行政院國家科學委員會專題研究計畫 成果報告

Visitant: 一個以代理人為基礎的泛用性點對點應用服務平台(3/3)

研究成果報告(完整版)

計 畫 類 別 ：個別型
計 畫 編 號 ：NSC 95-2221-E-002-058-
執 行 期 間 ：95 年 08 月 01 日至 96 年 10 月 15 日
執 行 單 位 ：國立臺灣大學資訊管理學系暨研究所

計 畫 主 持 人 ：莊裕澤

計畫參與人員：碩士班研究生-兼任助理：黃榮達、巫志彰

報 告 附 件 ：出席國際會議研究心得報告及發表論文

處 理 方 式 ：本計畫可公開查詢

中 華 民 國 97 年 02 月 20 日

Final Project Report on Visitant: An Agent-Based, General-Purpose, Peer-to-Peer Platform

Yuh-Jzer Joung
Department of Information Management
National Taiwan University
Taipei, Taiwan
(NSC 93-2213-E-002-096, 94-2213-E-002-036, and 95-2221-E-002-058)

Abstract

This is a final project report on "Visitant: An Agent-Based, General-Purpose, Peer-to-Peer Platform", a three-year project supported by the National Science Council. The goal was to design and implement an agent-based P2P system, called ***Visitant***, to integrate mobile agent technology with P2P computing. An important problem in the design is *resources discovery*, where mobile agents need resources to complete their tasks. Resources discovery is also a fundamental issue in P2P networks, where peers need to be able to efficiently find the objects, given only limited knowledge they have about the network. Therefore, the project also paid much attention in this problem. The report gives an overview of the main results we have established for the project.

I. Introduction

With the rapid growth of personal computers and network technologies, there are abundant resources such as computing power, network bandwidth, and storage space in the Internet, but most of them have low utilization. At the same time, there has been an increasing demand for massive parallel computation in, e.g., astronomy, biology, chemistry, high energy physics, and meteorology, which cannot be accomplished by any single computer. In light of this need, there are two approaches to integrate superfluous resources in the Internet: *Grid computing*, and *Peer-To-Peer (P2P) computing*.

Grid computing [8], [9] emphasizes on the integration of computational resources from geographically distributed organizations and presents them as a single, unified resource for solving large-scale and data intensive computing. For ease of management, resources are organized hierarchically and discovered in a centralized manner. The resources within each organization (e.g., computing clusters, storage systems, and data sources) are assumed to be relatively stable to reduce the overhead in management. Even so, the configuration of the resources is somewhat complex to ensure inter-operability.

P2P computing, on the other hand, assumes a more dynamic and loosely-coupled environment such as the Internet, and a potentially huge number (millions) of highly autonomous participants. To handle such scale of networks, fully decentralized management is essential. However, due to lack of an effective mechanism to manage computational resources over heterogeneous platforms, only storage and file sharing services (e.g., BitTorrent and eMule) are popular.

Our goal is to exploit computational resources in P2P networks and to increase flexibility of application programs deployment. To achieve this goal, we incorporate mobile agent technology into P2P computing. *Mobile agents* are active objects that have autonomous behavior, executing states, and network locations. They are able to

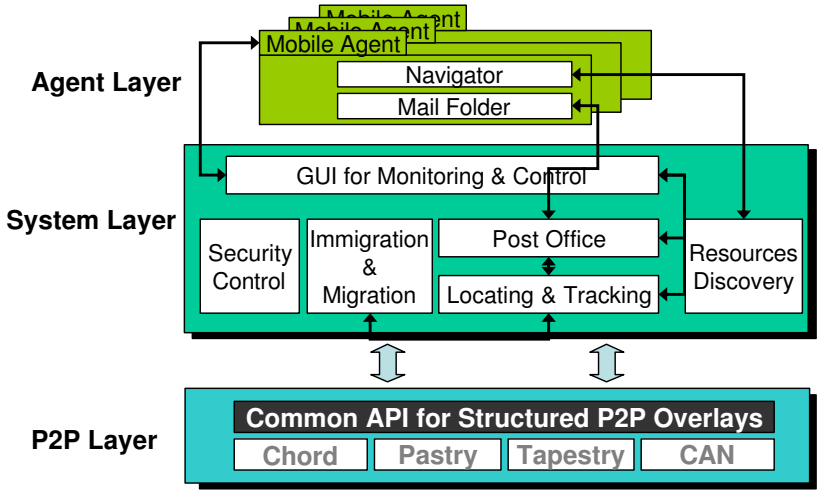


Fig. 1. Visitant system architecture.

move over different *mobile agent systems*—a distributed abstraction layer to provide mobile agents with mobility, communication, and security [1], [29]. If each peer has a mobile agent system to function as a middleware between mobile agents and the underlying execution platform, then P2P users can deploy their programs in mobile agent format to remote peers easily and flexibly without complex configuration.

To help mobile agents allocate resources (including agents themselves), we use structured P2P networks as the base layer over which agents are travelling and migrating between different hosts. *Structured P2P networks*, e.g. [37], [32], [30], organize their overlay topology into some distributed data structure, over which objects are placed via some globally-agreed scheme. They typically can provide a distributed object locating service that guarantees to find an object within $O(\log N)$ steps, and takes only $O(\log N)$ space per site for storing routing information, where N is the network size. Their counterpart, *unstructured P2P networks*, abandon efforts to maintain a specific topology, so as to increase system robustness in a highly dynamic network environment. This, however, is at the cost of search, as flooding is essential in such a chaotic network.

We call the agent-based P2P system we designed *Visitant*. Below we give an overview of the system. Section III presents some fundamental problem in the design and our results.

II. Visitant

As shown in Figure 1, Visitant is divided into, from bottom to top, the following three layers: *P2P Layer*, *System Layer*, and *Agent Layer*. P2P Layer plays as an execution platform of the Visitant system, and provides agent migration and communication between Visitant systems.

System Layer provides an execution environment for mobile agents. It is composed of six modules: *Locating & Tracking*, *Post Office*, *Immigration & Migration*, *Security Control*, *GUI for Monitoring & Control*, and *Resources Discovery*. The Locating & Tracking module locates a mobile agent in the system. The Post Office module is responsible for message delivery between mobile agents. The Immigration & Migration module allows mobile agents to migrate from one host to another. After migration, agents need to update their current location to the Locating & Tracking module. The Security Control module adopts *Java Security Architecture* as the basic resources access control mechanism, and on top of which it needs to handle clone management. A GUI is provided for monitoring and controlling mobile agents. It uses the Locating & Tracking module to get locations of mobile agents, and transmits messages to them via the Post Office module. The last module Resources Discovery provides a keyword search mechanism for mobile agents to search resources in the P2P network.

The top Visitant Agent Layer allows programmers to design their mobile agent applications to utilize resources in the P2P network. Each mobile agent has two components: *Navigator* and *Mail Folder*. The Navigator looks up resources needed by the mobile agent, and maintains traveling plans of the agent as well. The Mail Folder stores messages delivered to the agent.

A. P2P Layer

Existing structured P2P systems [37], [32], [30] have different designs and characteristics. These differences make it difficult for applications developed for one system to be ported to another. Frank Dabek et al. [7] therefore identify fundamental abstractions of structured P2P systems and propose a common API for general services of them, such as message transmission and routing. They define a *key-based routing API (KBR)* for programmers to develop applications regardless of subtle differences of structured P2P systems. Visitant adopts this API as the interface to avoid been locked to a specific structured P2P system.

B. System Layer

The Visitant system layer is the core of the Visitant, and below we describe its six modules in more detail.

1) Locating & Tracking: When a mobile agent was created and dispatched into the system, it may dynamically migrate between a number of sites to complete its task. In order to control and communicate with the agent, its location needs to be tracked. To do so, each mobile agent is assigned a unique identifier. For security reasons (see Section II-B5), agent identifiers are not assigned arbitrarily, but rather derived from their portable code using some hash function. When no confusion is possible, we simply use an agent's id to denote the agent. An agent also has some attributes that can be used by other agents/hosts to locate the agent through the Resource Discovery module when the agent's identifier is not known.

Each agent has a permanent host called its *home* to manage the agent. For security reasons, the home is determined by a globally-agreed, one-way hash function $\mathcal{H}$. So given an agent identifier a, one can easily determine its home $\mathcal{H}(a)$; but one cannot first find a host x, and then creates an agent of id a' such that $\mathcal{H}(a') = x$. This prevents a malicious host from colluding with another host to create mobile agents to attack the system. We will go back to this issue later in the security control module.

When a mobile agent a is created, it sends a message to its home $\mathcal{H}(a)$ to register in the host. The message contains the creating host's ID, the creation time, and capability of the agent. When a migrates to a new location, it must report the location to its home by sending an updating message. The agent cannot proceed until its home host acknowledges its update message. Therefore, to locate an agent, one needs only to know the identifier of the agent, and then use the identifier to find its home, which has the current location of the agent.

The above algorithm requires that every mobile agent host is existent and stable, but in practice, a mobile agent host could fail or leave at any time. Therefore, the system needs a mechanism to assign a "surrogate" for a failed host. Our method is based on the surrogate mechanisms supported by most structured P2P systems.[1] That is, if a host x fails, the system will assign a workable host *Surrogate*(x) to act as x. All messages targeting for x will then be automatically forwarded to *Surrogate*(x) via the P2P overlay. We can further increase the degree of fault tolerance by assigning a *secondary home* to every agent to backup the primary one.

In addition to security reasons, the use of an external host, rather than the creating host of an agent a, to be the home of a, has the following advantages: First, the agent can remain functioning even if its creating host is offline, and the surrogate mechanism further boosts the system's fault tolerance. Second, by using a uniform hash function

[1]For example, Chord [37] uses a function $successor(x)$ to determine a node responsible for a key x, where $successor(x)$ is the alive node with an ID most close to x in the identifier ring space.

for $\mathcal{H}$, the load of agent hosts in handling mobile agents can be made balanced even though the creation of agents may not uniformly occur in the system.

2) Post Office: The Post Office module of a host x is responsible for delivering messages sent to the agents of which x is their home, as well as transmitting the messages sent by the agents currently in the host to their targets.

In Visitant, messages between mobile agents are called *mails*. Each mobile agent a has a mailbox in its home $\mathcal{H}(a)$. All mails to an agent will first be delivered to the home host of the agent, and then delivered to the agent by the host according to the mail type. Two types are supported: *Express* and *Ordinary*. Express mails will be forwarded by home to agents immediately when they arrive at the home. They can be used for real time communication with the agents. Ordinary mails, on the other hand, will be stored at home and be taken away by agents when they contact with their homes; this occurs when agents migrate to a new host, or when they actively check their mailboxes.

3) Immigration & Migration: A distinguishing characteristic of mobile agents is migration. To support migration, a mobile agent system needs to stop an agent's execution, records its status, and then transmits the agent to the destination. The agent will then be restarted by the remote mobile agent system. The *Immigration & Migration* module of Visitant provides this functionality. For portability, we use JAVA as the implementation language in Visitant. When an agent finishes its tasks and is ready to migrate, it notifies the Immigration & Migration module for migration. The module then removes the agent from the host's visiting agent list and uses Java *serialization* for transmitting objects and variables in the program.

Specifically, Visitant supports weak mobility [15], meaning that agent systems do not record executing states such as program counter and stacks before agent migrates. Rather, agents migrate at specific part of programs. Most Java-based mobile agent systems use weak mobility because Java Virtual Machine (JVM) limits programmers to obtain agent execution information for security reasons.

4) Resources Discovery: Mobile agents need resources to complete their tasks. Resources include CPU cycles, storage space, bandwidth, as well as agents themselves. To obtain resources, agents need to know their locations, and the Resources Discovery module is designed for this purpose. In Visitant, each available resource must be registered into the system by the site that owns the resource. Registration information includes the name of the resource, its attributes, and the location of the resource. Visitant provides two types of search for resource discovery: *by name* (white page service) and *by attribute* (yellow page service).

To support name search, when a site registers a resource s, it inserts a name record of the resource into the site $\mathcal{H}(s)$. So when an agent needs to locate s, it can use the lookup service provided by the base layer to send its query message to the site $\mathcal{H}(s)$, and then obtains the location of s.

For yellow page service, we have developed a hypercube index and search scheme [18] to index attributes of resources to facilitate efficient search. More details of this will be given in Section III-B.

5) Security Control: Security threats in mobile agent systems can be roughly classified into two categories: *uncoordinated attacks*, and *coordinated attacks*. Uncoordinated attacks refer to individual attacks to hosts, such as unauthorized access to a host's resources. For uncoordinated attacks to individual hosts, Visitant uses JAVA sandbox to provide the basic security mechanism. Each mobile agent host can define its own security policy for incoming mobile agents, which must then meet the policy in order to execute on the host.

Coordinated attacks refer to systematic attacks to a host or to the entire system, typically done by consuming an extraordinary amount of resources to paralyze the target. Visitant's strategy to this problem is the following. First, every agent, when created, is given a *capability*, specifying its TTL (Time-To-Live)—the time it can live in the system, the maximum number of resources (e.g., CPU time, bandwidth, and memory space) it can use at each site, the number of times it can migrate, and the number of clones/decedents it can produce.

Second, each agent is assigned with a *guardian* to monitor its capability. Like agent homes, we use a one-way

hash function $\mathcal{G}$ to determine the guardian of an agent a so that one cannot first find a host, and then creates an agent that is to be guarded by the host. That is, the guardian $\mathcal{G}(a)$ of an agent a is independent with the owner of the agent. This prevents several hosts from colluding with each other to guard a set of malicious hosts.

When a host x creates an agent a, x must provide with a's capability to its guardian $\mathcal{G}(a)$. The guardian can reject the capability if it may pose a threat to the system, or to any host the agent may visit. For some applications that demand high capability agents, their guardians can verify if the owners of the agents have enough credit to create them via, e.g., some trust management system.

Recall that every agent also has a home host to track the agent. The two independent roles—"home" and "guardian"—may also be combined together and assumed by only one host.

Note that the security mechanism requires that both agent and host ids are not assigned arbitrarily. Otherwise, one can create an agent with id a and a host with id $\mathcal{H}(a)$ (as well as a guardian $\mathcal{G}(a)$) to manage/monitor the agent, so as to allow the agent to do some malicious task. In practice, a host's id is usually generated by hashing the host's IP address, which is not easily controlled by the end user. An agent's id is also not arbitrarily assigned, as it is derived from its code. So it is difficult for one to create a meaningful agent with some specific id from which the creator can allocate a host under his control.

6) GUI for Monitoring & Control: The graphical user interface mainly provides functions for users to monitor and send messages to their mobile agents.

C. Agent Layer

We provide an agent template and a set of basic methods for programmers to overwrite when they design their agents. Each mobile agent has two main components: *Navigator*, responsible for managing the agent's itinerary, and *Mail Folder*, for storing messages to the agent. Both *static* and *dynamic* itineraries are supported. The former allows programmers to code a specific route into an agent for the agent to travel, while the latter, in combination with our resource discovery module, allows agents to discover the next host on the fly.

D. Application Programming Interfaces

We have implemented a set of APIs (Application Programming Interfaces) for developing applications in Visitant. Details of this can be found in [3]. We have also built an application, called *Cyclone Rover* [19], over Visitant. Cyclone Rover simulates Fujiwhara effect—the tendency of two nearby tropical cyclones to rotate cyclonically about each other. The simulation requires large matrix operations, which can be decomposed into many small operations and computed concurrently to speed up the computation. Using this application, we also show how a computational-demanding program can be developed and deployed over Visitant.

III. Search in Peer-to-Peer Networks

As we commented earlier, *resource discovery* is a fundamental problem in P2P systems. In this project, we have also been focusing on designing an efficient and effective resource discovery mechanism for fully decentralized P2P systems. We have several results for different types of P2P networks—*unstructured* and *structured*.

A. Search in Unstructured Peer-to-Peer Networks

When the network is Gnutella-like unstructured, each node often locally maintains objects it shares to the network. Since query messages are passed around nodes to check if they have the desired objects, a variety of searches can be offered, including, of course, keyword search. However, since search is typically a blind process traversing

around the network, to avoid flooding the network, some TTL is set to limit the search space. So search cannot be guaranteed if target objects are distant. Much work on unstructured P2P networks has been devoted to search improvement (e.g., [17], [39], [6], [25]), with some specifically focusing on keyword search [28], [2].

Our contribution in this direction is a simple, practical, yet powerful index scheme to enhance search in unstructured P2P networks. The index scheme uses a data structure "Bloom filters" to index files shared at each node. Let B_u be the Bloom filter of node u. If B_u is replicated and distributed to other nodes in the network, then every node v that has B_u can answer queries about whether u might have a particular file one is looking for. If the answer is yes, v can ask u to perform an actual check to its local directory to see if u does have the file.

In general, the replication allows a query to u's files to be answered by any node having a copy of B_u. So if every node has a copy of B_u, then file search can be efficiently performed in the system. However, for the system to scale, only a limited number of replicas of B_u can be distributed. In practice, the number of replicas should allow a query initiating from a node to be answered within a reasonable search space. Here, the search space refers to the set of nodes to be visited for the query. It corresponds directly to the set of Bloom filters to be searched during the query resolving process.

Still, the distribution of a node's Bloom filters may not be so uniform to let the search space of any given node v contain a copy of a particular B_u. As a result, a query to u's files will not be resolved at v. To solve this problem, we let nodes dynamically exchange their Bloom filters so that the search space of each node is also dynamically changing. This then allows every query from a node to have some probability to be successfully resolved, regardless of the time the query is issued. By properly setting the system parameters, we can tune the probability to be reasonably high.

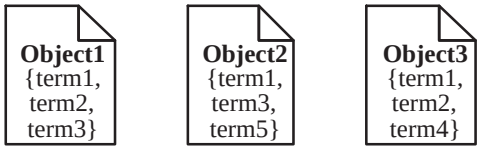
The experimental results show that our approach can improve the search in Gnutella by an order of magnitude. For example, in a typical Gnutella network consisting of about 89,000 nodes, by replicating a node's Bloom filter to less than 0.45% of the nodes in the network, 70% of the queries can be resolved within a search space of 200 nodes. In contrast, within the same search space size, only 1.6% of the queries can be resolved without the index scheme; or, alternatively, more than 48,000 nodes need to be searched in Gnutella in order to reach the same success rate as our index scheme. The result is published in [4].

B. Search in Structured Peer-to-Peer Networks

For structured P2P networks like DHTs , they basically support only exact name match [14], [24], as objects are given a unique identifier obtained by hashing their names to determine their location in the network. Keyword search must be built on top of the overlay to enhance search functionality. Several mechanisms have been proposed for keyword search in DHTs (e.g., [13], [40], [31], [38], [36], [11]), but all of them use inverted index as the primary data structure.

An *inverted index* is a set of entries of pairs (w, O), where w is a keyword, and O is the set of objects containing this keyword; see Fig. 2. Once an inverted index is built, a set of keywords can be entered to find all objects that contain these keywords. For example, in Fig. 2, by taking an intersection of the sets associated with keywords $\mathrm{term1}, \mathrm{term2}$, and $\mathrm{term3}$, we can find the object (i.e., $\mathrm{Object1}$) that has all these keywords.

To implement keyword search in a P2P network, a distributed version of inverted index can be built. A simple way is to distribute the entries so that each keyword is assigned a node to index the objects that have this keyword. By incorporating into DHT networks, one can use a given keyword as key to determine the node that is responsible for the keyword, and obtains objects that contain the keyword. By taking a join operation, one can retrieve objects with a given keyword set.



Key	List of Objects
term1	{Object1, Object2, Object3}
term2	{Object1, Object3}
term3	{Object1, Object2}
term4	{Object3}
term5	{Object2}
...	...

Fig. 2. An inverted index of three objects.

The above approach, although commonly used in existing P2P systems, suffers from several problems. The first one concerns load balance. In a real world corpus, keyword frequency—the count of a keyword's occurrence in objects—varies enormously. The distribution typically follows *Zipf's law*, meaning that a few keywords occur very often while many others occur rarely. So, simply mapping each entry in an inverted index to a node makes the indexing load extremely uneven.

The second problem concerns storage redundancy. If an object σ contains keywords $w_1, \ldots, w_k$, then creating entries for each keyword means that information about the object is repeatedly stored at k different places. A typical object has a few to dozens of keywords in its metadata. So this redundancy makes object insert, delete, and maintenance very expensive, as it has to deal with multiple peer accesses in the network. Note that redundancy is necessary in coping with fault tolerance. However, the redundancy incurred by the above index scheme does not solve fault tolerance in a natural way because the number of keywords of an object has no correlation with the failure probability of the object.

Moreover, even though an object is indexed at several places, each keyword is still handled only by a single node. Any failure to the node would then deny all queries involving this keyword. The system is also vulnerable to hot spots, as nodes responsible for some popular keywords may be queried much more frequently than the others.

The last problem concerns object ranking and query expansion. If the object space is huge, a query composed of a few popular keywords may yield a set with a very large number of objects. One would certainly prefer some ranking mechanism to help select relevant objects, but such feature is less addressed in existing P2P systems. Ranking, in general, requires some global knowledge. For example, in information retrieval, the concept of *inverse document frequency (IDF)* has been used to measure how importance a keyword is. It is defined as the logarithm of the ratio of number of documents in a collection to the number of documents containing the keyword [16]. So infrequent words have high IDF and common words such as 'mp3' have low IDF. IDF can be easily calculated when indexing service is centralized, but the cost is high to get a good measure of it in a decentralized environment.

On the other hand, short queries severely affect search precision [10]. In information retrieval, *query expansion* [26], [27], [12], [23] has been studied for decades to expand queries with some additional keywords to help describing the target and narrowing down the search scope. Query expansion is also less addressed in existing P2P systems, perhaps due to that co-occurrence relation between keywords is expensive to obtain in a fully distributed environment.

C. Hypercube-Based Index and Search scheme

Our main result for keyword search is to devise a general keyword index and search scheme for DHT networks. The idea is to represent each object as an r-bit vector according to its keyword set. Then we construct the index scheme over an r-dimensional logical hypercube $H_r(V, E)$. The hypercube can be constructed directly from a physical hypercube (e.g. HyperCuP [35]), or conceptually built on a DHT. The advantage of using a physical hypercube is that communication between two neighboring nodes in the logical layer costs only one hop of message transmission in the physical overlay.

To construct $H_r(V, E)$ over a physical DHT $G = (V', E')$, we simply need a mapping $g : V \to V'$ so that every logical node in the hypercube has a corresponding physical node in the network. However, as mentioned before, most DHTs offer $O(\log N)$ hop-to-hop delay for communication between any two nodes, where N is the size of the network. So a message transmission between any two nodes in the hypercube will cost $O(\log N)$ messages in the DHT. Another advantage is that the size of the hypercube can be decoupled from the size of the DHT. The former is often determined by the object set to be indexed, while the latter is determined by the number of participating nodes in the system.

Our index scheme does not impose any specific requirement on the mapping of hypercube nodes to DHT nodes, and thus makes it a general keyword search layer over any chosen DHT. Nevertheless, some guidelines may be provided to choose the mapping. For example, when the size of the hypercube is larger than the size of the DHT (i.e., there are more logical nodes than physical nodes), then for load balancing, we can use a hash function (e.g., SHA-1) to uniformly map logical nodes (by their IDs) to physical nodes. When the size of the hypercube is smaller, only a portion of the physical nodes will actually be responsible for indexing objects. This allows some leeway in selecting indexing nodes. For example, many researches have observed that nodes in P2P networks are not homogeneous: some are more stable/powerful than the others [34], [5]. So we may select stable/powerful nodes to serve as indexing nodes in the hypercube. The use of "supernodes" as index servers for ordinary nodes has been practically adopted in several popular unstructured P2P networks like KaZaA and eMule.

The index scheme works as follows. Let $\mathcal{W}$ be the set of all keywords considered in the system. Let $h : \mathcal{W} \to \{0, 1, \ldots, r-1\}$ be a uniform hash function that maps every keyword in $\mathcal{W}$ to an integer in $\{0, 1, \ldots, r-1\}$. We define a mapping $\mathcal{F}_h : 2^{\mathcal{W}} \to V$ as follows: $\mathcal{F}_h(K) = u$ if, and only if, $One(u) = \{h(w) \,|\, w \in K\}$. In other words, $\mathcal{F}_h(K)$ is the node whose bits are set by the hash function h according to the keywords in K. We say that u is *responsible* for K if $\mathcal{F}_h(K) = u$. Thus, for every possible set of keywords in the system, there is a unique node in the hypercube responsible for the set. Note that a node may be responsible for more than one set of keywords (as $\mathcal{F}_h(K)$ might be equal to $\mathcal{F}_h(K')$ for some K and K'). We use $\mathcal{R}_u$ to denote the set of keyword sets for which u is responsible; that is, $\mathcal{R}_u = \{K \subseteq \mathcal{W} \,|\, \mathcal{F}_h(K) = u\}$.

To build the index scheme, for each object σ that is associated with keyword set K_σ, we let the node $\mathcal{F}_h(K_\sigma)$ in the hypercube maintain an entry $\langle K_\sigma, \sigma \rangle$ in its index table. We say that σ is *indexed* at the node, and we use $\mathcal{O}_u$ to denote the set of objects that are indexed at u; that is, $\mathcal{O}_u = \{\sigma \in \mathcal{O} \,|\, K_\sigma \in \mathcal{R}_u\}$.

Fig. 3(a) illustrates the index scheme over a 4-dimensional hypercube. The mapping of hash function h is also shown in the figure. For example, $h(b) = 3$, $h(c) = 1$, and $h(e) = 0$. So the keyword set $\{b, c, e\}$ is responsible by node 1011. Objects x and y both have keyword set $\{b, c, e\}$, so they are indexed at node 1011. In addition, the node also indexes object w, which has keyword set $\{b, c, f\}$.

1) Search in the Hypercube: Given a keyword set K, we can locate a copy of object associated with K by first finding the node in H_r that is responsible for K. The node is determined by $\mathcal{F}_h(K)$. Once the node is located, its index table can be searched to obtain the ID of an object σ that is associated with the keyword set K. Then, a call $Read(\sigma)$ to the underlying DHT network will invoke the DOLR scheme to return a copy of σ. So pin search

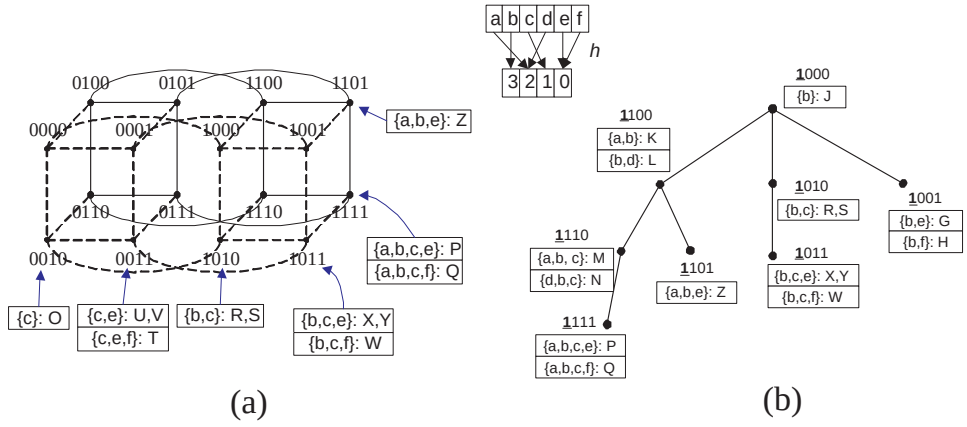


Fig. 3. (a) The hypercube index scheme and (b) a search tree for query $\{b\}$.

is directly supported by the scheme. For example, in Fig. 3, to search objects with keyword set $\{b, c, e\}$, since $\mathcal{F}_h(\{b, c, e\}) = 1011$, we can issue a query to node 1011 to retrieve the objects.

For superset search, we need to retrieve objects that can be described by K. To locate the objects, we need to find all the nodes that are responsible for a superset of K. Recall that a subhypercube $\mathcal{H}_r(u)$ of H_r induced by u consists of all nodes w in V that contain u (that is, $u[i] = 1 \Rightarrow w[i] = 1$). So every node in H_r that is responsible for a superset of K is in the subhypercube induced by $u = \mathcal{F}_h(K)$. This property allows us to search for only the subhypercube if we wish to find out any object that can be described by K.

Moreover, when searching the subhypercube, we can explore the spanning binomial tree $OSBT_{\mathcal{H}_r}(u)$ rooted at u. Recall that a node v at depth i in the tree has Hamming distance i from the root. For every keyword set $K_v \in \mathcal{R}_v$ and $K_u \in \mathcal{R}_u$, if $K_u \subseteq K_v$, then $|K_v| - |K_u| \geq i$; that is, K_v contains at least i more keywords than K_u. This means that if we search the tree $OSBT_{\mathcal{H}_r}(u)$ in a breadth-first style, we can locate objects whose associated keyword sets gradually enlarge, thereby allowing the upper-level applications to retrieve relevant objects more effectively.

Fig. 3(b) illustrates the search tree for a query of keyword b on the left hypercube. One can see that the keyword set size indexed at the nodes gradually increases along each path.

The following two lemmas summarize the above properties. Their proofs are straightforward.

Lemma 3.1: Let $u \in V$ be a node in $H_r = (V, E)$, and $K \subseteq \mathcal{W}$ be a keyword set for which u is responsible. Then, all objects that can be described by K are indexed at nodes in the subhypercube $\mathcal{H}_r(u)$ induced by u.

Lemma 3.2: Let $u \in V$ be a node in $H_r = (V, E)$, $\mathcal{H}_r(u)$ be a subhypercube induced by u, and $OSBT_{\mathcal{H}_r}(u)$ be a spanning binomial tree rooted at u. For every node v in the tree, if v is at depth d, then for every keyword set $K_u \in \mathcal{R}_u$, every keyword set $K_v \in \mathcal{R}_v$ satisfying $K_u \subseteq K_v$ has at least d more keywords than K_u.

Finally, when performing a superset search, a typical scenario is that a user starts by specifying a set of keywords, browses through some returned objects, and then adds more keywords to refine the search. The following lemma says that the second query has a search space within the first one. An implication of the lemma is that we can cache some information about the nodes visited in earlier queries for search refinement, so as to save bandwidth.

Lemma 3.3: Let $K_1, K_2 \subseteq \mathcal{W}$ be two keyword sets. If $K_1 \subseteq K_2$, then $\mathcal{H}_r(\mathcal{F}_h(K_2))$ is a subhypercube of $\mathcal{H}_r(\mathcal{F}_h(K_1))$.

2) Experiments and Conclusions: We have also evaluated our index scheme using real data collected from the Web. The results indicate that the scheme can result in quite balanced indexing load. This, however, also indicates that search space may increase in proportion to recall rate. To improve search efficiency, we investigated several

approaches, including GDFS, query expansion, and caching. All of them proved very effective in reducing the nodes to be contacted. In particular, with just a small size of cache, the number of nodes need to be contacted is significantly reduced: less than 1% of nodes per query in order to retrieve all the matching objects.

To summarize, our hypercube index and search scheme has the following interesting properties compared to the inverted index approach: First, the index entries of a single keyword are handled by a set of nodes. The population of this set depends on the popularity of the keyword: the more the popularity of a keyword, the more the number of nodes responsible for the keyword. As a result, the load of nodes can be balanced even though keyword distribution follows Zipf's Law, and no node is likely to be swamped even if it handles a very popular keyword. Moreover, since a number of nodes are responsible for a keyword, any failure of them cannot block queries involving the keyword.

Secondly, an object σ associated with a keyword set K can be efficiently 'pinpointed' if the set K is given. This is analogous to exact name search in DHT networks. As discussed earlier, DHT networks use an object's name to determine its handling node. Thereafter, locating the object is simply a message routing to the node, which can be done very efficiently in the networks. In our index scheme, we use the keyword set associating with an object to determine a unique node to index the object. When the set is known, locating the object is as efficient as exact name search in DHT networks. In contrast, this kind of 'pin search' is usually very expensive in existing P2P networks. Likewise, object insert, delete, and maintenance can also be done efficiently, as no unnecessary redundancy is introduced in our scheme to index objects.

Third, in a search operation, in addition to objects whose keyword sets match exactly with a given keyword set K, one may also wish to retrieve objects whose keyword sets *contain* K. All these objects can be easily and efficiently retrieved in our index scheme. Moreover, the larger the set K a user has specified, the more restriction a user has placed on his target objects. Accordingly, our index scheme will require fewer number of nodes to be contacted. On the other hand, when a small set of K is given, a large number of objects may satisfy the search request. In this case, a user often expects to see only a small subset of them. Our index scheme can also support this kind of operations effectively and efficiently.

Specifically, objects in our index scheme are easily distinguished by the number of keywords they associate. For example, let K be a set of keywords. Our index scheme can easily locate objects that are associated with exactly the set K of keywords, objects that are associated with K plus one more keyword, K plus two more keywords, and so on. Moreover, within each category, e.g., K plus one more keyword, objects can further be distinguished by the extra keyword they have, e.g., K plus a specific keyword σ_1, K plus a specific keyword σ_2, and so on.

This interesting feature allows upper level applications to retrieve objects in the order they wish. For example, an application might prefer more specific objects to be retrieved first. In this case, when a search request with a keyword set K is issued, our index scheme can return objects containing this keyword set K in the order by giving preference to those with more extra number of keywords. On the other hand, if an application prefers more general objects, then our index scheme can give preference to those with fewer number of extra keywords. Furthermore, our index scheme may also sample some objects in each category described above, e.g., objects that have an extra keyword σ_1, an extra keyword σ_2, ..., two extra keywords σ_1, σ_2, two extra keywords σ_1, σ_3, ..., and so on; and then return these sample objects along with their extra keyword(s) to help users refine their queries. Note that no global knowledge is required to implement this ranking mechanism. Moreover, the clustering effect of keyword sets also makes query expansion easy to achieve, because a node can obtain which additional keywords are likely to co-occur with its keyword set by contacting only its neighbors.

Part of the results have been published in the 25th International Conference on Distributed Computing Systems (ICDCS 2005) [18], and in the *IEEE Journal on Selected Areas in Communications (JSAC)*, Special issue on

Peer-to-peer Communications and Applications.

D. Keytoken-Based Index Scheme for Prefix Search

In addition, we have also studied prefix search in P2P networks. Prefix search is a fundamental operation in information retrieval (IR). A prefix query such as *comp** allows users to retrieve objects with keywords like *computer, company*, and *competitor* that begin with *comp*. Prefix search can also be used in combination with keyword search, for example, like "*ACM SIG* proceedings*" to search proceedings from all ACM special interest groups. This is very useful when people have only partial information about the objects they wish to retrieve.

Prefix search is more general than keyword search, as the latter can be viewed as a special case in prefix search. So a system that supports prefix search can easily facilitate keyword search, but not vice versa. There are, however, some techniques to extend keyword search to prefix search. The most common way is to use the *n-gram* technique [33], [14] to augment each keyword with all its prefixes. For example, if an object o has two keywords abc and acd, then we expand its keyword set to a, ab, abc, ac, and acd. Using this technique, a prefix query of "ab*" can be converted into an ordinary keyword search with query "ab". To implement keyword search, as discussed above, the inverted index data structure is commonly used. However, as also commented above, inverted index suffers from several problems, and the problems are magnified when taking prefixes into account.

Our solution uses a very simple yet novel technique: extracting characters and their position information in a keyword to index objects. Each character-position pair is referred to as a *keytoken*. We use 2,412,613 CD records collected in FreeDB (http://freedb.org) as experimental dataset to test our index scheme. For simplicity, we use only three fields from the records: DTITLE (performer and album name), DYEAR (publishing year), and DGENRE (category). We will use the following sample record (case-insensitive) for illustration: DTITLE="norah jones unforgettable", DYEAR="2002", and DGENRE="jazz". The DTITLE field contains three keywords. From these keywords we extract a set of keytokens $S_T = \{\langle c, i\rangle \mid \text{character } c \text{ occurs at the } i^{\text{th}} \text{ position in some of the keywords}\}$. So $S_T = \{\langle n, 1\rangle, \langle o, 2\rangle, \langle r, 3\rangle, \ldots\}$. For numeric data like that in the DYEAR field, we treat them as fixed-length strings (with padding '0'). So the keytoken set S_Y extracted from the field in the sample record is $S_Y = \{\langle 2, 1\rangle, \langle 0, 2\rangle, \langle 0, 3\rangle, \langle 2, 4\rangle\}$. For the DGENRE field, as there are only 11 categories in the database, we can encode them by 11 distinct keytokens, say, $\langle a, 1\rangle, \langle b, 1\rangle, \ldots, \langle k, 1\rangle$. Assume that the jazz category is encoded by the keytoken $\langle f, 1\rangle$.

To distinguish keytokens from different fields, we shift the position information in keytokens as follows. Suppose the maximum keyword length in the DTITLE field is 20. Then the keytokens in the DYEAR field will begin with position 21. After the shift, the keytokens in S_Y of the sample record become $S_Y = \{\langle 2, 21\rangle, \langle 0, 22\rangle, \langle 0, 23\rangle, \langle 2, 24\rangle\}$. Likewise, the keytoken for the DGENRE field is shifted to $\langle f, 25\rangle$. The keytoken extraction process is illustrated in Fig. 4. We then use the keytokens extracted from the fields of a CD record to represent the record. So each object o has a corresponding keytoken set S_o to represent the object.

To index the objects, we used the hypercube indexing technique developed for keyword search, and assume a logical hypercube $H_r = (V, E)$ of dimension r. Each node has a unique r-bit ID, and $|V| = 2^r$. Our index and search scheme will be performed on the hypercube. The hypercube can be physically built as a structured P2P network, or mapping to a physical DHT-based P2P network like Chord or CAN. For details, please see [18].

We need to find a mapping that maps every object o by its keytoken set S_o to a unique node u in V so that u is responsible for indexing o. Formally, let $\mathcal{T}$ be the set of all possible keytokens, and let $h : \mathcal{T} \to \{1, \ldots, r\}$ be a uniform hash function. We define a mapping $\mathcal{F}_h : 2^{\mathcal{T}} \to V$ as follows: $\mathcal{F}_h(T) = u$ if, and only if, $\{i \mid u[i] = 1, 1 \leq i \leq r\} = \{h(t) \mid t \in T\}$. That is, $\mathcal{F}_h(T)$ is the node ID whose bits are set by h according to the keytokens in T.

It can be seen that r is affected by the average keytoken set size. Large keytoken sets incur large r (which

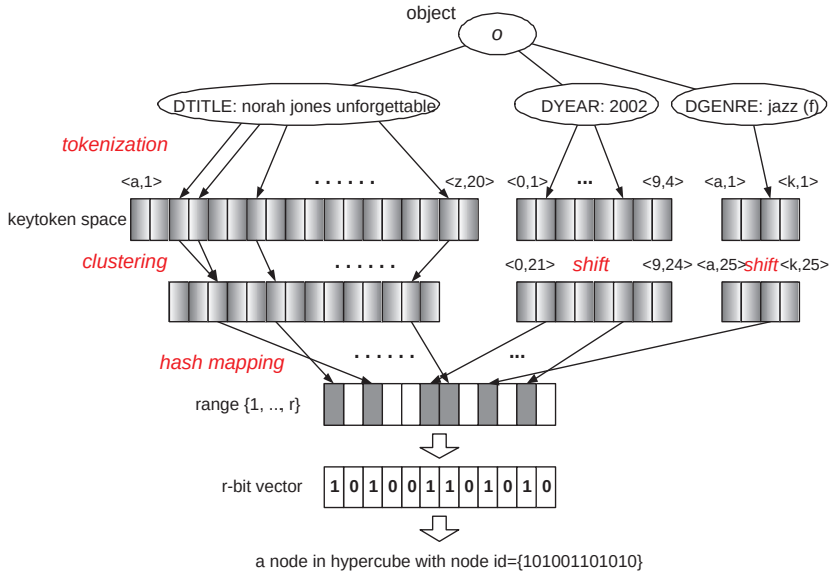


Fig. 4. Keytoken-based index scheme.

results in high search costs), or else hash collision would be high (which results in unbalanced index loads). To reduce keytoken set size, we observe that some keytokens are highly correlated. For example, in English, many words begin with $de*$, so $\langle d, 1\rangle$ has high probability to occur with $\langle e, 2\rangle$ simultaneously. We define the *correlation coefficient* of two keytokens t_j and t_k as follows

$$corr(t_j, t_k) = \frac{\sum_i f_{ij} \cdot f_{ik}}{\sqrt{\sum_i f_{ij} \cdot \sum_i f_{ik}}}$$

where f_{ij} is a binary value representing if t_j is in object o_i. Then, given a threshold thr, we can use some clustering technique, e.g., *simple-link* [33], to cluster highly correlated keytokens and use a single keytoken to represent each cluster.

We note that according to our preliminary experiments, keytoken correlation is quite stable for English words. This means that the correlation coefficients can be obtained from some sample dataset and so need not be calculated online.

1) Search Strategies: To search objects in the index hypercube, suppose, for example, one is looking for some record in the jazz category with *unforg** in the DTITLE field. Then, analogous to the index scheme, we first extract the keytokens from the query: $T = \{\langle u, 1\rangle, \langle n, 2\rangle, \langle f, 3\rangle, \langle o, 4\rangle, \langle r, 5\rangle, \langle g, 6\rangle, \langle f, 25\rangle\}$. We then reduce the keytoken set by merging keytokens that are highly correlated, and use the mapping $\mathcal{F}_h$ to determine the node that is responsible for indexing the set. Let us assume that this node is 000001101010.

Observe that any object that matches the query must have a keytoken set that is a superset of T. So the matching objects can only be indexed at nodes whose IDs are of the form xxxxx11x1x1x. These nodes form a "subhypercube" of the original hypercube H_r, and search over a hypercube can be done by traversing a corresponding *spanning binomial tree* [18]. Moreover, the information in keytokens helps "guide" and "prune" the search tree. For example, in our sample query *unforg**, the next matching keytoken must be of the form $\langle ?, 7\rangle$ (a word expanded from *unforg*), or of the form $\langle ?, 1\rangle$ (accompanied by a new keyword). For the character field, the keytoken correlation also helps us know which candidate keytoken is likely to follow. As such, we can develop an efficient search strategy to explore the search tree. Finally, the information stored in keytokens also allows us to rank and return objects in

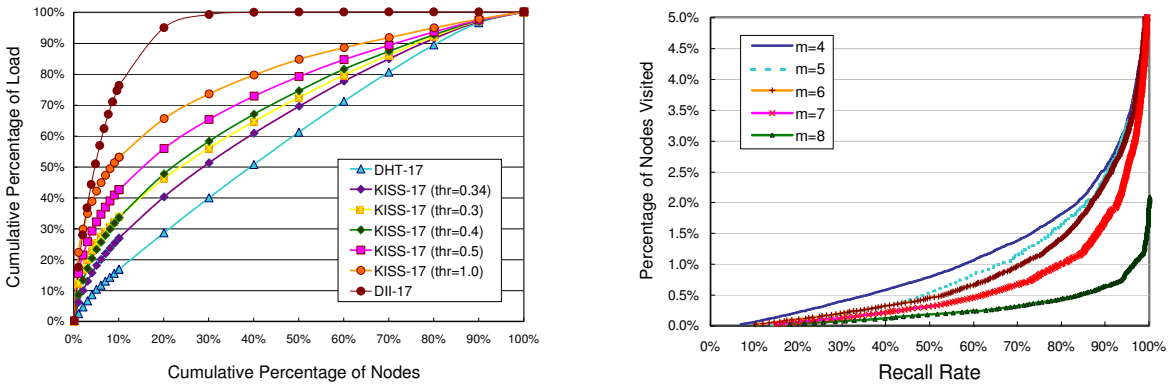


Fig. 5. Load distribution (left) and query Performance (right).

alphabetical or lexicographical order.

2) Experimental Results: We refer to our scheme as **KISS** (Keytoken-based Index and Search Scheme). Fig. 5 left shows load distribution of KISS. We set the hypercube dimension r to 17. (Results for $r = 16$ and 18 are similar.) "KISS-17 (thr=v)" means that a threshold v is used in clustering keytokens, and $v = 1$ implies that no clustering is used to reduce keytoken sets. For comparison, the scheme "DHT-17" represents the load distribution that simply uses hash function to distribute objects to nodes in the hypercube. Using hash function to distribute objects in P2P is generally considered as a good approach to balance loads, so we put it here as the benchmark. The scheme "DII-17" uses distributed inverted index in combination with the n-gram technique to index object, where each entry in the inverted index is handled by a randomly selected node. From the figure we see that DII-17 results in an extremely unbalanced load, while "KISS-17 (thr=0.34)" is more close to "DHT-17".

Fig. 5 right shows query performance of KISS (with $\mathrm{thr} = 0.34$). We measured the number of nodes visited with respect to a given recall rate. For the experiment we randomly sampled 2,500 words from the DTITLE field, extracted prefixes of length $m = 4, 5, 6, 7, 8$ from them. The prefixes are used together with some random decade in between 1960-2000 from the DYEAR field and some random category for DGENRE to form a multi-attribute query to search objects. From the figure we see that KISS offers an appealing search performance. For example, given a prefix of length 5, only 0.54% of nodes need to be visited to obtain a recall rate of 50%.

More details can of the results can be found in [20], [21].

E. Keytoken-Based Index Scheme for Wildcard Search

Keyword-based search has a variety of forms, among which, *wildcard search* provides the query functionality to locate and retrieve the desired objects by some part of a keyword, for example, *-net-* for *internet*, *cybernet*, *network*, etc. Wildcard search is usually presented by the question mark operator '?' for matching any single character, and by the asterisk operator '$*$' for matching any number of characters. The two operators are often used in combination with exact keyword match, like *ACM SIG* proceedings* to search proceedings from all ACM special interest groups. This is very useful when people have only partial information about their target objects, or wish to retrieve a sequence of objects.

It is easy to see that prefix search is simply a kind of wildcard search. Because wildcard search is considerably more difficult, to our knowledge, before our work, no research for character-based wildcard search has been reported for P2P networks. Our solution for wildcard search is based on the technique we developed for prefix search. We call our system **KISS-W** (*Keytoken-based Index and Search Scheme for Wildcards*), as it uses a novel technique to extract *keytokens*—character-position information—from keywords to index objects over a logical hypercube.

The index scheme of KISS-W relies on a very fundamental process called *tokenization* to extract characters and their position information in a keyword to index objects. Let $\mathcal{A}$ be the set of alphabets in consideration. A *keytoken* is a pair $\langle c, i\rangle$, where $c \in \mathcal{A}$ and i an integer. For notational simplicity, we sometimes write $\langle c, i\rangle$ as ci when no confusion is possible. Let $\mathcal{W} \subset \mathcal{A}^+$ be the set of keywords used in the system. For each keyword $w \in \mathcal{W}$, we use $w[i]$ to denote the i^{th} character of w, and $||w||$ to denote the length of w. A *tokenization* is a process to extract keytokens—the character-position—from a given word w. The position can be counted "forward" $(1, 2, \ldots)$ and "backward" $(-1, -2, \ldots)$. Therefore, we define two types of tokenization, forward and backward, and a combination of both:

- A *forward tokenization* is a function τ_f that extracts keytokens from a given keyword w by counting the position forward; that is
$$\tau_f(w) = \left\{ \langle w[i], i\rangle \,\middle|\, i \le ||w|| \right\}$$
We call $\tau_f(w)$ the *forward keytoken set* of w. For example, $\tau_f(jazz) = \{j1, a2, z3, z4\}$.
- A *backward tokenization* is a function τ_r that extracts keytokens from a given keyword w by counting the position backward; that is
$$\tau_b(w) = \left\{ \langle w[i], -||w|| + i - 1\rangle \,\middle|\, i \le ||w|| \right\}$$
We call $\tau_b(w)$ the *backward keytoken set* of w. For example, $\tau_f(jazz) = \{j\text{-}4,\ a\text{-}3,\ z\text{-}2,\ z\text{-}1\}$.
- A *symmetric tokenization* is a function τ_s that extracts all the keytokens from a given keyword w in both the forward and backward style. That is,
$$\tau_s(w) = \tau_f(w) \cup \tau_b(w)$$
We call $\tau_s(w)$ the *symmetric keytoken set* of w.

Given $\mathcal{W}$, the number of all possible keytokens that can be extracted from $\mathcal{W}$ is no greater than $2|\mathcal{A}| \times l_{\max}$, where $l_{\max}$ is the maximum length of a keyword. We shall use $\mathcal{T} = \bigcup_{w\in\mathcal{W}} \tau_s(w)$ to denote the set of all possible keytokens considered in the system. Clearly, $\mathcal{T}$ can be partitioned into two subsets $\mathcal{T}_f = \bigcup_{w\in\mathcal{W}} \tau_f(w)$ and $\mathcal{T}_b = \bigcup_{w\in\mathcal{W}} \tau_b(w)$. For any set of keytokens $T \subset \mathcal{T}$, we say that T is *valid* if the keytokens are extracted from some words in $\mathcal{W}$; i.e., $T = \bigcup_{w\in K} \tau_s(w)$ for some $K \subset \mathcal{W}$. Note that the character positions in a valid keytoken set must be continuous and span from 1 to some k and from -1 to $-k$.

To make our index scheme general and independent of the underlying physical network, we again present the scheme along with its search mechanisms over an r-dimensional logical hypercube $H_r(V, E)$. Implementation issues concerning the hypercube over an existing DHT network can be found in [18], [22].

To index objects in the hypercube, assume that each node in the hypercube has a unique r-bit binary string as its id. We use $u[i]$, $1 \le i \le r$, to denote the i^{th} bit of u (counting from the right), and $One(u) = \{i \,|\, u[i] = 1\}$ for the set of positions at which u has bit 1. Let $h : \mathcal{T} \to \{1, \ldots, r\}$ be a hash (and onto) function that uniformly and independently maps every keytoken in $\mathcal{T}$ to an integer in $\{1, \ldots, r\}$. We define a mapping $\mathcal{F}_h : 2^{\mathcal{T}} \to V$ as follows: $\mathcal{F}_h(T) = u$ if, and only if, $One(u) = \{h(t) \,|\, t \in T\}$. In other words, $\mathcal{F}_h(T)$ is the node with a binary id whose bits are set by the hash function h according to the keytokens in T. For example, suppose $h(w1) = 3$, $h(e2) = 1$, $h(b3) = 7$, $h(w\text{-}3) = 2$, $h(e\text{-}2) = 6$, $h(b\text{-}1) = 3$, and $r = 10$. Then the keytoken set $\{w1, e2, b3, w\text{-}3,\ e\text{-}2, b\text{-}1\}$ is mapped to the node 0001100111 in H_{10}.

We say that a node u is *responsible* for a keytoken set T if $\mathcal{F}_h(T) = u$. Thus, for every possible keytoken set in the system, there is exactly one node in the hypercube responsible for the set. Note that due to hash collision, a node may be responsible for more than one set of keytokens. A keytoken set T responsible by u is *maximal* if $\mathcal{F}_h(T) = u$ and for all T' such that $\mathcal{F}_h(T') = u$, we have $T' \subseteq T$.

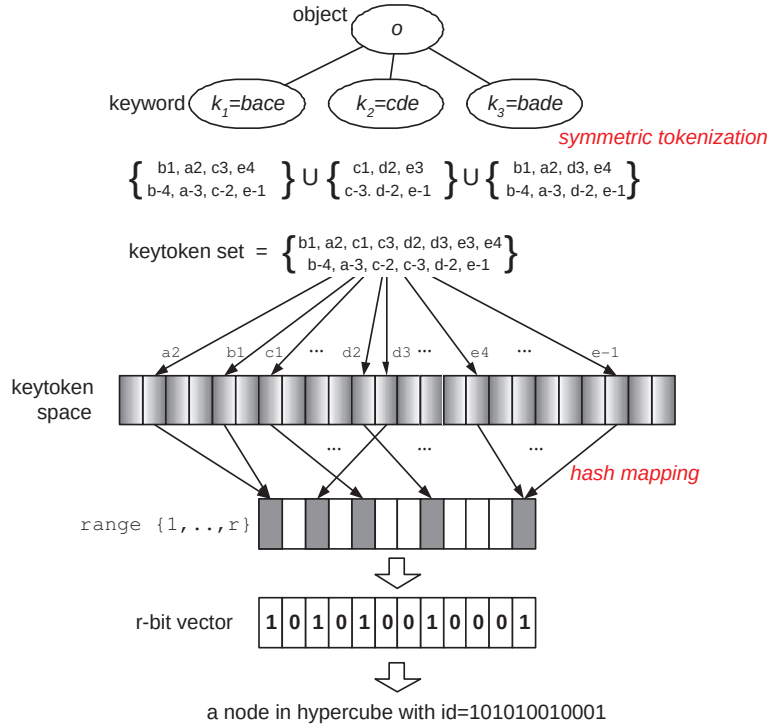


Fig. 6. The KISS-W index scheme.

To index objects at nodes, let σ be an object and K_σ be the set of keywords associated with σ. Let $T_\sigma = \cup_{w \in K_\sigma} \tau_s(w)$ be the set of keytokens extracted from the keywords of σ. Then, σ is *indexed* at the node u such that $\mathcal{F}_h(T_\sigma) = u$. It should be clear that when an object σ is indexed at u, u needs to maintain, in addition to the keyword set of σ, the actual location information of σ (e.g., source IP, port, and file path of the object) from where σ can be retrieved. Colloquially, we sometimes say that a node u *has indexed a keyword* w if w belongs to a keyword set K such that $\mathcal{F}_h(\bigcup_{\alpha \in K} \tau_s(\alpha)) = u$. In other words, u has indexed w if $\forall\, t \in \tau_s(w), u[h(t)] = 1$. Note that more than one node may index a keyword, but only one node can index an object. Fig. 6 illustrates the index scheme for an object o that has three keywords *bace*, *cde*, and *bade*. The object is indexed at node 101010010001.

1) Search Schemes: We identify two types of search: *pin search* and *superset search* [18]. Given a query of keyword set K, pin search returns the object(s) that are described exactly by K. Pin search is useful when one wishes to locate an object for maintenance, e.g., updating its index record, or when one has a precise description about his target object. Superset search, on the other hand, returns the set of objects whose keyword sets contain (but not limit to) K. For example, given a keyword set $\{ab, ce\}$ in Fig. 7, pin search returns only object O_3, while superset search additionally returns O_1. Note that a small keyword set in superset search often results in a large number of matching objects. Most applications will only return a portion of them, or return the results cumulatively in rounds (for example, Google returns 10 matching results per page view).

Orthogonal to the type of search is how a query keyword set is specified. In the paper we allow a query set to be specified via wildcards. Formally, a *query expression* α is of the form $(\mathcal{A} \cup \{?, *\})^+$, where '?' and '$*$' represent the question mark and asterisk operators. A keyword $w \in \mathcal{W}$ *matches* a query expression α if w can be obtained from α by replacing every occurrence of '?' in α by a character in $\mathcal{A}$, and every occurrence of '$*$' by a string in $\mathcal{A}^*$. We allow a query to be composed of query expressions in conjunction.[2] A keyword set K *matches* a query $\alpha_1 \wedge \ldots \wedge \alpha_l$ if every $w \in K$ matches some α_i, and every α_j has some $w' \in K$ to match it, $1 \leq i, j \leq l$. We

[2] Disjunctive query expressions can be easily implemented by taking the union of the results of each query expression.

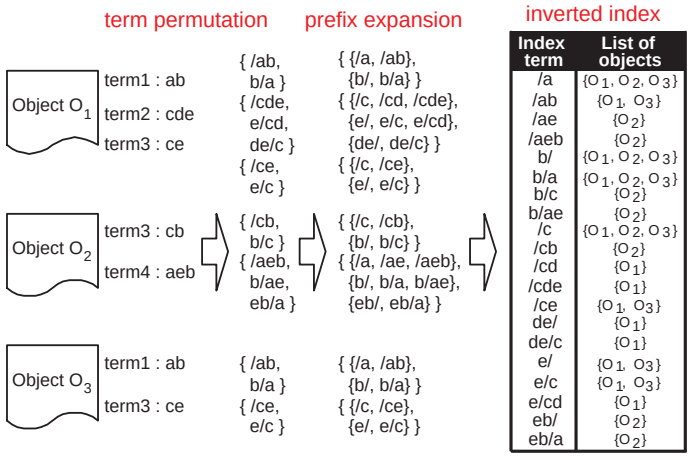


Fig. 7. Transforming wildcard query to prefix match.

will often represent a query $\alpha_1 \wedge \ldots \wedge \alpha_l$ simply as a set $Q = \{\alpha_1, \ldots, \alpha_l\}$. For example, the query $\{a*b,\ c?\}$ is looking for a keyword set containing a keyword with prefix a and suffix b, and another keyword that starts with c and is of length 2.

When a query Q is combined with pin search, an object σ *satisfies* Q if the keyword set associated with σ matches Q. Similarly, when combined with superset search, an object σ *satisfies* Q if σ's keyword set contains a subset that matches Q. The search is to retrieve objects that satisfy the query. For example, pin search for the query $\{a*b,\ c?\}$ in Fig. 7 will return $\{O_2, O_3\}$, while superset search for the same query will return $\{O_1, O_2, O_3\}$.

The detailed search algorithm, along with simulation results, can be found in [22].

IV. Conclusions

To conclude, in this project we have designed and implemented Visitant, an agent-based P2P system, to integrate mobile agent technology with P2P computing. Unlike existing agent-based P2P systems that are often built over an unstructured P2P network, Visitant adopts a structured topology. Likewise, communication can be done efficiently and search of an agent is *guaranteed* in the sense that the target can be located in bounded time and using only bounded resources.

What is more interesting and challenging in the design of the system is to study the resource allocation problem, which, not only important to the system, but is also a very fundamental issue in P2P networks. In this direction, we have achieved several important results, and some (prefix and wildcard search) are pioneer in the field. In addition to the work cited here, there are still some work yet to be published, and some (e.g., more complex search) left in the future work. We hope that more interesting results will be obtained in the near future.

References

[1] J. Baumann, F. Hohl, K. Rothermel, and M. Straser. Mole - concepts of a mobile agent system. *World Wide Web*, 1(3):123–137, 1998.

[2] H. Cai and J. Wang. Foreseer: a novel, locality-aware peer-to-peer system architecture for keyword searches. In *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, volume 3231 of *Lecture Notes in Computer Science*, pages 38–58, New York, NY, USA, 2004. Springer-Verlag.

[3] H.-C. Chang. On the implementation of application programming interface of visitant. Master's thesis, National Central University, Taoyuan, Taiwan, 2007.

[4] A.-H. Cheng and Y.-J. Joung. Probabilistic file indexing and searching in unstructured peer-to-peer networks. *Computer Networks*, 50(1):106–127, 2006.

[5] K. Chu, K. Labonte, and B. N. Levine. Availability and locality measurements of peer-to-peer file systems. In *Proceedings of SPIE (SPIE 2002)*, volume 4868. The International Society for Optical Engineering, August 2002.

[6] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceedings of the Twenty-Second International Conference on Distributed Computing Systems (ICDCS 2002)*, pages 23–32. IEEE Computer Society, July 2002.

[7] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a common api for structured peer-to-peer overlays. In *Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS '03)*, pages 33–44, Berkeley, California, United States, February 2003.

[8] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, first edition, 1998.

[9] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. Technical report, The Globus Alliance, 2002. Available from http://www.globus.org/research/papers/ogsa.pdf.

[10] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. The vocabulary problem in human-system communication. *Communications of the ACM*, 30(11):964–971, 1987.

[11] P. Ganesan, Q. Sun, and H. Garcia-Molina. Adlib: A self-tuning index for dynamic peer-to-peer systems. In *Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*, pages 256–257, Washington, DC, USA, 2005. IEEE Computer Society.

[12] S. Gauch, J. Wang, and S. M. Rachakonda. A corpus analysis approach for automatic query expansion and its extension to multiple databases. *ACM Transactions on Information Systems*, 17(3):250–269, 1999.

[13] O. D. Gnawali. A keyword-set search system for peer-to-peer networks. Master's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, United States, June 2002.

[14] M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, and I. Stoica. Complex queries in DHT-based peer-to-peer networks. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS 2002)*, volume 2429 of *Lecture Notes in Computer Science*, pages 242–259. Springer-Verlag, 2002.

[15] T. Illmann, F. Kargl, M. Weber, and T. Kruger. Migration in java: problems, classifications and solutions. In *Proceedings of the International ICSC Symposium on Multi-Agent and Mobile Agents in Virtual Organizations and E-Commerce (MAMA 2000)*, pages 281–287, Wollongong, Australia, December 2000.

[16] K. S. Jones. Index term weighting. *Information Storage and Retrieval*, 9(11):619–633, 1973.

[17] S. Joseph. NeuroGrid: Semantically routing queries in peer-to-peer networks. In *Proceedings of the International Workshop on Peer-to-Peer Computing (co-located with Networking 2002)*, volume 2376 of *Lecture Notes in Computer Science*, pages 202–214. Springer-Verlag, 2002.

[18] Y.-J. Joung, C.-T. Fang, and L.-W. Yang. Keyword search in DHT-based peer-to-peer networks. In *Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS 2005)*, pages 339–348. IEEE Computer Society, 2005.

[19] Y.-J. Joung, M. kai Lin, Y.-C. Lin, C.-W. Chang, and H.-H. Chen. A simulation of Fujiwhara effect on a structured agent-based peer-to-peer system. In *Proceedings of the Second International Conference on Communication Systems Software and Middleware (COMSWARE 2007)*, pages 1–7. IEEE Computer Society, 2007.

[20] Y.-J. Joung and L.-W. Yang. KISS: A simple prefix search scheme in P2P networks. In *Proceedings of the Ninth International Workshop on the Web and Databases (WebDB 2006)*, pages 61–66, June 2006.

[21] Y.-J. Joung and L.-W. Yang. Multi-dimensional prefix search in P2P networks. In *Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing (P2P 2006)*, pages 67–68, Sept. 2006.

[22] Y.-J. Joung and L.-W. Yang. Wildcard search in structured peer-to-peer networks. *IEEE Transactions on Knowledge and Data Engineering*, 19(11):1524–1540, 2007.

[23] S. Klink, A. Hust, M. Junker, and A. Dengel. Improving document retrieval by automatic query expansion using collaborative learning of term-based concepts. In *Proceedings of the 5th International Workshop on Document Analysis Systems (DAS)*, volume 2423 of *Lecture Notes in Computer Science*, pages 376–387, Princeton, NJ, USA, August 2002. Springer-Verlag.

[24] J. Li, B. T. Loo, J. M. Hellerstein, M. F. Kaashoek, D. R. Karger, and R. Morris. On the feasibility of peer-to-peer web indexing and search. In *Proceedings for the Second International Workshop on Peer-to-Peer Systems (IPTPS 2003)*, volume 2753 of *Lecture Notes in Computer Science*, pages 207–215. Springer-Verlag, 2003.

[25] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th International Conference on Supercomputing (ICS)*, pages 84–95. ACM Press, 2002.

[26] M. Magennis and C. J. van Rijsbergen. The potential and actual effectiveness of interactive query expansion. In *Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '97)*, pages 324–332, New York, NY, USA, 1997. ACM Press.

[27] M. Mitra, A. Singhal, and C. Buckley. Improving automatic query expansion. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '98)*, pages 206–214. ACM Press, 1998.

[28] K. Nakauchi, Y. Ishikawa, H. Morikawa, and T. Aoyama. Peer-to-peer keyword search using keyword relationship. In *Proceedings of the Third International Workshop on Global and P2P Computing (GP2PC 2003)*, pages 359–366. IEEE Computer Society, May 2003.

[29] V. A. Pham and A. Karmouch. Mobile software agents: An overview. *IEEE Communications Magazine*, 36(7):26–37, July 1998.

[30] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2001)*, pages 161–172. ACM Pres, August 2001.

[31] P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Proceedings of the 2003 ACM/IFIP/USENIX International Middleware Conference (Middleware 2003)*, volume 2672 of *Lecture Notes in Computer Science*, pages 21–40. Springer-Verlag, 2003.

[32] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 2001 IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350. Springer-Verlag, 2001.

[33] G. Salton. *Automatic Text Processing. The Transformation, Analysis and Retrieval of Information by Computer*. Reading, MA: Addison-Wesley, 1989.

[34] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of the 2002 Multimedia Computing and Networking (MMCN 2002)*. The International Society of Optical Engineering, January 2002.

[35] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. HyperCuP: Hypercubes, ontologies and efficient search on P2P networks. In *Proceedings of the 2002 International Workshop on Agents and Peer-to-Peer Computing (AP2PC 2002)*, volume 2530 of *Lecture Notes in Computer Science*, pages 112–124. Springer-Verlag, 2003.

[36] S. Shi, G. Yang, D. Wang, J. Yu, S. Qu, and M. Chen. Making peer-to-peer keyword searching feasible using multi-level partitioning. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS 2004)*, volume 3279 of *Lecture Notes in Computer Science*, pages 151–161. Springer-Verlag, 2005.

[37] 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 *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2001)*, pages 149–160. ACM Press, August 2001.

[38] C. Tang and S. Dwarkadas. Hybrid global-local indexing for efficient peer-to-peer information retrieval. In *Proceedings of the First Symposium on Networked Systems Design and Implementation (NSDI 2004)*, pages 211–224. USENIX, 2004.

[39] B. Yang and H. Garcia-Molina. Improving search in peer-to-peer systems. In *Proceedings of the Twenty-Second International Conference on Distributed Computing Systems (ICDCS 2002)*, pages 5–14. IEEE Computer Society, July 2002.

[40] F. Zhou, L. Zhuang, B. Y. Zhao, L. Huang, A. D. Joseph, and J. Kubiatowics. Approximate object location and spam filtering on peer-to-peer systems. In *Proceedings of the 2003 ACM/IFIP/USENIX International Middleware Conference (Middleware 2003)*, volume 2672 of *Lecture Notes in Computer Science*, pages 1–20. Springer-Verlag, 2003.

出席國際學術會議心得報告

計畫編號	NSC 95-2221-E-002-058
計畫名稱	Visitant: 一個以代理人爲基礎的泛用性點對點應用服務平台(3/3)
出國人員姓名 服務機關及職稱	莊裕澤, 國立台灣大學管理學院資訊管理學系教授
會議時間地點	Bangalore, India
會議名稱	THE SECOND IEEE/Create-Net/ICST International Conference on COMmunication System softWAre and MiddlewaRE (COMSWARE)
發表論文題目	A Simulation of Fujiwhara Effect on a Structured Agent-Based Peer-to-Peer System, Yuh-Jzer Joung, Meng-kai Lin, Yi-Chun Lin, Chia-Wei Chang, Huang-Hsu Chen

一、參加會議經過

THE Second IEEE/Create-Net/ICST International Conference on COMmunication System softWAre and MiddlewaRE (COMSWARE) (http://www.comsware.org/2007/index.html) is a new conference dedicated to addressing emerging topics and challenges in Communications Software. The goal of the conference is to create a world-class gathering of researchers from academia and industry, practitioners, business leaders, intellectual property experts and venture capitalists, providing a launch pad for new innovative business and technology.

This year the conference is held at Bangalore, India, from Jan. 7-12, 2007. The conference received 281 submissions, and accepts 95 papers. The conference opened up with five workshops. Seven keynote speakers were invited, include

- N. R. Narayana Murthy, Chairman of the Board and Chief Mentor, Infosys Technologies Limited, India
- Kiran karnik, President of NASSCOM
- Hamid Ahmadi, Motorola's Chief Architect
- Jim Kurose, Professor of Computer Science at the University of Massachusetts
- Dipankar Raychaudhuri, Professor, Electrical & Computer Engineering Department and Director, WINLAB (Wireless Information Network Lab) at Rutgers University.
- Ken Birman, Professor of Computer Science at Cornell University
- Arun Kant, Vice President of Agere Systems

The list represents a good combination of academia and industry.

二、與會心得

This is a relative broad conference. Several topics were covered, include: P2P, WWW, Middleware, Wireless Network Enhancement, Cooperation Schemes in Wireless Networks,

Multihop Topology, Location and Positioning, Agent-based Systems, Location Schemes, Network Security, Wireless Network Security, Sensor Network Security, WMAN/WWAN, Measurement, MIMO/OFDM,Ad hoc Network Enhancement, Multihop Routing, WLAN, Network Applications I, II, III, Sensor Network I, II, III, Advances in Internet, Network Routing, Grid Computing.

Our paper was characterized in the P2P category. The paper presents an agent-based P2P system, called **Visitant**, to integrate mobile agent technology with P2P computing. The use of structured P2P systems as the base layer allows agents to efficiently locate resources and communicate with each other. It also allows us to incorporate a security control mechanism to prevent malicious hosts from colluding with each other to paralyze the system. To demonstrate Visitant, we built an application to called **Cyclone Rover**, over Visitant. Cyclone Rover simulates Fujiwhara effect---the tendency of two nearby tropical cyclones to rotate around each other. We received several questions in the presentation, mostly related to the simulation scale, as, due to resource availability, here we only used a small number (<10) of nodes to implement the system. Large scale simulation would certainly worth for future work.

Overall, it is a pleasure to attend the conference. Although it is an international conference, I believed that half of the attendees are from India. It is also interesting to see Bangalore, the IT capital of India. I have met several researchers from local companies. The infrastructure has been established in quite an amazing speed. Still, I saw some uneven distribution of resources in the city, and lots of people are still living in poverty. In terms of IT human resources, India clearly has plenty of them. Compared to Taiwan, although we clearly outpace India in the infrastructure, but the gap has been shortened each year!