Keyword Search for Enhancing JXTA Discovery Service in Peer to Peer Networks [1]

Tsung-Hsuan Chang
Department of Computer Science
National Chiao Tung University
Hsinchu 300, Taiwan
sam0408.cs95g@nctu.edu.tw

Kuochen Wang
Department of Computer Science
National Chiao Tung University
Hsinchu 300, Taiwan
kwang@cs.nctu.edu.tw

Chung-Yuan Hsu
Department of Computer Science
National Chiao Tung University
Hsinchu 300, Taiwan
theone0615@gmail.com

Abstract—**JXTA (Juxtapose) is emerging as the next generation P2P (peer to peer) platform and has been adopted by many applications, such as instant messaging systems, file sharing systems, and real-time collaboration platforms, etc. However, JXTA is lack of keyword search. Without the support of keyword search, the query must contain the exact full name of a desired resource. It is inconvenient for users in this aspect. In this paper, we propose a mechanism called *JXTA Keyword Search (JKS)* with the support of *Chinese Keyword Partition* (CKP). JKS has two objectives. First, it is providing keyword search upon JXTA. New publishing and discovery schemes are proposed for keyword search. Second, it is designing a Chinese Keyword Partition method to enhance the number of exact matches. Experimental results, based on real data obtained from some notable portal sites, show that the number of exact matches of the proposed JKS is comparable to that of KAD with English queries. Furthermore, JKS is 237% less than KAD in terms of bandwidth cost with English queries. The proposed JKS performs much better than KAD with mixed queries (English + Chinese). The research results are applicable for P2P applications (e.g. file sharing, multimedia streaming sharing, internet service discovery, etc.) built on JXTA.**

Keywords — **Discovery service, JXTA, keyword search, peer to peer network.**

1. INTRODUCTION

JXTA is a set of protocols which provide different network devices to communicate and collaborate in a P2P manner. JXTA is a common framework provided for P2P application development. There are several advantages of JXTA. (1) It accepts heterogeneous devices like PCs and mobile devices [2], [3]. (2) There is a J2ME version for mobile devices to build applications to access JXTA networks [4]. (3) It provides firewall routing capabilities [2]. It is used by many projects to establish their applications [5], [6], [7]. It is emerging as a next-generation P2P platform [8].

There are two types of P2P overlay networks: *structured* and *unstructured* [9]. Most structured P2P networks are based on distributed hash table (DHT) technology. The advantage of DHT is its cost-effectiveness on query routing. Given a key, it guarantees to find an object within bounded cost. But it is useful only when the user has the exact file name. Most of the time, the user may only know partial information [10]. Thus, augmenting DHT with a keyword-based search capability is a valuable extension [11].

JXTA uses loosely-consistent DHT (LC-DHT) as the underlying query routing mechanism. The operation is like DHT which stores a file index in a selected peer according to some kind of hash values. However, it is also lack of keyword search. There are two objectives for this research. The first objective is designing the keyword search function upon JXTA. The second objective is designing a Chinese Keyword Partition (CKP) method to enhance the number of exact matches.

The rest of this paper is organized as follows. The preliminary knowledge of JXTA is presented in Section 2. Section 3 discusses related work. Section 4 depicts design approach. Experimental results and discussion are shown in Section 5. Section 6 gives concluding remarks.

2. PRELIMINARIES

2.1 Introduction of JXTA [5]

JXTA was started by Sun Microsystems in 2001. Network devices can join the JXTA network by applications they run if it conforms to the JXTA set of protocols and could parse XML. An advertisement

[1] This work was supported by the National Science Council under Grants NSC96-2628-E-009-140-MY3 and NSC97-3114-E-009-001.

is an XML document. JXTA uses advertisements to describe resources. Resources can include files, documents, pipes, and media etc. [12]. An example of a pipe advertisement is showed in Fig. 1.

```
<?xml version="1.0"?>
<!DOCTYPE jxta:PipeAdvertisement>
<jxta:PipeAdvertisement
xmlns:jxta="http://jxta.org">
<Id>
urn:jxta:uuid-59616261646162614E504720
503250338E3E786229EA460DADC1A176
B69B731504
</Id>
<Type>JxtaUnicast</Type>
<Name>TestPipe</Name>
</jxta:PipeAdvertisement>
```

Fig. 1. An example pipe advertisement [12].

The JXTA network is composed of connected peers. There are several kinds of peers. Two of them (rendezvous peer and edge peer) are shown in the Fig. 2. The other kinds of peers are not discussed in this paper. Most peers in a JXTA network are edge peers. An edge peer can perform search, process discovery requests from others, but doesn't involve forwarding discovery requests. Rendezvous peers have a list of other rendezvous peers and communicate with each other in order to maintain the list. They help propagate discovery requests to other peers.

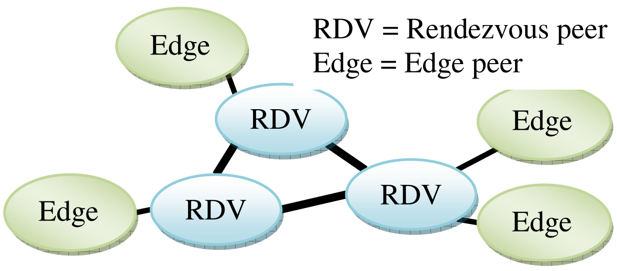

Fig. 2. JXTA network organization.

2.2 JXTA routing, publish and discovery service

Previous section gives an overview of JXTA. Here we briefly describe JXTA routing, publish and discovery service [13].

Routing

In JXTA, each rendezvous peer has a Rendezvous Peer View (RPV) which is an ordered list of known rendezvous peers by their peer IDs. A RPV is maintained through exchanging rendezvous peer information or through well-known seeding rendezvous peers [14]. Moreover, the message received from or sent to other rendezvous peers would update the RPV [13]. The RPVs on different rendezvous peers are not always consistent due to peer joining and leaving. But they will eventually converge to a consistent RPV when changes in the network are not too frequent [13].

For routing a key in a traditional DHT, suppose *h(key)* is the hash value of the key. The key is routed to a peer whose ID is close to the hash value of the key. In LC-DHT, the proportion of *h(key)* to the maximum hash value is computed first. A rendezvous peer is selected from the RPV according to the proportion. Then the key is routed to the selected peer.

Publish

The file owner allows others to search its files by publishing the index entries. Each entry contains a pair of *<field, value>* which is extracted from an advertisement. An advertisement is created from each file and contains information of the file. Table 1 shows some index entries extracted from an advertisement shown in Fig. 1.

Table 1. Some index entries extracted from Fig. 1.

Field	Value
ID	urn:jxta:uuid-59616261646162614....
Type	JxtaUnicast
Name	TestPipe

In an edge peer, the steps of publish are listed below:

(1) Create an advertisement of files and store it into the local advertisement database.
(2) Extract index entries from the local advertisement database and send these entries to a connected rendezvous peer.
(3) In a rendezvous peer: retrieve entries from index messages and store them into the index entry database.
(4) In a rendezvous peer: for each entry of index messages, use a routing mechanism to send each

entry to a different rendezvous peer and then the received peer stores it.

Discovery

Discovery service helps users to find out who owns files they are looking for. JXTA discovery service allows user input one set <*Field, Value*> (e.g. <name, "A Whole World">) and finds out advertisements which contain it. Since an RPV is not always consistent, it is possible to route a query to an incorrect rendezvous peer. JXTA designed a *limited-range walker algorithm* to cope with it. In this case, the query walks along rendezvous peers in both directions of the RPV. The walker is stopped when a result is found, the end of RPV is reached, or a defined hop count is run out [13].

In an edge peer, the steps of discovery are listed below:

(1) Create a query message that contains user input, and then send the query message to a connected rendezvous peer.
(2) In a rendezvous peer: search its local advertisement database.
  (2a) Find results: return found advertisements to the query peer.
  (2b) No result: go to step 3.
(3) In a rendezvous peer: search its index entry database.
  (3a) Find results: forward the query to the peers which have published the index entry. The owner peer then returns the desired advertisement to the query requester.
  (3b) No result: if the rendezvous peer is the responsible peer go to step 5; otherwise, go to step 4.
(4) Forward the query to a responsible rendezvous peer. The receiving peer will start from step 2.
(5) Start the walker algorithm (described above).

The JXTA publish and discovery scenario is illustrated in Fig. 3. The number nearby an arrow line is corresponding to the step described above.

3. Related work

3.1 JXSE-CMS [15]

JXSE-CMS (content manager service) provides file sharing and retrieving services on JXTA. It also provides metadata search, but this function can't be used in the remote search because an index is found only when a query is routed to the rendezvous peer which contains the index. JXSE-CMS define its own content advertisement which could contain *content id* (*cid*), length, description, name and etc. The *cid* field contains the unique 128-bit MD5 checksum of the file. The index of an advertisement is published by the underlying JXTA functions.

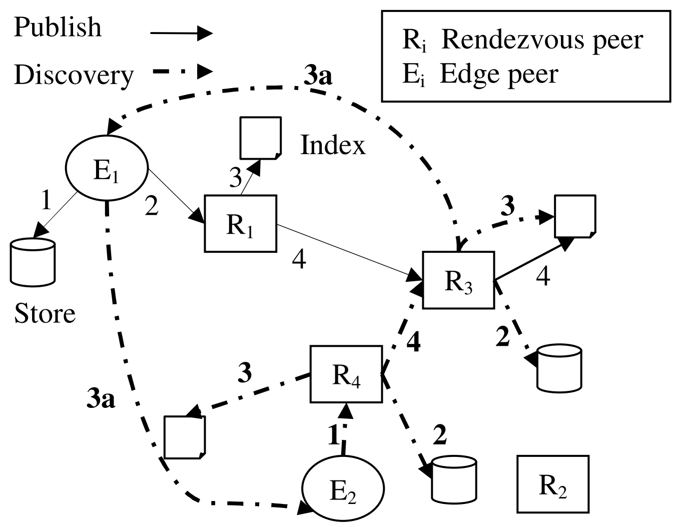

Fig. 3. JXTA publish and discovery scheme.

3.2 Rich metadata searches [16]

In Fig. 4, the metadata search layer is implemented based on the existing CMS. The abundant metadata is added into an advertisement so that much more searchable attributes could be supported in a query expression. The query could include author, publisher, etc. But this mechanism can't work in a recent JXTA version in the internet because the resource is stored in a selected peer according to some kind of a hash value. The metadata search only works in a local network.

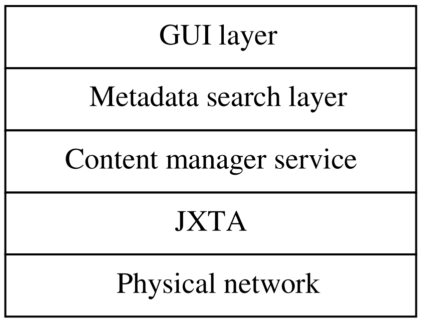

Fig. 4. Metadata architecture.

3.3 KAD [17]

There are many DHT-based methods that were proposed but very few of them are wildly deployed [18]. KAD is one of these and is based on Kademlia networks. Several papers pointed out that there are several million users using KAD [18]. In the following, we briefly describe about operations related to the keyword search in KAD. The overview below includes keyword partition, publishing objects and searching objects.

Keyword partition

The keyword sets are extracted by ripping names of objects. A name is ripped by some special characters, e.g. space, '.', ',', ':', '-', '_', '*', '?', etc. The keywords that consist of two or less letters are deleted [19]. For example, the keyword set of "A Whole World" is {"Whole", "World"}.

Publish objects

A peer enables other peers access its owning files by publishing references. There are two kinds of references: *source key* and *keyword key* [18].

(1) Source key

A source key is computed by hashing the content of the file. It is used to search the peers owning the file.

(2) Keyword key

A keyword key is computed by hashing the keyword gained from the keyword partition described above. It is used to search the files whose name contains the keyword.

The 2-level publishing scheme publishes these two kinds of references. The benefit of the 2-level scheme is decreasing number of publishing references [17]. The references are stored in the peers whose KAD ID agrees in the first 8 bits with the reference [20]. The 2-level publishing scheme is described as follows:

(1) Source publishing

It finds and stores a source key in the responsible peers. The address information of a publishing peer is stored with the key.

(2) Keyword publishing

It finds and stores a keyword key in the responsible peers. The source key and related information of a published file (e.g. size, type, name, etc.) are stored with the key.

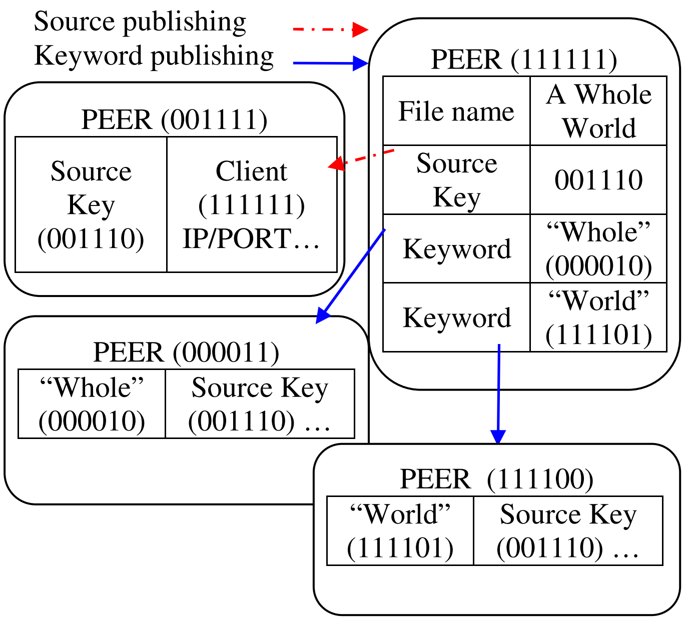

Fig. 5. KAD publishing scheme.

Search objects

Like publishing, searching files is also a 2-level scheme: *keyword search* and *source search.*

(1) Keyword search

The hash value of the first word of user input is computed. The rest of words and additional attributes are packed in form of a search tree. A query consists of a hash value of the first keyword and a search tree. The query is routed to the peers that have a KAD ID close to the hash value. The matching results are responded from the peers and carry the information of source keys and files. Fig. 5 describes the publishing scheme.

(2) Source search

A user chooses a desired file from returned results. Then the source key of the file is used for searching the peers who have the file. The returned results would be added into the download queue of the file.

Finally, Table 2 shows the comparison of the existing methods described above and the proposed JKS.

4. Design Approach

As mentioned before, the proposed JKS has two objectives: (1) adding a keyword search function upon JXTA; (2) designing a Chinese Keyword Partition (CKP) method into JXTA. The latest Java

Table 2. Qualitative comparison of existing approaches.

	JXSE-CMS **[15]**	Rich **[16]**	KAD [17]	JKS (Proposed)
P2P network	JXTA	JXTA	Kademlia	JXTA
Search capability	Metadata	Metadata	Keyword + Attribute	Keyword + Attribute
Query key	Content id or exact file name[1]		Keyword + Attribute	
Special feature	N/A	N/A	N/A	Chinese Keyword Partition (CKP)
Number of exact matches	Low	Low	Medium	High
Bandwidth cost	Low	Low	High	Medium
Storage cost	Low	Low	Low	High

implementation of JXTA is JXSE 2.5. JKS is based on JXSE 2.5.

4.1 Keyword search

The solution of how to add keyword search functionality upon JXTA is described in two parts: publish and discovery.

Publish

The publish scheme allows to publish several keywords for a single file. It involves adding keyword partition, a new advertisement, and modifying the publishing behavior of each rendezvous peer. The keyword partition algorithm will be described in Section 4.2. The publish scheme uses the content advertisement extended from JXSE-CMS to describe files, which is detailed as follows.

To let users search files by file attributes (e.g. author), the related pairs can added into advertisements, (e.g. <Author> David). These attributes are not index entries but are used when determining whether a query and an advertisement match. The index entries are extracted from "name" and "cid" fields of the content advertisement. Then the entries are sent to the connected rendezvous peer. When the rendezvous peer receives entries, it stores the entries with field "cid" into the index entry database. If the entry field is "name," the keywords are extracted by the keyword partition algorithm. These keywords are added into entries and also store into the index entry database with field "keyword." Then the entry with field "name" is discarded.

In the original JXTA source code, the rendezvous peer distributes each entry to other rendezvous peers to store. The 2-tier hierarchical architecture allows to further decrease network traffic and storage cost. If there are multiple same entries, only one entry is sent. Besides, the sources of entries are all changed to the rendezvous peer. During the discovery phase, the query would be routed to the rendezvous peer and then be forwarded to the real publisher.

The steps of file publishing are listed below:

(1) Create an advertisement and save it in a local advertisement database.

(2) Extract index entries from the local advertisement database and pack these entries into an index message.

(3) Send the index message to the connected rendezvous peer.

(4) In the rendezvous peer: retrieve entries from the index message and store them into the index entry database. If an entry field is "name," it is not stored. Keywords are obtained by ripping file name, and keyword pairs are stored into the index entry database.

In the rendezvous peer: for each entry of the index message, remove duplicated entries and change the source to the rendezvous peer. Then use the routing mechanism to send each entry to a different rendezvous peer and store it.Fig. 6 shows the publishing scenario and the number nearby each arrow line is corresponding to the step described above.

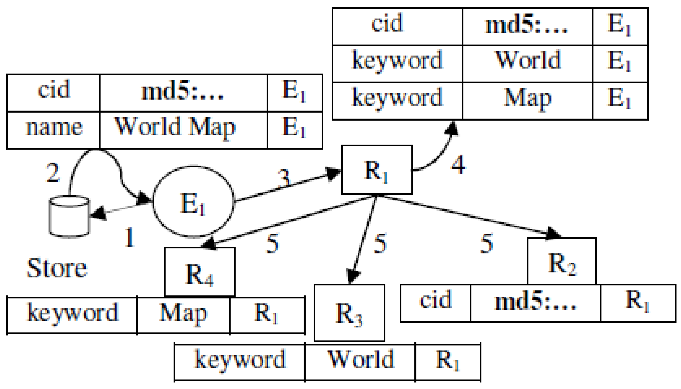

Fig. 6. Flowchart of file publishing.

Discovery

The discovery service enables users to lookup desired files by keywords and file attributes. To enable keyword and attribute discovery, the format of a query is changed. In the original JXTA, a query includes *type*, *threshold* (maximum response number), *attr*, *value*, *query ID*, *source peer ID*, etc. Attr and value is the pair of <field, value> which was mentioned before.

A new query format is defined by adding three new kinds of tags into the original query format. These tags are: <Number>, <Append> and <Attribute>. <Number> is used to ask how many entries of <Attr, Value> in the peer. <Append> contains keywords. <Attribute> contains file attributes. Both <Append> and <Attribute> are used to filter the results, searched by <Attr, Value>, to refine the results.

To start a query, a user inputs some keywords and file attributes. To avoid generating large network traffic, a query is created for each keyword and an added tag <Number> is included in the query. Queries are first routed to peers who might contain index entries for the keyword. The receiving peer checks if the tag <Number> is set and the number of entries for the keyword is returned. After receiving the number of entries for each keyword, the keyword with minimum number is set in tag <Value>. The tag <Attr> contains "keyword." The rest of keywords are set in tag <Append>, and file attributes are set in tag <Attribute>.

Then, a query is routed to the peer who might contain an index entry of <"keyword", Value>. The peer forwards the query to peers who published the entries. To decrease network traffic, only ten peers are selected for forwarding. Besides, only one query is sent if two entries are published from the same peer. After the query is forwarded to the entry publisher, the <"keyword", Value> specified in the query is used to retrieve advertisements which contain it. If either <Append> or <Attribute> is set, it is used to filter the results. The matching results are returned to the requester.

The steps of the discovery scheme are listed below:

(1) Rip keywords from user input (use the keyword partition algorithm described in the publish scheme).

(2) Ask for the index number of each keyword.

(3) If one of the numbers is zero, the discovery process terminates. Pack and send the query to a connected rendezvous peer.

(4) The connected rendezvous peer sends a query to the peer who is responsible for storing the index entries of the query keyword.

(5) The peer receiving the query finds out peers who published the index entry containing the query keyword. Then forwards the query to found peers.

(6) The peer receiving the forwarding query is a rendezvous peer who is connected to the real publisher. The peer finds out these publishers and forwards the query to the found peers.

(7) The peer sends found advertisements to the query peer.

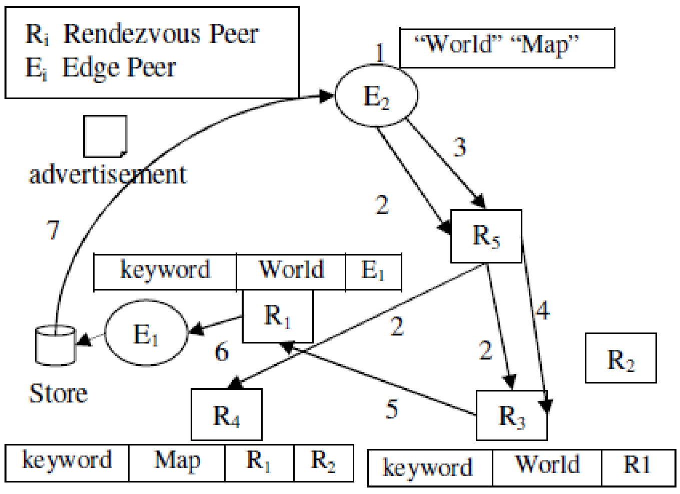

Fig. 7. Flowchart of discovering the file "World Map".

A user chooses a desired file from returned results. If there are not enough file owner peers acquired from returned results, the *cid* of the file is used for searching the peers who have the desired file. This process is like the KAD source key search. The discovery scheme is illustrated in Fig. 7. The number nearby an arrow line is corresponding to the step described above.

4.2 Chinese keyword partition (CKP)

The aim of CKP is to let files with Chinese names increase their possibilities of being searched out. In other words, increase the number of exact matching results for a query containing Chinese keywords. The character characteristic of Chinese sentences makes it difficult to be partitioned by the keyword partition algorithm of KAD because it does not have spaces between each character. Users who publish files need to manually add some special delimiters between Chinese characters. But some users may not know how to separate or are not diligent to change all files' names. It results in only a subset of files with Chinese names being searched out if a user uses a complete file name to query.

CKP provides a method to automatically distinguish Chinese keywords from file names. CKP involves a Chinese keyword list (CKL) and a keyword partition algorithm. Each rendezvous peer needs to maintain a CKL. The list contains records in form of <Keyword, Frequency>. The Keyword is Chinese. When a rendezvous peer receives a query from other peers, it gets a keyword set from the query by using the CKP algorithm. For each keyword in the keyword set, a rendezvous peer checks whether the keyword is contained in the CKL. If there is no record for the keyword, it adds a record <keyword, 1> into the CKL. Otherwise, it adds 1 to Frequency of the old record.

The CKL needs to be maintained to control the use of storage. When the number of records in the CKL exceeds 2,000, all records with frequency less than 3 are removed. When the total frequencies of all records exceed 100,000, the frequencies of all records are divided by 2. If the original frequency is 1, the record is deleted. If the number of records in the CKL is less than 200, a peer uses its RPV to ask for records from other rendezvous peers. Each transmission is limited to 200 records selected from the top of the CKL. A peer would continually ask for records until the number of records exceeds 200.

An array TK (Top Keywords) contains 200 keywords with the most frequencies in the CKL. It is updated once per hour and after asking for record transmission from one of rendezvous peers. The proposed keyword partition algorithm, the CKP algorithm, is presented in Fig. 8.

Chinese Keyword Partition (CKP) Algorithm:

```
INPUT file_name
OUTPUT R_Keyword_Set
Key_Set = Split file_name by special characters
        {space, '.', ',', ':', '-', '_', '*', '?', etc.}
For each Keyword in Key_Set
    Split Keyword if it consists of Chinese and
    English and then add results into
    Keyword_Set
For each Keyword in Keyword_Set
    IF Keyword is not Chinese THEN
        IF Keyword.length > 2 THEN
            Add Keyword into R_Keyword_Set
        ENDIF
    ELSE            //Keyword is Chinese
        Add Keyword into R_Keyword_Set
        FOR i = 1 to 200
            IF Keyword contains TK[i] THEN
              Add TK[i] into R_Keyword_Set
            ENDIF
    ENDIF
  return R_Keyword_Set
```

Fig. 8. Chinese keyword partition (CKP) algorithm.

From the CKP algorithm, if the file name contains any keyword of TK, the keyword is retrieved and added into the keyword set. The overhead of CKP can be examined from three aspects: network traffic, storage cost and CPU load.

Network traffic

It is produced by record transmissions among rendezvous peers. It only occurs when the number of records in the CKL is less than 200. If the number of records exceeds 200, it seldom goes down to the number less than 200 again because the list keeps growing while queries keep coming. If there are enough queries in the network, the demand for transmission will be very low and the network traffic is little.

Storage cost

It is mainly produced by the CKL. Suppose each record in the CKL is 50 bytes (46 bytes for Keyword (string) and 4 bytes for Frequency (Integer)). When the CKL has the maximum record number 2,000, the size of the CKL is 2,000 * 50 bytes = 100 KB. 100 KB is small compared to the memory size currently (512 MB, 1 G …).

CPU load

CPU load is evaluated by the elapsed system time from the beginning to the end of executing the CKP algorithm. A simple test program in Java was written to run the CKP algorithm. The program was run on a PC (AMD Athlon(tm) 64 X2 Dual Core Processor 3600+ (2.01 GHz) CPU, 1 GB RAM, Windows XP). Fig. 9 shows that it takes 6.5 seconds to run the CKP algorithm 100,000 times. The overhead of running the CKP is little.

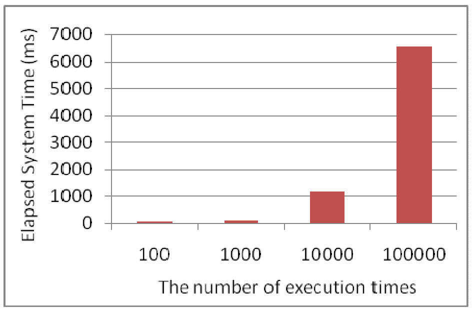

Fig. 9. The elapsed system times for running the CKP algorithm various number of times.

5. Evaluation and Discussion

5.1 Experiment environment

Table 3 gives description about four kinds of data collected for experiments. Mixed queries consist of Chinese and English queries with a ratio of approximately 5:2. Two kinds of files were produced in the same way. First, 2000 queries with most frequencies were selected from query data. Second, each query obtained five website titles by Google SOAP search API [23]. The total number of websites is 10,000 and the titles of websites are treated as file names. Third, each file name is assigned a replicate number according to Zipf's law, and each file name is replicated this number of times. The total number of file names after replication is 100,000. Finally, file names were distributed among peers according to Zipf's law at runtime. In addition, all peers in the system are stable (no leaving or joining) and the replication mechanism for publishing is removed for easy evaluation. Table 4 shows some system parameters.

Table 3. Experiment data.

Data type	Source	Amount	Date
English query	AOL query log [21]	10000	released on 2006.8.4
Mixed query	Yahoo [21]	10000	2008.5.29
English file	Google	100000	2008.5.29
Mixed file	Google	1000003	2008.5.29

Table 4. The system parameters used in the experiments.

Notation	Description	Value
	File popularity distribution	Zipf's law [24]
	File distribution among peers	Zipf's law
N_K	Number of KAD peers	5,000
N_{JE}	Number of JXTA edge peers	5,000
N_{JR}	Number of JXTA rendezvous peers	25 [25]
T_n	Number of keywords in TK	200

5.2 Experimental results

We wrote a simulation program in Java SE 6 to simulate and evaluate the performance of JKS, JXTA and KAD. Table 5 gives definitions for measured metrics.

Table 5. Definitions for measured metrics.

Measured metric	Definition
Publish cost	Average number of visited peers per file publishing
Discovery cost	Average number of visited peers per query
Bandwidth cost	Publish cost*10 + discovery cost
Storage cost	Total number of published references
Number of exact matches	Average number of exact matches per query (no duplicate)

Number of exact matches (no duplicate)

In the first experiment, it is to show how the CKP algorithm affects the number of exact matches. All queries were sent from one peer. After receiving responses, each result is counted as one while duplicate results were removed. Fig. 10 shows simulation results using two kinds of data (English and mixed: Chinese + English). JXTA is zero in both data because it needs a query that exactly matches to the file name.

For the number of exact matches, the proposed JKS is 1% less than KAD when data is English and 81% more than KAD when data is mixed. It is because JKS publishes more keywords according to most popular query keywords (TK). Besides, if a file name contains a string with a mix of Chinese, English or number, the string is partitioned and more keywords are published. The growth of published references increases with the number of exact matches.

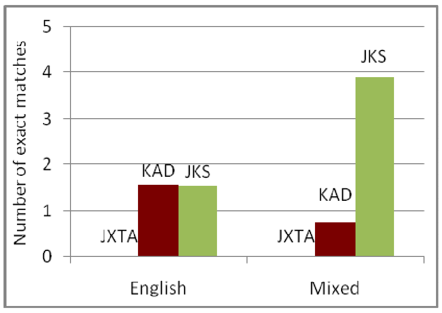

Fig. 10. Number of exact matches (no duplicate)

Bandwidth cost

In the second experiment, it aims to show the network traffic produced by the keyword search. The bandwidth cost consists of publish cost and discovery cost. Since the number of publish messages is ten times bigger than the number of discovery messages [26], [27], the bandwidth cost is computed as follows:

*Bandwidth cost = Publish cost *10 + Discovery cost* (1)

Fig. 11 shows the publish cost using two kinds of data (English and mixed). JXTA is 2 in both data because it publishes one index entry for each file and each index entry is transmitted twice. One is from an edge peer to a connected rendezvous peer; the other is transmitted to the responsible peer.

For the publish cost, JKS is 474% less than KAD with English data and is 163% less than KAD with mixed data. There are two reasons for this. One is because JKS uses a 2-tier hierarchical architecture. The rendezvous peer publishes the index entries from many connected edge peers. When there are duplicate index entries, only one is published. The other reason is that attributes and file information are attached to the keyword key published in KAD. So when there is a same keyword appeared in two file names, the keyword is published twice in KAD but only once in JKS. However, when mixed data is used, the performance improvement margin is smaller. The reason is that the CKP can retrieve more keywords from the file name so that more index entries can be published.

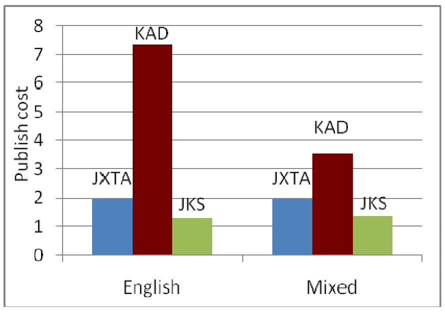

Fig. 11. Publish cost.

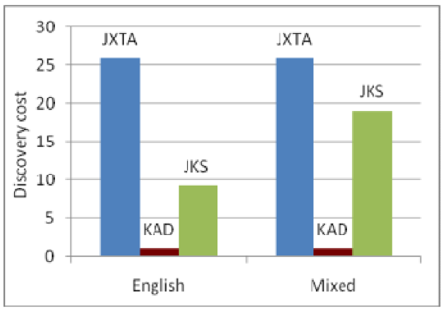

Fig. 12. Discovery cost.

Fig. 12 shows the discovery cost using two kinds of data (English and mixed). JXTA is around 26 in both data because when a query finds nothing in the responsible rendezvous peer it goes walking. In the walking, it visits other rendezvous peers.

JKS has 89% and 95% more discovery cost than KAD when the data are English and mixed, respectively. The reason is as follows. KAD is a 2-level search scheme that searches the source key in the first level. The abundant information published with the keyword key gives the query sufficient information to determine whether it matches the query. It avoids routing a query to too many peers to find out whether there are matched files.

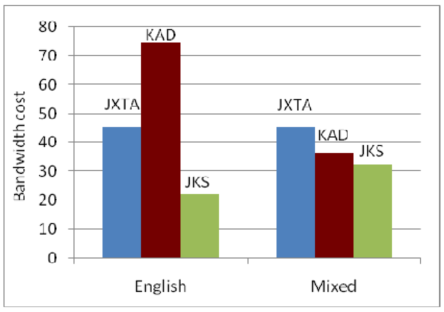

Fig. 13. Bandwidth cost.

Fig. 13 shows bandwidth cost computed according to equation (1). The bandwidth cost of JKS is 237% less than that of KAD with English data, and is 12% less than that of KAD with mixed data. The result shows JKS is better than KAD in both kinds of data.

Storage cost

In this experiment, the aim is to show the storage overhead. The storage cost is calculated after all files are published. In JXTA, all index entries stored in the index entry database are counted. In KAD, all stored references are counted. Fig. 14 shows storage cost using two kinds of data (English and mixed). The result of JXTA is near twice in the number of files in both data. It is because each entry is stored in a connected rendezvous peer and a responsible peer.

JKS has 70% more storage cost than KAD in both kinds of data. There are two reasons. One is the index entry is stored in a connected rendezvous peer and a responsible peer. The other reason is due to that KAD is a 2-level publish scheme. Assume that there is a file and the number of keywords in the file name is 3. KAD stores four references for the file (three keyword keys and one source key). If there is another identical file published, this file will produce only one stored reference (source key). The keyword keys are stored only once for all identical files. This feature saves large storage cost when there are many identical files. Nevertheless, the memory price is very cheap. Today, one 2G memory only costs US$33 [28].

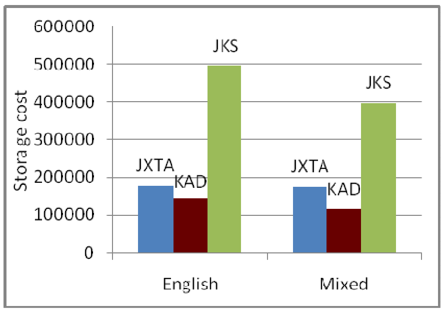

Fig. 14. Storage cost.

6 Conclusions

In this paper, we have presented the design and evaluation of keyword search with Chinese Keyword Partition (CKP) upon JXTA. Experimental results have shown that CKP helps JKS to achieve 81% more number of exact matches compared to KAD with mixed queries. In terms of bandwidth cost, JKS is 237% and 12 % less than KAD with English quires and mixed queries, respectively. The overhead is that JKS produces 70% more storage cost than KAD. Nevertheless, the storage cost is very cheap nowadays. The keyword search function can be used by many P2P applications, like video on demand (VOD) streaming, file sharing, etc. In addition, our CKP design is simple and can be extended to other languages.

References

[1] N. Nakamura, S. Takahama, L. Barolli, J. Ma, and K. Sugita, "A Multiplatform P2P System: its implementation and applications," in *Proc. of the 19th International Conference on Advanced Information Networking and Applications*, vol. 1, pp. 171-176, 2005.

[2] A. Akram and R. Allan, "Comparison of JXTA and WSRF," in *Proc. of the 7th IEEE International Symposium on Cluster Computing and the Grid*, pp. 761-766, May 2007.

[3] S. Venot and Y. Lu, "On-demand mobile peer-to-peer streaming over the JXTA Overlay," in *Proc. of the International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, pp. 131-136, November 2007.

[4] “JXTA CMS” [Online]. Available: https://jxse-cms.dev.java.net/

[5] “JXTA community projects,” [Online]. Available: https://jxta.dev.java.net.

[6] “JXTA company spotlight,” [Online]. Available: https://jxta.dev.java.net/companyarchive.htm.

[7] “SourceForge,” [Online]. Available: http://sourceforge.net/search/?type_of_search=soft&type_of_search=soft&words=jxta.

[8] T. Kim, H. Lee, and H. Cheon, "Implementation of a service oriented architecture based on JXTA for new business models (ICCAS 2007)," in *Proc. of the International Conference on Control, Automation and Systems* , pp. 2402-2406, October 2007.

[9] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *Communications Surveys & Tutorials*, *IEEE*, vol. 7, pp. 72-93, 2005.

[10] Y.-J Joung, L.-W. Yang, and C.-T. Fang, "Keyword search in DHT-based peer-to-peer networks," *IEEE Journal on Selected Areas in Communications,* vol. 25, pp. 46-61, January 2007.

[11] L. Liu, K. D. Ryu, and K.-W. Lee, "Keyword fusion to support efficient keyword-based search in peer-to-peer file sharing," *IEEE International Symposium on Cluster Computing and the Grid*, pp. 269-276, April 2004.

[12] “JXTA Java Standard Edition v2.5: Programmers Guide,” [Online]. Available: https://jxta-guide.dev.java.net.

[13] N. Théodoloz, "DHT-based Routing and Discovery in JXTA," Master’s Thesis, School of Computer and Communication Sciences, February 2004.

[14] M. Abdelaziz, B. Traversat, E. Pouyoul, "Project JXTA: A Loosely-Consistent DHT Rendezvous Walker," Mar. 2003. [Online]. Available: http://www.jxta.org/docs/jxta-dht.pdf.

[15] “JXSE CMS,” [Online]. Available: https://jxse-cms.dev.java.net.

[16] X. Xiang, Y. Shi, and L. Guo, "Rich metadata searches using the JXTA content manager service," in *Proc. of the 18th International Conference on Advanced Information Networking and Applications*, vol. 1, pp. 624-629, 2004.

[17] R. Brunner, “A performance evaluation of the Kad-protocol,” Master’s Thesis, University of Mannheim and Institut Eurecom, 2006.

[18] M. Steiner, T. En-Najjary, and E. W. Biersack, "A global view of KAD," in *Proc. of the 7th ACM SIGCOMM Conference on Internet Measurement*, pp.117-122, 2007.

[19] “eMule,” [Online]. Available: http://www.emule-project.net/home/perl/general.cgi?l=1.

[20] D. Carra and E. W. Biersack, "Building a reliable P2P system out of unreliable P2P clients: the case of KAD," in *Proc. of the ACM CoNEXT Conference*, No. 28, 2007.

[21] “AOL query log,” [Online]. Available: http://www.gregsadetsky.com/aol-data.

[22] “Yahoo,” [Online]. Available: http://tw.buzz.yahoo.com/live_kw.php.xml.

[23] “Google Soap Search API,” [Online]. Available: http://code.google.com/apis/soapsearch.

[24] D. Stutzbach, S. Y. Zhao, and R. Rejaie, "Characterizing files in the modern Gnutella network," *Multimedia Systems*, vol. 13, pp. 35-50, Sep. 2007.

[25] X. Jin, W.-P. K. Yiu, and S.-H. G. Chan, "Supporting multiple-keyword search in a hybrid structured peer-to-peer network," in *Proc. of the IEEE International Conference on Communications*, pp. 42-47, June 2006.

[26] M. Steiner, W. Effelsberg, T. En-Najjary, and E. W. Biersack. “Load reduction in the KAD peer-to-peer system,” in Proc. of *5th International Workshop on Databases, Information Systems and Peer-to-Peer Computing*, 2007.

[27] M. Steiner, T. En-Najjary, and E. W. Biersack, "Exploiting KAD: possible uses and misuses," *ACM SIGCOMM Computer Communnication Review*, vol. 37, pp. 65-70, 2007.

[28] “NOVA,” [Online]. Available: http://www.nova.com.tw.