Optimal Resource Placement in Structured Peer-to-Peer Networks

Weixiong Rao,Lei Chen, *Member, IEEE,* Ada Wai-Chee Fu, *Member, IEEE,* and Guoren Wang

Abstract—Utilizing the skewed popularity distribution in P2P systems, common in Gnutella and KazaA like P2P applications, we propose an optimal resource (replica or link) placement strategy, which can optimally tradeoff the performance gain and paid cost. The proposed resource placement strategy, with better results than existing works, can be generally applied in randomized P2P systems (Symphony) and deterministic P2P systems (e.g. Chord, Pastry, Tapestry, etc). We apply the proposed resource placement strategy respectively to two novel applications: PCache (a P2P-based caching scheme) and PRing (a P2P ring structure). The simulation results as well as a real deployment on Planetlab demonstrate the effectiveness of the proposed resource placement strategy in reducing the average search cost of the whole system.

Index Terms—Peer-to-Peer (P2P) Network, Distributed Hash Table (DHT), Popularity, Placement, Cache, Topology

1 Introduction

A Peer-to-Peer (P2P) system is a promising new platform that has the potential to support information exchange at a very large scale. Existing P2P systems, such as Gnutella and KazaA, also known as *unstructured P2P*, connect millions of machines to provide large-scale file sharing services. Similarly, *structured* P2P systems, such as Chord [28], Pastry [26], Tapestry [32] and CAN [23], are used for file-sharing or other large-scale applications. In these structured P2P systems, each peer and its stored content are structurally organized using a Distributed Hash Table (DHT).

For file-sharing, depending upon the contents of shared files, some files are highly popular; while some are rarely accessed. It means that *data popularities*, measured with respect to the proportion of the submitted queries that can be satisfied by the data contents [19], [9], [33], are typically skewed. In general, the distributions of data popularity in file sharing and many other applications are often non-uniform [19], [9], [33]. For example, web requests on the Internet space are found to be highly skewed with a Zipf-like distribution [6].

Most previous approaches consider the skewed popularity as a *challenge*, since popular objects create excessive workloads and result in the overloading problem. In this paper we consider the skewed popularity as an *opportunity* and not just a *challenge*: if we can make use of this popularity distribution, there can be substantial reduction on the search cost for popular contents, which may minimize the average search cost of the entire dataset. Of course, in order to reduce the average search cost, we have to make use of the available resources. In different applications, resources to be placed in a network can have various meanings. For example, in a content distribution network (CDN), the resources refer to the replicas of popular web objects; in unstructured P2P applications like Gnutella, the pointers or links among peers can be considered as resources; in World Wide Web (WWW), the hyper links among web sites can be also treated as a kind of resources. In this paper, we focus upon the center problem in structured P2P networks: *how to utilize the resources to reduce the search cost which is measured by the average lookup hops*? Here the average lookup hops can be treated as the performance requirement, termed as $\mathcal{P}$; in order to achieve the target performance $\mathcal{P}$, resources, denoted by $\mathcal{R}$, are consumed. $\mathcal{R}$ can be replicas, or links, etc. A key problem related to the resource placement strategy is where to place the resources $\mathcal{R}$ in structured P2P. To answer the *where* problem, we propose to place the resources $\mathcal{R}$ at the special *acceleration nodes* so that the search can be accelerated by those acceleration nodes.

We apply such a placement strategy to PCache (A Popularity-based Caching Approach in P2P) where replicas are taken as the resources to accelerate the search for popular contents. Then we propose solutions to the question of *"how many replicas are needed?"* based on the following two optimization criteria: (1) *MAX_PERF*: given a constant number of replicas, how to minimize the request lookup hops (i.e. maximize the performance gain)? (2) *MIN_COST*: given a targeted threshold of the request lookup hops, how to minimize the amount of replicas (i.e. minimize the paid cost)? In this paper, we give closed-form solutions for both optimization criterion.

When the links among peers are treated as resources and they are placed at acceleration nodes to accelerate the search for popular objects, we can achieve similar performance as PCache. Furthermore, when the total number of links in a structured P2P network, as a constraint, is set to $N \log N$, comparable with the number of total links in existing regular ring structures (Chord [28] or Symphony [20]), our proposed optimal ring structure, PRing, can achieve better results than Chord or Symphony do.

To sum up, we make the following contributions in this work:

- We propose a novel resource placement strategy for improving the search efficiency. From experiments, our strategy can achieve better results than current heuristic approaches (e.g. CFS [10] and Pastry [27]) and an approximate approach (Beehive [21]);
- We provide closed-form solutions for the two problems

TABLE 1
Meanings of Main Symbols Used

Symbol	Meaning
c_x	x-th object
n_i	i-th node
p_x	popularity of object c_x
l_x	number of replicas assigned for c_x
N	total number of nodes
L	total number of links (alinks and llinks)
M	total number of objects
R	total number of replicas
H	average number of hops to find all M objects
H_x	average cost to search c_x in number of hops
p_i	popularity of node n_i
ℓ_i	number of alinks assigned for n_i
$\mathbb{L}$	total number of alinks
$\mathbb{D}$	total number of links (alinks and llinks)
$\mathbb{H}$	average number of hops to find all N nodes
$\mathbb{H}_i$	average number of hops to search node n_i
k	number of *long links* in Symphony
γ	acceleration ratio

in PCache, MAX_PERF and MIN_COST, with similar results in both randomized P2Ps and deterministic P2Ps;
- We propose a P2P structure, PRing, which can achieve less average lookup hop number than Chord and Symphony with the equal total number of links.

We remark three particularly interesting results as follows:
- Our replica placement strategy can be generally applied to structured P2P networks including randomized P2P networks (e.g. Symphony, etc.) and deterministic P2P networks (e.g. Chord, Pastry, etc.);
- The optimal number of resources (replicas in PCache and links in PRing) is found to be *proportional* to the popularity. It is known that for unstructured P2Ps, the random walk-based technique is optimized by the *square-root* principle [8], [19], [33], [9]. However, here we arrive at a different optimal function of the popularity for structured P2P systems;
- Our analysis in structured P2P systems shows that the average number of search hops in a structured P2P system is related to the *entropy* of popularity. Intuitively this 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. 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 explains the general resource placement strategy. Section 3 and Section 4 respectively present PCache and PRing. Section 5 evaluates the proposed strategy and applications. In Section 6 we summarize the related work in this area. Finally, we conclude in Section 7.

2 Resource Placement Strategy

2.1 Structured Peer-to-Peer

Symphony: In Symphony [20], each node is assigned a uniform real number id within [0,1) and manages the key range corresponding to the segment on the unit ring as defined by its own id and that of its immediate clockwise predecessor. In each node there are two types of links: two *short links* connected with its immediate neighbors (i.e. anti-clockwise predecessor and clockwise successor), and k *long links* constructed by a Harmonic distribution to connect remote nodes, so that the probability that two nodes are connected by the long link is inversely proportional to their distance on the ring. To search a node with a particular key value y starting from node n_i, n_i first checks whether the target value y is inside its, local key range. If so, node n_i is the target node; otherwise, node n_i selects a link, among all its long links, whose other endpoint n_t has the closest key range towards the target value y; then the search request is forwarded to node n_t and node n_t performs a similar operation to get to y. Through this *greedy* algorithm, the target node will be found if it exists; otherwise NULL will be returned. With k long links constructed based on the Harmonic distribution in each node, the average search cost in Symphony is $O(\frac{1}{k} \cdot \log^2 N)$.

Chord: Chord [28] assigns each node an m-bit identifier using a hash function such as SHA-1. Node Identifiers are ordered in an identifier circle modulo 2^m. When each node is only aware of its *successor* node on the circle, queries for a given identifier can be passed around the circle via these successor pointers until they first encounter a node that succeeds the identifier; this is the node the query maps to. A portion of the Chord protocol maintains these successor pointers, thus ensuring that all lookups are resolved correctly. However, this resolution scheme is inefficient: it may require traversing all N nodes to find the appropriate mapping. To accelerate this process, Chord maintains additional routing information, i.e. m entries in the *finger table*. The i^{th} entry in the finger table at node n contains the identity of the first node, s, that succeeds n by at least 2^{i-1} on the identifer circle: i.e. $s = successor(n + 2^{i-1})$, where $1 \leq i \leq m$. With the finger table, the number of nodes that must be contacted to find a successor in N-node Chord network is $O(\log N)$.

2.2 Where to Place the Resources

With the introduction of Symphony and Chord in Section 2.1, we may find that both the long link in Symphony and the finger table in Chord are used to speed the lookup; otherwise the lookup in Symphony or Chord will walk along the circle node by node until the destination is found. Thus, without long links or finger tables, the lookup complexity is $O(N)$ where N is the total number of nodes. Considering the short link in Symphony and the connection between one node and its successor in Chord that are used to construct the base overlay structure, we intuitively call these links as the *base overlay link* (in short, overlay link). Based on such overlay link, to facilitate the introduction of the proposed resource placement strategy, we first define the *base overlay distance* as follows:

Definition 1 *(Base overlay distance) The base overlay distance from node n_i to n_j in a structured P2P, denoted as w_{ij}, is the number of hops from node n_i to reach n_j in the P2P overlay network connected only by the base overlay link.*

Take Symphony in Figure 1 as an example, the base overlay distance from r_i to n_i (in this figure, r_i is the predecessor of n_i) is 1 hop, and the base overlay distance from n_{t0} to n_i is 4 hops because there are 4 short links from n_{t0} to n_i. Note that if we consider the help of long links, the lookup from n_{t0} to n_i consumes only 2 hops: one hop from n_{t0} to r_i, and another hop from r_i to n_i. Similar situation holds for Chord.

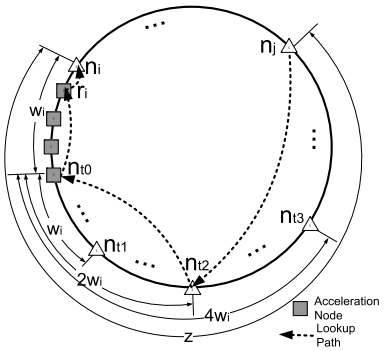
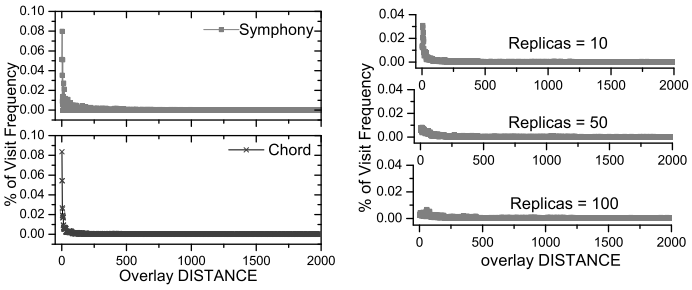


Fig. 1. Placement Strategy in Symphony

To clearly understand the definition of overlay distance, we first conduct an experiment as follows. Given a P2P network Symphony (or Chord) with N nodes $n_1, ..., n_N$, we fix a node, for example n_i, as the destination node with $1 \leq i \leq N$. By the P2P lookup algorithm, which utilizes both long links (or finger tables) and short links, for any $1 \leq j \leq N$ with $j \neq i$, a search is initiated the source node n_j to the destination n_i. The search may intermediately visit $O(\log N)$ nodes. For each $j \in [1, n]$ with $j \neq i$, we conduct total $(N-1)$ lookups and record all intermediately visited nodes from n_j to n_i. Among these intermediately visited nodes (except n_i itself), we count the visit frequency of each distinct intermediate node, denoted as n_t. This experiment illustrates a special scenario: the objects stored in destination n_i have the largest popularity $p_x = 1.0$ and objects stored in other nodes n_j having popularity $p_x = 0.0$. Thus all searches target a destination n_i with no search towards other nodes n_j.

We conduct the above experiment in Chord and Symphony, respectively. Figure 2(a) plots the frequency to visit an intermediate node n_t with regards to the overlay distance from n_t to n_i for $N = 2000$. Here, we sort the overlay distance of x-axis by an ascending order and study the relationship between the overlay distance and the visit frequency. As shown in Figure 2(a), those nodes with *smaller overlay distance* have *higher* visit frequencies. When considering the cumulative visit frequency, we find for the top 10 (i.e. 0.5%) nodes with the closet overlay distance to n_i, the cumulative visit frequency rate in Symphony and Chord is 11.49% and 16.71%, respectively.

In Figure 2(a), among all intermediate nodes n_t during the search from n_j to n_i, a higher frequency of n_t means n_t, having a smaller overlay distance to n_i, is more possible to intermediately *intercept* the lookup to n_i. Thus, if we consider placing resources $\mathcal{R}$ (like replicas or links) in the n_t, then with a high probability, n_t can intermediately intercept the searches before the search reaches the destination n_i. Hence, the resources in those intermediate nodes n_t can help answer the lookup towards n_i. For example when resources $\mathcal{R}$ are



(a) Before Replication (b) After Replication

Fig. 2. Resource Placement Strategy

replicas, the lookup for the object in n_i can be directly answered by the replicas placed in n_t; and the search hop numbers can be reduced as there is no need of continuing the remaining search towards n_i. When resources $\mathcal{R}$ are links pointing from n_t to the destination node n_i, the search can directly jump from n_t to n_i by only one hop, instead of continuing the remaining routing process. Now we derive a resource placement strategy as follows.

Definition 2 *(Resource Placement Strategy) Given l resources to be assigned to a node n_i, the resources are placed at the nodes with the l nearest overlay distances to n_i. These nodes are called the acceleration nodes of n_i.*

With the above strategy, the acceleration nodes can reduce the search hop numbers towards the destination node. It makes sense because the routing in structured P2P is a greedy algorithm, and when any routing method approaches the destination, the distance to the destination is smaller. In this situation, our acceleration nodes, just with smaller distances before the destination, can intercept the routing towards the destination. Taking the node n_i of Figure 1 as an example, l acceleration nodes are those nodes which continuously precede n_i along the ring. In details, the 1-st acceleration node of n_i is r_i, i.e. the predecessor of n_i; the 2-nd acceleration node of n_i is the predecessor of r_i; ... the l-th acceleration node of n_i is the predecessor of the $(l-1)$-th acceleration node of n_i.

To show the effect of acceleration nodes, we repeat the above experiment after the objects in n_i are replicated to the acceleration nodes of n_i, with one replica to each of the acceleration nodes. Figure 2(b) plots the experimental results with Symphony as the underlying P2P network (due to the similar results, we do not plot the results for Chord). Clearly, the results show that the placed replicas can reduce the visit frequency. When more replicas are placed, the distribution of visit frequency becomes more even. In the case that 2000 replicas are used, all lookups towards n_i will be answered by the local replicas, without visiting any other peers, and the distribution of visit frequency in Figure 2(b) becomes uniform. By this experiment, we can find that acceleration nodes can effectively intercept the lookups towards the destination n_i, and the replicas placed in acceleration nodes can directly answer the lookups for the contents in n_i, instead of continuing the remaining lookups towards n_i.

Besides the above empirical study, we will formally, in Section 3, show how these replicas placed in acceleration

nodes can improve the search efficiency, and further extend the placement strategy to a Popularity-based P2P Ring Structure (PRing), which is given in Section 4.

3 PCache: Popularity based P2P Web Caching

In this section we analyze PCache both in the randomized structured P2P system (Section 3.2) and in the DHT-based determinants structured P2P system (section 3.3), finally present the PCache application architecture.

Though web caching has been extensively studied in the literature, most works are related to the caching in a passive manner, instead of a proactive manner. The key point for the proactive caching in a P2P network is *where* and *how* to proactively place the copies of web objects. When we use replicas being resources $\mathcal{R}$, we can apply the proposed resource placement strategy in Section 2 to place the copies of web objects in the acceleration nodes as the solution of the *where* problem. Next based on the placement strategy, we propose the popularity-based optimal caching approach to tradeoff the consumed resource $\mathcal{R}$ and the achieved performance $\mathcal{P}$ measured by the average lookup hops.

3.1 Background of Web Caching

As an Internet application, web caching is widely used to reduce the object request latency, to decrease the amount of aggregate network traffic, and to balance the workload by distributing the heavy workload of 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 as follows: (i) in the local directory of the *client side*; (ii) 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); (iii) at the intermediate *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 CDN: content delivery network).

Web objects 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 objects which are *already* requested. This kind of caching is typically called *passive caching* at client side. On the other hand, *proactively caching* means that the replicas of web objects are *proactively* cached by the original web servers and the server side CDN to achieve the goals of improved performance and balanced workload. The proactively caching technique is widely utilized for the object providers like Google or the third-party CDN provider like Akamai. Though supporting load balancing and having better performance (e.g. reduced request latency and decreased bandwidth consumption), such caching involves high 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.

P2P technology is an attractive technology to avoid the expensive cost of the server side caching and CDN. By connecting a large number of volunteered nodes with low costs (for example desk top machines), P2P technology can be used to construct a cooperative web caching system. Squirrel [16] is an example of such a cooperative web caching system and it shares the local contents to form an efficient and scalable web caching. However, there is no consideration about the popularity skewness of the contents in Squirrel. Consequently, the overall workload of Squirrel is still unbalanced.

3.2 PCache in Symphony

As a proactive caching approach, PCache proactively replicates the copies of web contents in order to reduce the average lookup hops in P2P based Web Caching systems (e.g. Squirrel [16]). On the other hand, the more copies of web contents are replicated and fetched throughout the P2P network, more cost including message volume and content storage will be consumed. As a result, we use the copy number of web contents to explicitly indicate the consumed cost.

By the proposed strategy in Section 2, PCache proactively places the copies of popular contents in acceleration nodes: when some popular content c_x originally stored in the home node n_i, copies of c_x are proactively replicated in the acceleration nodes of n_i. Furthermore, PCache gives the solution about how many copies of c_x should be replicated in the acceleration nodes of n_i. When more replicas of c_x are used, more acceleration nodes of n_i will be used to store replicas. In order to optimally tradeoff the performance $\mathcal{P}$ and the consumed resource $\mathcal{R}$ (i.e. replicas), we give an optimal solution about how many replicas are created for c_x based on the popularity of c_x, denoted as p_x. The following sections show our optimal results in Symphony, Chord and other DHT based structured P2P networks, respectively.

3.2.1 Acceleration Theorem

Now we formally analyze the effect of acceleration nodes in Symphony. For a content c_x with popularity p_x, we call the node n_i where c_x is stored in P2P the *home node* of c_x. Suppose c_x is assigned with l_x replicas, we place l_x replicas in l_x acceleration nodes, i.e. one replica of c_x in each acceleration node. The following theorem shows the benefit of our placement strategy in Symphony by reducing the average number of hops to lookup c_x.

Theorem 1 *Suppose Symphony assigns k long links at each node and assigns l_x replicas for object c_x, then the average number of hops to search object c_x, $H_x = O(\frac{\log N/l_x}{k} \cdot \log N)$, if $l_x \geq 1$.*

Proof: We adopt some similar arguments as in Symphony [20] for our proof. Since each *long link* in Symphony is constructed by the pdf $\frac{1}{x \ln N}$, then the probability for any source node n_j having a *long link* to cut the distance

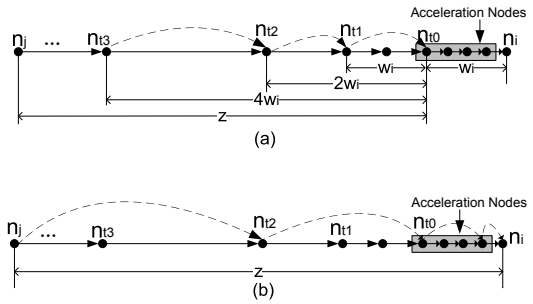


Fig. 3. Acceleration Nodes in (a) Symphony (b) Chord

$z \in [1/N, 1)$ between n_j and a destination node at least by half, denoted as p_{half}, is given by

$$p_{half} = \int_{z/2}^{z} \frac{1}{x \ln N} dx = \frac{1}{\log_2 N}$$

Note that p_{half} is independent of the value of z. If we treat the event of halving the distance as a series of trials with probability p_{half}, the expected number of trials before the event successfully occurs is the expectation of a *geometric* random variable: $1/p_{half} = \log_2 N$, i.e. on average $\log_2 N$ *long link*s are required to cut the distance by at least half. Since each node in Symphony is assigned k *long link*s, then on average the search of $O(\frac{\log_2 N}{k})$ nodes, i.e. $O(\frac{\log_2 N}{k})$ hops, are needed before the search from the source node n_i with distance z to the destination node arrives at any node with distance at most $z/2$ to the destination node.

Now we consider the effect of acceleration nodes to the average lookup hops. For simplicity, we transform Figure 1 to Figure 3(a), and assume that n_j is the source node and n_i is the destination node. All of l_x nodes between n_{t0} and n_i, except for n_i itself, are the acceleration nodes of n_i. Given the unit circle of Symphony with N nodes, the distance from n_{t0} to n_i is $w_x = l_x/N$.

Given node n_{t1} with a (base overlay) distance to n_i equal to $2w_x$, based on the definition of p_{half}, the probability for n_{t1} to have a *long link* pointing to one of l_x acceleration nodes of n_i (i.e. the nodes from n_{t0} to n_i) is p_{half}. Meanwhile, for any node with a distance to n_{t0} *smaller* than w_x, i.e. any node located between n_{t1} and n_{t0}, the probability for such a node having a *long link* to any acceleration node of n_i is *bigger* than p_{half}. Thus, for a node between n_{t1} and n_{t0}, the expected number of *long link*s used to reach one of acceleration nodes of n_j is at most $\frac{1}{p_{half}} = \log_2 N$, as the expected value of a *geometric* random variable. Since each node is assigned k *long link*s, the average number of hops to one acceleration node of n_i, starting from any node between n_{t1} and n_{t0}, is at most $O(\frac{\log_2 N}{k})$. After the search reaches such acceleration node, the replicas of c_x in the acceleration node can directly answer the search for c_x, instead of continuing the remaining search for c_x in node n_i. Thus, the average number of lookup hops from one of the nodes between n_{t1} and n_{t0} is $O(\frac{\log_2 N}{k})$.

Similarly, for node n_{t2} with a distance $2w_x$ to n_{t0}, the expected number of hops to reach some node between n_{t1} and n_{t0} is $O(\frac{\log_2 N}{k})$. In the same way, with distance $4w_x$ to n_{t0}, n_{t3} needs $O(\frac{\log_2 N}{k})$ hops to reach some node between n_{t2} and n_{t0}. As a result, from a starting node n_j with any distance z to n_{t0}, a search needs $O(\log_2 z/w_x)$ iterations to half its current distances until the search arrives at some node between n_{t1} and n_{t0}. For each iteration to cut the current distance by half, the average number of hops is $O(\frac{\log_2 N}{k})$, then the whole search require on average $O(\log_2 z/w_x \cdot \frac{\log_2 N}{k})$ hops. As described before, for the remaining path from some node between n_{t1} and n_{t0} to the destination node n_i, less than $O(\frac{\log_2 N}{k})$ hops are required. Thus the average number of hops for n_j to search n_i is $O(\log_2 z/w_i \cdot \frac{\log_2 N}{k})$, i.e. $O(\log N/l_i \cdot \frac{\log N}{k})$. ■

3.2.2 MAX_PERF Performance Maximization

Based on the definition of popularity, object c_x with popularity p_x receives a fraction p_x of all searches. Supposing c_x is assigned with l_x replicas, from Theorem 1, the average search cost for c_x, H_x, is $O(\frac{\log N/l_x}{k} \cdot \log N)$. Given some popularity distribution, a measure of the average search cost to a total of M data objects, H, can be given by:

$$H = \sum_{x=1}^{M} (p_x \cdot H_x) = \sum_{x=1}^{M} \left[p_x \cdot \left(\log N/l_x \cdot \frac{\log N}{k} \right) \right] \quad (1)$$

In the optimization problem MAX_PERF, given a constant number of replicas, the average number of lookup hops can be minimized in Symphony by the following theorem:

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

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

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

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

$$\nabla f = \sum_{x=1}^{M} \lambda \cdot \widehat{u_x} \quad (3)$$

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

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

Solving for l_x gives

$$l_x = -p_x \cdot \frac{1}{\ln k} \cdot \frac{1}{\lambda} \quad (5)$$

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

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

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

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

$$-\frac{1}{\lambda} = \frac{L}{\frac{1}{\ln k}} \quad (8)$$

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

Note the proportional result of Theorem 2 is different from the square root result in [9]. With Theorem 2, given totally L replicas, for some object c_x with popularity p_x, we can assign it with $l_x = \lceil p_x \cdot L \rceil$ replicas to minimize the average search cost H. For a large value of L and p_x, the result of $p_x \cdot L$ could be even larger than the maximum valid value of l_x, $(N-1)$. Thus for $p_x \cdot L > (N-1)$, we only assign $(N-1)$ replicas for c_x and allocate the remaining replicas to the other objects for further acceleration. When we substitute $l_x = p_x \cdot L$ into Equation 1, we get:

$$H = \frac{\log N}{k} \cdot \left(\log N - \sum_{x=1}^{M} p_x \log p_x - \log L \right) \quad (9)$$

Interestingly, we notice that the term $-\sum_{x=1}^{M} p_x \log p_x$ in Equations 9 is in fact the *entropy* of the popularities p_x. This makes sense since we have expected that the skew of the popularity distribution will play an important role in the optimization of the system settings, and taking the popularity as a probability function (of the query targets) entropy is a sound measure of the skew of the distribution. In conclusion, we can state that the average search cost H depends upon the values of L, N and the entropy of p_x.

3.2.3 MIN_COST for Replicas 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 target constant τ for the average search cost H. We derive the following theorem to solve the optimization problem of MIN_COST:

Theorem 3 *Given $H = \tau$, $L = \sum_{x=1}^{M} l_x$ is minimized if $\forall x$ $l_x = \frac{p_x \log N}{k} \cdot k^{\alpha}$, where $\alpha = (\log N - \tau k / \log N - \log\log N + \log k - E_{p_x})$ and $E_{p_x} = \sum_{x=1}^{M}(p_x \cdot \log_k p_x)$ is the entropy of p_x.*

Proof: Similar to the proof of Theorem 2 by using Lagrange multiplier method. ■

The benefit of our optimal solution for MIN_COST is that the value of τ can be a constant independent upon the node count N. Thus the optimal allocation of l_x for data object c_x in Theorem 3 can achieve $O(1)$ number of lookup hops. As Theorem 2, Theorem 3 also gives 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 Symphony.

3.3 PCache in DHT based Structured P2Ps

In this section we extend the placement strategy to DHT based structured P2P. Chord is chosen as the example of DHT based structured P2P due to: (1) similar ring structures and prefix based routing as in Symphony; (2) other prefix-based DHTs like Pastry, Tapestry and Kademlia can be generalized to a ring-like structure by ordering the NodeID per node. Therefore, our placement strategy can be similarly applied in these prefix routing DHTs.

3.3.1 Acceleration Theorem in Chord

Similar to Theorem 1 in Section 3.2.1, we have the following theorem for Chord:

Theorem 4 *Suppose Chord assigns a finger table with size of $\log_2 N$ at each node and assigns l_x acceleration nodes for object c_x with n_i as the home node, for $1 \le i \le N$, then the average hops to search object c_x, $H_x = O(\log_2 N - \log_2 l_x)$, if $l_x \ge 1$.*

Proof: Based on Chord's finger table allocation principle, let the i-th node in Chord be the node with a node ID i, the k-th item in the finger table points to the successor node of ID $(i + 2^{k-1})$ where $1 \le k \le \log_2 N$. As shown in Figure 3(b), by the lookup algorithm provided by Chord, the search from source n_j with distance z to destination n_i takes at most $\log_2 z$ hops. Thus, along the routing path from n_j to n_i, there are at most $(\log_2 z - 1)$ intermediate nodes denoted as n_{tx} where $1 \le x \le (\log_2 z - 1)$. During each hop from an intermediate node n_{tx} to a next intermediate node n_{tx+1}, the distance between n_{tx} and n_{tx+1} is 2^x. Since in Chord at most $\log_2 z$ hops are consumed from n_j to n_i, then sum of the distance for each hop between two continuous intermediate nodes is no less than the distance between n_j and n_i, i.e. $\sum_{x=1}^{\log_2 z - 2} 2^x \ge z$.

Suppose l_x acceleration nodes are assigned for object c_x in home node n_i. These acceleration nodes precede n_i with the nearest overlay distances (see Definition 1) , then these l_x acceleration nodes will cover at least $0.5 * \log_2 l_x$ hops along the routing path from n_j towards n_i because $l_x \ge \sum_{u=1}^{0.5*\log_2 l_x} 2^u$. Then, with l_x acceleration nodes assigned to n_i, the number of hops from n_j to n_i will be reduced by at least $0.5 * \log_2 l_x$, and the number of hops from n_j to n_i is $(\log_2 N - 0.5 * \log_2 l_x)$. Consequently, for any source n_j, the number of hops from n_j to n_i is $O(\log N - \log l_x)$. ■

3.3.2 MAX_PERF in Chord

The proportional principle based popularity p_x in Theorem 2 can be similarly derived for Chord. We need only substitute the results of H_i in Theorem 4 into Equation 1 in Section 3.2.2, which gives:

$$H = \sum_{x=1}^{M}(p_x \cdot H_x) = \sum_{x=1}^{M} p_x \cdot (\log N - \log l_x) \quad (10)$$

For the optimal performance problem MAX_PERF, we have the following result:

Theorem 5 *Given* $\sum_{x=1}^{M} l_x = L$, *H is minimized when* $\forall x \; l_x = p_x \cdot L$

Proof: Similar to the proving of Theorem 2 in Section 3.2.2. ■

Here for Chord we also achieve the popularity based proportional principle as Symphony. Again we substitute $l_x = p_x \cdot L$ into Equation 10 and get:

$$H = \log N - \log L - \sum_{x=0}^{M} (p_x \cdot \log p_x) \tag{11}$$

In Equation 11, we find $-\sum_{x=0}^{M} (p_x \cdot \log p_x)$, i.e. the entropy of p_x, again appears in the average lookup hops in Chord. It can be explained with the similar reasons as Symphony. In particular, Equation 9 for Symphony when $k = \log N$ will be consistent with Equation 11 for Chord.

3.3.3 MIN_COST in Chord

For the optimization problem MIN_COST, we get the following result:

Theorem 6 *Given* $H = \tau$, $L = \sum_{x=1}^{C} l_x$ *is minimized when* $l_x = p_x \cdot k^{(\log N - \tau - E_{p_x})}$, *where* $E_{p_x} = \sum_{x=1}^{C} (p_x \cdot \log_k p_x)$ *is the entropy of* p_x.

Proof: Similar to the proof of Theorem 3 in Section 3.2.3.■

3.4 PCache System Architecture

3.4.1 Overview

The target environment of PCache is a large scale distributed system with a large number of peer nodes. Such environment could be corporate networks or Internet service providers with well-managed and dedicated machines. In each node, PCache runs as a daemon program. There are three components in PCache: proactive cache proxy, local cache store, and the underlying P2P operation unit (see Figure 4). 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: (i) intercepting http requests from web browsers; (ii) caching requested web objects; (iii) optimally replicating popular web objects based on the replica placement strategy; and (iv) maintaining the consistency of cached Web objects. The cache store, used to locally store the cached web objects, is limited to a fixed storage size, and the Least Recent Used (LRU) algorithm is used to replace Web objects. Finally, the P2P operation unit provides the *get/put* API by which web objects are retrieved/stored from/to the *home* node of the web object. Note that web objects are typically of a reasonable size.

When a client node submits a http request, the client node itself, intermediate nodes, and the home node of the http URL can cooperatively serve the http request when the local cache stores of these nodes contain the replicas of the requested web object. If no replica of such requested object is found, PCache redirects the request to the original web server. As a result, PCache utilizes the underlying P2P overlay to cooperatively serve the http request by replicas of web contents. The key

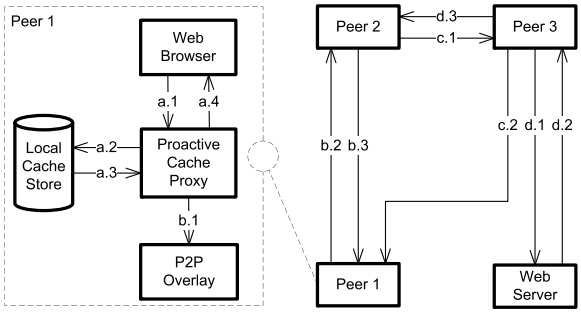


Fig. 4. PCache Architecture

point of PCache is to proactively replicate the web contents based on the optimal principle derived in previous sections. Therefore, PCache can optimally tradeoff the paid cost (i.e. the number of replicas) and the performance gain (i.e. average search hop number).

Through Figure 4, we illustrate the use of PCache as follows.

- The client user sends the http request by submitting a URL via the web browser to access the web object (a.1 in Figure 4). 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 (a.2 in Figure 4).
- If there is a replica in the local cache store, such replica is directly returned to the client (a.3 and a.4 in Figure 4);
- Otherwise, if no replica of the requested object exists , the proactive cache proxy redirects (b.1 in Figure 4) the http request to the P2P operation unit to lookup the home node (i.e. $peer\_3$ in Figure 4) with a node ID closest to the hash ID of the URL. The http request is greedily forwarded towards $peer\_3$. During the forwarding, if the local cache store of an intermediated node $peer\_2$ contains a replica of the requested object (b.2 in Figure 4), then the replica is returned to the requester (b.3 in Figure 4);
- When all intermediate nodes towards the home node $peer\_3$ do not contain the replica of such requested object, the request finally reaches $peer\_3$ (c.1 in Figure 4). Then PCache will check whether a replica in $peer\_3$ is available, and if so, the replica is directly returned to the requester (c.2 in Figure 4); otherwise, the request will visit the original web server (d.1 in Figure 4) and fetch the requested object to the home node $peer\_3$ (d.2 in Figure 4), then the home node responds the original requester $peer\_1$.
- The home node $peer\_3$ probatively replicates the object c_x to an acceleration node $peer\_2$ (d.3 in Figure 4).

3.4.2 Load Balancing

Though the skewed popularity can bring the benefit of reducing the average search cost, it creates the unbalanced workload. In particular, the home nodes for the most popular web objects will serve more the most requests. However, when the replicas of popular web objects are placed in acceleration nodes, the requests for popular web objects are cooperatively served by

the replicas in acceleration nodes. As a result, the workload of these home nodes for popular web objects are reduced. As shown in Figure 2(b), the replicas in acceleration nodes can efficiently intercept the requests towards the home node of the most popular web objects, and the workload of the home node is shared by acceleration nodes. In addition, we conduct the experiment in Section 4 to study the load balancing in the general case, instead of the special case of Figure 2(b) where a single home node has the highest popularity. The experimental result in Section 4 shows that the replicas in acceleration nodes can help achieve the balanced workload throughout the whole P2P network.

3.4.3 Popularity Estimation

We may periodically approximate the value of content popularity p_x and the popularity distribution. First each peer periodically measures the number of lookups received for each local content. If, the content is the raw object instead of replicated copy, the count number of received lookups can directly indicate the popularity of such content; otherwise, the lookups for some content c_x with l_x replicas may be answered by one of l_x replicas distributed in l_x acceleration nodes. The query count of c_x can be computed by aggregating the queries answered by all of these l_x replicas. We may use the aggregation protocol in [31] to estimate the total number of queries. For each node, the number of received queries can be measured by counting at a regular interval.

Since the popularity distribution is formed with a relatively long period, we can setup the long interval to collect the query count in order to reduce the message volume caused by popularity estimation. When copies of c_x are progressively replicated in l_x acceleration nodes, the query count of all l_x replicas can be aggregated and the final query count will be computed in home node of c_x. 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. After that, the home node of c_x will assign more replicas of c_x to relax the workload of a sudden burst.

3.4.4 Consistency Maintenance

A common concern in maintaining object copies at multiple acceleration nodes is the issue of content consistency. When a web object expires, its home node is responsible for fetching a new copy from the origin web server. Then this fresh copy is propagated proactively to all acceleration nodes. To resolve the content conflict, an extra field, i.e. the object version, can be attached to a web object. When the web object is refreshed, its home node increases the object version. An acceleration node always refreshes the web object by the received copy with a larger version. The object version is helpful for the acceleration node that may miss a fresh object to restore itself to a consistent state.

Though PCache mainly focuses on the optimization problem between the average lookup hop number and the replica number, Web objects are different in data size and update rates [12]. That could affect the optimal replica number l_x if the communication cost of updating replicas is considered. An approximation is to use a factor by the ratio between the communication cost of updating a web object against the the average communication cost. We need to measure the data size and update rate of each web object. Then the multiplication of the data size and update rate is treated as the communication cost of updating a web object, denoted as $\mathcal{C}_x$. Based on the optimal replica number for such web object, l_x (e.g. derived in Theorem 2), we can approximately tune the replica number as $l_x \cdot \frac{\mathcal{C}_x}{\overline{\mathcal{C}}}$, where $\overline{\mathcal{C}}$ is the average communication cost. Based on such idea, we could extend PCache to other situations. For example if the real storage is considered, we can find a similar storage factor and tune the replica number, correspondingly.

4 PRing: A Popularity Based P2P Ring Structure

In this section we consider the links to connect the peers in a P2P network as the resource $\mathcal{R}$. Following the resource placement strategy in Section 2, for a given node n_i, we assign extra links to connect n_i's acceleration nodes and n_i. For each acceleration node, the extra link, called *acceleration link* (*alink*), as shown in Figure 5, is constructed to point to n_i. When a search for some object in n_i arrives at one acceleration node of n_i, the search can directly jump to n_i by the *alink*. Compared with PCache where the request for popular objects is directly answered by the replicas, the search for popular node n_i in PRing only requires one more hop via *alink*.

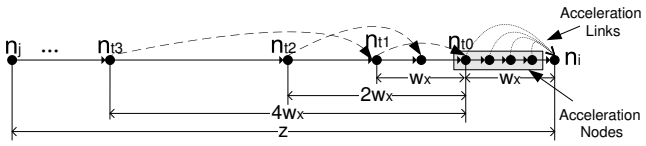


Fig. 5. Acceleration Links in Symphony

In PRing, we choose Symphony [20] as the base P2P structure due to the unique property of Symphony that the number of long links in each node can be flexibly configured. In Symphony, the number of long links, k, is the same for all nodes, and if it is set as $k = O(1)$ then the average search cost is $O(\frac{1}{k} \cdot \log^2 N)$. With this kind of flexility, we can adaptively set the varied number of long links in order to optimally achieve the best performance $\mathcal{P}$.

In PRing there are three kinds of links: *alinks* for acceleration purpose, original *short links* and *long link* (*llink*) in Symphony. Since short link is always used to construct the base ring structure by connecting two immediate neighbors, we can treat the number of short links assigned to each node fixed. For a given number of all links (*llinks* and *alinks*), we focus on how many fraction of all links are assigned for *alinks* and the remaining for *llinks*.

In PRing, for a given node n_i, we aggregate the object popularities p_x of all objects c_x stored in n_i as the node popularity of n_i, denoted by p_i. Following similar proofs for the properties in PCache, we can derive the following corollaries, respectively related to Theorem 1, Theorem 2 and Theorem 3:

Corollary 1 *The average search cost to node n_i in PRing is $\mathbb{H}_i = O(\frac{\log N/\ell_i}{k} \cdot \log N)$ where ℓ_i is the number of* alinks *assigned for node n_i.*

Let N be the number of nodes in PRing, and $\mathbb{H} = \sum_{i=1}^{M} (p_i \cdot \mathbb{H}_i)$.

Corollary 2 *If there are in total $\mathbb{L}$ alinks in PRing, $\mathbb{H}$ is minimized when $\forall i\ \ell_i = p_i \cdot \mathbb{L}$.*

Corollary 3 *If $\mathbb{H} = \tau$, then the total number of alinks is minimized when $\ell_i = \frac{p_i \log N}{k} \cdot k^{\alpha}$, where $\alpha = (\log N - \tau k / \log N - \log\log N + \log k - E_{p_i})$ and $E_{p_i} = \sum_{i=1}^{N} (p_i \cdot \log_k p_i)$ is the entropy of p_i.*

The above corollaries for PRing are similar to the theorems for PCache. Nevertheless, we are interested in how the performance of PRing, by the average search cost $\mathbb{H}$, compares with existing P2P structures like Symphony itself and Chord, particularly when the total number of links (including *alinks* and *llinks*) in PRing are equal to the total number of regular long links in Symphony, and that of finger table size in Chord.

Based on such problem, we apply the result of corollary 2 to further improve the average search cost $\mathbb{H}$ for PRing. Until now, for node n_i in PRing with total $\mathbb{L}$ number of *alinks*, we optimally assign $p_i \cdot \mathbb{L}$ number of *alinks* to n_i based on its popularity p_i. Actually such assignment scheme does not globally consider the assignment between *llinks* and *alinks*. When Symphony is used as the underlying P2P overlay network in PRing, each node is regularly assigned with k *llinks*. With consideration of the number of *alinks* and *llinks*, there are total $\mathbb{D} = (k \cdot N + \mathbb{L})$ links. On the constraint of a constant value of $\mathbb{D}$, we focus on how many links are assigned as *llinks* and how many are as *alinks*. Here we define the term acceleration ratio γ as following:

Definition 3 *The acceleration ratio, denoted by γ, is defined as the fraction of the number of* alinks *among all available links, i.e. $\gamma = \frac{\mathbb{L}}{\mathbb{D}}$.*

Then with the definition of acceleration ratio γ and the constant number of all links, we get $k = (1-\gamma)\mathbb{D}/N$ and $L = \gamma \cdot \mathbb{D}$, which can be substituted to Equation 9:

$$\mathbb{H} = \frac{N \log N}{\mathbb{D}(1-\gamma)} \cdot \left[\log N - \sum_{i=0}^{N} p_i \log p_i - \log(\gamma \mathbb{D}) \right] \qquad (12)$$

Theorem 7 *To minimize $\mathbb{H}$ in Equation 12, given the value of $\mathbb{D}$, the total number of links including* llinks *and* alinks*, the acceleration ratio γ where $0 < \gamma < 1$ satisfies:*

$$\log(\gamma \mathbb{D}) + \frac{1-\gamma}{\gamma} = \log N - \sum_{i=1}^{N} p_i \log p_i \qquad (13)$$

Proof: When the probability distribution of p_i and the values of $\mathbb{D}$ and N are given, the value of $\mathbb{H}$ in Equation 12 is dependent upon a single variance γ. To minimize $\mathbb{H}$, we can solve the equation $\mathbb{H}' = 0$, where $\mathbb{H}'$ is the derivative of $\mathbb{H}$, and then give equation 13. ■

The value of γ in Theorem 7 is within $(0.0, 1.0)$. Let us consider the extreme cases when $\gamma = 0.0$ or 1.0. If all $\mathbb{D}$ links are assigned as $N \cdot k$ *llinks*, i.e. $\gamma = 0.0$, Equation 12 then becomes $\mathbb{H} = \frac{N \log^2 N}{D}$. Note that when $\gamma = 0.0$, there is no *llink*, and we have the original Symphony structure. If $\gamma = 1.0$, then all $\mathbb{D}$ links are allocated for *alinks* with $\mathbb{L} = \mathbb{D}$, the average search cost $\mathbb{H}$ is $O(N)$, which is quite unacceptable.

As a special case that $\mathbb{D} = N \cdot \log N$ which is comparable to existing DHT systems like Chord, then Equation 13 and Equation 12 become

$$\log\gamma + \frac{1}{\gamma} = 1 - \log(\log N) - \sum_{i=1}^{N} p_i \log p_i \qquad (14)$$

$$\mathbb{H} = \frac{-\sum_{i=1}^{N} p_i \log p_i - \log\gamma - \log(\log N)}{(1-\gamma)} \qquad (15)$$

In the experiments we shall show that the optimal value of γ can give better performance results than other values of γ, including 0.0 and 1.0.

Discussion: It is well-known that the topologies of Internet, WWW, social networks, and cells are the so-called scale-free networks. Such networks display an unexpected degree of robustness, the ability of their nodes to communicate being unaffected even by unrealistically high failure rates. However, error tolerance comes at a price that these networks are vulnerable to attacks by the selection and removal of a few nodes playing a vital role in maintaining the network's connectivity [25]. Different from these scale-free networks with power-law organizational structures, our proposed topology PRing avoids such extremely skewed distribution of links by considering the optimal tradeoff between the number of *llinks* and *alinks* in Theorem 7. From both numeric results and simulation results, we can find that for the commonly appearing Zipf parameter $\alpha \approx 1.15$, only a small fraction (i.e. the acceleration ratio $\gamma = 0.1562$ when $\alpha = 1.15$ and N=8192) of all links are assigned as *alinks*, and the majority of links are still regular *llinks*. Furthermore, the larger value of N results in the smaller value of the acceleration ratio γ. Consequently, the purpose of PRing is to (i) accelerate the search for popular nodes thus minimizing the overall average lookup hops; (ii) avoid the extremely skewed linkage distribution towards a few popular nodes by setting up an optimal value of γ to tradeoff the link assignment between *alinks* and *llinks*.

With a skewed popularity p_i, nevertheless, PRing means the irregular topology where popular nodes have more *alinks* than unpopular nodes do. Though with such un-regular topology, our experimental results show that the average search cost $\mathbb{H}$ in PRing is less than Chord and Symphony with the equal number of links $N \cdot \log N$. However, due to the skewed popularity, PRing will produce an un-balanced workload to serve incoming requests. To overcome the unbalanced workload, as one of available solutions, we can apply PCache to enhance the maintenance of PRing. As a result, the workloads caused by popular objects can be cooperatively served by the replicas in acceleration nodes, that has already been given by PCache; meanwhile, the popularity distribution becomes less skewed

and PRing maintains the more balanced topology.

In addition, when considering the capacity heterogeneity, we can follow the technique in [14] to setup virtual nodes in a powerful physical machine. After such technique is adopted, an interesting situation is that the virtual nodes inside one machine could be simultaneously used as the acceleration nodes of a home node. In this situation, the physical machine maintains only one copy of the web object in the home node, instead of multiple copies with one copy for each of virtual node. Clearly, it can help reduce the maintenance cost to store the content copies and to check the content consistency.

Finally, considering the practical operation, each node can manage the construction and maintenance of *alinks* and *llinks* with differentiate protocols. First, to construct the *alinks* and *llinks*, based on the operational policy, each node sets the minimal number of *llinks* (e.g. at least one *llinks*) and the maximal number of *alinks* (e.g. no more than its capacity). In this way, the overall assignment of *alinks* and *llinks* can be practically manageable. Secondly, for the maintenance side, the *llinks* are kept as constant connections with a higher Qualify of Service (QoS) than the *alinks*. This makes sense since *llinks* always can guarantee the $O(\log N)$ number of routing hops.

5 EVALUATION

In this section we evaluate the performance of PCache and PRing. The evaluation is for four purposes:

- The comparison of PCache with three related approaches (CFS [10], PAST [27] and Beehive [21]) respectively on three P2P networks (Symphony, Chord and Pastry). All comparisons are based on two publicly available web trace files;
- The deployment of PCache in the real emulation platform PlanetLab;
- The numeric results of PRing by MatLab to study the optimality of PRing under various configuration parameters;
- The simulation results PRing with the comparison of two related ring structures (Symphony and Chord) with various P2P topologies;

5.1 Traces

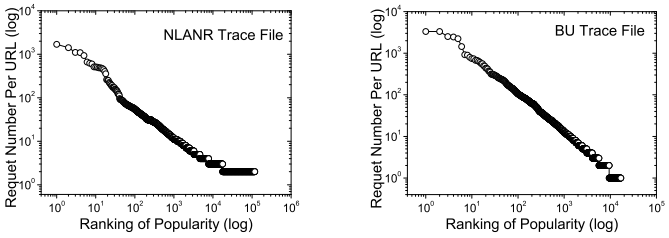
Table 2 lists the Web traces we have used for the performance evaluation.

- NLANR traces: NLANR (National Lab of Applied Network Research) provides sanitized cache access logs in the public domain [1]. 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 [2]. We have selected the contents in subdirectory condensed/272 of BU-www-client-traces.tar.gz

To clearly show the popularity distribution, we respectively plot the count of requests per URL ordered by the ranking of popularities of NLANR trace and BU trace in Figure 6. The ranking sequence in the x-axis of Figure 6 is ordered in a descending manner. For 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 $-\sum_{x=1}^{M}(p_x \cdot \log_k p_x)$, for $k = 7.6$, we compute the entropy as 6.2579 and 2.7637, respectively for NLANR trace file and BU trace file. Since the entropy is used to measure the randomness of the popularity distribution, we can find that the popularity distribution of BU trace file is more skewed than that of NLANR trace file.

TABLE 2
Statistics about the traces

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



(a) NLANR trace (b) BU trace

Fig. 6. Requests of URLs

5.2 Evaluation Approaches

During the evaluation, we mainly use three kinds of evaluation approaches: an event driven simulator, the deployment in PlanetLab, and the numeric result computed by Matlab. The metric used in the simulation and numeric result is the average hop number H. In the PlanetLab deployment, the evaluation metrics include the average hit rate and external bandwidth.

During the simulation, we use an event driven simulator to respectively simulate Chord protocol. Based on the simulator, we revise it to produce the Symphony protocol by connecting peers based on the Harmonic distribution, instead of the deterministic scheme in Chord. For Pastry implementation, we chose the open source code FreePastry[1]. In all implementations, the web object is stored and retrieved in the home node of its URL. Furthermore, the web objects are proactively replicated to the acceleration nodes (peers) based on PCache's optimal solutions for MAX_PERF and MIN_COST.

1. FreePastry:http://www.freepastry.org

TABLE 3
Solving MAX_PERF Problem with $N = 4000$ to Evaluate the Achieved Average Lookup Hops H .

Replicas($\times$ 1000)	NLANR Trace				BU Trace			
	PAST	CFS	Beehive	Our work	PAST	CFS	Beehive	Our work
Chord								
0	7.97255	7.97255		7.97255	7.97255	7.97255		7.97255
40	7.61557	7.29463		7.05315	6.9135	6.013		5.35135
80	7.39019	7.02557		6.61353	6.0389	5.2158		4.0135
160	6.82265	6.3123		6.0135	5.1931	4.1131		2.8131
400	5.66699	5.23435		4.63136	3.3151	2.351		1.8953
Symphony								
0	8.13513	8.13513		8.13513	8.13513	8.13513		8.13513
40	7.81557	7.41463		7.29123	7.351	6.51535		5.58135
80	7.59019	7.12557		6.93521	6.3389	5.5909		4.0135
160	6.9565	6.6133		6.31618	5.4133	4.2513		2.8131
400	6.13567	5.5135		5.2353	3.19315	2.3835		2.09
Pastry								
0	4.798	4.798	4.798	4.798	4.798	4.798	4.798	4.798
40	4.51557	4.29463	4.20391	3.63391	4.25135	4.113	4.05315	3.2135
80	4.29019	4.02557	4.05315	3.04315	3.91353	3.7135	3.6158	2.7389
160	3.82265	3.7123	3.66353	2.61353	3.0135	3.5131	3.1931	2.4131
400	3.46699	3.23435	3.11136	2.01136	2.03136	2.39	2.151	1.851

For the realistic deployment in Planetlab environment, we chose FreePastry as the example of P2P network and adopt the optimal solution of MAX_PERF to evaluate the performance of PCache. The reasons that we chose FreePastry as the example of P2P network are given as follows: (1) up to date, FreePastry is the most widely used and powerful open source implementation of DHT; (2) Pastry shares the similar ring structure as Symphony and Chord [7]; (3) though we only conduct the deployment of FreePastry-based PCache over PlanetLab, we believe the similar results will be achieved for the deployment of Chord or Symphony-based PCache over PlanetLab.

We deploy FreePastry based PCache implementation on a set of around 90 Planet-Lab nodes. In each PlanetLab node we start up 10 FreePastry instances (i.e. one port for each instance in the same physical machine) to standard for total around 900 peers, and measure their performance with respect to the three metrics including average lookup hops H, average hit rate and external bandwidth. For this experiment, we use the NLANR web trace of Section 5.1 with total 189034 replicas as the resource to improve the lookup performance. During the first hour during the experiment, all distinct URLs in the Web trace are published to simulate the web contents stored in the home nodes. After that, with consideration of the limited bandwidth of PlanetLab nodes, 10 special nodes are specially used to randomly chose 900 records from the trace file every one minute (i.e. 90 records per node), and then send the chosen records to 900 peers. After receiving the records, each peer simulates the search by finding the home node for the URL appearing in each received record. Since the records in the trace themselves are skewed distributed, such searching can be used to simulate the search with skewed popularity, following the original distribution in the web trace file. The whole experiment lasts for the total 14 days, and we collect the statistics every day, and every 30 minutes the content popularity p_x is estimated.

5.3 Comparison of Placement Strategy

To compare the resource placement strategies, we use replica as the resource to solve MAX_PERF and MIN_COST optimization problems. All experimental results in this section are based on the event driven simulation.

Solving MAX_PERF problem: We compare our placement strategies in Section 2 with other strategies: PAST strategy, CFS strategy, and Beehive strategy respectively with Symphony, Chord and Pastry as the underlying P2P overlay network platforms. Following the strategy described in [27], PAST strategy replicates the copies of web contents in randomly chosen nodes; CFS [10] which is a cooperative file system based on Chord [28], and we place replicas along the lookup path towards the destination. Since Beehive strategy [21] only provides the approximate solution for Zipf popularity distribution in Pastry, we apply Beehive strategy only in Pastry. Also for a fairness comparison, the replica number for each strategy is also based on the popularity-based proportional assignment, as we achieved in Section 3.

Table 3 illustrates the placement strategy comparisons for solving the MAX_PERF optimization problem in three P2P networks with two trace files: given a number as the total number of replicas, each item in Table 3 shows the average lookup hops. Based on these results, we can find:

- PCache can achieve the best average lookup hops H in three P2P networks. This is because: (a) PCache by placing the replicas along the acceleration nodes can intercept the lookups towards the destination nodes with a high probability, as shown in Figure 2. Thus it can thus reduce the average lookup number H; for CFS or random strategy, the heuristic replica placement is useful to some partial nodes. For example in Figure 1, for CFS strategy, though replicas placed in node r_i, n_{t0}, n_{t2} and n_j can benefit the search to n_i, the search from other nodes (except r_i, n_{t0}, n_{t2} or n_j) can not benefit from such placement; (b) our work, with the proportional replication, can be generally applied in both randomized P2P networks (Symphony) and deterministic

TABLE 4
Solving MIN_COST Problem with $N = 4000$ to Find the Consumed Replicas ($\times$ 1000)

Avg Lookup Hops H	NLANR Trace				BU Trace			
	PAST	CFS	Beehive	Our work	PAST	CFS	Beehive	Our work
Chord								
2					670	510		387
3					450	331.3		140
4					281	201		80
5	870	613.05		400	190	90		48.13
6	300	350		223	80	40		20.01
7	100	84		80	52.6	18.2		10.001
8	19.153	16.2		11.2	2.81	1.353		1.050
Symphony								
2					960	660		423
3					591	482.3		251.8
4					431	339.1		180
5	1013	871.2		533.4	359.6	190.4		93.13
6	522	423		283	180	86		20.01
7	280	229		130	66.3	43.2		16.3
8	37.153	26.2		15.2	10.81	5.353		1.050
Pastry								
1					429	350	199	113
2	631.5	484.5	401.3	330.7	214	259	130	92
3	390	281	120.6	90.1	73.6	68.1	53.3	23.1

P2Ps (DHT); (3) our work is better than the approximate solution Beehive [21].

- With the equal total number of replicas L, the value of H of BU is less than that of NLANR trace. It can be clearly explained by the value of popularity entropies, i.e. $-\sum_{x=1}^{M} p_x \log p_x$, of BU trace smaller than that of NLANR trace. Intuitively, for BU trace with a skew popularity distribution, some given number of replicas can improve the majority of all searches, which are targeted to a few number of highly popular objects; for NLANR trace with a less skewed popularity distribution, the equal number of replicas only improve minor searches which are targeted to much more objects.

Solving MIN_COST problem: Table 4 illustrates the required number of replicas L with the goal to satisfy the target lookup hops τ for solving MIN_COST problem. Naturally, to satisfy a smaller τ, more copies will be consumed. From Table 4, to achieve the same target number of hops τ, our strategy PCache consumes the fewest number of replicas L for both NLANR trace and BU trace. Furthermore, to achieve the reduction of 1 hop from $\tau = 8$ to $\tau = 7$, for PCache in Symphony with NLANR trace, about 1.2×10^4 more replicas are required; while to achieve the same reduction of 1 hop from $\tau = 6$ to $\tau = 5$, about 7×10^6 more replicas are required. It means that the same reduction of the average lookup hops consumes nearly 700 times of replicas. Also due to the different entropy values for BU trace and NLANR trace, more replicas are consumed for NLANR trace than BU trace. These experimental results provide a guidelines for a system designer about the tradeoff between the gained performance and the paid cost.

5.4 Evaluation of PCache in PlanetLab

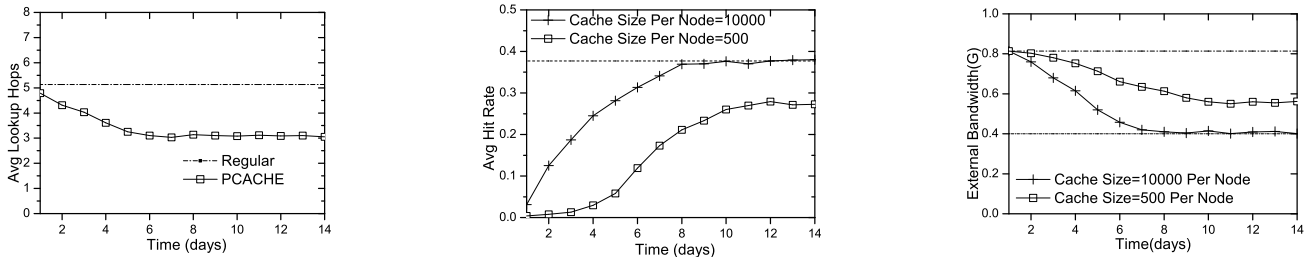
Figure 7(a) shows the average lookup hops. In this figure the statistical value of the average lookup hops gradually becomes stable equal to 3.08 after 6 days, because it is enough to capture the overall popularity distribution in the web trace file. We measure the simulated hit ratio and external bandwidth savings when the per-node cache storage size varies. 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 and be more possibly cached in the local cache store. Obviously the larger storage size of local cache store can accommodates more replicas, and increase the hit ratio, as shown in Figure 7(b). The average hit ratio in Figure 7(b) is stable after around 7 days. For the missed requests, we calculate the data size of such URL as the consumed external bandwidth, shown by Figure 7(c). From Figure 7(b) and (c) we can find that the larger local cache store size setup in each node can make the average hit rate and external bandwidth be stable in the a shorter period than the smaller allocated cache store.

In addition, we compute the overloading rate of a node as the rate of the maximal number of requests in such peer over the average number of request. In the above experiment, we find the overloading rate in the first day, 7.23, is gradually reduced and it becomes only 1.50 in last day.

5.5 Evaluation of PRing

To evaluate PRing, we respectively use the numeric results computed by Matlab, and simulation results by the event driven simulator. For Matlab result, we synthetically generate the node popularity p_i distribution following the Zipf distribution when the Zipf parameter α is given and the total number of links $\mathbb{D}$ (including *alinks* and *llinks*) is equal to $(N \cdot \log N)$. In order to compare with the Matlab results, we use the same synthetic data for simulation. By the same synthetic data, we can validate whether the simulation results are consistent with the simulation results.

Numeric Results: When the values of node count N, and the entropy of node popularity p_i are given, the average lookup hops $\mathbb{H}$ and the acceleration ratio γ in Equations 14 and 15 can be numerically solved. Figures 8(a) and (b) respectively plot the numeric values of γ and $\mathbb{H}$ for a given Zipf parameter



(a) Average Lookup Hops (b) Average Hit Rate(%) (c) External Bandwidth(GB)

Fig. 7. PCache in PlanetLab

value $\alpha = 0.95$. We can find that for the optimal numeric result, a larger number of node count N can make a less value of acceleration ratio γ. It means fewer links are allocated for acceleration links (*alinks*); also the larger value of Zipf parameter α, i.e the more skewed popularity distribution, makes the larger value of γ, meaning more *alinks* are allocated for the popular nodes. However, due to the optimal assignment, not all links are assigned as the *alinks* for a few number of highly popular nodes. For example, for $\alpha = 1.15$ and N=8192, the numeric value of γ is equal to 0.1562. It means that the majority of links are assigned to the regular long links (*llinks*) and a small fractions of links are for *alinks*. This result is important for PRing to avoid the extremely skewed linkage distribution.

Furthermore, by calculating the entropy value of p_i, we find the entropy value of p_i is the primary factor in Equation 15; when N grows, the entropy of p_i also grows and $\mathbb{H}$ then grows. Figures 8 (c) and (d) respectively show the values of γ and $\mathbb{H}$ by varying α with $N = 2048$ and $N = 8192$. We find that with a fixed node count N, the value of $\mathbb{H}$ is decreased when the popularity distribution becomes more skewed (i.e. a larger value of α). For $\alpha = 2.15$, $\mathbb{H}$ approach 1.

Simulation Results: In simulation experiments, we vary the value of acceleration ratio γ, Zipf parameter α, and node count N to generate multiple topologies. With a given value of Zipf parameter α to produce the values of popularity p_i, we assign each node in the generated P2P topology with some popularity p_i. By Equations 14 and 15, each node is assigned with the regular number of *llinks* and the irregular number of *alinks*. The node assigned with a high value of p_i will receive more requests to simulate the skewed popularity. For $\gamma = 1.0$, all links are assigned for *alinks* and none for *llinks*, resulting in the $O(N)$ average search cost $\mathbb{H}$. To avoid the high search cost $O(N)$ of $\gamma = 1.0$, we still allocate N *llinks* to make sure each node has $k = 1$ *llink*. Under each generated topology, we evaluate the average lookup hops $\mathbb{H}$ as the performance metric of such topology.

Effect of Acceleration Ratio: In this experiment, we vary the value of the acceleration ratio γ between [0.0,1.0) respectively for $N = 2048$ and $N = 8192$. Figure 9(a) shows that the link allocation with $0.1 \leq \gamma \leq 0.3$ can achieve the minimal search cost $\mathbb{H}$. Compared with the numeric results in Figure 8(a), the numeric values of $\gamma = 0.145$ for $N = 2048$ and $\gamma = 0.130$ for $N = 8192$ consistently fall within the range $0.1 \leq \gamma \leq 0.3$ of simulation results.

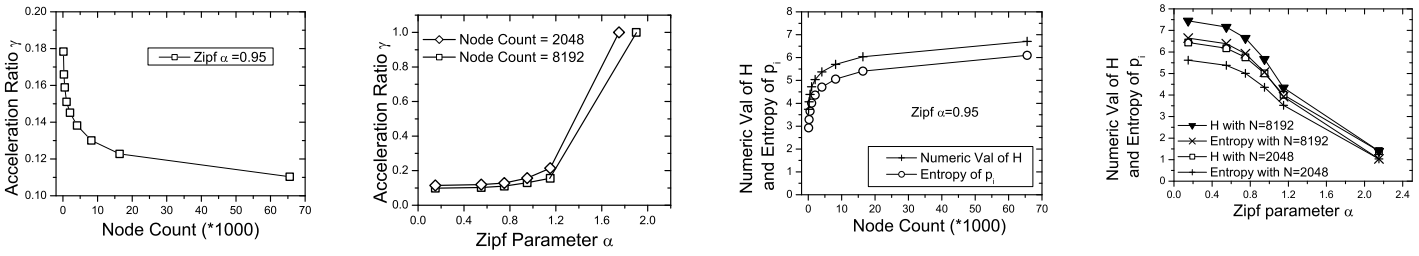
Effect of Popularity Skewness: In this experiment with $N = 2048$, we vary the value of the Zipf parameter α and respectively allocate the links with $\gamma = 0$, $\gamma = 1$, and the optimal numeric value of γ by Matlab. Figure 9(b) shows that the optimal value of γ can achieve the best performance with varied skewness of popularity. When the popularity distribution is extremely skewed with $\alpha = 2.15$[2], the average search cost H with $\gamma = 1.0$ is almost the same as with the optimal γ. Intuitively, in this extreme case where all searches target to the same destination node, all *alinks* are placed from all nodes to one node with the highest popularity, and H approaches 1. Compared with the numeric results in figure 8(b), the simulation values in figure 9(b) follow the numeric results.

Effect of Node Count N: In this experiment, we allocate links by three cases: $\gamma = 0$, the optimal numeric value of γ by Matlab and $\gamma = 1$. By conducting the searches following the Zipf-based popularity distribution, Figure 9(c) plots the average search hops H with the varied values of node count N. It can be seen that the optimal allocation can achieve better results than other tow cases with an equal number of links, particularly for the case with $\gamma = 0.0$ which produces the regular topology (i.e. Symphony and Chord).

Load Balancing: To study of the load distribution, we first conduct the experiments for PRing; next, we adopt PCache to PRing to show the effects of PCache. For an incoming request having a URL, node n_i servers the request in two following cases: i) n_i, as the *home node* of the URL, contains the raw content of the requested URL; ii) n_i, as the *acceleration node* for the URL, contains the replica of the requested raw content. For a node n_i, we set the workload rate of n_i, denoted as ω_i, to be the the number of incoming requests that are served by node n_i over all requests. For $N = 2048$, Figures 9(d) plots the load distribution respectively of PRing itself with no cache and PRing enhanced by PCache. The x-axis of Figure 9(d) represents the workload rate and the y-axis represents the percentage of nodes with a workload rate ω_i within two neighboring rates of x-axis in Figures 9(d).

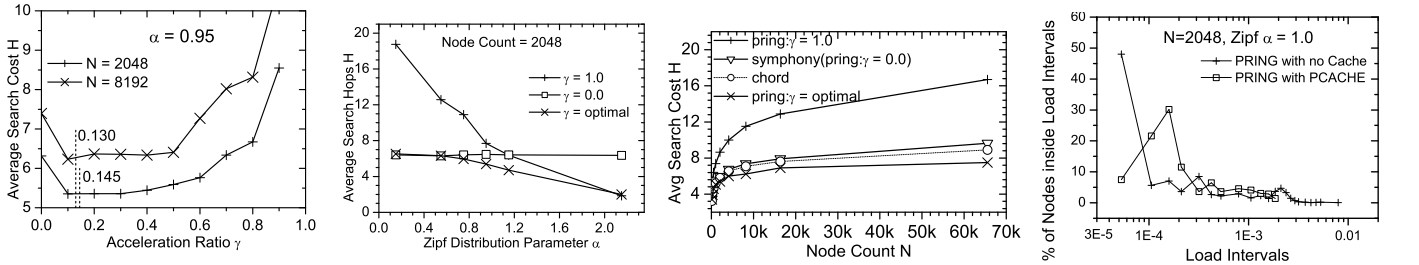
In Figures 9(d), it is not surprised to find that the load distribution of PRing with no cache is heavily tailed and is basically consistent with the Zipf distribution. It is because PRing is designed as an optimal topology to tradeoff between long links (*llinks*) and acceleration links (*alinks*) as to a better average search cost. In order to overcome the limitation of unbalanced workloads, PCache can be used to enhance PRing

2. the value of $\alpha = 2.15$ rarely appears in real applications. Here we intentionally use it for the purpose of the performance study



(a) γ Vs N (b) γ Vs α (c) $\mathbb{H}$ and Entropy of p_i Vs N (d) $\mathbb{H}$ and Entropy of p_i Vs α

Fig. 8. Numeric Results of PRing



(a) Effect of Acceleration Ratio γ (b) Effect of Zipf Parameter α (c) Effect of Node Count N (d) Load Distribution

Fig. 9. Simulation Results of PRing

with balanced workloads, as shown in Figure 9(d). Given 2048 nodes in this experiment, the average workload rate of a node, denoted as $\overline{\omega}$, is equal to 1/2048, and we treat the workload ω_i with $\overline{\omega}/5 \leq \omega_i \leq \overline{\omega} * 5$ as an allowable workload; otherwise either as an overloaded workload or as an underloaded workload. For PRing with no cache, only 49.58% nodes have the allowable workloads; after PCache is adopted, 93.11% nodes have the allowable workloads.

6 Related Work

In this section, we review the related work to ours in three aspects: p2p networks and topologies, replica placement strategies, and cooperative web caching.

Peer-to-Peer Networks and Topologies: A number of peer-to-peer routing protocols have been proposed recently, including Chord [28], Tapestry [32] and Pastry [26]. These self-organizing and 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. These DHT-based P2P networks essentially use an HashID-based prefix routing algorithm. Different from DHT-based deterministic P2P networks, randomized P2P networks like Symphony [20] and SkipGraph [4] choose the node neighbors based on some probability distribution. In all of these structured P2P protocols, each node is regularly assigned with equal $O(\log N)$ number of links, i.e. total $N \cdot O(\log N)$ links, and the average lookup hop number is guaranteed with $O(\log N)$. Overall, these structured P2P are typically *regular* network topologies with $O(\log N)$ links in each node.

Different from structured P2P, Freenet, Gnutella, FastTrack/KaZaA, BitTorrent, Overnet/eDonkey2000 are examples of unstructured P2P which organize peers in a random graph in flat or hierarchical manners (e.g. Super-Peers layer) and use flooding or random walks or expanding-ring Time-To-Live (TTL) search, etc. on the graph to query object stored by overlay peers. Typically the topologies of these unstructured P2P are *irregular*. For example, analysis in [24] shows that Gnutella networks have topologies that are power-law random graphs, and later measurement shows that there are too few peers with a low number of connectivity. Networks with power-law organizational structures, display an unexpected degree of robustness [25], i.e. the ability of peers to communicate unaffectedly by extremely high failure rates.

Replica Placement Strategies: Some works in P2P networks involve the placement of replicas or cached objects for popular objects. CFS [10], a cooperative file system over Chord [28], caches the popular objects along the lookup path towards the home nodes where popular objects are originally stored. In PAST [27], the storage system over Pastry [26], the search for some object is redirected to the nearest replicas of the targeted object. Such placement strategy is a random scheme. Based on the replication level l, Beehive [21] replicates the object copies to all nodes that have at least l common prefixes matching with the object hash ID. As a further extension of Beehive, the technical report [29] is based on deterministic structured P2P; however, it is unknown whether the approach in [29] can be adopted to the randomized structured P2P systems (e.g. Symphony).

Unlike the heuristical schemes like CFS [10] and PAST [27], both Beehive [21] and our work can provide optimal solutions based on the analytical model. Compared with Beehive [21], PCache can provide the closed form solutions for both MAX_PERF and MIN_COST problems; while Beehive only seeks for the approximate solution for

MIN_COST problem. Moreover, there is no assumption on the popularity distribution in our optimization solution; while Beehive [21] provides the analytical solution only for the Zipf distribution.

Our previous work [22] actually is an extension of CFS by the proportional principle. However, though with the similar proportional principle as PCache, the children at the same level of the k-ary tree in [22] do not have the same opportunity as the intermediate nodes to intercept the search towards the destination node. Actually, Figure 3 of Section 2 shows that the replicas placed in acceleration nodes closer the destination node, with a higher probability, can intercept the search to the destination node. Finally, as shown in Section 5.3, by placing replicas in the acceleration nodes closer to the destination node, PCache can perform better than CFS with a proportional replication, which is the replication approach of [22].

Unlike our work with the purpose of accelerating the search, [15] focused upon the load balancing issue by adaptive replication. In addition, though offering a proportional replication in [5], the solutions are only based on some intuitive observations, without any analytical bound.

In unstructured P2P networks, [8] proposes to optimize search efficiency by replication, where the number of replicas of an object is proportional to the square-root of the object popularity. [9] presents a square-root topology for unstructured P2P networks where the degree of a peer is proportional to the square root of node popularity. Their results show that the square-root principle can achieve the optimal performance by the random walk search.

Cooperative Web caching: Cooperative Web caching is the most common solution for augmenting the low cache hit rates due to a single proxy. There has been extensive work on cooperative web caching system as a technique to reduce request latency and to increase the hit rate. The design of cooperative web caching systems can be hierarchical (Harvest [11] and Squid [3]), hash ([17]), directory ([13]), and multicast ([30]). Different from these systems which still require a dedicated proxy infrastructure, P2P based web cache systems completely eliminate the need of proxy servers. Squirrel [16] aims to replace central demand-side web caches; however, a passive web cache system over Pastry [26], it does not consider the skewed popularity and performance optimization. Kache [18] is a cooperative web caching system built over Kelips [18]-based P2P overlay network and it can perform a lookup in one hop but with a high maintenance cost of $O(\sqrt{N})$ peers in each node.

7 Conclusion

In this paper, we present a novel resource placement strategy which can be applied in randomized and deterministic structured P2P networks. By this strategy, we propose the optimal web caching scheme, PCache, and the optimal ring structure, PRing, respectively. Our main contributions are formally analyzing the greedy search algorithm with the consideration of skewed popularity, and developing a resource allocation solution which can optimally tradeoff the performance gain and paid cost. The extensive evaluations demonstrate the effectiveness of PCache and PRing, with better results than existing works.

Acknowledgment

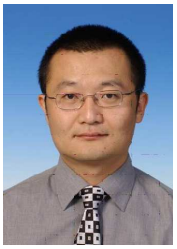
Funding for this work was partially supported by Hong Kong RGC Grants under Project 611608, NSFC Grants under Projects 60736013 and 60873011.

References

[1] In *ftp://ircache.nlanr.net/Traces/DITL-2007-01-09*.
[2] In *http://ita.ee.lbl.gov/html/contrib/BU-Web-Client.html*.
[3] In *http://squid.nlanr.net*.
[4] J. Aspnes and G. Shah. Skip graphs. In *SODA*, 2003.
[5] I. Bhattacharya, S. R. Kashyap, and S. Parthasarathy. Similarity searching in peer-to-peer databases. In *ICDCS*, pages 329–338, 2005.
[6] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM*, pages 126–134, 1999.
[7] M. Castro, P. Druschel, A.-M. Kermarrec, and A. I. T. Rowstron. One ring to rule them all: service discovery and binding in structured peer-to-peer overlay networks. In *ACM SIGOPS European Workshop*, pages 140–145, 2002.
[8] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *SIGCOMM*, 2002.
[9] B. F. Cooper. An optimal overlay topology for routing peer-to-peer searches. In *Middleware*, 2005.
[10] F. Dabek, M. F. Kaashoek, D. R. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *SOSP*, 2001.
[11] F. Douglis and T. Ball. Tracking and viewing changes on the web. In *USENIX Annual Technical Conference*, pages 165–176, 1996.
[12] F. Douglis, A. Feldmann, B. Krishnamurthy, and J. C. Mogul. Rate of change and other metrics: a live study of the world wide web. In *USENIX Symposium on Internet Technologies and Systems*, 1997.
[13] 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.
[14] B. Godfrey and I. Stoica. Heterogeneity and load balance in distributed hash tables. In *INFOCOM*, pages 596–606, 2005.
[15] V. Gopalakrishnan, B. D. Silaghi, B. Bhattacharjee, and P. J. Keleher. Adaptive replication in peer-to-peer systems. In *ICDCS*, pages 360–369, 2004.
[16] S. Iyer, A. I. T. Rowstron, and P. Druschel. Squirrel: a decentralized peer-to-peer web cache. In *PODC*, pages 213–222, 2002.
[17] 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.
[18] P. Linga, I. Gupta, and K. Birman. Kache: Peer-to-peer web caching using kelips. In *In submission*, June 2004.
[19] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *SIGMETRICS*, 2002.
[20] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *USENIX Symposium on Internet Technologies and Systems*, 2003.
[21] V. Ramasubramanian and E. G. Sirer. The design and implementation of a next generation name service for the internet. In *SIGCOMM*, 2004.
[22] W. Rao, L. Chen, A. W.-C. Fu, and Y. Bu. Optimal proactive caching in peer-to-peer network: Analysis and application. In *CIKM*, 2007.
[23] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A scalable content-addressable network. In *SIGCOMM*, 2001.
[24] M. Ripeanu and I. T. Foster. Mapping the gnutella network: Macroscopic properties of large-scale peer-to-peer systems. In *IPTPS*, pages 85–93, 2002.
[25] H. J. Rka Albert and A.-L. Barabsi. Error and attack tolerance of complex networks. In *Nature 406*, pages 378–382, 27 July 2000.
[26] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, 2001.
[27] 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.
[28] 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.

[29] S. Tewari and L. Kleinrock. Entropy and search distance in peer-to-peer networks. In *UCLA Computer Science Dept Technical Report UCLA-CSD-TR050049*, November 2005.
[30] J. Wang. A survey of web caching schemes for the internet. In *ACM Computer Communication Review*, 1999.
[31] P. Yalagandula and M. Dahlin. A Scalable Distributed Information Management System. volume 34, pages 379–390, New York, NY, USA, 2004. ACM Press.
[32] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph. Tapestry: a fault-tolerant wide-area application infrastructure. volume 32, 2002.
[33] M. Zhong and K. Shen. Popularity biased random walks for peer-to-peer search under the square root principle. In *IPTPS*, 2006.

Weixiong Rao studied in the Computer Science and Engineering Department of the Chinese University of Hong Kong from October 2005 to April 2009 and passed the PhD defense in April 2009. He received the BS and MS degrees in Mechanical Engineering from North JiaoTong University and Shanghai JiaoTong University. His research interests include distributed information dissemination, publish/subscriber systems and Peer-to-Peer computing.



Lei Chen received his BS degree in Computer Science and Engineering from Tianjin University, China, in 1994, the MA degree from Asian Institute of Technology, Thailand, in 1997, and the PhD degree in computer science from University of Waterloo, Canada, in 2005. He is now an assistant professor in the Department of Computer Science and Engineering at Hong Kong University of Science and Technology. His research interests include multimedia and time series databases, sensor and peer-to-peer databases, and stream and probabilistic databases. He is a member of the IEEE.

Ada Wai-Chee Fu received the BSc degree in computer science from the Chinese University of Hong Kong in 1983 and the MSc and PhD degrees in computer science from Simon Fraser University, Canada, in 1986 and 1990, respectively. She worked at Bell Northern Research, Ottawa, from 1989 to 1993 on a wide-area distributed database project. She joined the Department of Computer Science and Engineering, Chinese University of Hong Kong, Shatin, Hong Kong, in 1993. Her research interests include database systems and data mining. She is a member of the IEEE.

Guoren Wang received his BSc, MSc and PhD degrees, in computer science, from Northeastern University, China, in 1988, 1991 and 1996, respectively. He was an assistant professor (March 1991 - March 1996) and an associate professor (April 1996 - March 1999) in the Department of Computer Science, Northeastern University, China. Currently he is a professor in the School of Information Science and Engineering, Northeastern University, China. His major research interests are XML data management, query processing and optimization, bioinformatics, high-dimensional indexing, parallel database systems, and P2P data management. He has published more than 80 research papers in international conferences and journals.