Optimal Proactive Caching in Peer-to-Peer Network: Analysis and Application

Weixiong Rao
Department of Computer Science and Engineering
The Chinese University of Hong Kong
wxrao@cse.cuhk.edu.hk

Lei Chen
Department of Computing Science
Hong Kong University of Science and Technology
leichen@cs.ust.hk

Ada Wai-Chee Fu, YingYi Bu
Department of Computer Science and Engineering
The Chinese University of Hong Kong
{adafu,yybu}@cse.cuhk.edu.hk

ABSTRACT

As a promising new technology with the unique properties like high efficiency, scalability and fault tolerance, Peer-to-Peer (P2P) technology is used as the underlying network to build new Internet-scale applications. However, one of the well known issues in such an application (for example WWW) is that the distribution of data popularities is heavily tailed with a Zipf-like distribution. With consideration of the skewed popularity we adopt a proactive caching approach to handle the challenge, and focus on two key problems: *where*(i.e. the placement strategy: where to place the replicas) and *how*(i.e. the degree problem: how many replicas are assigned to one specific content)? For the *where* problem, we propose a novel approach which can be generally applied to structured P2P networks. Next, we solve two optimization objectives related to the *how* problem: MAX_PERF and MIN_COST. Our solution is called **PoPCache**, and we discover two interesting properties: (1) the number of replicas assigned to each content is proportional to its popularity; (2) the derived optimal solutions are related to the entropy of popularity. To our knowledge, none of the previous works has mentioned such results. Finally, we apply the results of PoPCache to propose a P2P base web caching, called as Web-PoPCache. By means of web cache trace driven simulation, our extensive evaluation results demonstrate the advantages of PoPCache and Web-PoPCache.

Categories and Subject Descriptors

H.3.7 [**Information storage and retrieval**]: Information Search and Retrieval - Search process; C.2.4 [**Computer-Communication Networks**]: Distributed Systems - Distributed applications

General Terms

Design, Performance

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CIKM'07, November 6–8, 2007, Lisboa, Portugal.
Copyright 2007 ACM 978-1-59593-803-9/07/0011 ...$5.00.

Keywords

Peer-to-Peer, Web Caching, Placement Strategy

1. INTRODUCTION

As a promising new technology, Peer-to-Peer (P2P) systems have the potential to build applications at a very large scale. Existing P2P applications like Gnutella and KazaA connect millions of machines to provide Internet-scale file sharing services. Due to P2P's unique distributed, autonomous, and heterogenous characteristics, efficient search algorithms are essential to improve the usability of a P2P system. Therefore, many research works have been conducted on addressing the search efficiency issue, including Chord [23], Pastry[21], Tapestry[28], CAN[20], etc. In these approaches each peer and its stored contents are organized using a distributed hash table (DHT), and they are examples of structured P2P systems. For such a system with N nodes, the search cost (i.e. number of lookup hops) is bounded by $O(\log N)$. For the current search solutions in the structured P2P, all peers are assumed to submit queries to uniformly search the contents stored in all nodes. However, this assumption is not valid in practice. Often, the popularities of the contents (measured with respect to the proportion of all submitted queries that can be satisfied by the content [16, 8, 29]) are quite skewed. For example, web requests on the Internet space are highly skewed with a Zipf-like distribution. Therefore, in the reality, this skewed popularity will cause the workload of whole network unbalanced, moreover, due to the limited bandwidth, though the search cost to some "hot" peers (the peers have content that can satisfy more queries) is still bounded to $O(\log N)$ in terms of the number of hops, there will be a long latency for getting the data.

So far, we have listed the challenges brought by the skewed popularity distribution. In fact, this skewed popularity distribution also brings opportunity. If we can replicate these popular contents, there will be substantial reduction on the search cost for popular contents and the workload of the whole network can be balanced as well. Therefore, in this paper, in order to make use of the skewed popularity distribution for proper replication, we propose an analysis model to handle two key problems related to caching the replicates in structured P2P systems: *where* and *how* to cache the replicas of popular contents in the P2P nodes? For the first problem involving where to place the replicas of the contents, i.e. the placement strategy problem, we propose a novel approach which can be generally applied in the struc-

tured P2P network by placing the replicas in the so-called k-ary tree $\mathcal{T}_i$. Then with such placement strategy, we handle the problem that how many replicas are assigned for one object, i.e. the degree of replication, and we consider two optimization criteria:

– MAX_PERF: given a constant number of replicas, how to minimize the request latency measured by the average lookup hops in P2P network (i.e. maximize the performance)?

– MIN_COST: given a targeted threshold of the request lookup hops, how to minimize the copy number of replicas (i.e. minimize the cost)?

Replicas, as the paid cost to diminish the average lookup hops of P2P systems, will consume the bandwidth by shipping the copies to multiple destination nodes; on the other hand, the average lookup hops of P2P, as the gained benefit from replication, is one of the most important factors to measure the performance of P2P systems and useful to diminish the request latency in P2P applications. Consequently, how to tradeoff the gained performance(i.e. average lookup hops) and the paid cost(number of replicas) is the key problem during the design of P2P based Caching system. In this paper, we formate such tradeoff as two orthogonal optimization problems: MAX_PERF and MIN_COST; then, the system designers can choose either one criterion based on the system design objective with our closed-form solutions for both optimization criteria. We refer our proposed replica placement strategy together with the optimal closed form solutions as *the popularity-based P2P based proactive caching*, PoPCache in short.

In order to demonstrate the real usage of PoPCache, we develop a P2P-based web caching scheme, called as Web-PoPCache. We have shown through the extensive experiments with respect to the feasibility and optimization of Web-PoPCache based on our proposed analysis model. In summary, we make the following contributions in this paper:

- We propose a novel replica placement strategy of PoPCache, by which we can achieve better results than currently known placement strategies such as CFS [9](the cooperative file system over Chord [23]), Beehive [19], and the PAST [22].
- Under such a placement strategy and the optimization criteria MAX_PERF and MIN_COST: (1)we derive a proportional principle, i.e. the number of replicas for a web content is proportional to its popularity. (2) we also find that the entropy of content popularity is related to the optimal closed form solutions.
- We propose to a PoPCache based proactive web caching solution, Web-PoPCache, which replicates web contents among a large number of connected clients. The proposed web caching solution can optimally tradeoff the performance gain and the cost.

Among all contributions, two are interesting: (1) For both the objectives of MAX_PERF and MIN_COST, the optimal number of replicas is *proportional* to the *object popularity* p_x. It is known that for unstructured P2P, the random walk based technique is optimized by the *square-root* principle [7, 16, 29, 29]; here we arrive at a different optimal function of the popularity for a structured P2P system. (2) For both optimization problems MAX_PERF and MIN_COST, the derived closed form solutions are related to *entropy* of popularity p_x. To our knowledge, none of previous P2P works has mentioned such a relationship. Intuitively it makes sense since we have expected that our approach can accelerate the search for popular nodes, then the skew of the popularity distribution will play an important role in the optimization of the search performance, and taking the popularity as a probability function of the query targets, entropy is a sound measure of the skew of the distribution.

The rest of this paper is organized as follows: Section 2 introduces the background of web caching system. In Section 3 we present the related work in this area. In Section 4 we present PoPCache's novel replica placement strategy and the closed form solution for the optimization objectives. In Section 5 we present Web-PoPCache, a P2P based proactive web caching scheme based on PoPCache. Section 6 evaluates the performance of PoPCache and Web-PoPCache. Finally, Section 7 is a conclusion.

2. BACKGROUND

As an Internet application, web caching is a widely utilized technology to reduce the content request latency, to decrease the amount of aggregate network traffic between the client side and the server side where the requested contents are hosted, and to balance the workload by distributing the heavy workload of the busy web servers. Given a request from the web browser of a client side to the original web server of the server side, web caching can be implemented in various locations: (1) in the local directory of *client side web browser*; (2) at the *origin web server* (for example, the contents, or portions of contents, can be stored in a server-side cache to reduce the server load); (3) at the intermediate the *proxy servers* located between the client side and the original web server, including *client side proxy servers* (the organization proxy, and the forward proxy cache of client side ISP:Internet Service Provider), and the *server side proxy servers*(reverse proxy cache of server side ISP, and such a network is called a content delivery network: CDN).

Web contents can be cached while passing to the *client side web browser* from *client side proxy servers*(the organization proxy, and the forward proxy cache of client side ISP), hence there can be caching only for those contents which are already requested. This kind of caching is typically called *passive* caching at client side. On the other hand, *proactive caching* means that the replicas of web contents are *proactively* cached by the original web servers and the server side CDN to achieve the goals of improved performance and load balancing. The proactive caching technique is widely utilized for the content providers like Google or the third-party CDN provider like Akamai. Though supporting load balancing and having better performance including reduced request latency and decreased bandwidth consumption, such caching involves expensive costs, which include the expensive dedicate hardware devices (for example high performance servers and network devices), the operational or administrative cost and the associated network bandwidth consumption.

To avoid the expensive cost to construct the server side caching and CDN, P2P technology is an alternative to connect a large number of volunteered nodes with low costs (for example desk top machines) to construct a cooperative web caching system. Squirrel [12] is an example of such a web caching system and it shares the local contents by Pastry [21]

to form an efficient and scalable web caching. However, the passively web caching technique, such as the one applied by Squirrel, only stores the web content to the node with node ID numerically closest to the hash ID of the URL of such web content. There is no consideration about the popularity skewness of the contents. Consequently, the workload of the whole network may still unbalanced.

Cooperative Web caching is the most common solution for augmenting the low cache hit rates due to single proxies. There has been extensive work on cooperative web caching system as a technique to reduce request latency and increase the hit rate. The design of cooperative web caching systems can be hierarchical like Harvest [10] and Squid [1], hash-based [14],directory-based [11], and multicast based [24]. Different from these cooperative web caching systems which still require a dedicated proxy infrastructure, P2P based web caching systems completely eliminate the need of proxy servers. Kache [15], a cooperative web caching system built over Kelips [15], can perform a lookup in one hop, however, with the cost of maintaining a list of $O(\sqrt{N})$ peers in each node. Squirrel [12], a peer-to-peer web cache targeted at replacing central demand-side web caches within a local area network, is a passive, opportunistic cache with no consideration of the skewed popularity and performance optimization.

3. RELATED WORK

In this section we briefly review the related works in three aspects: structured P2P systems, replica placement strategy, and the degree of replication.

Structured Peer-to-Peer System: A number of peer-to-peer routing protocols have been proposed recently, including CAN [20], Chord [23], Tapestry [28] and Pastry [21]. These self-organizing, decentralized systems provide the functionality of a scalable distributed hash-table (DHT), by reliably mapping a given object key to a unique live node in the network. The systems have the desirable properties of high scalability, fault tolerance and efficient routing of queries. Besides these DHT-based P2P, randomized P2P networks like Symphony [17], SkipGraph [6],and more structured P2Ps like BATON [13], P-Grid [4] etc are all examples of structured P2P. In these structured P2P protocols, each node is regularly assigned with equal number of neighbors, and the average lookup hop number is guaranteed with $O(\log N)$.

Replica Placement Strategy: Some works in P2P network propose to place the replica or cached objects to smooth the load in the hot-spot nodes caused by popular queries. CFS [9], a cooperative file system over Chord [23], caches the popular objects along the lookup path towards the nodes where the popular objects are stored. In PAST [22],the storage system over Pastry [21], the search for some object is redirected to the nearest replicas of the targeted object. Such placement strategy [22] is a random placement strategy. Based on the replication level l, Beehive [19] replicates the object content copies to all the nodes that have at least l matching prefixes with the object. Unlike the heuristical schemes like CFS [9] and PAST [22], Beehive [19] and our work can provide optimal solutions based on the analytical model. Compared with Beehive [19], our proposed solution for an optimization problem MIN_COST makes use of less replicas to achieve the same targeted performance and provide better granularity to control the number of replicas; furthermore, there is no assumption on the popularity distribution in our optimization, while Beehive [19] only provides the analytical guarantee for the Zipf distribution. Different from our work, under the proposed cost model, [5] presented the replication of dynamic XML documents in the context of the ad-hoc connected web services which can be treated as unstructured P2P, and does not provide the global optimality.

Symbol	Meaning
n_i	i-th node in the system
N	the total number of nodes in the system
c_x	x-th content in the system
p_x	the request popularity of c_x
C	the total number of contents in the system
l_x	the number of replicas assigned to content c_x
L	the total number of replicas
H_x	the hop number to lookup content c_x
H	the average hop number to lookup all contents
k	number of links regularly assigned at each node
τ	the targeted number of average lookup hops
α	parameter in Zipf distribution $z^{-\alpha}$

Table 1: Meanings of main symbols used

The Degree of Replication: In unstructured P2P networks, [7] proposes to optimize search efficiency by replication, where the number of replicas of an object is proportional to the square-root of the object's popularity. [8] presents a square-root topology for unstructured P2P networks where the degree of a peer machine is proportional to the square root of the node popularity. Their results show that for random walk search, the square-root principle can achieve optimal performance. Compared with these related works, our work and Beehive [18, 19] solve the MIN_COST problem with closed form solution, however, Beehive [19] only arrives at the approximate numeric results for MIN_COST under the assumption of the Zipf popularity distribution. Furthermore, for the optimization problem MAX_PERF, we derive the proportional principle, in contrast to the square-root principle un-structured P2P network in the above.

4. THEORETICAL ANALYSIS

In this section, we first present our novel replica placement strategy in Section 4.1, then propose the analytical solutions of PoPCache to two optimization objectives related to the replication degree problem in Section 4.2. Table 1 summarizes the main symbols used in this section.

4.1 Replica Placement in PoPCache

Our proposed placement strategy is a general framework which can be applied in structured P2P networks like DHT based P2P(Chord [23], Pastry [21], Tapestry [28], CAN [20]), randomized structured P2P (Symphony [17], SkipGraph [6]), or BATON [13]. This is difference from the work in Beehive [19], which is limited to the prefix matching based DHT like Pastry[21].

In general, each node in structured P2P network is regularly assigned with k links for routing. In Chord [23], the value of k is the size of the finger table; in Pastry [21] (Tapestry [28]) the value of k can be treated as the number of rows (levels) in the routing table (neighbor set); in Symphony [17], the value of k is the number of long links. A similar situation holds for other structured P2P network.

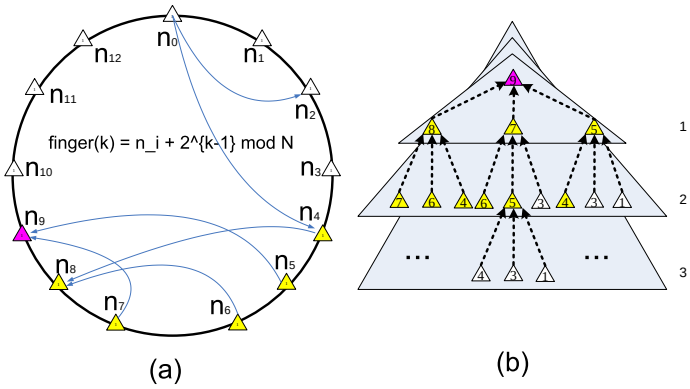

Figure 1: The proposed replica placement strategy

With k-regular links for routing in each node, the content object c_x stored in the structured P2P network, can be found with the average $O(\log N)$ number of hops, where N is the total number of nodes.

The key observation in structured P2P is that each node n can be treated as the root of a k-ary tree with the k direct neighbors of n connected by k links as the 1st level children, the neighbors of neighbors of n added with k^2 links as the 2nd-level children... As a result, given total N nodes in the structured P2P network, any node n_i can be treated as the root of a k-ary tree, which has at most $\log_k N$ levels with k^ℓ remote neighbors as the ℓ-level children where $\ell = 1, 2, ..., \log_k N$. Such a tree with node n_i as the root is denoted as $\mathcal{T}_i$. The search from some node n_j to n_i is the process of greedily approaching the root n_i along the bottom-up path of $\mathcal{T}_i$ towards n_i.

For simplicity, we take Chord [23] as example. In Figure 1(a), each node has a finger table with size equal to $k = 3$(Note: in Figure 1(a) we omit the arrows between one node and its successor, for example n_0 to n_1, for the clear figure). Based on the finger table allocation principle, the i-th node with nodeID n_i is connected directly to nodes with nodeID equal to $(n_i + 2^0)$ MOD N, $(n_i + 2^1)$ MOD N, and $(n_i + 2^2)$ MOD N, where $N = 13$. For example node n_0 with nodeID equal to 0 in Figure 1(a) has 3 links respectively pointing to nodes n_1,n_2, and n_4, which alternatively means nodes n_1,n_2, and n_4 are connected by n_0 . With the same situation, node n_9 is connected by nodes n_8, n_7, and n_5; n_7 is connected by n_6, n_5, and n_3. The same situation holds for other nodes. Then n_9 in Figure 1(a) is the root of $\mathcal{T}_9$, with the direct neighbors of n_8, n_7, and n_5 as the 1st-level children, the direct neighbors of n_8,n_7, and n_5 as 2nd-level descendants... The lookup for n_9 from any source node can be treated as the process of greedily approaching the root of $\mathcal{T}_9$, i.e. n_9 in the bottom-up manner. The routing from any node towards the root of $\mathcal{T}_i$ always finds the closest path to greedily reach the root based on the local routing information of n. For example, to find n_9, n_4 can look up the local routing information (n_5,n_6, and n_8) and consider n_8,instead of n_4 or n_5, as the closest intermediate forwarder to reach n_9. As a result, any node located at the ℓ-th level of the tree of n_i will consume on average average $O(\ell)$ hops to reach the root n_i. For other structured P2P network, such kind of k-ary tree can also be conceptually constructed for each node.

Now we show how to utilize the k-ary tree $\mathcal{T}_i$ for replica placement. For a structured P2P network with k regular links in each node, to place the replicas for some content object c_x, we can first find the *home* node n_i of the content object c_x, i.e. the node where the content object c_x is stored in the structured P2P network. The tree $\mathcal{T}_i$ with n_i as the root can be utilized for replica placement. Suppose l_x replicas of the content object c_x are required. First, at most (due to the fact that one node can appear in multiple locations of the k-ary tree $\mathcal{T}_i$) k replicas are placed in the 1st level children of n_i in the tree $\mathcal{T}_i$, next at most k^2 replicas are placed in the 2nd level children of n_i in the tree $\mathcal{T}_i$,..., until all l_i replicas of c_x are consumed. The corresponding l_x descendant nodes of n_i to store the replicas of c_X are called the *delegate* nodes, and the area with these l_x delegate nodes is called the *acceleration area* of c_x. In Figure 1(b), if 5 replicas are allocated for node n_9 for caching, first the 1st level children (n_8,n_7, and n_5) are respectively placed with one replica; then the two more replicas are placed in two 2nd level nodes (n_6 and n_4). During the greedy search towards the designation node, the nodes with less distance to the destination node have a higher probability to be visited as the intermediate nodes towards the destination node than those node with larger distance to destination node. As a result, n_6 and n_4, instead of n_3 and n_1, are chosen to place the replicas for node n_9. Based on our placement strategy we have the following theorem:

Theorem 1 *In structured P2P with N nodes and k regular links for routing in each node, for content object c_x with l_x replicas placed along levels of the tree $\mathcal{T}_i$ with the home node n_i of c_x as the root, the average hop number H_x to lookup c_x is $O(\log N - \log_k l_x)$.*

Proof: Since the content object c_x is stored in the home node n_i, c_x is located at the root n_i of the k-ary tree $\mathcal{T}_i$. Based on our proposed replica placement strategy, suppose the 1-st level children of n_i are placed with at most k replicas of c_x, the search for c_x from any source can be reduced on average by 1 hop; again if both the 1-st and 2-nd levels descendants of n_i are assigned at most $(k^1 + k^2)$ replicas, then the search for c_x can be reduced on average by 2 hops,..., if the 1-st level, the 2-nd level,..., the ℓ-th level descendants of n_i are in total assigned $(k^1 + k^2 + ... + k^\ell)$ replicas of c_x, the search for c_x can be reduced on average by ℓ hops. Given l_x replicas for c_x, they are placed at the 1-st, 2-nd,...,$\log_k(1 + l_x(1 - 1/k))$-th level descendants of n_i, the search for c_x can be reduced on average by $\log_k(1 + l_x(1 - 1/k))$ hops.

For any request to lookup c_x, once it reaches any one of l_x delegate nodes of n_i, it can be terminated and the rest of the search from the current node to n_i is skipped. Consequently, the average hop number to lookup c_x will be reduced by at most $\log_k l_x$, i.e. the average hop number H_x to lookup c_x is $H_x = O(\log N - \log_k l_x)$. ■

4.2 Optimal Proactive Cache

As discussed in the above section, given content object c_x with copy number equal to l_x, the average hop number to lookup c_x is $H_x = O(\log N - \log_k l_x)$. The choice of the optimization criterion MAX_PERF or MIN_COST is dependant upon the system design principle, the optimization objective can be either oriented to the best performance or the cost minimization. In this section, we proactively place the

replicas of popular contents with the placement strategy in Section 4.1 and provide our solutions for both optimization criteria MAX_PERF and MIN_COST based on the analytical model.

4.2.1 MAX_PERF: Performance MAXIMIZATION

Given the total number of L replicas in the structured P2P network, the optimal objective of MAX_PERF is to maximize the performance by minimizing the average hop number to lookup all C content objects in the structured P2P network. In this paper, given the content object c_x, the content popularity of c_x can be defined as the proportion of all submitted queries that can be satisfied by c_x. This definition is consistent with [8, 29]. Obviously, $0 \leq p_x \leq 1$ and $\sum_{i=1}^{C} p_x = 1$ where C is the total number of content objects in the system. Since the queries follow the popularity distribution, the average hop number to lookup all content objects, H, is given by:

$$H = \sum_{x=1}^{C} (p_x \cdot H_x) = \sum_{x=1}^{C} p_x \cdot (\log N - \log_k l_x) \tag{1}$$

Then we achieve the following proportional principle:

Theorem 2 *Given $\sum_{x=1}^{C} l_x = L$, H is minimized when $l_x = p_x \cdot L$.*

Proof: This is an optimization problem of H with the subjective $\sum_{x=1}^{C} l_x = L$. We use the Lagrange multiplier method to solve for the optimal value of l_x in terms of p_x. We find the Lagrange multiplier λ that satisfies $\nabla H = \lambda \cdot \nabla f$ where $f = \sum_{x=1}^{C} l_x - L = 0$. First, treating p_x, $\log N$ and k as the constants,

$$\nabla H = \sum_{x=1}^{C} p_x \cdot \frac{1}{\ln k} \cdot (-\frac{1}{l_x}) \cdot \widehat{u_x} \tag{2}$$

where $\widehat{u}_x$ is a unit vector. Next,

$$\nabla f = \sum_{x=1}^{C} \lambda \cdot \widehat{u_x} \tag{3}$$

Since $\nabla H = \lambda \cdot \nabla f$, then

$$p_x \cdot \frac{1}{\ln k} \cdot (-\frac{1}{l_x}) = \lambda \tag{4}$$

Solving for l_x gives

$$l_x = -p_x \cdot \frac{1}{\ln k} \cdot \frac{1}{\lambda} \tag{5}$$

Substituting the above equation into $f = \sum_{x=1}^{C} l_x - L = 0$ gives

$$\sum_{x=1}^{C} l_x = \sum_{x=1}^{C} (-p_x \cdot \frac{1}{\ln k} \cdot \frac{1}{\lambda}) = L \tag{6}$$

since $\sum_{x=1}^{C} p_x = 1$, we have

$$\sum_{x=1}^{C} (-p_x \cdot \frac{1}{\ln k} \cdot \frac{1}{\lambda}) = (-\frac{1}{\ln k} \cdot \frac{1}{\lambda}) = L \tag{7}$$

$$-\frac{1}{\lambda} = \frac{L}{\frac{1}{\ln k}} \tag{8}$$

By substituting the above equation back to Equation 5, we arrive at Theorem 2. ■

If we substitute $l_x = p_x \cdot L$ in Theorem 2 into Equation 1, then we get

$$H = \log N - \sum_{x=1}^{C} (p_x \cdot \log_k p_x) - \log_k L \tag{9}$$

Compared with the well-known square-root principle in unstructured P2P network, our replica placement strategy in structured P2P network arrives at a different proportional principle. In unstructured P2P with the random walk based search scheme, the average hop number to search c_x is proportional to the reverse of l_x, the number of n_i's replicas, while we achieve the average search cost as shown in Theorem 1. Hence for the optimization problem MAX_PERF, we derive the proportional principle.

Furthermore, in Equation 9, we find that $-\sum_{x=1}^{C} (p_x \cdot \log_k p_x)$ is in fact the entropy definition of p_x. Intuitively it makes sense since we have expected that our approach can accelerate the search for popular nodes, then the skew of the popularity distribution will play an important role in the optimization of the search performance, and taking the popularity as a probability function (of the query targets) entropy is a sound measure of the skew of the distribution. To our knowledge, no previous P2P work has mentioned such a relationship.

4.2.2 MIN_COST: Cost Minimization

In contrast to the problem of MAX_PERF, the optimization problem of MIN_COST is to minimize the total number of replicas, L, to achieve the targeted constant result τ for the average search cost H given in Equation 1. We derive the following theorem to solve the optimization problem of MIN_COST:

Theorem 3 *With the targeted constant result τ to make $H = \tau$, $L = \sum_{x=1}^{C} l_x$ is minimized when $l_x = p_x \cdot k^{(\log N - \tau - E_{p_x})}$, where E_{p_x} is the entropy of p_x with the value of $E_{p_x} = \sum_{x=1}^{C} (p_x \cdot \log_k p_x)$.*

Proof: Similar to the proof of Theorem 2, we use the Lagrange multiplier method to achieve the same result of l_x as Theorem 2:

$$l_x = -p_x \cdot \frac{1}{\ln k} \cdot \frac{1}{\lambda} \tag{10}$$

Then substituting the result of l_x by Equation 10 into $H = \tau$ can give:

$$H = \sum_{x=1}^{C} [p_x \cdot (\log N - \log_k \frac{-p_x}{\lambda \ln k})] = \tau \tag{11}$$

Since $\sum_{x=1}^{C} p_x = 1$, by the similar steps in the proof of Theorem 2 we have

$$-\lambda = k^{E_{p_x} + \tau - \log N - \log_k \ln k} \tag{12}$$

By substituting the result of λ into Equation 10, we arrive at Theorem 3. ■

The benefits of our optimal solution for MIN_COST is that the value of τ is independent of the value of node count N. In the very skewed case, the optimal allocation of l_x for content object c_x in Theorem 3 can achieve $O(1)$ average

search hops. Compared with Beehive [19], our optimal solution from Theorem 3 has the following the advantages: (1) the distribution of popularity p_i can be the general probability distribution, unlike Beehive which only gives the numeric result for the Zipf like distribution; (2) our optimal solution can be applied to the general structured P2P; while Beehive [19] is limited to the prefix matching based DHT like Pastry [21].

As in Theorem 2, Theorem 3 also give rise to the proportional principle where $l_x \propto p_x$, and the entropy term of p_x appears as well. Thus, the proportional principle and the relationship to the entropy of p_x are related to the optimal solutions to both MAX_PERF and MIN_COST in structured P2P.

5. OPTIMAL PROACTIVE WEB CACHING

In this section, we apply the results of PoPCache to a real Internet application, web caching, by proposing a P2P-based web caching scheme, which can optimally tradeoff the cost and performance gain. We shall call this web caching system Web-PoPCache.

5.1 System Architecture

The target environment of our system is a large scale cooperate network typically with 100 to 10,000 or more client desktop machines. All of the clients are assumed to access the Internet web server through a direct connection or through a firewall. In each node Web-PoPCache runs as a daemon program. There are three components in Web-PoPCache: proactive cache proxy, local cache store, and the underlying P2P operation unit(see Figure 2). The web browser in each node is configured to use the proactive cache proxy to access the web objects. The proactive cache proxy is responsible for: (1)intercepting the http request from the web browser; (2) caching the requested web contents, (3) optimally replicating the popular web contents based on the replica placement strategy; and (4) maintaining the consistency of local caches. The cache store, used to locally store the cached web contents, is limited to a fixed storage size and it uses an LRU cache replacement policy. Finally the P2P operation unit provides the get/put operations by finding the *home* node with node ID numerically closest to the hash ID of a web content URL. Note that web contents are typically of a reasonable size. In [25] there has been a study of the sizes of web contents of a real data set taken from the internet. In their study, the size of each HTML document was measured and for the entire data set, the mean size was 4.4KB, the median size was 2.0KB, and the maximum size was 1.6MB.

When a client node submits a http request, the client node itself, intermediate nodes, and home node can sever the http request when the local cache stores of the client node, intermediate nodes or the home node contain the replicas of the requested web contents. Even if no replicas of such contents are found, Web-PoPCache redirects the requests to the original web server. As a result, Web-PoPCache utilizes the underlying P2P overlay to cooperatively serve the http request with the replicated web contents. The key point of Web-PoPCache is to adopt the replica placement strategy of PoPCache (Section 4.1) to setup the caching location, and the proportional principle derived in Section 4.2 to place the replicas of the web content. Therefore, our system can optimally tradeoff the cost (i.e. the number of replicas) and the performance gain (i.e. average search hop number).

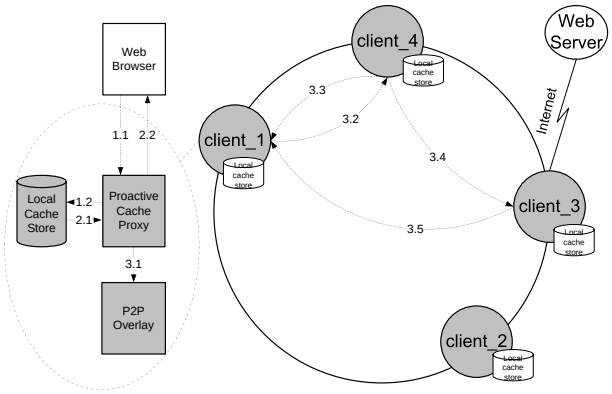

Figure 2: System Architecture of Web-PoPCache

Through Figure 2, we illustrate the use of Web-PoPCache as the following.

1. The client user sends the http request by submitting one URL via the web browser to access the web contents (1.1 in Figure 2). After intercepting the http request from the web browser, the proactive cache proxy checks the local cache store to determine whether a replica belonging to such a URL is available (1.2 in Figure 2).

2. If there is a replica in the local cache store, such content is directly returned to the client (2.1 and 2.2 in Figure 2);

3. Otherwise if no replica of such a URL exists , the proactive cache proxy redirects (3.1 in Figure 2) the http request to the P2P operation unit to lookup the home node i.e. *client_3* with a node ID closest to the hash ID of the URL. The http request is greedily forwarded towards *client_3*, but if the intermediated node *client_4* contains a replica for the request in its local cache store (3.2 in Figure 2), then the replica is returned to the requester (3.3 in Figure 2);

4. When all intermediate nodes towards home node do not contain the replica of such a URL, the request finally reaches home node *client_4* (3.4 in Figure 2), then Web-PoPCache will check whether a replica in *client_4* is available to serve the http request, and if so, the replica is directly returned to the requester (3.5 in Figure 2); otherwise, *client_4* will visit the original web server to fetch the requested content and return the content to the requester (3.4 in Figure 2).

5.2 Load Balancing

With no careful consideration of the load balancing scheme, the skewed popularity of web content request(i.e the well-known Zipf distribution) can aggravate the load of the node hosted with the most popular web contents. However, since the replicas of these popular web content are placed along the levels of neighbors based on the results of Section 4, the requests for such web contents can be terminated at the levels of neighbors, which reduce the workload. However, to handle a sudden burst of request for some popular contents, each node can set up a threshold for the query rate based

Traces	NLANR	BU
Time	Jan 9-10, 2007	Nov 1,1994- Jan 17, 1995
# Requests	189034	107578
# Contents	117765	16939
Total (GB)	2.66406	0.55371
Infinite Cache (GB)	2.0332	0.3945
Avg Requests per Content	1.60518	6.3500973
Max Requests per Content	1696	3328
Min Requests per Content	1	1
Avg content popularity (%)	.000849149	0.005902784
Max content popularity (%)	0.8971931	3.093569317
Min content popularity (%)	0.000529005	0.000929558
Hit ratio (%)	37.8239	72.5603748

Table 2: Statistics about the traces

on its capacity. Before the load of the client is out of its capacity, the client can progressively copy the requested web contents to its direct neighbors. Similar situation holds for these direct neighbors. As a result, the popular web contents are progressively replicated throughout the levels of nodes from its home node.

5.3 Popularity Estimation

Following the idea in [27], the total number of queries issued in the system in a time interval can be estimated by gossip based sampling protocol or [26] like deterministic protocol. The number of queries received at each local node n_i divided by the total number of queries is the popularity p_x of such a node. For each node, the number of received queries can be measured by counting at regular interval. However a sudden burst of queries can rapidly change the rate, hence we measure the query arrival time to respond to the change in query rate. Then, the popularity p_x can be estimated by the query rate against the total query rate.

6. EVALUATION

The evaluation includes two parts: (1) PART-A (Section 6.2): Comparison of our proposed cache placement strategy in Section 4.1 and several current cache placement strategies including CFS [9], Pastry [22](in spirit, [22] is a random strategy), and Beehive [19]; then further present the experimental results to evaluate our solutions for MAX_PERF and MIN_COST. For this part of the evaluation, we adopt the average search hops as the metric to evaluate the lookup efficiency. (2) PART-B (Section 6.3): Comparison of Web-PoPCache with Squirrel [12] to optimally tradeoff the better performance and load balancing against the cost. In this part, we adopt three main metric types: performance (request latency, external bandwidth and hit ratio), load balancing (query count per machine), and overhead (cache storage per machine).

The underlying structured P2P is implemented with a Chord [23] based DHT protocol. All the experimental results are obtained by a trace-driven event simulator. To model the skewed popularity distribution, we have downloaded several publicly available web request trace files. For the evaluation of PART-A, we directly store all distinct URLs that appeared in the web request trace files to the *home* nodes; based on the request count for each URL we calculate the request popularity for such URL, then apply the result of Section 4 to place the copies of URLs in the *delegate* nodes. Then to model the skewed request popularity, for each row of the trace file, one request for the URL that appeared in the row is sent to lookup the URL; For PART-B, we build the web cache application upon the underlying P2P simulator. Instead of statically allocated before the requests, the URL copies are progressively placed in its *delegate* nodes based on the incoming requests(see Section 5.2).

6.1 Traces

Table 2 lists the Web traces we have used for the performance evaluation.

- NLANR traces: NLANR (National Lab of Applied Network Research) provides sanitized cache access logs in the public domain [2]. We have used 2 days' traces of Jan 9, 2007 and Jan 10, 2007 from the "bo", "pa", "sd" and "uc" proxies.
- BU traces: Boston University collected traces from a similar computing facility and user population in 1995 and 1998, which can be found in [3]. We have selected the traces in a period of 2 months in two different years, which are denoted by BU-95 and BU-98, respectively.

To clearly present the distribution of requests, we respectively show the count of requests per URL ordered by the ranking of popularities of the trace file NLANR(pa.sanitized-access.20070109) and trace file BU(the contents in subdirectory condensed/272 of BU-www-client-traces.tar.gz) in Figure 3 and Figure 4. The traces can be publicly downloaded from IRCache [2] and BU Trace [3]. The ranking sequence presented in the x-axis of Figure 3 is ordered in a descending manner. To plot the points with a zero hit, we add the original requests by 1 so that the points with zero value can be shown on the y-axis with log-scale. From this figure, we can find that the requests count basically follows the well-known Zipf like distribution. Furthermore, based on the definition of entropy $E_{p_x} = -\sum_{x=1}^{C}(p_x \cdot \log_k p_x)$, for some value of $k = 7.6$ we compute the entropy for NLANR trace file and BU trace file:6.2579 and 2.7637. Since entropy is used to measure the randomness of the popularity distribution, we can find the popularity distribution of BU trace file is more skewed than NLANR trace file. In addition, the larger value of k also reduces the entropy of popularity distribution.

6.2 Evaluation for the Analysis Model

6.2.1 Comparison of placement strategy

We compare the placement strategy of PoPCache with other three strategies: the CFS strategy [9], Beehive strategy [19] and the random strategy. Following the strategy described in CFS [9] which is a cooperative file system based on Chord [23], we place replicas along the lookup path towards the destination. For random placement, which essentially is the strategy taken by Pastry [21], replicas are placed at randomly chosen nodes. In the experiment setting, $N = 4000$ and $k = 8$. For each web content uniquely identified by a URL, we adopt the proportional principle to assign the number of replicas based on p_x and place the replicas by four strategies as described above. By varying the total number of replicas, we measure the average lookup hops H for each strategy.

Figure 5 plots the average lookup hops for the above four cases by varying the value of D, it can be found that PoPCache outperforms other three strategies. This can be explained that: (1) the PoPCache placement achieves the least average lookup hops because the replicas placed in the the k-ary tree $\mathcal{T}_i$ can benefit searching from any source nodes;

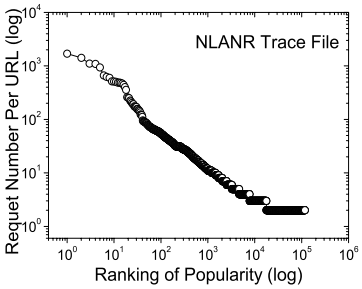

Figure 3: Requests By URL in NLANR trace

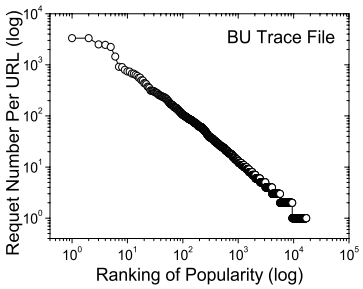

Figure 4: Requests By URL in BU trace

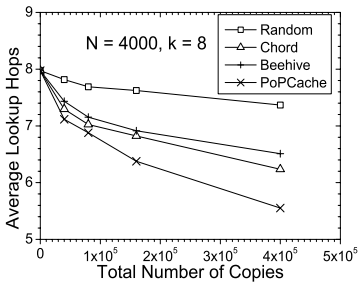

Figure 5: Comparison of Placement Strategy

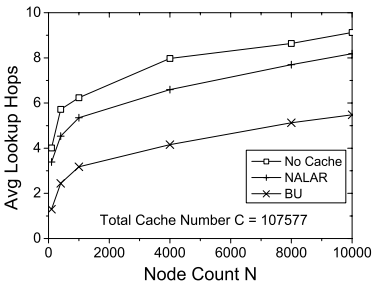

Figure 6: Maximization Performance

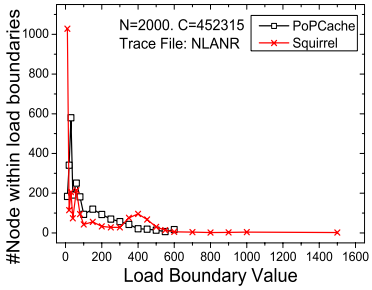

Figure 7: Load Distribution for NLANR Trace

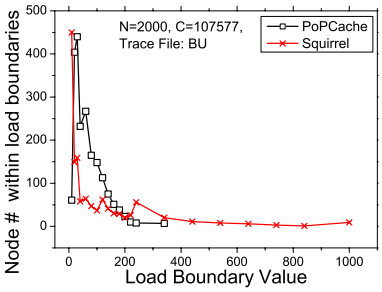

Figure 8: Load Distribution for BU Trace

(2)in the Chord or random strategy, only the searches from some specific source nodes can be accelerated by the *alink*s; (3) for Beehive [19], only roughly half of the cache copies can be helpful to the lookup due to the limitation that it is target for only the prefix based DHT.

6.2.2 MAX_PERF

In Figure 6 we compare the our proposed optimal solution for MAX_PERF under two trace files NLANR and BU, we also plot the results of no caching in Figure 6. By no caching we mean that no replicas is utilized in PoPCache. Theoretically the adoption of the proportional principle in PoPCache can achieve less lookup hops for the same number of replicas. From this figure, we can find that the average lookup hop number for trace file NLANR is larger than that for trace file BU. This result can be consistently explained by Equation 9 in Section 4.2.1, because the entropy of p_x for popularity in trace file NLANR is larger than that in trace file BU.

6.2.3 MIN_COST

Figure 9 plots the total copy number C with the goal to satisfy the target lookup hops τ respectively for $N = 800$ and $N = 8000$. Naturally, to satisfy a smaller τ, more copies will be consumed. From Figure 9, for $N = 8000$, to achieve less than 1 hops from $\tau = 8$ to $\tau = 7$, about 4.5×10^7 more replicas are needed; while to achieve below 1 hops from $\tau = 3$ to $\tau = 2$, about 3.1×10^9 more replicas are needed: the same reduction of the average lookup hops consumes nearly 100 times of replicas. Similar situation holds for $N = 800$, though when N grows from 800 to 8000, to achieve the average lookup hops H, more replicas are required for N=8000 than N=800. Also due to the different entropy values for BU trace and NLANR trace, more replicas are consumed for NLANR trace than BU trace for both N=800 and N=8000. This experiment provides the guideline for system administrator to tradeoff the performance gain and cost.

6.3 Evaluation for the Web Cache Application

In this section, we evaluate Web-PoPCache for cooperative web caching in terms of the load balancing, the consumed external bandwidth and hit ratio. Each participated node in Web-PoPCache is assumed to be a desktop machine with similar capacity. For simplicity, Web-PoPCache adopts the LRU policy.

6.3.1 Load Balancing

For a node n_i, we set the load of n_i, L_i, to be the the number of incoming http requests with URLs that are served by n_i. Figures 7 and 8 respectively plot the load distribution of NLANR and BU trace data for Web-PoPCache and Squirrel [12]. The x-axis of Figure 7 and Figure 8 present the load (number of requests to server in each node) boundary values, where the left most point on the x-axis corresponds to a load (request count in some node) of less than 10 both for the NLANR and BU trace, and the right most point refers to a load larger than 1500 and 1000 respectively for the NLANR and BU trace; and y-axis represents the count of all nodes with load L_i within the load boundary values.

From Figures 7 and 8, we can find that the load distribution of Squirrel [12] is heavily tailed and is consistent with the Zipf distribution given in Figures 3 and 4. It is because Squirrel [12] does not consider the problem of load balancing. For example in Figure 8, the rate of nodes with load within [10,250] among all nodes is 98.039%. When overloading is considered, for one threshold load value(=250 in Figure 8), 7.698% of the nodes in Squirrel jave loads greater than the threshold value and they have to serve 49.885% of the requests! While in Web-PoPCache, only 0.35% of the nodes exceed the load threshold and they only serve 1.862% of the requests. It is found that Web-PoPCache can effec-

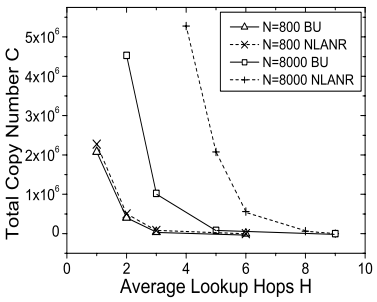

Figure 9: **Minimization Cost for BU Trace File**

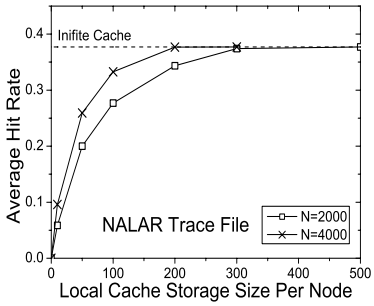

Figure 10: **Average Hit Rate for NLANR Trace**

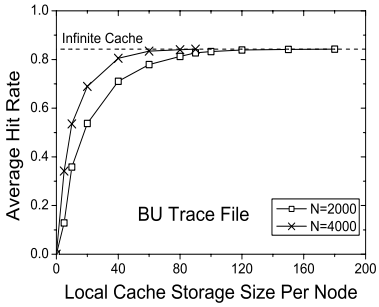

Figure 11: **Average Hit Rate for BU Trace**

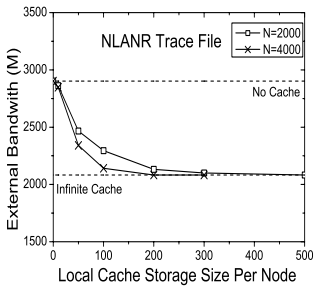

Figure 12: **External Bandwidth for NLANR Trace**

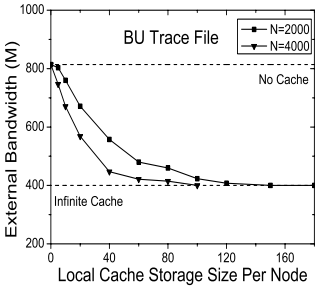

Figure 13: **External Bandwidth for BU Trace**

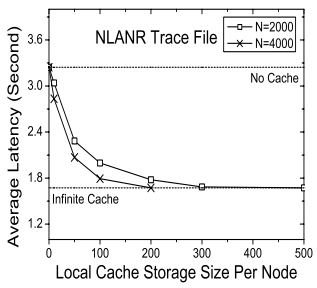

Figure 14: **Average Latency for NLANR Trace File**

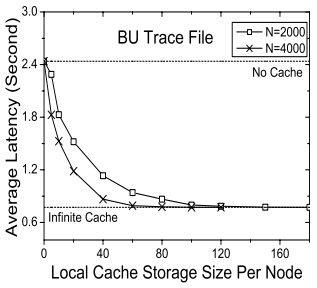

Figure 15: **Average Latency for BU Trace**

tively balance the load. A similar situation holds for the NLANR trace file seen in Figure 7.

6.3.2 *Hit ratio and external bandwidth*

We calculate the external bandwidth as the bytes transferred between Web-PoPCache and the external origin servers. In this section, we compare Web-PoPCache with Squirrel [12] on the hit ratio and external bandwidth savings by measuring the impact of the per-node cache storage size. Since the cache replacement strategy is implemented with LRU algorithm, the rarely requested contents in the local cache store are always replaced with the new incoming popular contents which are more frequently requested to be placed in the local cache store. Obviously the larger storage size of local cache store can accommodate more replicas and can increase the hit ratio; also a large value of node count N can provide more nodes to accommodate the replicas and can also increase the hit ratio. Both Figures 10 and 11 show the impact of local cache storage size and node count N. When N grows, the total size of all local storage in PoPCache is also increased. As a result, the hit rate grows. Since the data size varies from web object to web object, the x-axis in Figures 10 and 11 is represented as the cache storage size (i.e. size of the priority queue to implement the local cache store), instead of the total data size of cached web contents. In our experiments, the first time to request a URL always is required to visit the original web server outside the Web-PoPCache, as a result, the infinite hit ratio appears in Figure 10 and Figure 11 is different in that of the computed hit ratio in table 2 where the downloaded trace file only records the requested URL within some interval, and this could miss some of the first-time requests.

Since the external requests for the original web server are produced when the requests for some URL are missed in the Web-PoPCache, the less hit ratio means that more external bandwidth will be consumed as plotted in Figure 12 and Figure 13.

6.3.3 *Request Latency*

Since Web-PoPCache is targeted to a large corporate intranet environment by connecting a large number of desktop machines to provide the cooperative web caching. The communication latencies between the desktop machines within the Web-PoPCache are of the order of a few milliseconds, while the latencies between one desktop machine within the Web-PoPCache and the external web severs are at least an order of magnitude larger than the internal latency. Consequently, the request latency between issuing the HTTP request and receiving the response is determined by two factors: (1) hit ratio; (2) the external latency. If the HTTP requests are hit by the cached web objects placed in Web-PoPCache, the request latency is within several milliseconds; if missed, the request latency is dominated by the external latency. Figures 14 and 15 respectively plot the average request latency for NLANR trace and BU trace.

It can be found that the trends of the average latency in Figure 14 and Figure 15 follows that of Figure 12 and Figure 13, and are verse to that of Figure 10 and Figure 11. From Figure 10, 11, 12, 13, 14 and 15, we can demonstrate the benefits achieved by Web-PoPCache with increased hit rate, reduced bandwidth and smaller request latency.

6.3.4 *Fault Tolerance*

Since the nodes in Web-PoPCACHE could leave or fail suddenly, the replicated web contents in those nodes will be lost. In this experiment, we assume the nodes fail randomly with a given failure rate. Figure 16 shows the effect of hit ratio by node failure rate. From this figure we can find that the hit ratio is non-nearly reduced as the node failure ratio grows. It can be explained as following. Due to the proportional replication principle, the more popular contents are placed to more nodes and less popular contents

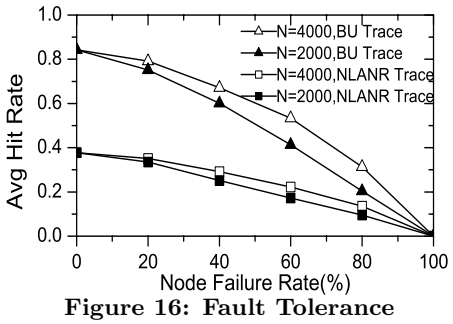

Figure 16: Fault Tolerance

are placed to less nodes. With a given rate of node failure, the more popular contents can survive with a higher probability than the less popular contents, i.e. the hit ratio for popular contents is higher than the hit ratio of un-popular contents. Simultaneously, the http requests are issued by with skewed popularity, the high popular contents receive more request than those less popularity. As a result, the hit ratio curve vs node failure rate is the convex shape. Moreover, we can also find that with the same node failure ratio, the hit ratio for N=4000 is higher than the hit ratio for N=2000. For N=2000, each node are placed with more contents than N=4000; and more contents are lost due to the node failure. Consequently, the larger number of nodes in Web-PoPCACHE can help the high hit ratio under node failure. The reason that we chose hit ratio as the measure metric is that when the failure happens in the intermediate lookup path towards the destination node instead of no failure of the destination node, more hops will be consumed but no effect of the hit ratio, and the replicated web contents can be always found in the destination node. As a result, the missed requests caused by node failure will visit the original web servers by Internet, and then we focus on the effect of hit ratio caused by node failure.

7. CONCLUSION

In this paper, with consideration of the skewed popularities of data contents in a P2P environment, we propose a proactive caching method PoPCache with a novel placement strategy based on the closed form solutions for two optimization objectives: MAX_PERF and MIN_COST. In the optimal solutions, there are the following interesting properties: (1) the copy number assigned to each specific content is proportional to its popularity, deviating from the well-known square-root result achieved in unstructured P2P; (2) the achieved optimal results are related to the entropy of popularity. Next based on these results we propose an optimal proactive web caching scheme, Web-PoPCache, which can optimally tradeoff the performance gain and cost. By means of web cache trace driven simulations, our extensive evaluation results demonstrate the advantages of PoPCache and Web-PoPCache.

8. REFERENCES

[1] In *http://squid.nlanr.net*.

[2] In *ftp://ircache.nlanr.net/Traces/DITL-2007-01-09*.

[3] In *http://ita.ee.lbl.gov/html/contrib/BU-Web-Client.html*.

[4] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-grid: a self-organizing structured p2p system. In *SIGMOD Record*, volume 32, 2003.

[5] S. Abiteboul, A. Bonifati, G. Cobena, I. Manolescu, and T. Milo. Dynamic xml documents with distribution and replication. In *SIGMOD Conference*, pages 527–538, 2003.

[6] J. Aspnes and G. Shah. Skip graphs. In *SODA*, 2003.

[7] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *SIGCOMM*, 2002.

[8] B. F. Cooper. An optimal overlay topology for routing peer-to-peer searches. In *Middleware*, 2005.

[9] F. Dabek, M. F. Kaashoek, D. R. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *SOSP*, 2001.

[10] F. Douglis and T. Ball. Tracking and viewing changes on the web. In *USENIX Annual Technical Conference*, pages 165–176, 1996.

[11] L. Fan, P. Cao, J. M. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. In *SIGCOMM*, pages 254–265, 1998.

[12] S. Iyer, A. I. T. Rowstron, and P. Druschel. Squirrel: a decentralized peer-to-peer web cache. In *PODC*, pages 213–222, 2002.

[13] H. V. Jagadish, B. C. Ooi, and Q. H. Vu. Baton: A balanced tree structure for peer-to-peer networks. In *VLDB*, 2005.

[14] D. R. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi. Web caching with consistent hashing. *Computer Networks*, 31(11-16):1203–1213, 1999.

[15] P. Linga, I. Gupta, and K. Birman. Kache: Peer-to-peer web caching using kelips. In *In submission*, June 2004.

[16] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *SIGMETRICS*, 2002.

[17] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *USENIX Symposium on Internet Technologies and Systems*, 2003.

[18] V. Ramasubramanian and E. G. Sirer. Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays. In *NSDI*, 2004.

[19] V. Ramasubramanian and E. G. Sirer. The design and implementation of a next generation name service for the internet. In *SIGCOMM*, 2004.

[20] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A scalable content-addressable network. In *SIGCOMM*, 2001.

[21] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, 2001.

[22] A. I. T. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *SOSP*, 2001.

[23] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, 2001.

[24] J. Wang. A survey of web caching schemes for the internet. In *ACM Computer Communication Review*, 1999.

[25] A. Woodruff, P. M. Aoki, E. Brewer, P. Gauthier, and L. A. Rowe. An investigation of documents from the www. In *In Proceedings of the Fifth International WWW Conference*, page 963âV979, 1996.

[26] P. Yalagandula and M. Dahlin. A Scalable Distributed Information Management System. volume 34, pages 379–390, New York, NY, USA, 2004. ACM Press.

[27] V. R. Yee Jiun Song and E. G. Sirer. Optimal resource utilization in content distribution networks. In *Cornell University, Computing and Information Science Technical Report TR2005-2004, Ithaca, New York*, November 2005.

[28] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph. Tapestry: a fault-tolerant wide-area application infrastructure. volume 32, 2002.

[29] M. Zhong and K. Shen. Popularity biased random walks for peer-to-peer search under the square root principle. In *IPTPS*, 2006.