Leopard: A Locality-Aware Peer-To-Peer System With No Hot Spot

[Extended Abstract]

Yinzhe Yu*, Sanghwan Lee, and Zhi-Li Zhang
Department of Computer Science & Engineering, University of Minnesota, Minneapolis
{yyu,sanghwan,zhzhang}@cs.umn.edu

I. Introduction

A fundamental challenge in Peer-To-Peer (P2P) systems is how to locate objects of interest, namely, the *look-up service* problem. A key break-through towards a scalable and distributed solution of this problem is the *distributed hash table (DHT)*, including Chord, CAN, and Tapestry, among others. However, since both object id and network node are *randomly hashed* to a same id space, "locality-awareness" is *not inherent* in the basic DHT design. As a result, *routing stretch* of object look-up, defined as the ratio of network distance traveled by a look-up message and distance between the requester and the nearest object copy, can be high. Besides, since in standard DHT an object is randomly hashed to a single (or a few) id, the node responsible for that object can be overwhelmed by a sudden surge of user requests, often called a "flash crowd," and creating a "hot spot" in the system.

Recent research advances in Internet virtual coordinates [3], [2], [1] has shown that Internet nodes can be efficiently assigned virtual coordinates such that the inter-nodal geometric distance in the virtual space accurately approximates their real IP network distance, e.g., [3] shows that more than 90% of inter-nodal virtual distances are within 50% error margin of real network distances. This provides a tremendous opportunity in solving the aforementioned issues. For example, a straightforward application of virtual coordinates system is to gradually select close-by routing neighbors in the virtual space, thus *incrementally* improves the neighbor relationship in a P2P system such as Chord, *after* the P2P network is created.

In this paper, we propose an alternative approach of using virtual coordinates system to build a P2P system called Leopard. To inherently incorporate locality-awareness, we separate the *object id space* from the *node space*: each object is assigned a unique id in an object id space; while each node is assigned a *coordinate* in a so called *(node) geo space*, as it joins the network. During the join process, the node obtains neighbor relationships that reflects "network proximity" *from the beginning*. A simple position-based greedy forwarding scheme can then be used to route a packet between any two points in the node geo space with low routing stretch. The object id space and the node geo space are then "weaved" together via a novel hashing technique called *geographically-scoped hashing* (GSH): each object id is mapped into multiple coordinates in the node geo space with varying geographical scopes; the nodes that are closest to these points are responsible for maintaining "pointers" to the object and performing look-up queries for it. Our initial analysis and simulation results show that: i) Leopard has a constant IP-level routing stretch; this holds not only in an *average* or *probabilistic* sense, but also in the *worst-case*. ii) Leopard always locates a *near-by* copy when multiple copies exist. iii) Leopard effectively handle "flash crowd" with near optimal load balancing.

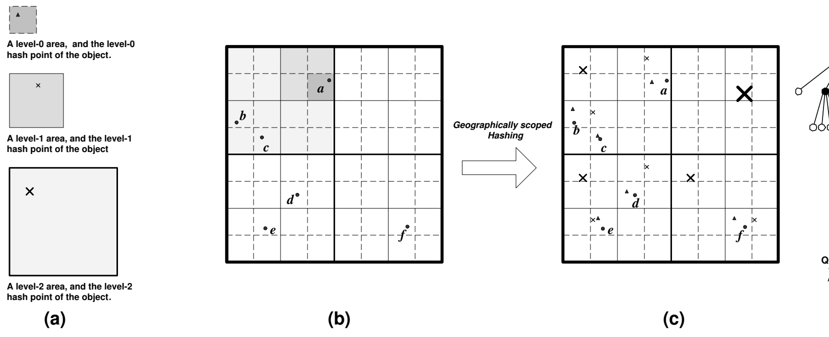
* Corresponding author.

II. Leopard Location Service

Same as standard DHT, we assign each object a unique *identifier* from an *id space*, based on either application semantics or random hashing. We use $\omega.id$ to denote the identifier of an object ω. On the other hand, nodes form the *node geo space* – a finite d-dimensional metric space – upon which a coordinate system is defined. A nodes's *coordinate* is determined when it joins the system, via a virtual coordinates service such as described in [3], [2], [1].

We introduce a *hierarchical grid* over the node geo space by dividing it into a hierarchy of (d-dimensional) *areas*: at the highest level, the entire space is the level-L area, where L is a system parameter specifying the total number of levels in the hierarchy. For $1 \leq l \leq L$, each level-l area is divided into 2^d level-$(l-1)$ areas, obtained by cutting the level-l area into half along each of the d dimensions. Fig.1(b) illustrates such a hierarchy of areas with $L = 3, d = 2$, where square areas of different levels are delineated with different line styles. For $l = 0, 1, \ldots, L$, let A_l be the level-l area containing a node a. In Fig. 1(b), we use four different levels of shades to show $A_0 \ldots A_3$. We see that $A_l \subset A_{l+1}, l = 0, ..., L-1$, and A_L is the entire node geo space. Let r_l denote the *size* of a level-l area A_l (i.e., its side length), then $r_{l+1} = 2r_l$. We will use $\mathcal{O}(A_l)$ to denote A_l's *origin* coordinate, i.e., point in A_l with the smallest coordinates.

We now introduce the concept of *geographically scoped hash* functions. For $l = 0, 1, \ldots, L$, let $\mathcal{H}_l$ be a d-dimensional random hash function with the range $[0, r_l)^d$. Given an object ω and a level-l area A_l, the *hash point* of ω under (the geographical scope) A_l is a point within A_l given as $\mathcal{H}(\omega, A_l) = \mathcal{O}(A_l) + \mathcal{H}_l(\omega.id)$. In Fig. 1(a), we illustrate the hash functions for three areas (scopes) of different levels. Fig. 1(b) shows the locations of six nodes ($a-f$), each owning a copy of the object ω. Fig. 1(c) shows the hash points of the six nodes in various areas. The node in the node geo space that is *closest* to the hash point $\mathcal{H}(\omega, A_l)$ of object ω is referred to as the *level-l pointer node* for object ω in area A_l, and is denoted by $\mathcal{P}(\omega, A_l)$. Pointer nodes of an object are responsible for maintaining object "pointers" and answering look-up queries.



A level-0 area, and the level-0 hash point of the object.
A level-1 area, and the level-1 hash point of the object
A level-2 area, and the level-2 hash point of the object.
(a)
(b)
Geographically scoped Hashing
(c)

Fig. 1. An example of geographically-scoped hashing. (a) An object is hashed to different relative points in different levels of areas. (b) Locations of six owners ($a - f$) of a same object ω. Note that the level-0, 1, 2 areas of node a are highlighted with different shades. (c) Hash points (pointer nodes) of the six owners at various levels of area.

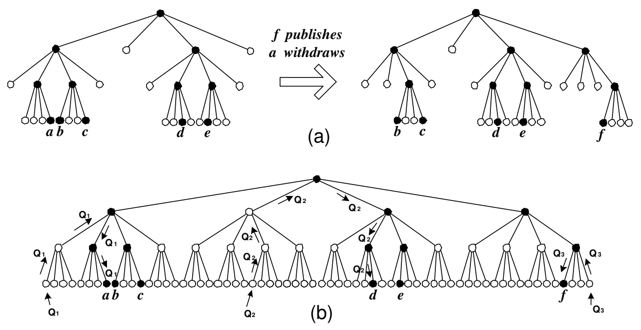


Fig. 2. (a) Initially, a object search tree (left) have five owners (a, b, c, d, and e). After f publishes and a withdraws, the object search tree becomes the one on the right. (b) Three examples (Q_1, Q_2, Q_3) of query message path.

We now illustrate how these pointer nodes cooperate to facilitate object location in Leopard. When a node a wishes to share an object ω, it *publishes* ω by "planting" a *pointer* at pointer node $\mathcal{P}(\omega, A_0)$, and becomes an *owner* of ω. For $l = 1, \ldots, L-1$, each level-l pointer node $\mathcal{P}(\omega, A_l)$ in turn computes the next-level hash point $\mathcal{H}(\omega, A_{l+1})$ and plants a pointer at $\mathcal{P}(\omega, A_{l+1})$. A pointer structure contains the object id ω, level l, and an owner information field of ω. On the level-0 pointer node $\mathcal{P}(\omega, A_0)$, the owner information field is simply a list of all owners (their IP addresses) of ω in the area A_0; while on a level-l ($l > 0$) pointer node $\mathcal{P}(\omega, A_l)$, the owner information field stores 2^d TRUE/FALSE values, indicating whether each of the 2^d lower level areas of A_l contains at least one copy of ω. In a sense the pointer nodes of an object ω form a distributed search tree embedded in the node geo space, where each edge connects $\mathcal{P}(\omega, A_l)$ to $\mathcal{P}(\omega, A_{l+1})$, as shown in Fig. 2(a). The left pane of Fig. 2(a) shows the search tree after five owners $a - e$ have published ω. Each black node in the figure represents a pointer node with a pointer stored on it. Note that the pointer nodes at the lowest level have the detailed knowledge (IP addresses) of owners, while pointer nodes at higher levels have the aggregate information of "which next lower level areas contain ω". The right pane of Fig 2(a) shows the tree after a new owner f publishes and the owner a withdraws ω. We can see that not all the L pointer nodes need to be notified when an owner publishes/withdraws, since aggregate information is stored on high level pointer nodes.

Now suppose a node x (let X_l denote its level-l areas) is interested in ω. It first sends a look-up query to $\mathcal{P}(\omega, X_0)$. Note that if a and x reside in the same level-0 area, then $\mathcal{P}(\omega, X_0)$ (=$\mathcal{P}(\omega, A_0)$) will be able to direct node x to node a for the object. Otherwise, $\mathcal{P}(\omega, X_0)$ computes the level-1 hash point and forwards the query to $\mathcal{P}(\omega, X_1)$. The process goes on recursively. If nodes a and x reside in the same level-l area, i.e., $A_l = X_l$, then the level-l pointer node $\mathcal{P}(\omega, X_l)$ (= $\mathcal{P}(\omega, A_l)$) will have a pointer to object ω. It thus directs the query to one of its next lower level pointer node with TRUE indicator in the owner information field. By tracing down a series of pointer nodes in lower and lower level, the IP address of an owner of ω can finally be pinpointed. As $A_L = X_L$, in at most $2L$ steps, node x will be able to locate the IP address of an owner. Fig. 2(b) illustrates three examples of such query paths. In fact, the distance traveled by a query message is also bounded by $O(r_l)$, where r_l is the size (side length) of a level-l area, as stated in the following theorem[1]:

Theorem 1: Suppose a node x queries for an object ω in Leopard, and the located object owner shares a level-l area with x. The total geometric distance traveled by the query message (summing up all the $2l$ steps) in the node geo space is bounded in worst case by $4\sqrt{d}r_l$.

Theorem 1 gives a *worst-case* bound on the geometric distance traveled by a query message from the requester to the *located owner*. Leopard also has the property that the located owner is near-optimal (by a constant factor) compared with the optimal owner (closest to the requester), as stated in the following theorem.

Theorem 2: Suppose a node x is querying for an object ω. Let the owner located by Leopard be s_1, and the closest (to x) owner of ω in the network be s_2. Let $D(a, b)$ denote the distance between two points a and b in the geo space. Then we have either $D(x, s_1) - D(x, s_2) \leq 2\sqrt{d}r_0$ or $\frac{D(x,s_1)}{D(x,s_2)} \leq 4\sqrt{d}$.

From Theorem 1 and 2, the distance traveled by a query message to locate an object in Leopard is at most a constant factor of the optimal distance, i.e., the geometric distance between the requester and the closest object owner in node geo space.

A. Mitigating Hot Spots

To cope with the "flash crowd" problem, Leopard imposes the following simple rule: *A node x starting the transfer (downloading) of an object ω from another node located through Leopard must publish ω for at least the duration of the object transfer.* This rule creates an object propagation model similar to the popular Bit-Torrent system. However, Leopard's object search tree helps it to achieve good load balancing naturally without maintaining complex object tracker, as used in Bit-Torrent. To analyze the problem, we define *owner service count* as the number of object transfer requests an object owner serves during its publishing time (assuming it withdraws immediately after finishing downloading the object). Ideally, this metric can be as low as 1 even with extremely high query rate, i.e., each requesting node can serve the next new

[1]Due to space limit, we refer [4] for all proof of theorems in this paper.

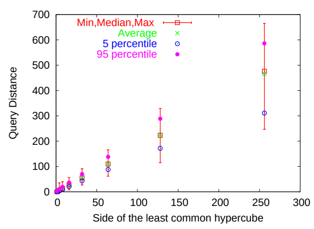


Fig. 3. Query distance vs. smallest common area size. d=2,L=8,uniform.

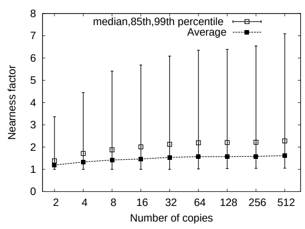


Fig. 4. Nearness factor vs. number of object copies. d=2, L=8, uniform.

rate\count	1	2	3	4	5	6	7	9	10	11	12	19
1	3160	830	64	2	0	0	0	0	0	0	0	0
4	12869	3144	205	16	5	0	0	0	0	0	0	0
16	53232	12203	679	59	7	3	0	0	1	0	0	0
64	216818	47483	2278	160	13	5	1	1	2	1	1	1

TABLE I

HISTOGRAM OF OWNER SERVICE COUNTS AT DIFFERENT QUERY RATES.

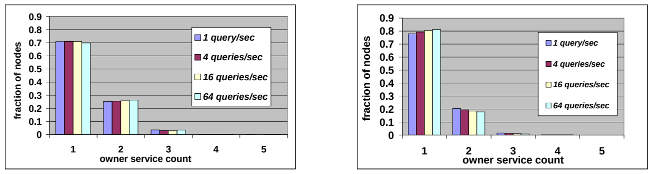


Fig. 5. Fraction of nodes that has various owner service counts. Left: uniform dist.; Right: non-uniform dist.

requester. We have the following theorem regarding Leopard's owner service count upper bound.

Theorem 3: The number of object transfer requests an object owner a serves during its publishing period is bounded by $L + n_0$, where n_0 is the number of nodes in the level-0 area A_0.

If we denote the total number of nodes in the network by N, then $N \approx n_0 \cdot (2^d)^L$, or $L = O(\frac{\log{(N/n_0)}}{d})$. Therefore, Theorem 3 guarantees that *regardless of the object request rate*, the upper bounds of the owner service count is $O(\frac{\log N}{d})$.

III. SIMULATION EXPERIMENTS

In this section, we evaluate Leopard through packet level simulations. When we generate random nodes in the node geo space, we use node distributions that models the Internet nodes (called non-uniform hereafter) in addition to a simple uniform distribution. The non-uniform node distribution is based on coordinate distributions of the GNP [3] data sets.

1) Query Distance: To verify our analysis of the Leopard *routing stretch*, we define *query distance*, which is computed by summing up the geometric distances in node geo space of each forwarding hop of a query message packet. We construct a network of 10^5 uniformly distributed nodes, and distribute $1,000$ objects into random nodes such that the i^{th} ($1 \leq i \leq 1000$) object has i copies. We generate $100,000$ queries from random node for a random target object, and record the query distances. Fig. 3 shows the distribution of the query distances grouped by the *smallest common areas* of the querying nodes and the *located* owner. The x-axis in Fig. 3 represents the size (i.e., side length) of the smallest common area, normalized by r_0. We observe a clear linear relation between the query distance and the common area size, i.e., Leopard has a constant routing stretch. In addition, the "constant" is small: less than 2 for average, and less then 2.5 for 95 percentile. Results of non-uniform distribution are similar.

2) Locating Nearby Copies: We first define a metric called *nearness factor*, which is the ratio of the distance from the querying node to the *located* owner and the distance to the actual *nearest* owner in the network. We construct a system ($d = 2, L = 8$) with 10^5 uniformly distributed nodes and 100 unique objects, each with 2^k copies in the network (we vary the value of k from 1 to 9 in nine different experiments). For each experiment, we generate 5000 queries from random nodes for a random object. Fig. 4 shows the statistics of nearness factors. The average, median, 85th percentile and 99th percentile are computed for the nine experiments with different k's. We see that the nearness factor is small: the *medians* are 1 for all k values, indicating at least half of the querying nodes being able to find the actual *nearest* copy.

3) Flash Crowd Mitigation: We generate queries to a *single* popular object with different query rates. The simulation starts with a 10^5 node network. After a warm up period of 4000 seconds, a random chosen node publishes the target object. After that, random queries are generated with a Poisson distribution with a mean rate of 2^q queries per second (q varies from 0 to 6). We performs five separate runs, each lasts $1,000$ seconds. Object downloading time is fixed at 100 seconds and nodes always withdraw immediately after downloading. Two different sets of parameters are used: i) $d = 2, L = 8$, uniform distribution and ii) $d = 8, L = 13$, non-uniform distribution. Table I shows the histograms of nodes with different owner service counts at four different query rates for the non-uniform case. The histograms are accumulated over five runs, and nodes with zero object service count are not shown. For example, when query rate is 1/s, a total of 5020 $(3160 + 2 \times 830 + 3 \times 64 + 4 \times 2)$ requests are served in the five runs. Only two nodes ever served four requests, the largest service count. For all query rates, the vast majority of nodes (99.9%+) serve three requests or less. We next plot the fraction of nodes has owner service counts of 1 through 5, as shown in Fig.5 (uniform case on the left and non-uniform on the right). We observe that the fractions are almost identical across all four query rates, indicating Leopard's capability to achieve near-optimal load balancing regardless of query rate.

IV. CONCLUSIONS

We have proposed to incorporate locality-awareness inherently into P2P network, and designed Leopard with this paradigm. We have demonstrated its many desirable properties: 1) constant routing stretch in worst case; 2) always locates a nearby copy of object; 3) effectively cope with "flash crowd."

REFERENCES

[1] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *Proc. of ACM SIGCOMM*, 2004.
[2] H. Lim, J. C. Hou, and C.-H. Choi. Constructing internet coordinate system based on delay measurement. In *Proc. of ACM IMC*, Oct. 2003.
[3] T. E. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *Proc. of IEEE INFOCOM*, June 2002.
[4] Y. Yu, S. Lee, and Z.-L. Zhang. Leopard: A locality-aware peer-to-peer system with no hot spot, Tech. Report CSE Dept., U of Minnesota, 2004.