ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/4318550

Application Based Distance Measurement for Context Retrieval in Ubiquitous Computing

Conference Paper · September 2007

DOI: 10.1109/MOBIQ.2007.4450999 · Source: IEEE Xplore

CITATIONS
4

READS
121

4 authors, including:

Shaxun Chen
Facebook
21 PUBLICATIONS **524** CITATIONS
SEE PROFILE

Tao Gu
RMIT University
154 PUBLICATIONS **6,028** CITATIONS
SEE PROFILE

Xianping Tao
Nanjing University
101 PUBLICATIONS **1,843** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project Human Activity Recognition View project

Project Sensor-based Motion Recognition View project

All content following this page was uploaded by Xianping Tao on 04 July 2014.

The user has requested enhancement of the downloaded file.

Application Based Distance Measurement for Context Retrieval in Ubiquitous Computing

Shaxun Chen[1], Tao Gu[2], Xianping Tao[1], Jian Lu[1]
[1]State Key Laboratory for Novel Software Technology, Nanjing University, P.R.China
[2]Institute for Infocomm Research, Singapore
csx@ics.nju.edu.cn; tgu@i2r.a-star.edu.sg; txp@ics.nju.edu.cn; lj@nju.edu.cn

Abstract—**Building large-scale smart environments is one of the long-term goals of ubiquitous computing. The widespread of context information in such environments necessitates an effective context retrieval mechanism. This paper proposes a novel context retrieval method based on applications' query patterns. We propose high dimensional vector to model contexts from applications' perspective, and apply the normalized inner product of high dimensional vectors to measure context distance. Contexts with similar query patterns are clustered into the same group. To improve the performance of context retrieval, we build distributed indices on each node to speed up a local search, and create shortcuts based on clustering results to facilitate query routing. We show how our proposed methods can be applied to existing context retrieval mechanisms. Our experimental results show that our method can significantly reduce retrieval cost.**

Keywords-context distance; clustering; ubiquitous computing; context retrieval

I. INTRODUCTION

Ubiquitous computing creates an appealing view of a new computing paradigm, which decreases a user's attention to computational devices and helps the user concentrate on the task through implicit collection of various contexts. According to Mark Weiser's vision, ubiquitous computing provides users with adequate services anytime and anywhere [1]. This vision implies that a smart environment may expand to a very large area across multiple application domains. With the advancement of sensing techniques and embedded devices, large-scale smart environments become possible, such as emerging smart hospitals [2] and outdoor guiding systems [3].

In a large-scale smart environment, a vast amount of context-aware applications may be interested in searching and utilizing different types of context information which is widespread across multiple domains. As a result, an efficient context retrieval mechanism becomes a key issue in ubiquitous computing. One approach is to store or index all the contexts in a centralized node and resolve the search requests. Although it will provide fast responds to context queries, the deficiencies are obvious. Centralized context management suffers from processing bottleneck, limited scalability and single point of failure. For example, in a context-aware car navigation system, it is not realistic for a single server to manage traffic loads of all the roads, and information about all the spots in a large city. It is more natural and reasonable to store and update context information in a distributed fashion in such large-scale smart environment.

This paper aims to propose efficient methods to improve context retrieval and reduce retrieval cost in a large-scale smart environment. Our proposed method is based on applications' query patterns, and contexts are clustered according to context distance. The query pattern of a context refers to the set of applications that query it and the respective query frequencies. To the best of our knowledge, this is the first attempt to study the problem of context retrieval from applications' perspective.

We define context distance as logical distance measured from applications' perspective. As we know, context information may be expected to serve multiple applications simultaneously. For example, an outdoor smart space equipped with sensors may assist in many applications such as ecosystem monitoring, weather reporting, precision agriculture, etc. If applications tend to access two contexts at the same time, i.e., these two contexts have high possibility to be queried simultaneously, then we say that they have a small context distance between each other. In the above example, the weather reporting application may be interested in humidity, temperature and light contexts, and the application for precision agriculture may need temperature, light and presence of chemicals. Hence, we know that temperature and light contexts are "close-by" to each other in context distance. However, the distance between two contexts may not be that intuitionistic when the number of applications increases and query frequencies are taken into account.

In this paper, we model contexts using high dimensional vector. Different from traditional Euclidean distance, we propose the sine of high dimensional vectors for measuring context distance. Based on this distance measurement, contexts which are close-by in context distance are grouped into the same cluster using a *k*-means based algorithm. Then we build distributed indices on each node to speed up a local search, and create shortcuts to facilitate inter-cluster routing. Our proposed methods can be applied to existing context retrieval mechanisms to enhance their performance. Our simulation results show that our method can significantly reduce query costs.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 presents our approach for context clustering. Section 4 describes how to improve query

Funded by 973 Program of China (2002CB312002) and 863 Program of China (2006AA01Z159), NSFC (60403014, 60603034), JSFC (BK2006712).

performance using clustering results. We evaluate our method in Section 5, and finally conclude the paper in Section 6.

II. Related Work

Most of the existing work is built upon centralized context information repositories, such as CoBrA [4] and Jena [5]. These approaches are easy to design and provide fast responds in small or moderate applications, but they have limitations in a large-scale smart environment.

Some researchers proposed distributed context retrieval approaches leveraging existing Peer-to-Peer (P2P) architectures. Edutella [6] is a combination of unstructured P2P network and the Semantic Web. It is schema-based and queries are routed through the peers organized in hypercubes. Gu et al. [7] proposed a two-tired P2P network for context lookup. The upper tier is Chord-like [8] while the lower tier is unstructured. These work made efforts to improve query performance by grouping or indexing contexts in distributed environments. However, these clusters or indices are based on context semantics, which requires a set of pre-defined ontologies. The lack of standard in context ontology becomes a bottleneck. Our approach intends to cluster contexts based on query patterns, which differs from existing work.

Biswas et al. [9] studied correlations between attributes by calculating applications' query probabilities in the context of wireless sensor networks. Attributes are allocated in GHT (Geographic Hash Table) based networks conforming to the following principle: frequently used attributes are put more closely to the center of the net and tightly correlated attributes are neighbored each other in the net. However, the algorithm described in this paper is heuristic. Since they use a greedy policy, the results obtained is not globally optimized. Further more, the research in wireless sensor networks is different from context retrieval in a way that the former focuses more on energy saving instead of responding time.

Most of the traditional clustering algorithms adopt Minkowski distance (i.e., generalized Euclidean distance) as the mathematic model. But this model is sensitive to the vector elements with large values. Although weights can be used to balance the effect, the choosing of weight values is somewhat arbitrary [10]. We propose an inner product based distance measurement, which is able to describe the correlation of contexts more appropriately. As compared to [9], this measurement method considers the globe conditions rather than a heuristic and greedy manner.

III. Context Clustering Using Query Patterns

In this section, we introduce a mathematic model to measure context distance, and then describe how to cluster contexts which are close-by in distance into a cluster based on the measurement.

A. Formalization of Contexts

Assuming that there are m types of contexts $\{C_1, C_2, \ldots, C_m\}$ and a set of q context-aware applications $\{A_1, A_2, \ldots, A_q\}$ in a large-scale smart environment. C_i $(1 \leqslant i \leqslant m)$ is defined as a q-dimensional vector:

$$Ci = (p_1, p_2, \ldots, p_q) \tag{1}$$

where p_j $(1 \leqslant j \leqslant q)$ is the number of times A_j accesses C_i per unit time. Obviously, $p_j \geqslant 0$ for all j. This vector provides a formal definition of the query pattern of Ci.

B. Context Distance Measurement: Sine

Given two types of contexts S and T (e.g., S is light context and T is temperature), let $S = (s_1, s_2, \ldots, s_q)$ and $T = (t_1, t_2, \ldots, t_q)$, firstly we calculate the normalized inner product of S and T.

$$\cos(S,T) = \frac{\sum_{j=1}^{q} s_j t_j}{\sqrt{\sum_{j=1}^{q} s_j^2}\sqrt{\sum_{j=1}^{q} t_j^2}} \tag{2}$$

Defined in this way, $cos(S, T)$ will vary between 0 (when two vectors are orthogonal) and 1 (when two vectors are identical or proportional). More generally, $cos(S, T)$ will increase as S and T share more non-zero components s_j and t_j. Shared components mean that S and T have associations with the same applications. Actually, Formula (2) calculates cosine of the included angle of two vectors and is an accuracy description of correlations between two types of contexts in the sense of applications' queries.

To explain it more clearly, we will show the special case of Formula (2) when both S and T are 2D (or 3D) vectors, that is, there are only two or three context-aware applications in this smart environment (q = 2 or 3). First, we let $q = 2$, $S = (s_1, s_2)$ and $T = (t_1, t_2)$, angles α, β, γ are shown in Figure 1. We have

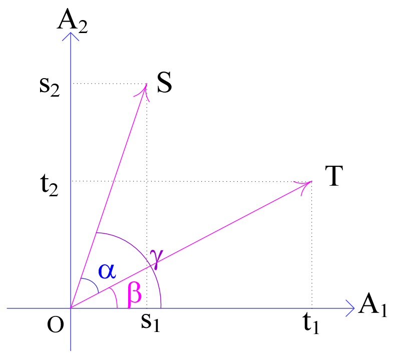

Figure 1. Cosine of 2D vectors

$$\begin{aligned}
\cos(S,T) &= \cos\alpha = \cos(\gamma - \beta) \\
&= \cos\gamma\cos\beta + \sin\gamma\sin\beta \\
&= \frac{s_1}{\sqrt{s_1^2 + s_2^2}} * \frac{t_1}{\sqrt{t_1^2 + t_2^2}} + \frac{s_2}{\sqrt{s_1^2 + s_2^2}} * \frac{t_2}{\sqrt{t_1^2 + t_2^2}} \\
&= \frac{s_1 t_1 + s_2 t_2}{\sqrt{s_1^2 + s_2^2}\sqrt{t_1^2 + t_2^2}}
\end{aligned}$$

We know that the above result conforms to the Formula (2) when q = 2. The more closer the proportions of frequencies where each applications query two contexts are, the more smaller the included angle of two vectors is, and more larger the cosine of this angle will be, given an acute angle. Intuitively, cosine gives similarity of two contexts in the aspect of its correlations with applications. Similarly, cosine of included angle of two 3D vectors is shown as follows.

$$\cos((s_1,s_2,s_3),(t_1,t_2,t_3)) = \frac{s_1t_1+s_2t_2+s_3t_3}{\sqrt{s_1^2+s_2^2+s_3^2}\sqrt{t_1^2+t_2^2+t_3^2}}$$

If we generalize the cosine expression to high dimensional vectors, we will have Formula (2). Next, we give the definition of the context distance between S and T.

$$D(S,T) = \sin(S,T) = \sqrt{1-\cos^2(S,T)} \qquad (3)$$

where *cos(S, T)* is defined in Formula (2). Formula (3) calculates sine of two vectors and we use this value to define context distance.

A well-defined distance measurement should conform to following four conditions.

1) Nonnegative. distance(S, T)⩾0;

2) Distance from an object to itself is 0. distance(S, S) = 0;

3) Symmetric. distance(S, T) = distance(T, S);

4) Triangular inequality. distance(S, T) ⩽ distance(S, R)+distance(R, T).

Now we prove $D(S$, T) is a well-defined distance measurement.

Prove. *1)* Trivial; *2)* $sin(S, S) = sin0 = 0$; *3)* Trivial; *4)* Let the included angle of S and T is γ, that of S and R is α, and that of R and T is β, then $\gamma = \alpha+\beta$ or $\gamma = |\alpha-\beta|$. When $\gamma = |\alpha-\beta|$, the proof is straightforward. When $\gamma = \alpha+\beta$, $sin\gamma = sin(\alpha+\beta) = sin\alpha cos\beta + cos\alpha sin\beta$. Since all vector components are nonnegative, so $0 \leqslant cos\alpha \leqslant 1$, $0 \leqslant cos\beta \leqslant 1$. Then $sin\gamma \leqslant sin\alpha + sin\beta$. □

While cosine depicts correlations between two contexts from applications' queries, sine is defined as a distance measurement of them in the same sense. $D(S, T)$ will decrease when two types of contexts S and T share more common points in their query patterns. In the example mentioned in Section 1, we may guess that D(*light*, *temperature*) is smaller than D(*light, humidity)* as the former pair appears in two applications. Of course, it depends on the queries of other applications in the same environment and the query frequencies as well. We will compare the measurement of $D(S, T)$ with that of traditional Euclidean distance in Section 5.

C. Context Clustering

Based on our distance measurement represented in the Formula (3), we can cluster "close-by" contexts into a group. Generally speaking, contexts that belong to the same group exhibit more similarity in their query patterns.

We apply $D(S, T)$ to obtain context distance, where S and T are q-dimensional vectors as defined before. Our clustering algorithm is based on k-means, and the steps are presented as follows.

1) Randomly select k contexts as the centers of the clusters;

2) For each remaining context, assign it to the cluster whose center is the nearest to the context. The distance between a context and the center is calculated by D function;

3) Compute the means of the new clusters and assign them with the new centers;

4) If there is no change, exit. Otherwise go to step 2).

This algorithm is convergent. The value of k (number of clusters) should be decided before clustering. Here follows a segment of the algorithm.

```
boolean exchange = true;
while (exchange){
    exchange = false;
    for (int i = 0; i < numOfObjects; i++){
        tempClu = object[i].getItsCluster();
        tempDis = D(mean[tempClu], object[i]);
        for (int j = 0; j < numOfCluster; j++){
            dis = D(mean[j], object[i])
            if (tempClu != j && tempDis > dis){
                tempDis = dis;
                object[i].setItsCluster(j);
                exchange = true;
            }
        }
    }
    If (exchange) calculateMeans();
}
```

Figure 2. A segment of the clustering algorithm

To illustrate the clustering algorithm, we take an example of 2D space, i.e., there are only two context-aware applications in a smart environment, say, A_1 and A_2. In practice, the dimension of a smart space could be very high. We assume there are five types of contexts, i.e., C_1, C_2, C_3, C_4, and C_5. The frequency that A_1 queries C_1 is 30 and that of A_2 is 0. Other frequencies are shown in Figure 3.

Let k = 2, after executing the clustering algorithm, five contexts are clustered into two groups, i.e., *Custer_1* = {C_1, C_2, C_3}, *Cluster_2* = {C_4, C_5}. The result is expected since the included angles of intra-cluster vectors are smaller than those of inter-cluster vectors. *Cluster_1* can be regarded as the set of contexts which application A_1 queries much more often than application A_2 dose. Similarly, *Cluster_2* is the set of contexts which A_2 queries more often than A_1 does. If we substitute Euclidean distance for our sine distance measurement and execute the algorithm again (still let k = 2), five contexts will be re-clustered. C_1 and C_5 are in the same group, while C_2, C_3, C_4 in the other (marked with dashed cycles in Figure 3), because the physical distance between end-points of those intra-group vectors are smaller.

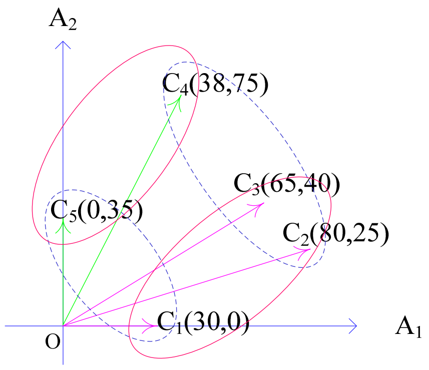

Figure 3. Clustering by two measurements

From the above comparison, we can see that clusters produced by our sine distance measurement are more accurate to the extent of query patterns and have more readable meanings. In Figure 3, it seems that clusters grouped by Euclidean distance reflect the difference of total query frequency. Actually, it may not the case for more complex examples. We will compare these two distance measurements in our experiments.

The clustering results can be used to improve context retrieval performance. In addition, it can be applied for measuring the semantics between two contexts. For example, if we execute our algorithm based on a large amount of queries which are collected from real-world context-aware applications, we can study semantic relations among contexts in the same cluster without any ontology interference. Since two words appeared frequently in the same web pages are proved to have some degree of semantic relations [11], it is conceivable that two kinds of contexts often queried together are also correlated in semantics. However, in this paper, we only focus on performance improvement of context retrieval as an illustration example of context clustering.

IV. Apply Clustering Result to Context Retrieval

In this section, we introduce how to improve the performance of context retrieval by using our clustering results. We first describe the network structure, and then explain the indices and the shortcuts constructing according to the context clusters.

A. *Network Structure*

We choose the unstructured P2P network as the underlying network substrate. In this network, a peer can be context providers which store contexts and handle the query requests, context consumers (i.e., context-aware applications), or both. There is no centralized controller or any form of DHT (Distributed Hash Table) applied.

Centralized repositories and query management may not be suitable for large-scale smart environments because of single point of failure, processing bottleneck and poor scalability, as we discussed in Section 1. DHT-based structured P2P networks like Chord [8], CAN [12], or Pastry [13] may account performance lost when peers join or leave the network frequently. In the ubiquitous computing environment, many nodes are wireless and handheld devices. As these nodes may be moving from place to place, the network topology changes dramatically and maintenance costs of DHT become very high. Unstructured P2P network is much easier to build and maintain, which is suitable for context-aware environments.

We assume that contexts are stored near where they are generated. Since the generating rates of some contexts are very high, such as noise level or the location of a person, local repository will have least costs.

B. *Context Indexing*

To speed up a local context search within a peer, we propose an indexing mechanism to summarize local contexts, and therefore reduce the local retrieval costs.

We use RDF [15] to present contexts, which are composed of <*subject*, *property*, *object*> triples, such as <Tom, locateIn, Room11> and <Room06, temperature, 21>. For simplicity, namespaces are omitted. We add the generating time for each context. The indices are built on the local node where contexts are stored, and the structure of indices is shown in Table 1.

To build indices, we divide local contexts into two *categories*. One is those whose *objects* are consecutive numerical values, such as the temperature of Room506; the other is those whose *objects* are enumerated values, such as the place Tom locates in.

TABLE I. Conetxt Indices in Peer_1

TypeID	***Subject***	***Property***	***Object***		***TimeFrom***	***TimeTo***
0x10af	Room506	tempera	12	23	020145723	021378107
0x2045	Jan	locateIn			020137589	024589341
0x20c7	Tom	locateIn			020137589	023337582
0x2123		locateIn	Room506		020137589	023337582
0x219a		locateIn	Room507		020334325	024589341

Shows three kinds of indices

For the former *category*, all the contexts (in the same peer) with identical *subject* and *property* are summarized into one record in the index table. For instance, assuming the set of contexts (stored in peer_1) with consecutive numerical-value *objects* is the first three lines in Table 2, the summarized index of them is just as the first line of table 1 shows. In this case, *Object* field describes the min and max value of objects of the contexts summarized by this record. *TimeFrom* and *TimeTo* fields give the generating time of the first and the last contexts of this type.

For the latter *category*, contexts are summarized according to both <*subject*, *property*> and <*property*, *objects*> pairs respectively. For instance, assuming the set of contexts (stored in peer_1) with enumerated value *objects* is the last six lines of table 2. First, we build indices of these contexts according to <*subject*, *property*> pairs. The result is shown as the second and third lines of Table 1. The *Object* field is vacant. *TimeFrom* and *TimeTo* fields refer to the same meaning as the former *category*. Then we build indices according to <*property*, *objects*>. The result is the fourth and fifth lines of Table 1. Similarly, *Subject* field is vacant.

In the above example, there are nine contexts in peer_1 (see Table 2), and only five records in the index (see Table 1). In real-life applications, the number of index records may be much fewer than the number of contexts. When a new context about temperature in Room506 generates, what we need to do is merely changing the *TimeTo* field (and possibly the *Object* field) of the fist line in Table 1. We do not need to add a line to the index. When a query arrives, the index table will be checked first. Only when a record in the indices matches the query will local contexts be scanned. By this way the search cost can be greatly reduced using our index method.

TABLE II. CONTEXTS IN PEER_1

Subject	*Property*	*Object*	*Timestamp*
Room506	tempera	12	020145723
Room506	tempera	16	020145856
Room506	tempera	23	021378107
Tom	locateIn	Room506	020137589
Jan	locateIn	Room506	020137589
Tom	locateIn	Room506	023337582
Jan	locateIn	Room507	024589341
Tom	locateIn	Room507	020334325
Jan	locateIn	Room507	022335628

Actually, contexts are stored in the order of their timestamps. This table is just for illustration.

In our approach, contexts whose *objects* are consecutive numerical values are only indexed by <*subject*, *property*> combination, while others whose *objects* are enumerated values are indexed by both <*subjects*, *property*> and <*property*, *object*> combinations. However, as we know, $C_3^2 = 3$. i.e., for a triple, there are three cases when looking up one item as other two items are known. Actually, RDFpeers [16] indexes every context by all three combinations: <*subject*, *property*>, <*property*, *object*> and <*subject*, *object*>. Each context has three copies in the whole network. Nevertheless, we do not build all these indices because some of them are unnecessary as explained as follows.

For a query like "which students are located in Room507" (requires contexts with enumerated-value *objects*), we build the index based on <*property*, *object*> pair for the related contexts. However, a query such as "which rooms have the temperature of 21.5" (requires contexts with consecutive numerical value *objects*) is seldom used in applications, because it wiser to query consecutive numerical item by a range instead of a particular value. Further more, indexing consecutive numerical values may greatly increase the storage cost of indices. Hence, we handle two *categories* of contexts differently. In addition, we do not index <*subject*, *object*> pair for both *categories*. In reality, applications never query "what is the relationship between Room507 and 21.5". Hence, <*subject*, *object*> pair will not be used. One may argue that the query "what is the relationship between Tom and Room507" can exist. However, since the answer can only be "locate in" or "not locate in", users can query "who are in the Room507" or simply "where Tom locates".

In summary, the design of our indexing method aims to improve the efficiency of local contexts retrieval and minimize the index volume.

C. *Shortcuts Based on Context Clusters*

In an unstructured P2P network such as Gnutella [14], queries usually are flooded to all the nodes in the network, which results in a large amount of unnecessary messages. In this paper, we propose to create shortcuts to facilitate query routing .

First, we define what is a *type* of context. We define a record in the index table (see Table 1) as a *type* of context. For example, <*, temperature、*> or <*, locateIn、*> is not a *type* of context (* is wildcard), but <Room507, temperature、*> or <*, locateIn、Room506> or <Tom, locateIn、*> is a valid *type* of context. Each *type* of context is assigned with a globe unique *typeID*, as shown in the first column of Table 1. Contexts in different peers with the same records in *subject*, *property*, *object* fields in their index tables are treated as the same *type* (the min and max value of *object* can be neglect). For instance, a record <Tom, locateIn, *> that exists in both peer_1's index table and peer_2's index table will be treated as the same *type* of contexts (Note: the namespaces of Tom should be the same).

After gathering enough query information from various applications, we model each type of context with a *q*-dimensional vector (assuming there are *q* context-aware applications in such smart environment), and use them as the input of our clustering algorithm presented in Section 3.

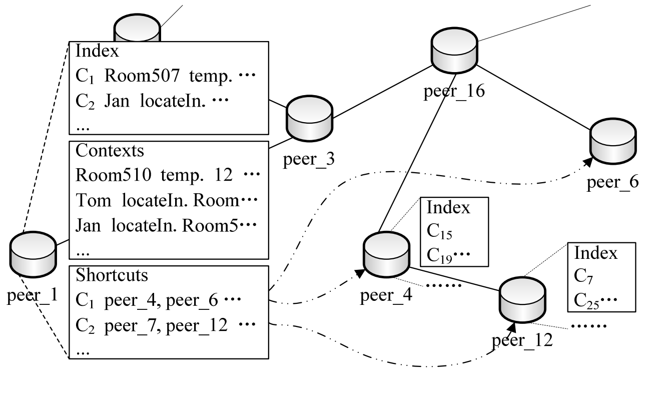

Figure 4. Overview of network and data structure of peer_1

The results of clustering will broadcast to every node in the network. Peers with contexts that belong to the same cluster will cache each other. For example, peer_1 has three *types* of contexts C_1, C_2, and C_3. The clusters these contexts are respectively located in are {C_1, C_{15}, C_{23}}, {C_2, C_3, C_7, C_{16}, C_{32}} (C_2 and C_3 are in the same cluster). Then peer_1 should cache the ID and IP address of all the peers that store C_{15}, C_{23}, C_7, C_{16} and C_{32}. In this way, most queries can find their way to destination by cached shortcuts instead of using blind flooding. Figure 4 shows the overview of the network and the data structure of peer_1. Solid lines stand for physical connection while dashed lines with arrows indicate the shortcuts.

When parameter *k* (i.e., the number of clusters) is smaller, a cluster may contain more *types* of contexts and a node may cache more peers, resulting in greater performance improvement of context retrieval. On the other hand, we should notice a larger *k* may result in higher cost of memory and more risk of cache invalidation in case of topology change.

V. EVALUATION

Now we move on to evaluate our proposed methods through simulation studies. First, we evaluate the effects of shortcuts based on our context clusters. We then present the performance improvement of our context indices. Next, we compare the sine distance measurement with Euclidean distance. Finally, we study the scalability of our clustering method.

We use the Autonomous System model to generate network topologies as previous studies have shown that the P2P overlay topology follows both small world and power law properties [17]. During our experiments, peers join and leave the network at the same rate to simulate dynamic characters of the P2P overlay. We assume that there are 30 applications and 128 *types* of contexts in an unstructured P2P network for the first three experiments. Query patterns are randomly assigned to each *type* of context.

A. Effect of Cluster-based Shortcuts

We compare our proposed methods presented in Section 3 and 4 with Gnutella in this experiment. In our methods, contexts are clustered according to our algorithm with sine distance measurement, and shortcuts between peers are cached based on the results of clustering. In a Gnutella-like network, the results of a query are cached along the query path and no shortcuts are built. Both methods have the same cache size. The average number of hops traversed by a query request to the destination is used as a metric.

As we know, query success rate (*QSR*) in an unstructured P2P network can hardly reach 100%. We use *QSR*s = 85% for both methods. We set parameter *k* (i.e., number of clusters) to 20, 10, and 5 respectively for our method. Figure 5 plots search path length vs. number of nodes for both two methods.

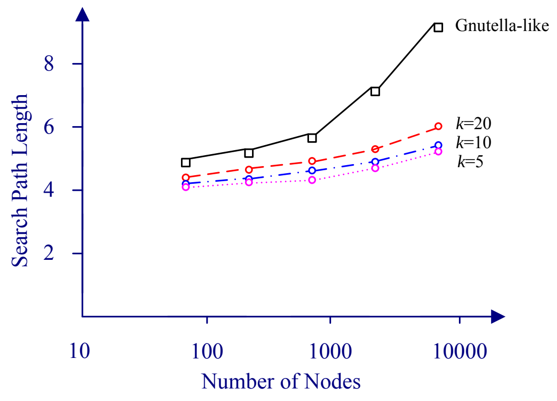

Figure 5. Average hops when *QSR* is 85%

From Figure 5, we can see that the average search hops of our method are much less than a Gnutella-like overlay. Moreover, the rising rate of our method is much less than that of Gnutella. It demonstrates the use of shortcuts can greatly improve system scalability. In addition, a smaller number of clusters results in better performance. This result probably can be explained as follows: When contexts within a cluster increase and a node tends to cache more peers. However, when *k* gets smaller and smaller, the performance improvement is limited, but the memory costs rise quickly. Therefore, a better tradeoff needs to be studied further.

B. Effect of Context Indices

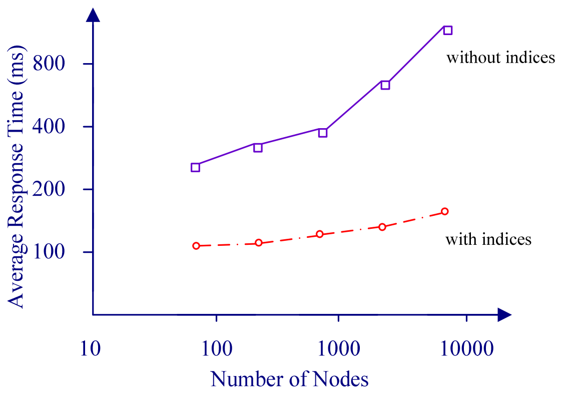

Figure 6. Effect of context indices

In this experiment, we evaluate the impact of our context indices presented in Section 4.B. As mentioned before, some contexts generate rapidly and we reserve the historical data, so data volume in a peer may be very large. Hence, local processing time takes a considerable part in response time. The comparison is performed between two cases. In the first case, we use our approach, including indices and shortcuts. In the other case, we shield the indices (shortcuts are reserved). We let *k*=20 and *QSR*=85% for both cases. As the routing of two cases are the same, the difference of their average response time reflects the impact of the indices. As shown in Figure 6, the case of using indices reduces the total response time significantly.

C. Performance of Sine Distance Measurement

In this experiment, we compare the sine distance measurement with Euclidean distance in the context of our clustering algorithm. We let k = 20, and use Formula (3) to calculate *D* function for the first case, and leverage Euclidean distance for the second case. Other settings are identical and *QSR* = 85%. The results of two cases are applied for building shortcuts respectively. We use search path length as the metric. The comparison result is shown in Figure 7.

The result reveals that the sine distance measurement performs better than the Euclidean distance, especially when the number of peers is large. In addition, context clusters based on the sine distance have clearer meanings, to the query pattern extent, compared to the Euclidean distance as we have discussed in Section 3.

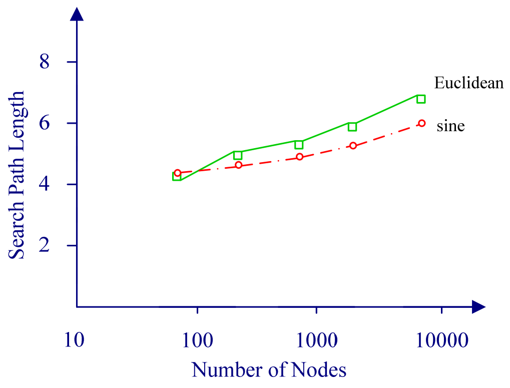

Figure 7. Sine distance vs. Euclidean distance

D. *Scalability of Our Context Clustering Method*

Finally, we test to evaluate the scalability of our approach. In this experiment, we measure the cost when the number of applications and *types* of contexts increase.

Figure 8 plots the cost for different numbers of applications and different *types* of contexts. The increase of clustering time costs (i.e., time cost for the program to execute the clustering algorithm) is linear to that of the number of applications and *types* of contexts, and the increase of response time is almost neglectable. Above all, context clustering only need to run once, while the response time is involved in every query and is the key measurement. From the above results, we can see that our sine distance measurement and clustering methods have good scalability and they are capable to deal with high dimensional vectors.

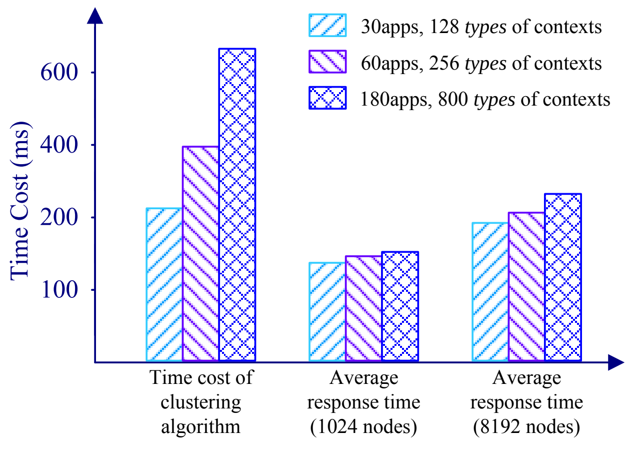

Figure 8. Scalability measurement

VI. Conclusion and Future Work

In this paper, we propose a context clustering method from a novel perspective. Contexts with similar query patterns abstracted from the applications are grouped into the same cluster. We improve the context retrieval performance by constructing logic shortcuts between peers based on clusters, in order to illustrate the sense of context clustering. Moreover, we also propose a new distance measurement--sine for clustering algorithms, which performs better than Euclidean distance in our domain.

In our future work, we will focus on two aspects. First, we will make efforts to collect query information, perform clustering and build shortcuts on the fly (including incremental clustering algorithms), in order to adapt to the dynamical characters (e.g., changes of context requirements and provision) in the real-life applications. Second, as discussed in Section 3, we will try to study the semantic clues provided by context clusters, which can be derived from our clustering algorithm, without any prior semantic knowledge.

References

[1] Weiser M., “The Computer for the 21st Century,” Scientific American, pp.94-100, September 1991.

[2] J Patrik Fuhrer, Dominique Guinard, “Building a Smart Hospital Using RFID Technologies,” presented at the 1st European Conference on eHealth (ECEH06), Fribourg, Switzerland, October 12 - 13, 2006.

[3] Gregory D. Abowd, Christopher G. Atkeson, Jason Hong, Sue Long, Rob Kooper and Mike Pinkerton, “Cyberguide: A Mobile Context-aware Tour Guide,” Wireless Network, Springer, 1997.

[4] Harry Chen, Tim Finin, Anupam Joshi, “Semantic Web in the Context Broker Architecture”, In Proceedings of PerCom 2004, Orlando FL., March 2004.

[5] http://www.hpl.hp.com/semweb/jena2.htm.

[6] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Lser, “Super-peer-based Routing and Clustering Strategies for RDF-based Peer-to-Peer Networks,” in Proceedings of the 12th World Wide Web Conference, May 2003.

[7] T. Gu, H. K. Pung, and D. Zhang, “A Peer-to-Peer Overlay for Context Information Search,” in Proceedings of the 14th IEEE International Conference on Computer Communications and Networks, San Diego, California, October 2005.

[8] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications,” in Proceedings of ACM SIGCOMM, 2001

[9] Ratnabali Biswas, Kaushik Chowdhury, Dharma P. Agrawal, “Attribute Allocation in Large Scale Sensor Networks,” in Proceedings of the 2nd International Workshop on Data Management for Sensor Networks (DMSN '05) August, 2005.

[10] Han JW, Kamber M, Data Mining: Concepts and Techniques. San Francisco, CA: Morgan Kaufmann, 2000.

[11] F. Heylighen, “Mining associative meanings from the web: from word disambiguation to the global brain,” in Proceedings of Trends in Special Language and Language Technology, R. Temmerman (ed.), Standaard Publishers, Brussels, 2001.

[12] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A Scalable Content Addressable Network,” in Proceedings of ACM SIGCOMM, 2001.

[13] A. Rowstron and P. Druschel, “Pastry: Scalable Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems,” in Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing, November 2001.

[14] Gnutella. http://gnutella.wego.com.

[15] http://www.w3.org/RDF.

[16] Min Cai, Martin Frank, “RDFPeers: A Scalable Distributed RDF Repository Based on a Structured Peer-to-Peer Network,” in Proceedings of the 13th International World Wide Web Conference, New York, May 2004.

[17] S. Saroiu, P. Gummadi, and S. Gribble, “A Measurement Study of Peer-to-Peer File Sharing Systems,” in Proceedings of Multimedia Computing and Networking, 2002.

View publication stats