Similarity Discovery in Structured P2P Overlays

Hung-Chang Hsiao Chung-Ta King*

Department of Computer Science
National Tsing-Hua University
Hsinchu, Taiwan 300
hchsiao@cs.nthu.edu.tw

Abstract

Peer-to-peer (P2P) overlays are appealing, since they can aggregate resources of end systems without relying on sophisticated infrastructures. Services can thus be rapidly deployed over such overlays. Primitive P2P overlays only support searches with single keywords. For queries with multiple keywords, presently only unstructured P2P systems can support by extensively employing message flooding.

In this study, we propose a similarity information retrieval system called Meteorograph for structured P2P overlays without relying on message flooding. Meteorograph is fault-resilient, scalable, responsive and self-administrative, which is particularly suitable for an environment with an explosion of information and a large number of dynamic entities. An information item stored in Meteorograph is represented as a vector. A small angle between two vectors means that the corresponding items are characterized by some identical keywords. Meteorograph further stores similar items at nearby locations in the P2P overlay. To retrieve similar items, only nodes in nearby locations are located and consulted. Meteorograph is evaluated with simulation. The results show that Meteorograph can effectively distribute loads to the nodes. Discovering a single item and a set (in size k) of similar items takes $O(\log N)$ *and* $(\frac{k}{c})\cdot O(\log N)$ *messages and hops respectively, where N is the number of nodes in the overlay and c is the storage capacity of a node.*

1. Introduction

Peer-to-peer (P2P) overlays have recently attracted much attention due to features such as self-administration, reliability and responsiveness. They can efficiently aggregate resources across the Internet without sophisticated management. Each node in a P2P system contributes some resources (storage space or processor cycles, for example) to the system. Functionally, the nodes are identical—they can act as a client, a server or a router. Participating nodes from various administrative domains may dynamically join and depart the system. Example P2P systems include CAN [14], Chord [16], Freenet [3], Gnutella [9], Pastry [15], Tapestry [19] and Tornado [11].

P2P overlays can be classified as *unstructured* and *structured*. Unstructured P2P overlays such as Gnutella and Freenet do not embed a logical and deterministic structure to organize the peer nodes. Consequently, they need a certain kind of message flooding to search for interested items stored in the overlay. For example, Gnutella adopts a breath-first approach to flood the requests, while Freenet uses a depth-first approach. To prevent the high cost of flooding the entire network, both systems use a time-to-live (TTL) value to limit the scope of a search.

In contrast, structured P2P overlays such as CAN, Chord, Pastry, Tapestry and Tornado manage the peer nodes with an implicit logical and deterministic structure. CAN is based on a multi-dimensional coordinate space, and the others are based on an *m*-way tree. These systems provide powerful lookup services by managing hash key and value pairs. A hash key is generated by applying a uniform hash function to the searched keyword. Given a hash key, a lookup request can be resolved by a node whose hash key is the closest to the requested key.

Structured P2P overlays offer several desirable features. First, they do not rely on the flooding mechanism and, therefore, do not generate large network traffic. A lookup request in most proposed overlays takes $O(\log N)$ hops and messages. Second, a lookup request can be resolved with a high probability and the associated cost is predictable. On the other hand, unstructured overlays cannot discover a requested item if this item is out of the search scope. Even if requested items can be discovered, the cost is unpredictable. Third, results of a search are deterministic in structured overlays. In unstructured overlays, different peers may receive different results when issuing the same search request.

A serious problem with structured overlays is that they can only support searches with a single keyword. For example, they can search for and return all papers with the keyword "distributed processing". This is done by first obtaining the hash key of "distributed processing", and then storing all such papers in a peer node whose node ID is the closest to the hash key. This creates several problems. First, if there are many papers on "distributed processing", then the hosting peer node will be overloaded. Second, if a paper on "distributed processing" can also be characterized as "computer architecture", then we have to decide which keyword to use to publish the paper. This then precludes the use of the other keyword to find the paper, unless we duplicate the paper to both sites. Third, we cannot issue a search with multiple keywords, such as <"distributed processing", "computer architecture">, and find all papers that exactly match this query. It is even difficult to find papers characterized by <"distributed processing", "computer architecture", "something else">.

One solution is to build multiple sub-overlays on top of the structured overlay. Each sub-overlay handles items that are characterized by the same keyword. To search with multiple keywords, the corresponding sub-overlays are consulted and each return items that match a specific keyword. The inquirer then examines the received items and filters out those that do

* This work was supported in part by the National Science Council, R.O.C., under Grant NSC 90-2213-E-007-076 and by the Ministry of Education, R.O.C., under Grant MOE 89-E-FA04-1-4.

not match all the specified keywords. Clearly, this approach will result in large traffic in transmitting items that do not fully match the specified keywords. Besides, if the number of keywords in the system is large, this approach requires a huge number of overlays. A node that participates in *k* overlays will require *k* times the overhead to maintain these sub-overlays.

In this study, we propose a novel information retrieval system, *Meteorograph*, for searches with multiple keywords (or similarity searches). It is based on a structured P2P "storage" overlay called Tornado [11]. Meteorograph characterizes an item as a vector in the vector space model [1] and stores the item in a single structured overlay. To map items into the structured overlay, each item in Meteorograph is transformed to a single value called the *absolute angle*. Two items are "similar" if they share some common keywords, and the two corresponding vectors in the vector space have a very small angle. By controlling the locations, represented by absolute angles, in which items are stored, Meteorograph can rapidly locate a search item. Moreover, it can aggregate similar items together at nearby locations in the overlay.

The contributions of this study are as follows.

- A reliable information retrieval system, Meteorograph, is proposed. It can be built on top of structured P2P overlays, especially those using a linear hash addressing space.
- Meteorograph can aggregate similar items in an overlay. It can thus provide similarity searches that cannot be supported by a naive structured overlay. The evaluation results indicate that Meteorograph takes only $(\frac{k}{c}) \cdot O(\log N)$ messages and hops to discover *k* items, where *N* is the number of peers in an overlay and *c* is the space capacity of a node. Moreover, Meteorograph can discover all items stored in an overlay that match the specified keywords.
- Meteorograph avoids problems commonly found in unstructured P2P overlays for similarity searchers. Note that many such unstructured P2P overlays have been proposed, e.g., associative overlay [4], PlanetP [7], routing index [5], semantic overlay [6] and YAPPERS [8]. Their problems are large network traffic due to message flooding, limited search scope, and nondeterministic research results. Meteorograph avoids these problems.
- In a P2P system, if loads are not uniformly distributed to the system, some nodes may be overloaded with published items. Meteorograph can evenly distribute items into the structured overlay. The load-balancing feature enables a search of single items to complete in $O(\log N)$ hops and messages.
- Meteorograph supports ranked searches, such as finding the *k* most similar items of a given key.

One problem with the vector space model used in Meteorograpah is that to add a new item may result in expansion of the vector space. Each published item then must be republished. Meteorograpah can simply employ a universal set of keywords in a dictionary to characterize each item without using a high-dimensional vector space. It thus needs not republish items.

To our knowledge, Meteorograph is the first system to implement similarity searches for structured P2P overlays that especially employ single-dimensional hash address space (such as Chord [16], Pastry [15], Tapestry [19] and Tornado [11]). We also provide an extensive experimental study on the performace of Meteorograph. The remainder of the paper is organized as follows. Section 2 overviews the design concept of Meteorograph. Section 3 presents the Meteorograph design. Evaluation for Meteorograph is given in Sections 4, and Section 5 discusses the related works. Conclusions of the paper are given in Section 6, with possible future research directions.

2. Overview

In the vector space model [1], given a set of items $S = \{t_1, t_2, t_3, \cdots, t_n\}$, a set of keywords $K = \{k_1, k_2, k_3, \cdots, k_m\}$, and the associated weights $W = \{w_1, w_2, w_3, \cdots, w_m\}$, each item t_i in S can be represented as a vector $\vec{d}_i = [v_1, v_2, v_3, \cdots, v_m]$, where $v_j = w_j$ ($1 \le j \le m$) if k_j can characterize $\vec{d}_i$; $v_j = 0$ otherwise. Thus, the set $M = \{\vec{d}_1, \vec{d}_2, \vec{d}_3, \cdots, \vec{d}_n\}$ can be used to represent *S*.

Given a query vector $\vec{q} = [q_1, q_2, q_3, \cdots, q_m]$ to search for a set of similar items *U* from *S*, we can apply the dot product (denoted by $\bullet$) to $\vec{q}$ and each $\vec{d}_i$ in *S*, obtaining the result $r = \vec{q} \bullet \vec{d}_i$. The angle ∂ between $\vec{q}$ and $\vec{d}_i$ is calculated by $\partial = \cos^{-1}(r)$. Note that $0^\circ \le \partial \le 180^\circ$. Cosine is thus a one-to-one and onto function, and the inverse function, $\cos^{-1}$, exists. The value ∂ can then be used to evaluate whether the two vectors are similar. If ∂ is smaller than a predefined threshold τ, we say that $\vec{q}$ and $\vec{d}_i$ are similar and thus $\vec{d}_i$ must be in the set of *U*. Other similarity measurements are possible, for instance, finding top-ten items similar to a query from *S*.

Meteorograph is based on the vector space model and employs the dot-product concept. It logically maintains a set of nodes in a *half circle* over a 2-dimensional *X*-*Y* space. Each item (denoted by the vector $\vec{d}$) in Meteorograph is represented as an angle ϖ with respect to the axis $Y = 0$ by $\varpi = \cos^{-1}(\vec{d} \bullet \vec{x})$. $\vec{x}$ is the projection vector of $\vec{d}$ in the vector space *M*. Items in *S* that are similar will have nearly identical angle ϖ and will thus be *published* in the same vicinity of the half circle (i.e., the nearby nodes). To retrieve a set of items by the giving query vector, Meteorograph calculates the angle between the query vector and the unity $\vec{1}$. Then it locates the node (or a set of nearby nodes) in the circle to retrieve those items closely matching the query.

3. Meteorograph

Meteorograph is based on Tornado. However, due to space constrain, Tornado can be referred to [11].

3.1 Absolute Angle

Given a vector $\vec{d} = [d_1, d_2, d_3, \cdots, d_m]$ in an *m*-dimensional space *M*, we define the *absolute angle*, θ, as

$$\theta = \sqrt{\frac{\theta_1^2 + \theta_2^2 + \theta_3^2 + \cdots + \theta_m^2}{m}}, \quad (1)$$

where θ_i is the angle between $\vec{d}$ and the unit vector $I_i = [0_1, \cdots, 0_{i-1}, 1_i, 0_{i+1}, \cdots, 0_m]$, for $1 \le i \le m$. Note that $0^\circ \le \theta \le 180^\circ$. The angle θ_i is calculated as

$$\theta_i = \cos^{-1}\left(\frac{\vec{d} \bullet \vec{d}_{proj(i)}}{|\vec{d}||\vec{d}_{proj(i)}|}\right), \quad (2)$$

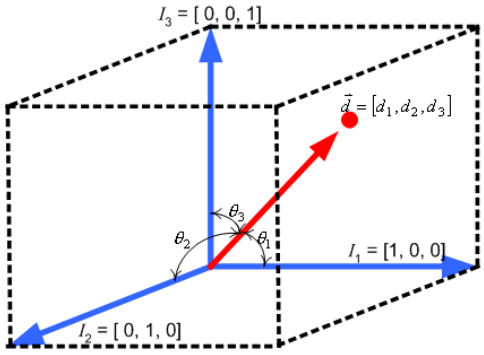


Figure 1: An example of a 3-dimensional vector space

where $\vec{d}_{proj(i)} = [d_{1i}, d_{2i}, d_{3i}, \cdots, d_{mi}]$ is the projection vector of $\vec{d}$ onto the subspace spanned by I_i. Let $|\vec{d}| = \sqrt{\sum_{i=1}^{m} d_i^2}$ and $|\vec{d}_{proj(i)}| = \sqrt{\sum_{k=1}^{m} d_{ki}^2}$. We have $\vec{d} \bullet \vec{d}_{proj(i)} = \sum_{k=1}^{m} d_i d_{ki}$, where $\vec{d}_{proj(i)}$ is

$$\vec{d}_{proj(i)} = \left(\vec{d} \bullet \frac{I_i}{|I_i|} \right) \frac{I_i}{|I_i|}. \quad (3)$$

Figure 1 illustrates an example of a 3-diminsional vector space and the angles between a vector d and the linear subspaces spanned by I_1, I_2 and I_3.

Using the vector space model, we can see that items with similar vector representations have nearly identical absolute angles. Meteorograph exploits this property to aggregate similar items by publishing them to logically clustered nodes in Tornado.

3.2 Naming

Given a vector $\vec{v} = [v_1, v_2, v_3, \cdots, v_m]$ that represents a query or an item, Meteorograph computes its absolute angle θ_v using Equation 1. The corresponding hash key, $\hbar_v$ of $\vec{v}$ in Tornado is then calculated as follows

$$\hbar_v = \left\lceil \left(\frac{\theta_v}{\pi} \right) \Re \right\rceil. \quad (4)$$

From Equations 2 and 3, $\vec{v}$'s projection vector in the subspace spanned by I_i is $\vec{v}_{proj(i)} = [0, \cdots, 0, v_i, 0, \cdots, 0]$ for $1 \le i \le m$. Thus Equation 4 can be further simplified as

$$\hbar_v = \left\lceil \left(\frac{\left(\sum_{i=1}^{m} \left(\cos^{-1} \left(\frac{v_i^2}{\sqrt{A} v_i} \right) \right)^2 \frac{1}{m} \right)^{\frac{1}{2}}}{\pi} \right) \Re \right\rceil, \quad (5)$$

where $A = \sum_{i=1}^{m} v_i^2$.

3.3 Publishing and Searching

To publish an item represented by the vector $\vec{p}$, Meteorograph performs the following steps.

- **Step 1:** Resolve the item's hash key $\hbar_p$ via Equation 5.
- **Step 2:** Publish the item to a node n_p with the hash key closest to $\hbar_p$.
- **Step 3:** If n_p cannot satisfy the publishing request due to a shortage in its storage space, n_p replaces the least alike item with the published item $\hbar_p$. Node n_p then asks its closest neighbor to help store the replaced item. That neighbor then performs similar operations. Note that the originating node of the publishing request can specify a *hop count* value to constrain the maximum number of neighbors visited. If the publishing request can be accomplished within the specified hop count, the publishing is successful. Otherwise, the originating node informs the application of the failure of publishing.

The replacement policy in Step 3 guarantees that most simi-

```
_publish (vector p̄, payload d, integer hop)
  // resolve p̄'s hash key via Equation 5
  ħ_p = _resolve ( p̄ );
  // issue a message with the publishing request from s to n
  // towards the node closest to ħ_p
  if (_forward (s, n, ħ_p, d, hop, "publish") is failed)
    inform the application of the failure of publishing;

_retrieve (vector q̄, integer amount)
  // resolve q̄'s hash key via Equation 5
  ħ_q = _resolve ( q̄ );
  // issue a message with the retrieving request from s to n
  // towards the node closest to ħ_q
  return _forward (s, n, ħ_q, q̄, amount, "retrieve");

_forward (node s, node n, key id, payload d, integer c, request type)
  // Does there exist a node with the hash key closest to id?
  if (∃t ∈ n's routing table such that p is closer to id)
    // forward to the node with the hash key closer to id
    _forward (s, t, ħ_p, d, hop, "publish");
  else
    // n is the node with the hash key closest to id
    switch (type)
      case "publish":
        if ( c = 0 )
          reply a publishing failure to s;
          return;
        if (n' storage space is not available)
          replace the least similar item u in n with d;
          b = n's closest neighbor;
          _forward (s, b, ħ_u, u, c − 1, "publish");
        else
          // adopt VSM or LSI for local indexing
          store d in n;
      case "retrieve":
        // manipulate the local index of n
        r = the number of most relevant items to d̄;
        send the resultant matched items to s;
        if ( c − r > 0 )
          b = n's closest neighbor;
          _forward (s, b, ħ_q, d̄, c − r, "retrieve");
```

Figure 2: The publishing and retrieving algorithms

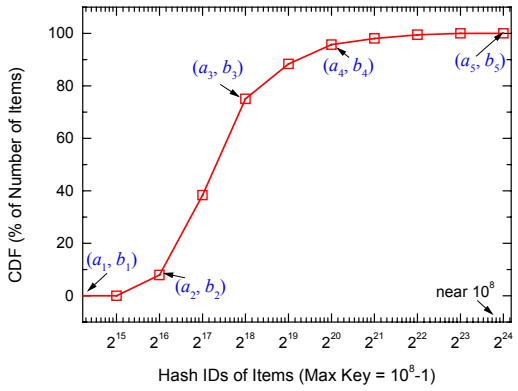


Figure 3: The CDF versus hash keys that represent 0.5% of the items out of the collected traces

lar items are clustered together and stored in the same node or the nearby nodes. Figure 2 presents the algorithm (see the **_publish**). As mentioned in Section 3.1, similar items have nearly identical absolute angles. They thus have similar hash keys and are published to the nearby nodes. Note that nodes may further implement the vector space model (VSM) or the latent semantic indexing (LSI) to manipulate the items stored locally.

To search for items that match the given keywords, the issuing node simply calculates the hash key representing the query vector q (see the **_retrieve**). Then, it forwards the search request to node n_q whose hash key is the closest to the hash key of q. Depending on the "amount" of items requested, n_q can simply look up its local index to retrieve the requested items. If n_q cannot fulfill the designated amount, it consults its closest neighbor to further process the query. Since items that are more alike will replace those more dissimilar (see the **_publish** algorithm), the most similar items must be stored in a node or a set of close nodes. Meteorograph exploits this aggregation feature and combines it with the linear ordering relationship between nodes of Tornado. It can thus discover the most similar k items for a given key.

3.4 Load Balance

A naive structured overlay names each participating peer by a uniform hash function. It publishes an item to a peer whose hash key is the closest to the key representing that item. If the distribution of the items' hash keys is uniform, each peer will host about the same amount of items. However, if some keywords are particularly popular, the distribution of the items may be biased towards some particular peers. This thus causes unbalanced load in the peer nodes and renders the hash addressing space underutilized.

By investigating a small sampled data set, Meteorograph tries to evenly scatter hash keys to the whole hash addressing space (Section 3.4.1). To further relieve the hot regions in the hash addressing space, Meteororgraph places more nodes into those regions to share the load (Section 3.4.2). We assume that the sample data set examined by Meteorograph can be obtained from an operating overlay such as Gnutella in advance.

3.4.1 Exploiting Unused Hash Space

Items may share some identical keywords. Popular keywords may result in skew distribution of the absolute angles. Figure 3 depicts a cumulative distribution function (CDF) of the number of items versus hash keys that represent 0.5% of the items out of the collected traces (see Section 4). It shows that near 65% and 20% of items are represented by keys from 2^{16} to 2^{18} and from 2^{18} to 2^{20}, respectively. These hash keys only takes 1.9% and 7.8% of the hash addressing space. That means 85% of items will be published to 5.9% of nodes that participate in the system.

Meteorograph tries to evenly scatter items into the system without scrambling those similar items that are aggregated. As Figure 3 shows, Meteorograph firstly identifies several points of knees (i.e., (a_1,b_1), (a_2,b_2), (a_3,b_3), (a_4,b_4) and (a_5,b_5)) for the distribution. A hash key, h, of an item is recalculated by applying a linear function f that is defined as follows

$$f(h) = \Re\left(a_i + (a_j - a_i)\frac{h - b_i}{b_j - b_i} \right), \qquad (6)$$

where $b_i \le h < b_j$, $a_i = CDF(b_i)$ and $a_j = CDF(b_j)$.

In this study, five points of knees are selected, that are $(0,0)$, $(0.079,2^{16})$, $(0.079,2^{16})$, $(0.75,2^{18})$, $(0.957,2^{20})$ and $(1,10^8)$.

3.4.2 Relieving Hot Regions

Figure 4 shows the CDF function after each item is named by applying Equation 6. It indicates that Meteorograph thoroughly exploits the hash keys provided by the structured overlay. Ideally, the CDF should scale linearly with a slope equal to one. That means the hash keys that actually represent items are uniformly distributed and therefore each peer node in the system perceives nearly identical workload.

Since some keywords are particularly hot, the hash keys of those items characterizing by those hot keywords are thus not uniformly scattered. Consequently, some particular regions (denoted as *hot regions*) in the hash addressing space may contain excessive items. Meteorograph solves this problem by introducing more nodes into those hot regions.

The idea is to firstly identify several points of knees, e.g., (x_{B1},y_{B1}), (x_{B2},y_{B2}), (x_{B3},y_{B3}), (x_{C1},y_{C1}), (x_{C2},y_{C2}) and (x_{C3},y_{C3}) in Figure 4, for the corresponding hot regions (**B** and **C**). Meteorograph then maps more nodes to hash keys in the range between x_{B1} and x_{B2} than those between x_{B2} and x_{B3} for the hot region **B**. Similarly, for region **C** more nodes with hash keys between x_{C1} and x_{C2} are mapped.

Figure 5 shows the naming algorithm for a joining node. The joining node employs a uniform hash function to name itself when the hash key received is outside a hot hash region. Otherwise, it will use a hash key within a hot region to join. It thus recalculates its representing hash key based on a probability (i.e., r) and the *degree of hotness* in that hot region. For instance, suppose in Figure 4 that node v randomly obtains a hash key k which is between x_{B1} and x_{B2} within the hot region **B**. Assume that the degrees of hotness in **B** are 0.8 and 0.2

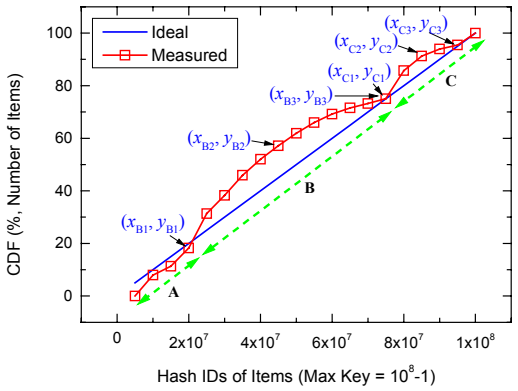


Figure 4: The CDF versus hash keys after applying Equation 6 to name the sampled items

for the two sub-regions $[x_{B1}, x_{B2})$ and $[x_{B2}, x_{B3})$, respectively. Node v randomly regenerates its representing hash key within $[x_{B1}, x_{B2})$ if it evaluates the probabilistic value (r) and finds it less than 0.8. Otherwise, it generates a hash key in $[x_{B2}, x_{B3})$.

Let (x_{ia}, y_{ia}) and (x_{ib}, y_{ib}) be the two subsequent points of knees that identify a sub-region $[x_{ia}, x_{ib})$ of a hot region G_i. The degree of hotness, p_{ia}, is defined as

$$p_{ia} = \frac{y_{ib} - y_{ia}}{y_{it} - y_{i1}}, \quad (7)$$

where $x_{ia} < x_{ib}$, y_{it} is the largest CDF value in G_i. Clearly, $\sum_{1 \le j < t} p_{ij} = 1$. The degree of hotness is proportional to the difference of CDFs corresponding to the two subsequent knees. Consequently, with a higher probability, Meteorograph enables nodes with hash keys within the hot sub-regions to participate in the system.

This study identifies two hot regions (i.e., **B** and **C**) based on the sampled item set. For **B**, 12 knees are used, that are $(2 \cdot 10^7, 18)$, $(2.5 \cdot 10^7, 31)$, $(3 \cdot 10^7, 38)$, $(3.5 \cdot 10^7, 46)$, $(4 \cdot 10^7, 52)$, $(4.5 \cdot 10^7, 57)$, $(5 \cdot 10^7, 62)$, $(5.5 \cdot 10^7, 66)$, $(6 \cdot 10^7, 69)$, $(6.5 \cdot 10^7, 72)$, $(7 \cdot 10^7, 73)$ and $(7.5 \cdot 10^7, 75)$. For **C**, six knees are selects, that include $(7.5 \cdot 10^7, 75)$, $(8 \cdot 10^7, 86)$, $(8.5 \cdot 10^7, 91)$, $(9 \cdot 10^7, 94)$, $(9.5 \cdot 10^7, 95)$ and $(10^8, 100)$.

Note that a node intending to join in a structured overlay needs to consult first a bootstrap node. This bootstrap node is responsible for maintaining information of the investigated items. The information includes the identified knees to exploit the unused hash addressing space (Section 3.4.1) and to relieve the hot regions (Section 3.4.2). When a joining node receives such information from the bootstrap node, it calculates its representing hash key using Equation 7. After it joins the system, it publishes items using Equation 6 based on this statistical information.

3.5 Optimizations for Similarity Search

3.5.1 First Hop

Consider a search using multiple keywords. Meteorograph resolves the representing vector and then issues a query with the corresponding hash key of the vector (**_retrieve** in Figure 2). However, if the number of keywords specified by the query is far smaller than that characterizing the published items, the resultant hash key of the query vector will be distant from those of the matching items.

Our solution for this problem is as follows. Before a node issues a search with multiple keywords, it first selects an item that matches the designated keywords from a given sample data set such that this item's representing hash key is the smallest. This node then sends this query with the designated keywords towards a node whose hash key is the closest to the resolved hash key. The latter node then performs a local search and uses the **_forward** algorithm to forward the query.

We expect that the size of the sampled data set in a node is small. This data set can be stored in the bootstrap node and downloaded to a new node at joining.

// given a set of hot regions denoted by $G = \cup\{G_i\}$ and a set
// of knees $K = \cup\{K_i\}$, where each region G_i is associated
// with a $K_i = \{(x_{i1}, y_{i1}), (x_{i2}, y_{i2}), \cdots, (x_{it}, y_{it})\}$
_name ()
// pick a hash key k by a randomly hash function, e.g., SHA-1
k = **_random ()**;
// determine k whether is within a hot region of hash address
if (k is within a hot region G_i)

$$p_{ij} = \frac{y_{i(j+1)} - y_{ij}}{y_{it} - y_{i1}}, \text{ for all } 1 \le j < t;$$

Let r be a random value between 0 and 1;

Let $1 \le s < t-1$ such that $\sum_{u=1}^{s} p_{iu} \le r < \sum_{u=1}^{s+1} p_{iu}$;

while ($x_{is} \le k < x_{i(s+1)}$ is not true)
k = **_random ()**;
return k;

Figure 5: The naming algorithm for a peer node

3.5.2 Directory Pointers

Meteorograph uniformly distributes items to the system in which each node obtains its represented hash key using a randomly uniform hash function except those appearing in hot regions. This uniformity leads to discover items that match specified keywords by crawling the entire system. Rather than merely publishing items with represented hash keys by applying Equation 6, each Meteorograph node additionally publishes a *directory pointer* associated with each published item. A directory pointer comprises of the associated item's represented hash key that is resolved by Equation 6 and the keywords that characterize the item. The represented hash key of a directory pointer, however, is the associated item's represented hash key by applying Equation 5. Consequently, Meteorograph aggregates directory pointers of similar items, but evenly distribute items into the system. A similarity search can be thus firstly forwarded to a node whose hash key is closest to the key resolved by applying Equation 5 to the corresponding query vector. The node that receives the query then performs a local search on locally stored items and directory pointers. If the associated keywords with a directory pointer satisfy the query, the node forwards this query to a node whose hash key is closest to the hash key indicated by the directory pointer.

We believe that a directory pointer is quite small in size and Tornado [11] has provided directory pointers that can thus leverage similarities searches. Clearly, to discover a node that stores an item matching the search keywords takes $2 \cdot O(\log N)$ hops and messages, i.e., $O(\log N)$ hops and messages to discover a node responsible for the directory pointer and $O(\log N)$ hops and messages to locate a node that stores the matching item. Hence, consider a similarity search in size k (i.e., discover k items). Assume a worst setting in which k similar items are stored in k various nodes. Such a search in Meteorograph takes $(k+1) \cdot O(\log N)$ messages[1], i.e., it takes $O(\log N)$ messages to send the query to the node hosting the directory pointer and $k \cdot O(\log N)$ messages to discover all k items. Possibly, these k discovery requests can be issued in parallel and this leads to $2 \cdot O(\log N)$ hops to search these k items. Meteorograph, however, does not blindly issue query request in parallel since k parallel discovery requests may redundantly sent to those nodes that have received the query if some of k items are stored at the same node. Instead, node a responsible for those matched directory pointers issues one query at a time to node b that can provide the matched items. Node a waits for a reply that involves the number (say k') of items matching the keywords specified by the query from node b and these items' represented hash keys by Equation 5. Node a then issues the same query to another node d for those undiscovered items if $k - k' > 0$. The search is complete, otherwise. This scheme concludes that a similarity search takes $(1+\frac{k}{c}) \cdot O(\log N) = O(\log N) + (\frac{k}{c}) \cdot O(\log N)$ messages and sequential hops, where c is per node mean storage space[2].

3.6 Reliability

Meteorograph leverages data reliability by constantly replicating and maintaining k replicas for each data item. The probability of completely losing a given data item is thus $1/p^k$, where p is a ratio to lose a particular replica. Once a virtual home receives a publishing request, it will firstly construct $k-1$ routes to $k-1$ virtual homes whose IDs are numerically closest to itself. To publish replicas from a virtual home, $k-1$ publishing requests with the hashing keys are routed to the replication homes. The virtual home will periodically monitor these replicas via the associated $k-1$ vectors. Since a data owner will periodically republish data items it generated, the corresponding virtual home also needs to periodically republishing replicas to $k-1$ nodes. If a virtual home fails, subsequent requests to the virtual home will be forwarded to one of its replicas by utilizing Tornado's routing infrastructure, i.e., one of the virtual homes responsible for the replications will have the numerically closest home ID to the requested data ID.

3.7 Changes of Vector Space

Consider adding a new item to a vector space. Possibly, since the keywords characterizing the newly introduced item may not appear in the keyword set K, K must be expanded to include those new keywords. This thus varies each absolute angle of those previously published items. That means items need to be republished. If the number of published items is huge, this may overwhelm an overlay by generating a huge amount of traffic for republishing.

Meteorograph does not need to republish each item stored in an overlay. It simply uses a comprehensive set of keywords from a dictionary. This is based on the assumption that each item can be characterized by the words that appear in the dictionary. To publish an item or search a set of items, the absolute angle that represents an item can be simply calculated by Equation 5. Clearly, a vector that represents an item in Meteorograph must be quite sparse and thus needs no sophisticated computations to calculate the corresponding absolute angle.

4. Performance Evaluation

Meteorograph is evaluated by simulation. Since there is no publicly available keyword-item data set, we use another similarly structured "market-basket" data set, the Web access log from the World Cup Web Site on July 24 in 1998, to synthesize the desired workload. The Web log comprises of a large number of requests and each logs a Web object (for example, a

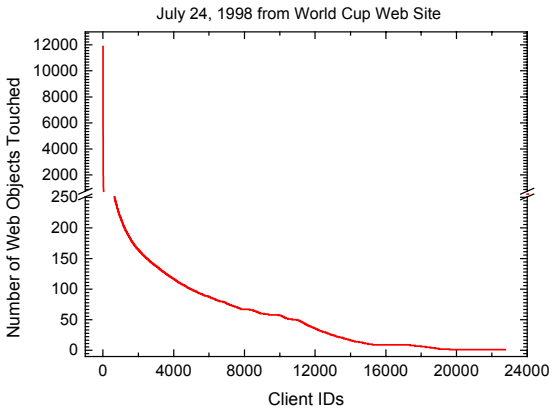


Figure 6: The number of web objects accessed in decreasing order versus the client IDs

Table 1: The statistics of the World Cup Web logs on July 24, 1998

Number of clients	2,760K
Number of Web objects accessed	89K
Average number of Web objects accessed by a client	43
Maximum number of Web objects accessed by a client	11,868
Minimum number of Web Objects accessed by a client	1

[1] Ideally, a Gnutella-like flooding scheme without TTL requires $N-1$ messages. This is assumed that each node has a global knowledge about which node has received the query request and knows how to forward the query to those nodes that have not received the query.

[2] Given a constant c, When $k << Nc$, Meteorograph considerably outperforms a Gnutella-like system in terms of messages since $(1+\frac{k}{c}) \cdot O(\log N) \approx O(\log N) < N-1$. When $k >> Nc$, $(1+\frac{k}{c}) \cdot O(\log N) \approx k \cdot O(\log N) >> N$. Note that all the nodes along a query route in Meteorograph may be the closest neighbor (Figure 2) of each other. Such a query will then take nearly $\frac{N}{2} \cdot O(\log N)$ messages rather than $k \cdot O(\log N)$ if each node can forward the query according to its directory pointers.

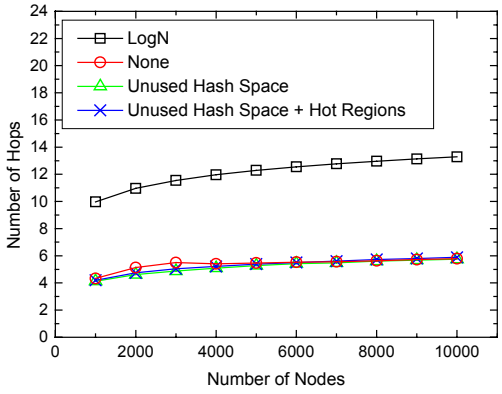


Figure 7: The performance of searching for a single keyword

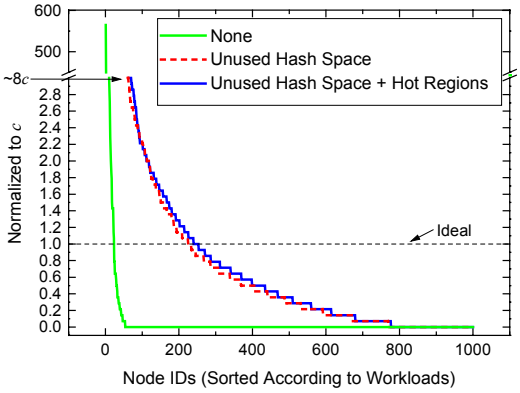


Figure 8: The load of each node

Web page, an icon, etc.) accessed by a client. We refer to those Web objects as the *keywords* and clients as the *items* published by nodes. This thus allows constructing a matrix of Web objects (keywords) versus clients (items), where the number of Web objects and clients are about 89K and 2,760K, respectively. We assume that each item has the identical size. Figure 6 shows the distribution of the number of Web objects versus the accessing IDs of clients. The resulting statistics is summarized in Table 1. Each client accesses 43 Web objects in average, i.e., each item is characterized by 43 keywords.

The structured P2P overlay (i.e., Tornado) simulated has the number of peers from 1,000 to 10,000 nodes (N). The 2,760K items with associated 89K keywords are published to the simulated overlays. Note that ideally each peer node simulated can be responsible for $c \approx \frac{2{,}760{,}000}{N}$ items. The hop count of each publishing is infinite, i.e., all 2,760K items are completely published to the system.

4.1 Discovery of a Single Item

We firstly investigate the performance of exactly searching by randomly picking a node from the overlay to retrieve a randomly selected item from 2,760K ones. The simulator measures the number of hops taken by each query. There are total 100K queries studied and the metrics (i.e., the number of hops) measured are averaged.

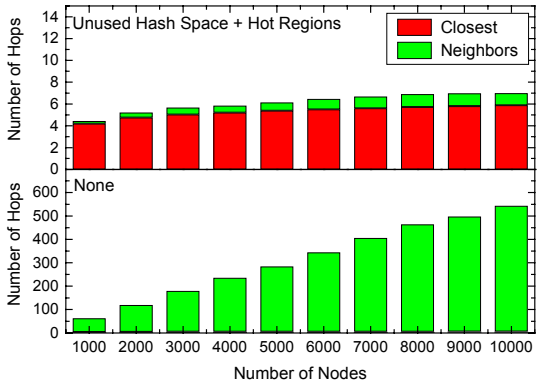
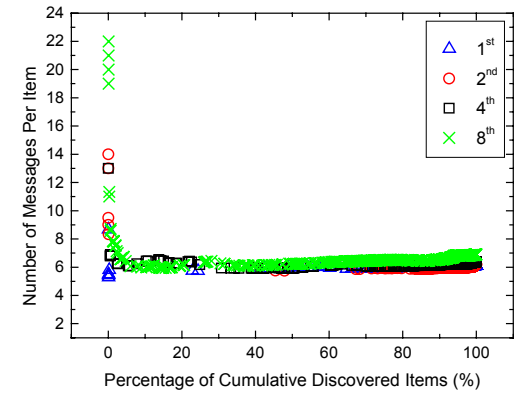


Figure 9: The effect of load balancing

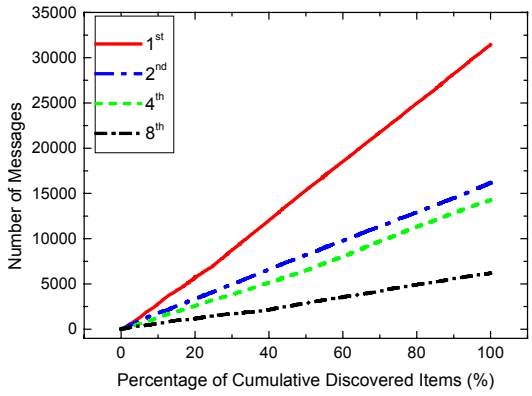
Figure 7 depicts the simulation results. Notably, each node simulated is equipped with infinite storage space (the effect of limited storage capacity is presented later). "None" denotes that the system is not optimized by any schemes for placement of items while "Unused Hash Space" and "Unused Hash Space + Hot Regions" represent the system is optimized by the naming scheme for an item (Section 3.4.1) and the one for a node (Section 3.4.2), respectively. The results present that "None", "Unused Hash Space" and "Unused Hash Space + Hot Regions" can retrieve a particular item in $O(\log N)$ hops and thus messages.

The workload of each node in terms of a ratio from the number of items the node stores to c is further investigated. Figure 8 illustrates the results for a system with 1000 nodes. It shows that "None" cannot uniformly place items to each peer. Most items are stored at a few of nodes in the system. "Unused Hash Space", however, can more evenly scatter items into the system. "Unused Hash Space + Hot Regions" delivers more even distribution than those provided by "Unused Hash Space". Both enable near 75% of nodes in the system host no more $2c$ items (ideally, each host hosts c items), and near 98.7% of nodes host $8c$ items.

As aforementioned, an item is published to the node i with the hash key closest to the key that represents the item. Node i may be overflowed due to limited storage space and thus a most un-similar item, stored in node i is migrated from node i to node i's closest neighbor peer. Consequently, an item requested might not be stored in the node whose hash key is closest to this item. Figure 9 shows the effect of limited storage capacity for "None" and "Unused Hash Space + Hot Regions". Each node simulated in this experiment can host up to $8c$ items. "Closest" indicates the number of hops required to route a query to a node whose hash key is closest to the one that represents a randomly requested item. "Neighbors" denotes the number of hops to discover a requested item along neighbor pointers. The results presents that to search a particular item still takes $O(\log N)$ hops (i.e., with high probability a node



(a)



(b)

Figure 10: (a) The number of hops required per discovery of a similar item and (b) the total messages required of discovery of a set of similar items

whose hash key is closest can resolve a query) when Meteorograph exploits unused hash space and places relatively more amounts of nodes into hot hash regions. However, if no schemes for balancing load are adopted, the performance to access a particular item becomes quite poor.

4.2 Discovery of Similar Items

Given a keyword, we measures the number of hops traversed versus the number of items that match this designated keyword. Queries with the *n*-th popular keyword from 43 ones are investigated, where *n* is one, two, four and eight. Figure 10 illustrates the simulation results for an overlay with 10,000 nodes. "Percentage of Cumulative Discovered items" indicates a ratio from the number of items that are found to match a specified keyword to the total number of items that appear in an overlay and involve the same specified keyword. Notably, each node simulated in this experiment can host at most 8*c* items.

The results (Figure 10(a)) firstly show Meteorograph can discover all items that match a specified keyword. Secondly, each of over 97% of similar items can be located by $O(\log N) = 6.91$ hops and thus messages. Figure 10(b) depicts the number of messages required to discover *k* similar items. Since items involving a specified keyword (1st, 2nd, 4th and 8th popular ones) is smaller than the system size (10,000 nodes). Thus the overheads of messages by discover *k* similar items are $(\frac{k}{c}) \cdot O(\log N)$, which linearly scale with *k*.

4.3 Effects of Failure

The experiment replicates each item that is published to Meteorograph with 10,000 nodes by generating 1, 2, 4 and 8 copies of each item. Nodes dynamically depart from Meteorograph. Queries to a dynamically selected item (the experiment studies queries to a single item) from those published ones are randomly generated. A successful query means Meteorograph effectively use the routing that is provided by Tornado to issue a query to one of those nodes that store replicas; this replica matches specified keywords.

The results show that when 50% of nodes fail, Meteorograph delivers up to 80% of availability of items if each item has two copies in the system. When the number of replicas is increased to four, up to 95% of queries are successful. The percentage further improves up to 99% when there are eight replicas. Notably, even 90% of nodes depart the system, there are still 20%, 30% and 45% of successful ratios for each item with 2, 4 and 8 copies, respectively.

5. Related Works

Service discovery frameworks such as Jini [12] and SLP [18] depend on the client and server model. They use a centralized server to host all resources that register themselves into this server. As aforementioned, this approach suffers from a single point of failures and introduces a performance bottleneck. Additionally, they rely on IP multicast.

In contrast, [4], [5], [6], [8] and [13] recently proposed are based on the peer-to-peer model. They enhance searches in unstructured P2P overlays (particularly for Gnutella [9]). They extensively rely on a flooding mechanism. To search items, query messages are flooded to nodes that have high probability to deliver matched items. The forwarding is based on either the random probability [13] or some heuristics [5]. Although works in [4], [6], [8] use multiple overlays to constrain scopes of searches, their searches still rely on the flooding mechanism in each sub-overlay. As aforementioned, they introduce three major issues. (1) These works considerably generate network traffic. The cost (the number of messages, for instance) of a search is unpredictable. (2) They cannot guarantee in successfully searching for a singly specified item that does appear in an overlay. (3) Results from a search are not deterministic ([8] can deliver complete results, but it broadcasts query to a sub-overlay). Consequently, these works do not guarantee quality of a search in terms of performance and results. Based on the vector space model, [7] can provide similarity search. It, however, is based on an unstructured overlay and relies on a flooding mechanism.

Perhaps, the work most relevant to Meteorograph is pSearch [17]. pSearch is also based on the vector space model and uses a structured P2P overlay (i.e., CAN [14]). CAN structures an overlay using a highly dimensional coordinate space

and requires employ multiple hash addressing space (i.e., *m* hash addressing spaces are necessitated by an *m*-dimensional coordinate space). Clearly, based on an structure overlay using a coordinate space with fixed *m*, in pSearch if a new item is added, then the coordinate space needs to be restructured and each item that are stored in the overlay must be republished. It is impossible that to choose a large *m* by adopting a universal dictionary set that is employed by Tornado. Additionally, when pSearch locates a particulate peer, it must use an expanded ring search (i.e., a localized flooding mechanism) to discover requested items from neighbor peers. This is because peers in CAN do not maintain a linear ordering for participating peers. Perhaps, the most important issue is that it is unclear how to apply pSearch to other structured overlays such as Chord, Pastry, Tapestry and Tornado that particularly use a single-dimensional hash space. Moreover, pSearch cannot deliver ranked results for a search. It is also unclear the performance of pSearch without a detailed experimental study.

The detailed comparisons for various designs can be found in [10].

6. Conclusions and Future Works

This study proposes a similarity information search system called Meteorograph to discover resources in a P2P computing environment. Meteorograph is based on a vector space model and implemented on top of the structured P2P overlay, Tornado. It can search for a particular item in $O(\log N)$ hops/messages and a set of similar items in $(\frac{k}{c}) \cdot O(\log N)$ hops/messages. Meteorograph provides ranked searches and can discover all items that match designated keywords. Based on a structured P2P overlay, Meteorograph guarantees that results from a search are deterministic. Moreover, it can evenly distribute items to the participating nodes. Although this study is based on Tornado, we believe that the concept can also be applied to other overlays (for instance, Chord, Pastry and Tapestry) that have a linear hash addressing space.

Currently, Meteorograph does not support range searches, such as discovering machines that have memory in size between 1G and 8G bytes. Mapping the range of values into the linear structure provided by Tornado may solve this problem. Meteorograph does not support notification to resource consumers either. Notification can rapidly transfer the states of resources to subscribed consumers. We are currently extending Meteorograph to support the above features. We also try to incorporate security mechanisms into Meteorogrpah.

Acknowledgements

We thank Chunqiang Tang for his valuable and insightful comments on discussing the designs of pSearch and Meteorograph.

References

[1] M. F. Arlitt and C. L. Williamson. "Web Server Workload Characterization: The Search for Invariants," In *ACM SIGMETRICS*, pages 126-137, May 1996.

[2] M. W. Berry, Z. Drmac, and E. R. Jessup. "Matrices, Vector Spaces, and Information Retrieval," *SIAM Review* 41(2):335-362, 1999.

[3] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. "Freenet: A Distributed Anonymous Information Storage and Retrieval System," In *Workshop on Design Issues in Anonymity and Unobservability*, pages 311-320, July 2000.

[4] E. Cohen, A. Fiat, and H. Kaplan. "A Case for Associative Peer to Peer Overlays," In *ACM Workshop on Hot Topics in Networks*, October 2002.

[5] A. Crespo and H. Garcia-Molina. "Routing Indices for Peer-to-Peer Systems," In *International Conference on Distributed Computing Systems*, pages 19-28, July 2002.

[6] A. Crespo and H. Garcia-Molina. "Semantic Overlay Networks", In *Submission for Publication*, 2002.

[7] F. M. Cuenca-Acuna and T. D. Nguyen. "Text-Based Content Search and Retrieval in ad hoc P2P Communities," In *International Workshop on Peer-to-Peer Computing*, May 2002.

[8] P. Ganesan, Q. Sun, and Hector Garcia-Molina. "YAPPERS: A Peer-to-Peer Lookup Service Over Arbitrary Topology," In *IEEE INFOCOM*, March 2003.

[9] Gnutella. http://www.gnutella.com/.

[10] H.-C. Hsiao and C.-T. King. "Similarity Discovery in Structured Peer-to-Peer Overlays," *Technical Report*, October 2002. http://www.cs.nthu.edu.tw/~hchsiao/projects.htm.

[11] H.-C. Hsiao and C.-T. King. "Tornado: Capability-Aware Peer-to-Peer Storage Networks," In *IEEE International Conference on Parallel and Distributed Processing Symposium*, April 2003.

[12] Jini™. http://www.sun.com/jini/.

[13] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. "Search and Replication in Unstructured Peer-to-Peer Networks," In *International Conference on Supercomputing*, pages 84-95, June 2002.

[14] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. "A Scalable Content-Addressable Network," In *ACM SIGCOMM*, pages 161-172, August 2001.

[15] A. Rowstron and P. Druschel. "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," In *IFIP/ACM International Conference on Distributed Systems Platforms*, November 2001.

[16] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," In *ACM SIGCOMM*, pages 149-160, August 2001.

[17] C. Tang, Z. Xu, and M. Mahalingam. "pSearch: Information Retrieval in Structured Overlays," In *ACM Workshop on Hot Topics in Networks*, October 2002.

[18] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. "Service Location Protocol," June 1997. RFC2165. http://www.ietf.org/rfc/rfc2165.txt.

[19] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. "Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing," *Technical Report UCB/CSD-01-1141*, April 2000.