Semantic P2P Overlay for Dynamic Context Lookup

Shubhabrata Sen, Hung Keng Pung,Wai Choong Wong
School of Computing, Department of Electrical and Computer Engineering
National University of Singapore, Singapore
{shubhabrata.sen, dcsphk, elewwcl}@nus.edu.sg

Wenwei Xue
Nokia Research Center
Beijing, China
wayne.xue@nokia.com

***Abstract*— Context-aware applications generally need to retrieve various kinds of dynamic context data from a large number of context sources. A middleware managing context sources must provide an efficient context lookup mechanism to ease application development. In this paper, we categorize the context sources as operating spaces and propose semantic peer-to-peer overlays over these spaces to accelerate dynamic context lookup in a context-aware middleware. Our proposed overlay structure is specially designed to deal with dynamic sensory context such as a person's location and temperature that are frequently changing and difficult to be promptly retrieved using traditional peer-to-peer protocols. Our overlay and indexing method has a low maintenance overhead. Measurement results show that the proposed overlay achieves a good response time and accuracy for context lookup as well as requires low maintenance overhead.**

Keywords-context-awareness; context-aware middleware; operating spaces; semantic peer-to-peer overlays

I. INTRODUCTION

The recent advances in pervasive computing have lead to an increased research focus on the development of sophisticated context-aware applications. Initially restricted to user location, *context* [7] is generally defined as any data that can be used to characterize the situation of an entity involved in the user-application interaction. We call such an entity a *context source*.

Context-aware middleware systems [2][14] have been explored to provide effective support for the development of context-aware applications by providing mechanisms to efficiently manage various data retrieved from context sources. We define an *operating space* as a person, object or place in the physical or virtual world having a software module hosted in a computing device through which the affiliated context sources in the space can communicate with and provide data to external consumers. Example classes of operating spaces include persons, homes, offices and shops.

We define a *context attribute* as a kind of context data provided by an operating space. The context attributes can be static or dynamic. *Dynamic attributes* refer to sensory data that change asynchronously and frequently, such as the *location* of a person and the *temperature* in an office. *Static attributes*, in contrast, refer to data that seldom changes, such as the *name* of a person and the *location* of a shop. Most attributes involved in context-aware applications are dynamic and continuously changing over, as the main focus of these applications is to ensure that they can adapt to the context changes in an unattended fashion.

An important component of a context-aware middleware system is the context lookup mechanism. *Context lookup* is defined as the process of searching and identifying the operating spaces that an application that have the required context data, as well as the actual operation of obtaining the data from each of these spaces. The context lookup process is quite challenging since it requires dealing with dynamic attributes and it is important to ensure that the data being retrieved is up to date. For example, a healthcare application monitoring a patient's heartbeat by data from body sensors triggers an alarm upon any abnormality. The application needs to ensure that it always reads the most recent data because stale data might lead to a severe consequence.

In this paper, we propose to overcome the limitations of traditional database methods for dynamic data management by using a semantic *peer-to-peer (P2P)* overlay [18] as a dynamic context lookup mechanism over operating spaces. The intuition is that by classifying the operating spaces into different semantic domains and having a distributed index for context data enables an efficient context lookup over these spaces. The main reason for us to use a P2P technology for context lookup is to utilize the dynamic leaving and joining facilities in P2P to cope with the dynamic changes in context data. We have implemented and evaluated this overlay structure in a research prototype of context-aware middleware infrastructure named C*oalition* [5] under ongoing development in our project.

The remainder of the paper is organized as follows. We discuss the related work in Section II. We provide an overview of our Coalition system and explain its various components in Section III. Section IV describes our detailed design of the semantic peer-to-peer overlay and the maintenance operations associated with the overlay. We present our current experimental results over the proposed overlay structure in Section V. Section VI concludes the paper with future research directions.

II. RELATED WORK

Traditional database indexes suffer from heavy update cost when data values associated with an index are dynamic and frequently change as in context-aware computing. We need the design of specialized distributed "sparse" index

 ISBN: 978-1-61208-100-7

structures that focus on minimizing index maintenance overhead upon the continuously arriving new data values.

Indexing moving objects: A recent research focus in database community is the data management and indexing techniques for moving objects. This is an enabling technology mainly for location-aware applications. The R-tree style index structures [13][16][19][21] designed for moving objects seek to minimize the index update overhead by minimizing the number of update operations that actually need to be applied to an index. This is realized by representing the motion of a moving object as a function and updating the index when the parameters of the function change or by utilizing physical properties of moving objects like trajectories to make indexing decisions. Trajectory information can be used to predict the motion path of objects.

Indexing dynamic web documents: Inverted index structures are often used to index web documents that provide a mapping between words and their positions in the document. Inverted indexes designed for web documents that change frequently [15][17] seek to minimize the index update overhead by reducing the number of memory operations that occur during an index update. This is achieved by selective rebuilding of the index and by reducing the information to be updated via forward indexing techniques.

The indexing techniques for moving object environments or dynamic web documents are not generic in nature since they are designed for a particular application class and largely take special properties of that application class into account when designing the index. Moreover, they utilize centralized indexing strategies, which could prove to be a very critical bottleneck for a context-aware middleware operating at the Internet scale.

Indexes for P2P systems: The indexes built for P2P systems are generally distributed in structure. Since the P2P systems are intended for information sharing, the indexes are built on the file metadata information like names and sizes [3] which are relatively static. P2P index structures like the routing index [6] attempt to minimize the number of peers to be looked up for a query by associating the routing information with links and do selective forwarding based on the usefulness of neighboring peers. These index structures are same as those designed for working with static data as they focus on the nature of the data content rather than the actual content itself.

Distributed hash table (DHT) based data lookup techniques for P2P was described in Chord [20] and a few other systems like CAN, Pastry and Tapestry. DHT uses hash functions to assign data to nodes as well as to perform the lookup operation. Since a change in data will change the associated hash value, static data is used as the key value. DHT based techniques are first designed to support point queries while later there are DHT variants [10] that work with range queries as well, but neither case explicitly handles data items frequently change in nature and might result in a large update overhead if used with dynamic data.

Indexes for sensor networks: Another area where the problem of dynamic data management is encountered is sensor networks [8][9][12]. Sensor networks require distributed index structures that can efficiently manage dynamic sensor data while ensuring that the data management procedure is energy-efficient. Sensor network indexes are query-driven and operate in a proactive or reactive mode based on the query nature and frequency.

A different approach to managing sensor network data is taken by visualizing the sensor network as a database. This involves processing each query issued to the sensor network similar to a database query and directing the query to specific sensors as determined by the query plan. Since the index structures are constructed based on the query issued, the data management aspect does not address the concern of indexing the actual dynamic sensor data. The scope of these index structures is generally limited to a single sensor network.

III. SYSTEM OVERVIEW

The overall architecture of our Coalition [5][24] context-aware middleware is illustrated in Figure 1. The operating spaces in Coalition are categorized and clustered into multiple domain classes such as persons and shops. Each class is called a *context domain* and all spaces in a domain provide a similar set of context attributes [24]. The system introduces the concept of an *operating space gateway (OSG)*, which is a software program that provides a single point of interaction between an operating space and the "outside" applications or other operating spaces. The OSGs of all operating spaces having a common attribute in a domain are organized into a P2P network. This P2P network having OSG peers is termed as a *semantic cluster*. The semantic clusters for different attributes in a domain are connected with a ring topology to form an overall semantic P2P overlay.

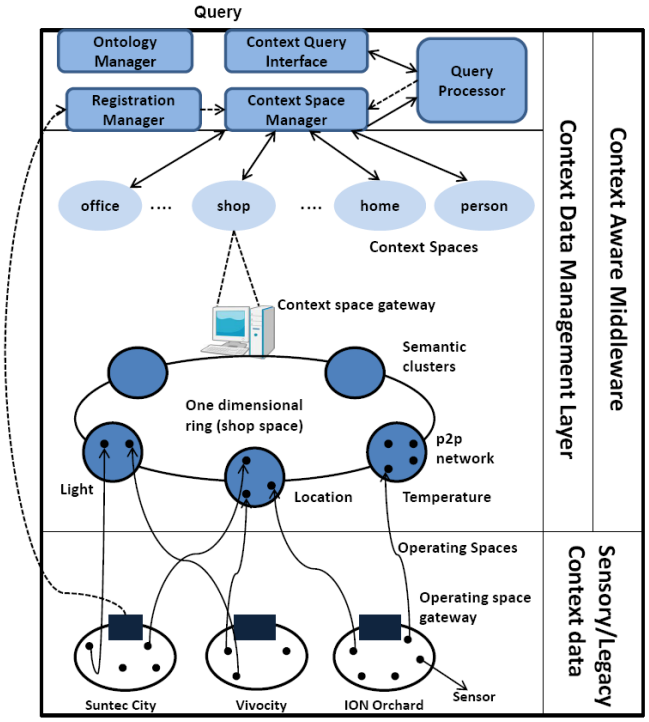

Figure 1. Overview of Coalition System Architecture.

The context data at all operating spaces is modeled using a simple key-value model to make the design generic and extensible. The clustering process of operating spaces into different context domains is automatic. There exist a set of global context schemas at the system server, each of which corresponds to one of the context domains and is incrementally updated from the local schemas submitted by individual operating spaces during registration. The integration of local space schemas into global domain schemas is handled by a context schema matcher that we developed. The technical details of the schema matcher are beyond the scope of this paper and available in our prior paper on schema matching [23].

The context domain manager at the system server manages the data structures of individual context domains and the "pointers" to their corresponding semantic P2P overlays: each overlay of a domain is associated with a *context domain gateway (CSG)* that acts as the single entry of query injection from the server into the overlay. The overlay for the SHOP context domain is amplified in Figure 1 as a first-hand example. More details of the overlay structure and operations are presented in Section IV.

Coalition supports an SQL-based declarative interface [24] for context-aware applications to acquire data from operating spaces or to subscribe to the events of interest that occur in these spaces. The context query engine identifies the proper context domains involved in a query and forwards the query to be executed onto OSG peers in the desirable semantic clusters of the domain's P2P overlay. Because a semantic cluster may contain numerous OSGs, context lookup in a cluster can generate large overhead if a simple flooding protocol is used to route the query throughout the P2P network wherein no distributed indexing is implemented for acceleration. The semantic overlay based lookup mechanism we propose in this paper aims to address this inefficiency issue by reducing the query routing cost via the introduction of attribute value-based sub-clustering and ordering of OSGs within a semantic cluster.

IV. SEMANTIC PEER-TO-PEER OVERLAY

A. Overall Overlay Structure

The structure of our proposed semantic P2P overlay for dynamic context lookup is exemplified in Figure 2.

As we have discussed earlier, the global schema maintained by the Coalition system is integrated from all the local schemas of operating spaces in the domain. The global schema of a domain is then mapped into a semantic P2P overlay with all attributes in the schema mapped to the semantic clusters of P2P networks in the overlay.

A semantic cluster groups gateways for operating spaces that have a common attribute. For example, all OSGs of the shops providing attribute temperature will join as peers in the semantic cluster "temperature", as shown in Figure 2. In order to facilitate a better context lookup operation, a semantic cluster is further partitioned into a number of disjoint range clusters. Each range cluster corresponds to a specific range of values for the attribute (See Figure 2). Each range cluster forms a sub-cluster of smaller P2P network that is contained in the overall P2P network of a semantic cluster. The peer membership of operating spaces in a semantic overlay is flexibly maintained in a distributed manner.

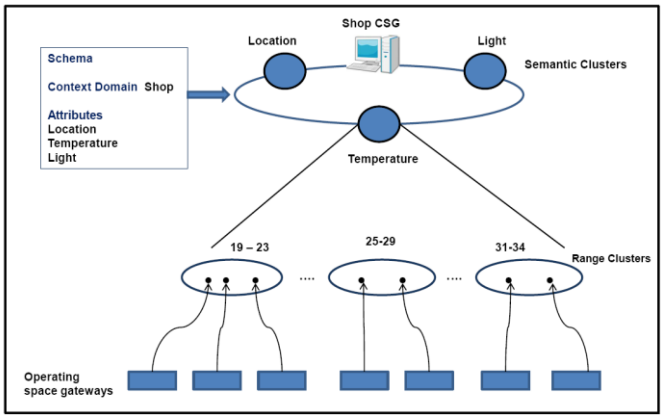

Figure 2. Illustration of the Overlay Structure.

An operating space joins a semantic cluster when it provides the attribute of the cluster, and it leaves the cluster when it no longer provides the attribute. An operating space joins a range cluster of a semantic cluster when its current attribute value falls into the particular value range of the cluster, and it leaves the cluster when its attribute value falls outside the range.

B. Context Lookup in semantic and range clusters

The notion of semantic cluster is introduced in our system to accelerate the context lookup process by grouping operating spaces according to the context data semantics they provide. A semantic cluster for an attribute of a domain is created when the first operating space registers the attribute to Coalition. This 'on-demand' approach bypasses the need to create the semantic clusters beforehand. A semantic cluster is removed when there is no space left in the cluster due to the OSG de-registrations.

An OSG providing an attribute of a domain must be assigned to a proper range cluster in the corresponding semantic cluster of the domain's P2P overlay. The assignment is based on the real-time value of the attribute at the OSG when it registers. Later, each OSG must monitor its attribute value periodically and move itself to a new range cluster when the new attribute value moves out of the bounds of the current range. The operations of an OSG joining a range cluster and switching to a different one upon attribute value change are shown in Figure 3.

Each range cluster maintains a range of numeric values in our approach. Attributes having symbolic string values are handled by hashing the strings to a numeric representation using a hash function. We set two system-defined parameters to restrict the maximum and minimum numbers of OSGs that a range cluster can contain. These values are used to ensure that a cluster is not too heavily or lightly loaded.

The context lookup in Coalition is carried out by an application issuing a query that specifies the required data acquisition from operating spaces satisfying certain range-

 ISBN: 978-1-61208-100-7

search conditions. In the absence of range clusters, the query must be flooded to all OSGs in a semantic cluster and is not scalable with large cluster size. The usage of range clusters addresses this problem by minimizing the search space. The query processing operation is depicted in detail in Figure 4.

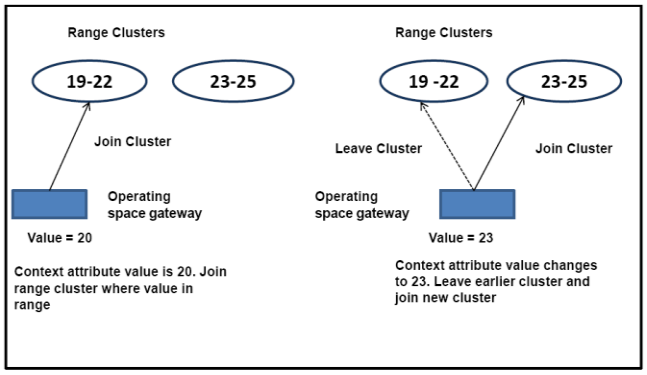

Figure 3. Joining and leaving of range clusters for OSG.

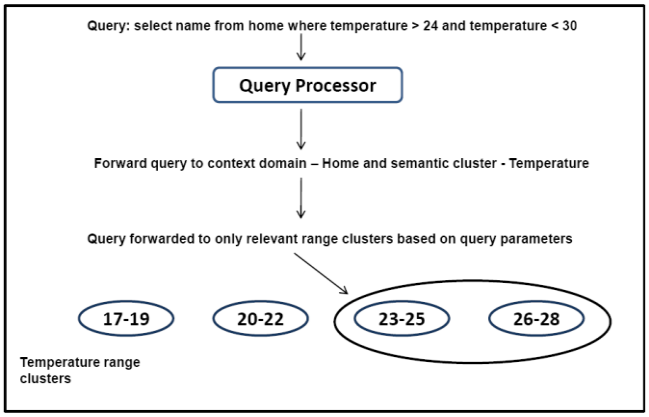

Figure 4. Query processing with range clusters.

Since the query is only flooded in the relevant range clusters rather than the whole semantic cluster now, the context lookup process is more efficient than the flooding based approach.

C. *Maintenance of range clusters*

The basic operations required for the range clusters include the provision for allowing an OSG to join/leave a particular range cluster. Moreover, in order to ensure that the load on a range cluster is not too heavy or too light, a range cluster can either be merged with a neighboring cluster or split into two clusters.

The joining and leaving of OSGs in a range cluster can trigger the splitting and merging of the cluster: if the operation causes the cluster size to fall above/below our system-defined values for maximum/minimum cluster sizes, the cluster is adjusted. The functional pseudo-codes realizing the joining and leaving of an OSG in a range cluster are shown Functions 1-2.

The cluster splitting operation requires OSGs in a range cluster to be redistributed among the two split clusters, as in Figure 5. The operation is carried out by sorting all OSGs in the cluster to be split in an ascending order of their current attribute values. The cluster is then divided into two smaller clusters. The first half of the sorted list is assigned to the first cluster and the second half the second cluster. The cluster bounds of the new clusters are updated accordingly. The cluster splitting operation only affects the cluster being split and does not affect other range clusters. The number of range clusters is increased by one due to this operation.

Function 1: Join_Range_Cluster
Begin
  Locate required context domain and semantic cluster
  If no range cluster present
    Create a new range cluster
  Else
    Locate range cluster by comparing attribute value against cluster bounds
    Assign operating space to the range cluster
    If (range cluster size > Maximum Cluster Size Threshold)
      Split Cluster
End

Function 2: Leave_Range_Cluster
Begin
  Locate range cluster by checking attribute value against cluster bounds
  Remove the operating space from range cluster
  If (range cluster size < Minimum Cluster Size Threshold)
    Merge the cluster with an adjacent one
  Else if (range cluster size == 0)
    Delete corresponding semantic cluster

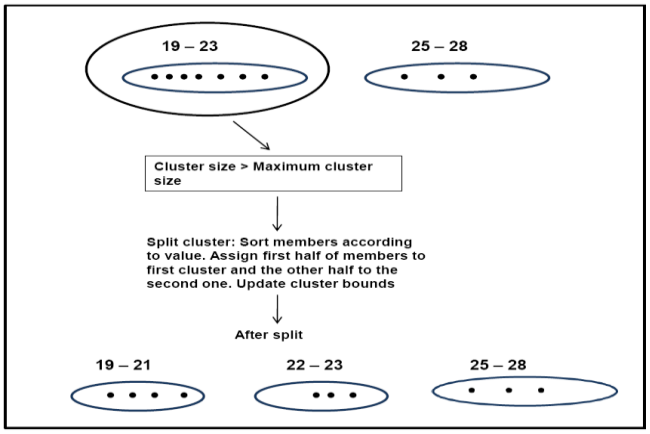

Figure 5. Splitting of a range cluster

In the range cluster merging operation, the cluster is merged with its adjacent cluster to form a bigger cluster. The operation is depicted in Figure 6.

If the cluster to be merged is the first or last range cluster in the sequence, there is only one option for choosing the adjacent cluster to be merged. In case a cluster having two adjacent clusters needs to be merged, the cluster is merged with the adjacent cluster having the lower size among the two. Also, in case the merging process creates a new cluster with a size greater than the maximum cluster size, a

subsequent splitting operation is required. The new cluster bounds are obtained by merging the cluster bounds of the clusters being merged. This operation decreases the number of range clusters by one.

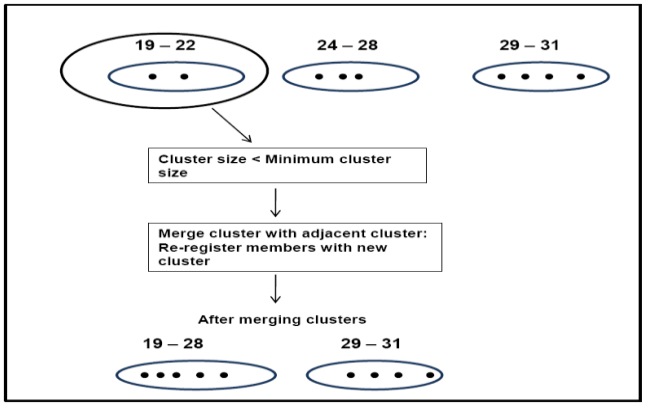

Figure 6. Merging of a range cluster

D. *Further discussions on range clusters*

Our current implementation of range clusters defines the range bounds of clusters as a numerical value range. Simple string attributes can also be partitioned using this scheme by hashing a string to get a hashed numeric value. The range clusters can then be constructed on these hashed values. However, such hashing method is only useful for answering queries that look for exact string matching but not for wildcard based queries.

To handle composite attributes composed of one or more sub-attributes (e.g. *location* composed of *city* and *street*), a simple solution is to hash the composite representation of the attribute value to a real number and generate the range clusters based on the hash values. This approach does not prove to be useful for queries interested in a sub-attribute. The current range cluster approach needs to be extended to support more sophisticated queries with regard to string attributes and composite attributes. The changes to be made to the range cluster structure to handle different types of context in practice also need to be studied.

The proposed current range cluster design generates clusters based on attribute values. Since the values of context attributes tend to be dynamic, the range cluster bounds need to be carefully chosen to minimize the number of cluster update operations. An alternative to this technique would be using the statistical properties of data, e.g. mean and variance, to generate the range clusters. This idea has been explored in [22] where the statistical properties of data are used to build an R-tree like index structure.

V. EXPERIMENTAL RESULTS

We present our current performance evaluation results of the proposed semantic P2P overlay in this section.

A. *Experimental Setup*

We implemented the semantic P2P overlay structure on top of Gnutella [11] in our Coalition prototype. We used a desktop PC as the middleware server and other four PCs to host the OSGs in the experiments. Each PC has an Intel Core 2 Duo 2.83 GHz CPU and runs the Windows XP OS. As our OSG is a software module, we uniformly installed and ran multiple OSG instances onto the four PCs to emulate a number of "operating spaces" in different experimental rounds. Each value in a figure or table shown in the following is the average of tens of independent experimental runs.

B. *Query Response Time*

We first studied the query response time over the semantic P2P overlay. The *response time* of a query is measured as the time interval between the issuing of a query from an application and the reception of query result by the application. In each round of this experiment, a random percentage of OSGs in a few range clusters of a semantic cluster had required results for the query. The total number of OSGs emulated in each run of experiment is the network size for that run.

We compared the query response time achieved by three P2P schemes: the original Gnutella flooding implemented using two different underlying protocols – TCP and UDP, as well as our proposed overlay structure using UDP. The results are shown in Figure 7 with the total number of OSGs in the semantic cluster where the query is routed and executed inside varying.

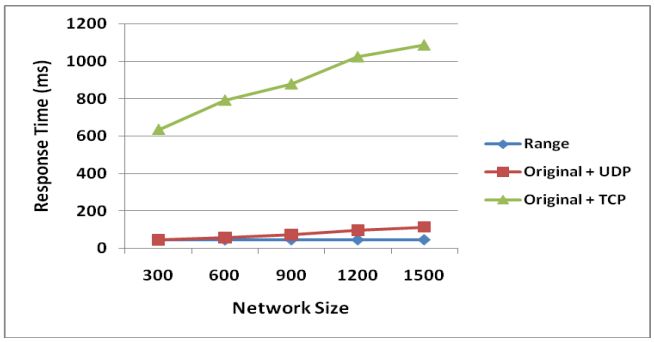

Figure 7. Query response time with different network sizes

The results show that the query response time increased with network size for all three schemes. This is intuitive as network size increase effectively increases the size of search space for a query. The increase curve of our semantic overlay with range clustering (denoted as "Range" in Figure 7) is flatter than the other two schemes. The response time increase of our approach is almost negligible in the experiment, because our distributed indexing based on range clusters largely reduce the search space of query flooding.

We further examined the response time of different schemes by varying the percentage of OSGs with the required results for a query in a semantic cluster. The network size of the semantic cluster was fixed to be of 100 OSGs in this experiment. The results are in Figure 8. Since

 ISBN: 978-1-61208-100-7

the response time of a query depends on the number of OSGs reporting a valid answer, the time intuitively increases with the growing percentage of valid answer sources.

This increase magnitude is quite steep for the original Gnutella scheme, either using TCP or UDP, compared to our approach whose increase curve is smooth after the initial sharp rise. This indicates our semantic overlay could achieve an overall better response time due to the ordering of OSGs according to attribute values, which enables the faster localization of OSGs having valid answers.

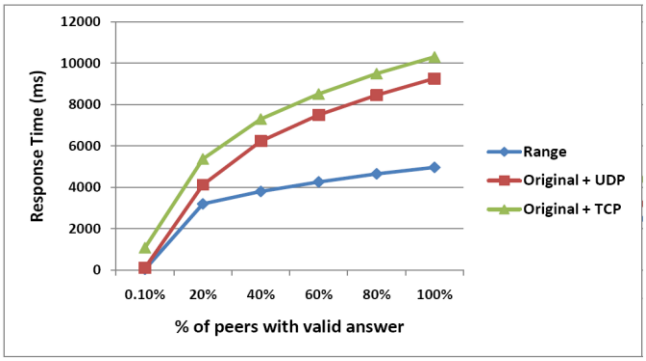

Figure 8. Query response time with different number of qualifying OSGs

C. *Time breakdown for query processing and maintenance operations*

We next analyzed the time breakdown for query processing to identify the time taken in different stages of the operation. Table I shows the breakdown of query response time in our approach and UDP-based Gnutella flooding. The dominant operation in the time breakdown is observed to be the P2P lookup operation since it involves the query flooding within the underlying P2P network. The results clearly illustrate that our approach reduces the P2P lookup time compared to the original UDP version of Gnutella, and achieves an overall faster query response time. Since our approach also requires maintenance like splitting and merging of range clusters, we analyzed the time breakdown of these maintenance operations in Tables II-III, respectively. We also measured the time breakdown of an OSG joining/switching a new range cluster in Table IV.

It is observed that the cluster splitting and merging operations require a similar maintenance overhead with the merging requiring slightly more. The regrouping costs of OSGs in the newly merged/split clusters dominate the total time taken for these operations, as expected. Unless the dynamism of the context data and the memberships of OSGs are high, the frequency of merging/splitting will be low and hence the latency of the operation will not affect the overall system performance. The operation of an OSG joining a new range cluster is comparatively faster than merging or splitting.

TABLE I. TIME BREAKDOWN FOR QUERY PROCESSING

Operation	Time Taken (Range) (ms)	Time Taken (UDP) (ms)
Query parsing	1	1
Context Domain Lookup	2	2
Semantic Cluster Lookup	7	9
Range Cluster Lookup	11.2	
P2P Lookup	26.06	95.4
Total	47.26	105.4

TABLE II. TIME BREAKDOWN FOR CLUSTER SPLITTING

Operation	Time Taken (ms)
Sorting	120
Cluster update	6.3
PSG reclustering	757.9
Total	884.2

TABLE III. TIME BREAKDOWN FOR CLUSTER MERGING

Operation	Time Taken (ms)
Cluster update	0
Re-Grouping of affected PSGs	647.4
Total	647.4

TABLE IV. TIME BREAKDOWN FOR JOINING RANGE CLUSTER

Operation	Time Taken (ms)
Range request	15.9
Neighbour notification	26.5
Neighbour reconnection	18.7
Total Time	61.1

VI. CONCLUSION

We have presented a semantic P2P overlay structure to support dynamic context lookup in a context-aware middleware. The P2P overlay is created for each context domain and represents the set of attributes in the domain. Each semantic cluster of P2P network for an attribute in the domain is further partitioned into a number of range clusters to support the efficient lookup of dynamic context data via simple SQL-based queries. Our simulation results demonstrate the proposed overlay structure achieves a good response time for context lookup. The results also suggest the time required for the range cluster maintenance operations is reasonably small. Our future work includes more extensive testing of the range cluster-based semantic P2P overlay, extending the structure to support symbolic or composite context attributes and the impact of dynamism of OSGs on the system performance.

ACKNOWLEDGMENT

This work is partially supported by National Research Foundation grant NRFIDM-IDM002-069 on "Life Spaces (POEM)" from the IDM Project Office, Media Development Authority of Singapore.

 ISBN: 978-1-61208-100-7

REFERENCES

[1] M. Ali and K. Langendoen, "A case for peer-to-peer network overlays in sensor networks," Proc. WWSNA, 2007, pp. 56-61.

[2] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," International Journal of Ad-Hoc and Ubiquitous Computing 2 (4) (2007), pp. 263-277.

[3] R. Blanco, N. Ahmed, D. Hadaller, L. Sung, H. Li, and M. Soliman, "A survey of data management in peer-to-peer systems," Tech. rep., University of Waterloo (2006), pp. 1-51 .

[4] G. Chen, M. Li, and D. Kotz, "Data-centric middleware for context-aware pervasive computing," Pervasive Mob. Computing. 4 (2) (2008), pp. 216-253.

[5] Context-Aware Middleware Services and Programming Support for Sentient Computing, http://lucan.ddns.comp.nus.edu.sg:8080/PublicNSS /researchContextAware.aspx (last accessed 03/06/2010)

[6] A. Crespo and H. Garcia-Molina, "Routing indices for peer-to-peer systems," Proc. ICDCS, 2002, pp. 23.

[7] A.K. Dey, "Understanding and using context," Personal and Ubiquitous Computing 5 (1) (2001), pp. 4-7.

[8] Y. Diao, D. Ganesan, G. Mathur, and P.J. Shenoy, "Rethinking data management for storage-centric sensor networks," Proc. CIDR, 2007, pp. 22-31.

[9] V. Dyo and C. Mascolo, "Adaptive distributed indexing for spatial queries in sensor networks," Proc. DEXA, 2005, pp. 1103-1107.

[10] J. Gao and P. Steenkiste, "An adaptive protocol for efficient support of range queries in DHT-based systems," Proc. ICNP, 2004, pp. 239-250.

[11] Gnutella Protocol Development, http://rfc-gnutella.sourceforge.net (last accessed 10/07/2010)

[12] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, and S. Shgenker, "DIFS: A distributred index for features in sensor networks," Ad Hoc Networks 1 (2-3) (2003) , pp. 333-349.

[13] A. Guttman, "R-trees: A dynamic index structure for spatial searching", Proc. SIGMOD, 1984, pp. 47-57.

[14] K.E. Kjær, "A survey of context-aware middleware," Proc. SE, 2007, pp. 148-155.

[15] N. Lester, J. Zobel, and H. Williams, "Efficient online index maintenance for contiguous inverted lists", Information Processing & Management 42 (4) (2006), pp. 916-933.

[16] W. Liao, G. Tang, N. Jing, and Z. Zhong, "VTPR-tree: An efficient indexing method for moving objects with frequent updates", Proc. CoMoGIS, 2006, pp. 120-129 .

[17] L. Lim, M. Wang, S. Padmanabhan, J. Vitter, and R. Agarwal, "Efficient update of indexes for dynamically changing web documents", World Wide Web 10 (1) (2007), pp. 37-69.

[18] E.K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," IEEE Communications Surveys & Tutorials 7 (2) (2004), pp. 72-93.

[19] S. Šaltenis, C.S. Jensen, S.T. Leutenegger, and M.A. Lopez, "Indexing the positions of continuously moving objects", Proc. SIGMOD, 2000, pp. 331-342.

[20] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications," IEEE/ACM Transactions on Networking 11 (1) (2003), pp. 17-32.

[21] Y. Tao, D. Papadias, and J. Sun, "The TPR*-tree: An optimized spatio-temporal access method for predictive queries", Proc. VLDB, 2003, pp. 790-801.

[22] Y. Xia, S. Prabhakar, S. Lei, R. Cheng, and R. Shah, "Indexing continuously changing data with mean-variance tree", Proc. SAC, 2005, pp. 263-272.

[23] W. Xue, H.K. Pung, P.P. Palmes, and T. Gu, "Schema matching for context-aware computing," Proc. Ubicomp, 2008, pp. 292-301.

[24] W. Xue H.K. Pung W.L. Ng, and T. Gu,"Data management for context-aware computing", Proc. EUC, 2008, pp. 492-498.

[25] W. Xue, H.K. Pung, W.L. Ng, C.W. Tang, and T. Gu, "Gateways of physical spaces in context-aware computing," Proc. ISSNIP, 2008, pp. 441-446 .

 ISBN: 978-1-61208-100-7