before leaving Squirrel (announced failure), then no content is lost when the node leaves.

When a node joins Squirrel then it automatically becomes the home node for some objects but does not store those objects yet (see details in [7]). In case a request for one of those objects is issued, then its two neighbors in the node-Id space transfer a copy of the object, if any. Therefore, we can consider that there is no performance degradation in Squirrel due to a node arrival, since the transfer time between two nodes is supposed to be at least one order of magnitude smaller than the transfer time between any given node and the origin server.

From now on the terms "node" and "client" will be used interchangeably.

2.2 Related work

Performance evaluation of P2P systems has recently attracted a lot of attention. Because of the complexity of these systems (in terms of number of documents, number of users, etc.), many traditional models such as queuing models, Markovian models and branching processes require numerical methods [5, 10] or model simulations [11].

A fluid approach allows one to simplify these systems and may offer meaningful analysis at a low computational cost. We already used this approach in [3] to

study cache clusters then in [2] for a first study of Squirrel. Here we propose an enhanced version of this model and relax some major assumptions.

2.3 The fluid model

The basic idea is to model HTTP requests as a fluid flow modulated by the random arrivals and departures of the Squirrel nodes. Specifically, cached objects are replaced with fluid. This approximation requires the unicity of a cached copy in the Squirrel system, i.e., that objects are not replicated in the system. This is indeed the case in the home store scheme since a single node is elected to be the home node for a given object.

In [2] we modeled the node dynamics by an Engset model. This model has two main problems. First, it requires the existence of a bound N on the number of nodes which can simultaneously be active. In general there does not exist such a bound and, if it did, then it would be very difficult to find. Second, the calculation of the hit rate induced by the Engset model poses serious computational complexity issues as N becomes large. As an illustration, it took more than one day to compute the hit rate (given in [2, Prop. 4.1], using a realistic value of $\rho = 100$) on a Intel 4 2GHz/768Mo workstation for 10,000 nodes, a small population for a corporate network.

To overcome the shortcomings of using the Engset model (i.e. need to have a bound on the number of users and scalability issue), in this paper we model the node dynamics by a M/M/∞ queuing system [8, p. 101]. In the M/M/∞ setting, nodes become active according to a Poisson process with intensity λ (referred to as the arrival process) and each node remains active for an exponentially distributed amount of time, with mean $1/\mu$. It is a natural model since it assumes nodes join the system at arbitrary times, independently of each other. Node activity periods are assumed to be mutually independent, and further independent of the arrival process.

The number of active nodes at time t is denoted by $N(t)$. Let $0 \leq T_1 < T_2 < \cdots$ be the successive jump times of the process $\{N(t), t \geq 0\}$, where a jump corresponds either to a node arrival or to a node departure. We assume that the sample paths of the process $\{N(t), t \geq 0\}$ are right-continuous, so that $N_n := N(T_n)$ gives the number of active nodes just after the n-th jump (i.e. at time T_n+).

Let N^∞ denote the number of nodes which are active in steady-state. It is known that N^∞ has a Poisson distribution with parameter $\rho := \lambda/\mu$ [8, p. 101]. In particular, the expected number of active nodes in steady-state is given by

$$\mathbb{E}\left[N^\infty\right] = \rho. \quad (1)$$

We assume that each node can store an unlimited number of objects. (Currently, disk storage capacity is abundant for most caching systems, and capacity misses are very rare compared to misses due to stale objects.)

The fluid object model is the same as in [2]. It is based on the observation that the process of nodes joining and leaving Squirrel evolves on a much larger timescale than the rate at which documents are requested. Like in [2] we assume that each active node produces a continuous and deterministic stream of requests with rate σ, so that σs requests are generated by each active node throughout the time-interval $[t, t+s)$. Each miss (i.e., unsatisfied request) brings a new document into the Squirrel cache. This follows from the fact that a missing document has to be retrieved from the origin server and brought back into the Squirrel cache, thereby increasing the number of stored documents. It may happen that two concurrent requests for the same document will generate two misses but only one cached copy. This event is assumed rare enough to be neglected. (We validate this claim in Section 6.) The total number of documents in Squirrel at any time is represented by a fluid, with $X(t)$ the amount of fluid at time $t \geq 0$. We assume the sample paths of the process $\{X(t),\ t \geq 0\}$ are right-continuous. In particular, $X(T_n)$ is the amount of fluid just after the n-th jump.

During the time-interval (T_n, T_{n+1}) ($n \geq 0$), each active node generates requests at the constant rate σ, so that σN_n is the total request rate in (T_n, T_{n+1}). Thus $X(t)$ increases at the constant rate σN_n in (T_n, T_{n+1}), multiplied by the probability that a requested document is not found in the Squirrel cache. We also assume that each stored document has a Time-To-Live (TTL) equal to $1/\theta$, so that $\theta X(t)$ is the total expiration rate at time t.

Hence, in the time-interval (T_n, T_{n+1}), $X(t)$ satisfies the following first-order differential equation

$$\frac{d}{dt}X(t) = \sigma N_n(1 - \mathbb{P}[\text{hit}|N_n, X(t)]) - \theta X(t) \quad (2)$$

if $N_n > 0$, where $\mathbb{P}[\text{hit}|N_n, X(t)]$ is the hit probability at time t given that there are N_n nodes active at time T_n+, and that the amount of fluid at time t is $X(t)$. Different expressions for $\mathbb{P}[\text{hit}|N_n, X(t)]$ will be given in Sections 3 and 4 depending on whether or not documents are equally popular.

When $N_n = 0$ (all nodes are inactive in (T_n, T_{n+1})), then the amount of fluid remains equal to zero in this time-interval: $X(t) = 0$ for $T_n < t < T_{n+1}$ when $N_n = 0$.

Let us now examine the behavior of $X(t)$ at jump times T_n, $n \geq 1$. To this end, we introduce two mappings, $\Delta_u : \{0, 1, \ldots\} \to [0, 1]$ and $\Delta_d : \{1, 2, \ldots\} \to [0, 1]$.

If the jump occurring at time T_n corresponds to a new node joining Squirrel, then the amount of fluid in the Squirrel cache at time T_n drops by a factor $\Delta_u(N(T_n-))$, that is $X(T_n) = \Delta_u(N(T_n-))X(T_n-)$. Similarly, if the jump occurring at time T_n corresponds to a departure, then the amount of fluid drops by a factor $\Delta_d(N(T_n-))$, so that $X(T_n) = \Delta_d(N(T_n-))X(T_n-)$. (The subscripts u and d in Δ_u and Δ_d refer to "up" and "down", respectively.)

As discussed in Section 2.1 there is no loss of content when a new node joins Squirrel, since its neighbors (in the node-Id space) can transfer the documents for which the new node is now the home node. Moreover, we assume that a node joining Squirrel does not bring any document with it. Though this assumption can be regarded as restrictive, it has the following motivation: the minimum time during which a node is inactive is orders of magnitude higher than the request inter-arrival time. Therefore, most of the documents stored in an inactive node i, especially the most popular ones, are very likely to be requested and added to their new home nodes, while node i is inactive. As a result, when node i joins the system with its own set of documents, it will not *add* any content to the Squirrel system, but merely become the new home node for these objects. The combination of these two properties gives that $\Delta_u(i) = 1$ for $i \geq 1$.

On the other hand, there is no loss of content if a departure is announced, so that $\Delta_d(i) = 1$ ($i \geq 2$) when such an event occurs. In the case of an abrupt departure the content of the departing node is totally lost. Since it has been observed that Squirrel reaches a good load balancing among the active nodes [7], we can assume that a fraction $1/i$ of the content is lost if a node departs without warning when there are i nodes active. Hence, $\Delta_d(i) = (i-1)/i$ ($i \geq 1$) in this case.

In the following we will analyze both the situations where $\Delta_d(i) = 1$ and $\Delta_d(i) = (i-1)/i$, with $\Delta_u(i) = 1$ in both cases.

We denote by $c < \infty$ the total number of objects that Squirrel clients may request (i.e., the total number of existing objects in the universe).

3 Uniform popularity case

We consider the stationary hit probability as a performance measure of the system. A precise definition of this metric will be given shortly. We first assume that all objects are equally popular, which implies that the probability that a given object o is requested is $1/c$. This assumption is relaxed in Section 4, where a more realistic Zipf-like popularity distribution is considered.

Under the uniform document popularity assumption, the (conditional) hit probability at time t, $\mathbb{P}[\text{hit}|N_n, X(t)]$, is a simple linear function of $X(t)$, given by

$$\mathbb{P}[\text{hit}|N_n, X(t)]) = \frac{X(t)}{c}. \tag{3}$$

Using this value of $\mathbb{P}[\text{hit}|N_n, X(t)]$, (2) becomes a first-order linear ODE. We are interested in the steady-state behavior of the system. We show in [4, Appendix A] that the stationary expected amount of fluid in the system – denoted by $\mathbb{E}[X]$ – exists and is independent of the initial state $(N(0), X(0))$.

Hence, we define the *stationary hit probability* p_H (or simply the hit probability) as the ratio of the stationary expected amount of fluid in the system to the number of available documents, that is

$$p_H = \frac{\mathbb{E}[X]}{c}. \tag{4}$$

For the sake of convenience we introduce the new parameters

$$\gamma := \frac{\sigma}{\mu c} \quad \text{and} \quad \alpha := \frac{\theta c}{\sigma}. \tag{5}$$

A first expression for the hit probability p_H is derived in the following proposition.

Proposition 3.1 *The hit probability is given by*

$$p_H = e^{-\rho} \sum_{i=1}^{\infty} \frac{\rho^i}{i!} v_i \tag{6}$$

where the constants $v_1, v_2, \ldots$ satisfy the infinite linear recursion

$$\begin{aligned}(\rho + \alpha\gamma + (\gamma+1)i)\, v_i \quad &= \quad \gamma i + i\Delta_u(i-1)v_{i-1} \\ &\quad +\rho\Delta_d(i+1)v_{i+1}\end{aligned}$$

for $i \geq 1$, with $v_0 := 0$. ◇

Due to space constraints, the proof of Proposition 3.1 is given in [4, Appendix A]. The expression in (6) is not amenable to efficient computation, since it involves the solution of an infinite system of linear equations and the computation of an infinite series.

Building on Proposition 3.1, the next result provides an alternative expression for the hit probability, which will turn out to be more tractable than (6). This is done for the cases (i) $\Delta_d(i) = (i-1)/i$, $\Delta_u(i) = 1$ and (ii) $\Delta_d(i) = \Delta_u(i) = 1$.

For the sake of compactness we define $\eta := \gamma + 1$.

Proposition 3.2 *Assume that $\Delta_u(i) = 1$ (no loss of content at node arrival).*

If node departures are not announced (i.e. $\Delta_d(i) = (i-1)/i$) then

$$p_H = e^{-\frac{\gamma\rho}{\eta}} \gamma^{-(1+\kappa)} \int_{\frac{1}{\eta}}^{1} \gamma\rho e^{\frac{\gamma\rho t}{\eta}} (t\eta - 1)^{\kappa} \, dt \qquad (7)$$

where $\kappa := \gamma(\alpha\eta + \rho)/\eta^2$.

If node departures are announced (i.e. $\Delta_d(i) = 1$) then

$$p_H = \rho e^{-\frac{\rho\gamma}{\eta}} \gamma^{-\nu} \int_{\frac{1}{\eta}}^{1} (\gamma t e^{\rho t} - v_1) e^{-\frac{\rho t}{\eta}} (\eta t - 1)^{\nu-1} dt \qquad (8)$$

with $v_1 := \dfrac{\int_0^{1/\eta} \gamma t e^{\frac{\rho\gamma t}{\gamma+1}} (1-\eta t)^{\nu-1} \, dt}{\int_0^{1/\eta} e^{\frac{\rho t}{\gamma+1}} (1-\eta t)^{\nu-1} \, dt}$ *and* $\nu := \frac{\alpha\gamma\eta + \rho}{\eta^2}$. ◇

Due to space limitations the proof is given in [4, p.11]. Proposition 3.2 provides a low-complexity formula for the computation of p_H. The only difficulty lies in the evaluation of the various exponentials, especially when the expected number of active nodes ρ is large (see (1)). In this case, a good accuracy can be achieved by rewriting p_H in the form $p_H = \int_{1/\gamma+1}^{1} e^{f(t,\rho,\alpha,\kappa)} dt$, where the mapping f can easily be identified from (7) (resp. (8)). Using this method, the average CPU time needed to compute the hit probability using (7) or (8) is typically less than a second with a Intel 4 2GHz/768Mo workstation, even for networks as large as a million nodes.

4 Zipf-like popularity case

Following [1] we now assume that the popularity of the documents follows a Zipf-like distribution. This implies that the probability ψ_n that the n-th most popular object is requested, is given by

$$\psi_n = \frac{\Omega}{n^\beta} \quad \text{for } n = 1, \ldots, c \qquad (9)$$

with $0 < \beta \leq 1$, where $\Omega := 1/\sum_{i=1}^{c} i^{-\beta}$ is a normalization factor. When $\beta = 1$ then we have the Zipf's law. (For sake of comparison, note that $\psi_n = 1/c$ under the uniform popularity assumption – see analysis in Section 3.)

The next step is to replace (3) by an expression that takes into account the popularity of the documents, as defined in (9). If we assume that the $X(t)$ cached objects at time t are the most popular ones, a natural choice for $\mathbb{P}[\text{hit}|N_n, X(t)]$ is (with $\lfloor x \rfloor$ the largest integer less than or equal to c)

$$\mathbb{P}[\text{hit}|N_n, X(t)] = \sum_{i=1}^{\lfloor X(t) \rfloor} \frac{\Omega}{i^\beta} \approx \frac{X(t)^{1-\beta} - 1}{c^{1-\beta} - 1} \qquad (10)$$

by using the approximation $\sum_{i=1}^{\lfloor x \rfloor} i^{-\beta} \approx \int_1^x t^{-\beta} dt = (x^{1-\beta} - 1)/(1-\beta)$ for $x \geq 1$.

Unfortunately, with this hit probability function equation (2) has no closed-form solution, which does not allow us to develop the same kind of analysis as in Section 3.

Instead, we suggest to approximate the hit probability by dividing the set of c documents into K popularity classes of size c_k, $1 \leq k \leq K$ ($\sum_{k=1}^{K} c_k = c$) and to assume that documents belonging to the same class have the same popularity. The hit probability within each class can then be computed using Proposition 3.2.

More specifically, assume that the K classes are ordered according to the popularity of their documents, with class 1 containing the most popular documents, class 2 the second most popular documents, etc. We define the global hit rate p_H as a weighted sum of the intra-class hit probabilities, that is,

$$p_H = \sum_{k=1}^{K} q_k \, p_H^k \qquad (11)$$

with p_H^k the hit rate for documents of class k, and q_k the probability that a document of class k is requested. From (9) we see that

$$q_k = \sum_{i=1}^{c_K} \frac{\Omega}{(\sum_{l=1}^{k-1} c_l + i)^\beta} \qquad (12)$$

for $k = 1, 2, \ldots, K$. This formula is obtained by summing up the popularities of all documents in class k, with $\Omega/(\sum_{l=1}^{k-1} c_l + i)^\beta$ the popularity of the i-th most popular document of class k.

The intra-class hit probability p_H^k is obtained from Proposition 3.2 by replacing the parameters α and γ in (7) and (8) by $\alpha_k = \theta c_k/(\sigma q_k)$ and $\gamma_k = \sigma q_k/(\mu c_k)$, respectively.

It remains to determine the number of classes K and the number of objects assigned to each class. We first select the number of classes K. Clearly, the accuracy of this approximation will only increase with the number of classes. As a result, we simply choose the highest value of K that leads to an affordable computation. In Section 6 we will search for an acceptable number of classes through a comparison with a simulation of the real system.

Once K is chosen, we need to calculate the number of objects c_k assigned to each class k. This is a classical clustering problem that can be solved with a scalar quantization algorithm (see e.g. [6]), which also readily provides the q_k coefficients. Given the initial popularity vector $(\psi_1, ..., \psi_c)$, the vector quantization algorithm

aims at finding the class vector $(\phi_1, ..., \phi_K)$ that minimizes

$$E = \sum_{n=1}^{c} d(\psi_n, Q(\psi_n)),$$

where $d(.)$ is a distance measure (in our case the Euclidean distance) and $Q(\psi_n)$ the quantified version of ψ_n in the set $\{\phi_1, ..., \phi_K\}$, namely,

$$Q(\psi_n) = \arg\min_{\phi_k} d(\psi_n, \phi_k). \quad (13)$$

The quantity ϕ_k can be understood as the average popularity of documents in class k. Therefore the q_k coefficients are given by $q_k = c_k \phi_k, \quad 1 \leq k \leq K$.

In order to determine the set $\{\phi_1, ..., \phi_K\}$ we used the Lloyd algorithm [6, page 189] that can be seen as an application of the Expectation-Maximization (EM) algorithm. This algorithm is composed of the following four steps:

S1 Initialize $(\phi, \ldots, \phi_K)$ (for example by using random sampling);

S2 For $n = 1, \ldots, N$, estimate $Q(\psi_n)$ from (13): for each ϕ_k we obtain c_k corresponding objects;

S3 For $k = 1, 2, \ldots, K$, re-estimate ϕ_k: $\phi_k = (1/c_k) \sum_{n:Q(\psi_n)=\phi_k} \psi_n$;

S4 Go back to step 2 (S2) until convergence.

Since this algorithm is based on EM, the error will decrease at each iteration so that the set $\{\phi_1, ..., \phi_K\}$ will converge to a local optimum. In practice, this algorithm provides the optimal vector $(\phi_1, ..., \phi_K)$ along with the corresponding $(c_1, ..., c_K)$ values.

This approximation is validated in Section 6.

5 Qualitative observations

We now investigate the impact on the hit probability of the document popularity distribution (Section 5.2) and of announced/unannounced departures (Section 5.3).

5.1 Experimental setup

We used Matlab to compute the hit probability from (7), (8) and (11) with the following parameters:

$$\begin{cases} c &= 10^7 \text{ files} \\ \sigma &= 10^{-3} \text{ requests per second and per user} \\ \theta &= 10^{-6} \text{ s}^{-1} \text{ (11-day TTL)} \\ \mu &= 10^{-7} \text{ s}^{-1} \text{ (3 abrupt failures per year and per user)} \end{cases}$$

With the above values, we see from (5) that $\gamma = 10^{-3}$ and $\alpha = 10^4$. For the Zipf-like distribution we used $\beta = 0.7$ (cf. [1]) and an approximation of $K = 10$ classes for 10^7 documents (cf. Section 6 for a discussion on the choice of K).

We also investigate the role of the mean online time on the hit probability in Section 5.3 by setting $\rho = 10^5$ and varying μ instead. This case will be explicitly mentioned.

5.2 Impact of the Zipf-like popularity

Figure 1 displays the hit probability for the Squirrel system with unannounced departures as a function of the expected number of active nodes ρ, for uniform and Zipf-like document popularity distributions. In both cases, the hit probability is an increasing function of the size (i.e. ρ) of the network, which reflects the self-scaling nature (and therefore the scalability) of the Squirrel system.

We can see on Figure 1 that the Zipf-like document popularity distribution generates higher hit probability than the uniform popularity for small and medium-sized networks (say up to 10^4-10^5 active nodes on average). This is a rather intuitive result since when the popularity is skewed, many requests can be served with only a few popular cached documents.

From this, we infer that the document probability distribution is a crucial performance factor, which must be carefully modeled.

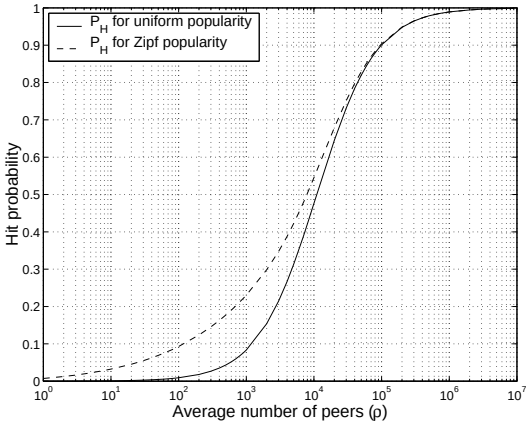

Figure 1. Hit probability of Squirrel for various document popularity distributions.

One can also use Figure 1 to determine the minimum network size necessary for an acceptable performance. For instance, with the experimental setting in Section 5.1, 8000 nodes must be active on the average with the

Zipf-like distribution if one wants the hit probability to exceed $1/2$.

5.3 Utility of announced departures

In this section we evaluate the benefit of announcing departures on Squirrel performance. We compare the hit probability of the Squirrel system in the case of abrupt failures and announced departures. We do this for the uniform popularity case, using (7)-(8).

When the nodes disconnect rarely from the system (e.g., 3 times a year) the relative performance gain achieved by announcing their departure is very small, around 5% as shown in [4]. The interest of announcing departures thus has to be balanced with the overhead cost that this feature will induce, and which is due to departing nodes transferring their content to their neighbors. We can expect this tradeoff to depend strongly on the mean online time of peers $1/\mu$. Intuitively, the smaller the mean online time, the greater the benefit of announcing departures - and the greater the overhead cost.

In Figure 2 we compare the hit probability of the Squirrel system for announced and unannounced departures for various departure rates, ranging from 10^{-7} (3 departures per year) to 10^{-5} (around 1 departure per day). For this experiment we used a network size of $\rho = 10^5$ nodes and $\theta = 10^{-5}$ (24 hours TTL).

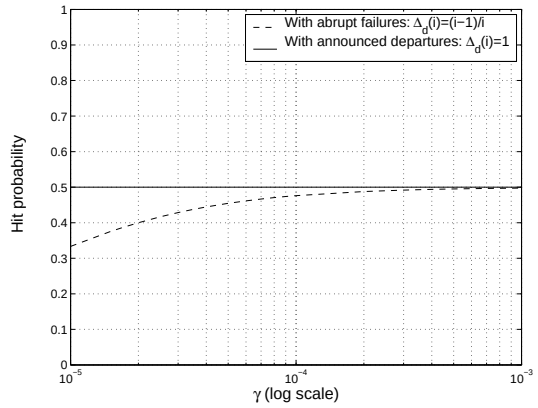

Figure 2. Hit probability of Squirrel for announced and unannounced node departures as a function of the mean online time (network of 100,000 nodes).

We observe that the performance of the system does not depend on γ (i.e., on μ in this experiment) when nodes are able to announce their departure. While this property is not directly visible from the expression in (8), it is fairly intuitive since an announced departure does not generate performance degradation, unlike abrupt failures. We also observe that the performance degradation due to abrupt failures only becomes significant for $\gamma \leq 10^{-4}$, corresponding to $\mu \geq 10^{-6}$, or a mean online time of 11 days at most.

6 Experimental validation

The goal of this section is to validate the fluid model approximation of requests, as well as the clustering approximation of document popularity, against a discrete-event simulation of the Squirrel system. Regarding the validity of the M/M/∞ model instead of the earlier Engset model, we compared the hit probability when the node dynamics are modeled as the number of customers in a M/M/∞ queuing system, given in (7), to the corresponding formula in [2, Prop. 4.1] where node dynamics are represented by the Engset model. To do so we fixed the mean number of active nodes to the same value in both models and varied it between 1 and 11000 nodes (range of tractability of the hit probability obtained via the Engset model). The hit probability with the Engset model was computed using Maple V.

We found that both models consistently predict the same hit probability over all range of loads (i.e. mean number of active nodes), even for very small networks. The relative error was always smaller than 10^{-4}. Therefore, we can expect both models to describe the Squirrel system with the same accuracy.

In [2], we compared the theoretical results obtained via the Engset model to a discrete-event simulation of the Squirrel system with uniform popularity distribution. The simulation validates the fluid model approximation by using Poisson arrivals for requests and by allowing concurrent requests. We found that the theoretical hit probability was remarkably close to the hit probability obtained through simulations (see [2] for details).

We can therefore safely conclude from the above that the hit probability computed via the M/M/∞ model offers the same accuracy as the one obtained via the Engset model, at least in the uniform popularity case (the analysis in [2] was only carried out for uniformly popular objects). In particular, we can reasonably extrapolate that the model developed in this paper is a good approximation of Squirrel's behavior when deployed on large networks (say larger than 10,000 users), a situation where both discrete-event simulations and the model in [2] fail to work.

In Figure 3 we compare our multiclass approach (Section4) to a discrete-event simulation of the Squirrel system with a Zipf-like popularity distribution. The simulation uses Poisson arrivals for requests (with rate

$\sigma N(t)$ at time t) and allows concurrent requests for popular files. The parameters were: $c = 40,000$ files, $\rho = 9.99$ nodes, $\theta = 10^{-3}$ s,$^{-1}\mu = 10^{-7}$ s,$^{-1}$ $\beta = 0.7$. We varied the request rate σ. Simulation results are accurate at 0.2% with 99% confidence.

Figure 3 shows that our multiclass model is able to approximate very closely the hit probability of the simulated system: with 10 classes the curve follows already closely the same shape as the curve obtained by simulation, and with 100 classes the relative error amounts to 1%. We conclude that the combination of fluid approximation of request streams and of the multiclass approach for modeling the different document popularities provides an accurate estimation of Squirrel performance.

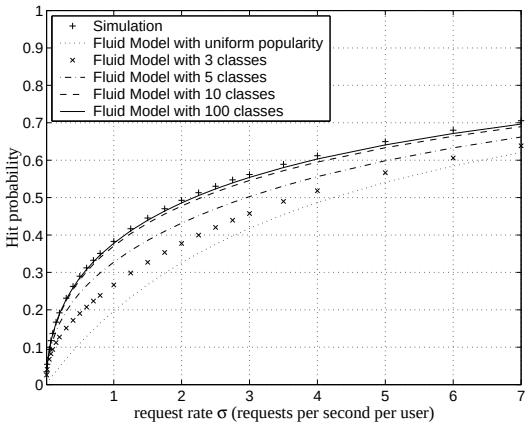

Figure 3. Comparison of the multiclass M/M/∞ model with a discrete-event simulation of Squirrel under a Zipf-like popularity distribution.

7 Conclusion

We proposed a stochastic fluid model for the Squirrel peer-to-peer cooperative caching system. This model, based on M/M/∞ node dynamics, approximates the request streams of the individual nodes by a fluid flow. The model can be viewed as a scalable extension of our previous Engset-based fluid model. The new model turns out to be tractable for any network size and is also more convenient than our previous model. In addition, the model allowed us to study the effect of nodes announcing their departure on the resulting hit probability.

Furthermore, we proposed, implemented and evaluated a multiclass approach to take variable object popularity into account. We found that this method gives accurate results even with a small number of classes.

Finally, this paper illustrates a stochastic fluid model approach through the example of the Squirrel system. This approach can also be straightforwardly applied to other systems based on distributed hash tables, provided these systems exhibit load balancing properties and do not replicate objects.

Future work will intend to adapt the fluid model approach to analyze other types of content distribution systems which require to take into account document replication.

References

[1] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of IEEE INFOCOM '99*, pages 126–134, New York, 1999.

[2] F. Clévenot and P. Nain. A simple model for the analysis of the Squirrel peer-to-peer caching system. In *Proceedings of IEEE INFOCOM 2004*, Hong Kong, March 2004.

[3] F. Clévenot, P. Nain, and K. W. Ross. Stochastic fluid models for cache clusters. *Performance Evaluation*, 59(1):1–18, January 2005.

[4] F. Clévenot-Perronnin and P. Nain. Stochastic fluid model for P2P caching evaluation. Technical Report 5533, INRIA, Sophia Antipolis, January 2005.

[5] Z. Ge, D. Figueiredo, S. Jaiswal, J. Kurose, and D. Towsley. Modeling peer-peer file sharing systems. In *Proceedings of IEEE INFOCOM 2003*, San Francisco, California, April 2003.

[6] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992.

[7] S. Iyer, A. Rowstron, and P. Druschel. Squirrel: A decentralized, peer-to-peer Web cache. In *Proceedings of of ACM Symposium on Principles of Distributed Computing (PODC 2002)*, pages 213–222, Monterey, California, 2002.

[8] L. Kleinrock. *Queueing systems*, volume 1. J. Wiley and sons, 1975.

[9] A. Rowstron and P. Druschel. Pastry: Scalable distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of Int. Conf. on Distributed Systems Platforms (Middleware)*, Heideberger, Germany, November 2001.

[10] X. Yang and G. De Veciana. Service capacity of peer-to-peer networks. In *Proceedings of IEEE INFOCOM 2004*, Hong Kong, March 2004.

[11] L. Zou and M. Ammar. A file-centric model for peer-to-peer file sharing systems. In *Proceedings of ICNP 03*, Atlanta, Georgia, USA, November 2003.