A Mean-Variance Based Index for Dynamic Context Data Lookup

Shubhabrata Sen and Hung Keng Pung

School of Computing, National University of Singapore
{shubhabrata.sen,dcsphk}@nus.edu.sg

Abstract. primary functionality of context aware applications is the retrieval of different types of context data from various context sources and adapting their behavior accordingly. In order to facilitate context aware application development, a context aware middleware must provide an effective context data management and lookup strategy. The use of a traditional index for indexing dynamic context data is not feasible due to the high update overhead. In this paper, we propose a context data indexing mechanism that utilizes the statistical properties of data viz. the mean and variance to cluster similar data values together and minimizes the need for frequent index updates. Experimental results indicate that the performance of the proposed index structure is satisfactory with respect to the query response time and query accuracy together with a low maintenance overhead.

Keywords: Context-awareness, Dynamic data, Context data management, Context Lookup.

1 Introduction

The recent advances in the fields of pervasive and ubiquitous computing have led to an increased focus towards the development of sophisticated context-aware applications. Context-aware applications need to efficiently retrieve and manage context data from different context sources. A context source here refers to an entity (physical or virtual) that is responsible for the generation of context data. In order to facilitate the development of context-aware applications, there has been an increased research interest towards the development of context-aware middleware [1][2]. An integral component of a context-aware middleware is a data management component that can effectively manage multiple different types of context data from a variety of context sources as well as support the context lookup operation over multiple context sources. The context lookup process can be defined as the process of identifying the context sources that an application should acquire data from and the retrieval of the data from these sources.

The notion of 'context' [3] can be defined as any data that can be used to characterize the situation of an entity involved in the user-application interaction. Context data can be either static or dynamic. Dynamic context data changes asynchronously and frequently such as the location of a person and the temperature in an office. As context-aware applications primarily operate by adapting to context

A. Puiatti et al. (Eds.): MobiQuitous 2011, LNICST 104, pp. 101–112, 2012.

changes, the context data involved in these applications is primarily dynamic. The involvement of dynamic attributes makes the context lookup process more challenging as the applications need to deal with continuously changing data values. An index structure designed to work with dynamic data needs to handle the update overheads caused due to frequently changing data.

Context-aware applications often need to identify a subset of context sources from a collection that satisfy some particular conditions. Such context lookup queries are similar to the range queries observed in databases. As part of our current research project Coalition [4] aimed towards developing a prototype of a context-aware middleware, semantic peer-to-peer overlays are used to aid context data lookup. Context sources are grouped according to the semantics of the data provided by them. However, even within a single semantic group, there can be a large number of context sources that can hamper the processing of range queries as the query needs to be sent to every context source.

In this paper, we propose an indexing mechanism that attempts to provide an index structure similar to that of databases for dynamic context data. The index functions by partitioning context data into multiple range clusters and reduces the search space for a query. In order to minimize the index update overhead for dynamic data, we utilize the observation presented in [5] that states that the statistical properties of data viz. the mean and variance are more resistant to change than the actual data values and can be used to construct indexes for dynamic data. Our proposed index utilizes this observation to develop an index for dynamic context data that can be utilized by applications to resolve range and point queries over context data.

The rest of the paper is organized as follows. We give an overview of our Coalition system in Section 2. Section 3 gives the description of the mean-variance calculation process over dynamic data. We discuss the index construction and context lookup operations in Section 4. The current experimental results are described in Section 5. The related work is discussed in Section 6. Section 7 concludes the paper with future research directions.

2 System Overview

The Coalition middleware uses two concepts to provide an abstract representation of context data – *operating spaces* and *context spaces*. The context data representation techniques used in Coalition is illustrated in Fig. 1. An operating space is defined as a person, object or place that provides a common interface for managing the context data retrieved from the context sources affiliated to it. This interface is maintained as a software module defined as the *operating space gateway (OSG)*. Coalition categorizes the operating spaces into multiple domain classes termed as *context spaces* such as shops and person where each context space has a set of attributes. OSGs sharing an attribute within a particular context domain are grouped into a P2P network with the OSGs as the peers. This P2P network is defined as a *semantic cluster*. Context-aware applications can use Coalition to acquire context data through a SQL-based query interface [6]. The query processor of Coalition receives and parses queries to identify the relevant context space and semantic cluster.

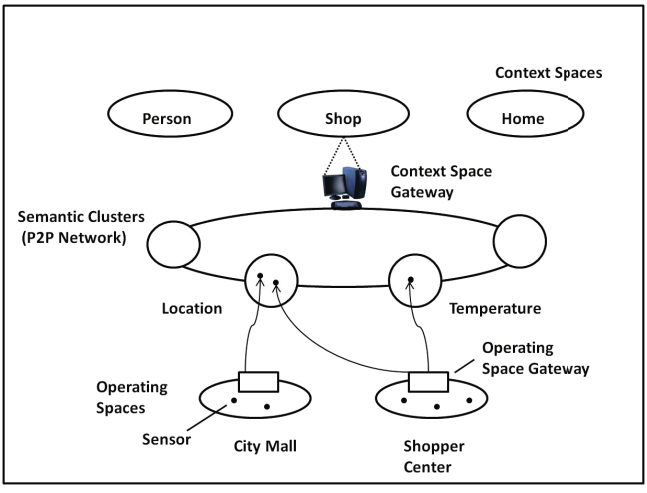


Fig. 1. Context Data representation in Coalition

The query is then distributed within the semantic cluster and each OSG processes the query against its local data store as no data is stored within the middleware. In the absence of any ordering within a semantic cluster, the only option for resolving the query is by flooding the entire p2p network which can prove to be expensive if the number of OSGs within a semantic cluster is large. In this paper, we attempt to address this issue by providing an index that creates an ordering amongst the OSGs in a semantic cluster.

3 Using Mean and Variance to Index Dynamic Data

As mentioned previously, the mean-variance based indexing scheme proposed in this paper is partly based on the idea described in [5].The motivation behind using the mean and variance properties of data to build an index is that even though a data item may be dynamic, the associated mean and variance values are relatively stable. For example, sensory data like temperature changes continuously but the changes are confined within a range for a significant period of time. Hence, indexes constructed using the statistical properties of data are expected to have a lower index updating cost. If a data attribute A is dynamic and continuously changing and has a set of historical values A_1, A_2, A_3, …, A_n, the mean value of A is denoted by μ_A and is calculated as the sum of the values divided by the sample size. The variance for attribute A is denoted by δ_A and the formula for calculating the bias-corrected sample variance is given by

$$\delta_A = 1/(n-1)\sum_{k=1}^{n}(A_k - \mu_A)^2 \quad (1)$$

The mean and variance values calculated for a given context attribute are used to generate the range clusters that constitute our proposed index. We utilize the basic structure of the mean and variance calculation process as discussed in [5]. The calculations are carried out within the OSG of an operating space. An overview of the mean-variance calculation process is given in Fig. 2.

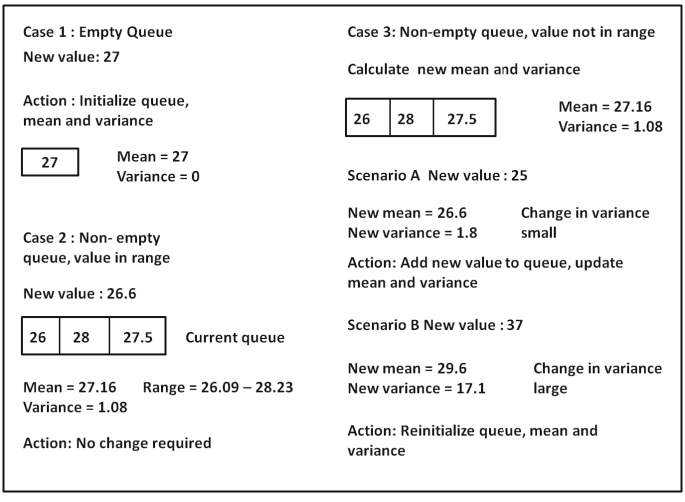


Fig. 2. The mean-variance calculation process

A dynamic data attribute results in a continuous influx of data values and attempting to use the entire set of data values for calculation will be expensive in terms of memory as well as processing. We use a dynamic queue to hold the values over which the mean and variance is calculated as compared to the sliding window approach adopted in [5]. As observed, no changes are required as long as the data value stays within the mean-variance interval. Since the data changes according to the expected pattern, the mean-variance parameters are left unchanged. If the data value exceeds the interval, the mean-variance values are updated and the change in the variance is used to assert whether the data pattern is changing. The change in variance is compared against a pre-defined threshold. A large change in the variance causes the queue to be reinitialized and the process is restarted whereas in the other case, the mean-variance values are updated and the new data value is added to the queue.

4 Mean-Variance Based Index for Context Data Lookup

4.1 Index Structure for Context Lookup

The proposed index structure utilizes the mean and variance values calculated within each OSG to partition the OSGs into multiple range clusters that can reduce the search space for a query. As the index is required to operate within a context aware middleware where OSGs can leave and join the system in a dynamically, a clustering algorithm that can generate clusters incrementally as new data values are received needs to be used. The sequential leader clustering algorithm [7] generates clusters incrementally according to the sequence of input values and can produce a large number of small sized clusters. We propose a clustering algorithm as part of our index structure that is a variant of the leader clustering algorithm that strives to avoid this problem.

The proposed index structure operates within the scope of a single semantic cluster. As of now, only single valued numerical context data attributes are supported. As every OSG in a semantic cluster contains the data value for the corresponding

attribute, each OSG represents a data sample. All the OSGs are assigned to a single cluster during the initial phase of index construction when no clusters are present. This process is continued till the size of the cluster exceeds a system-defined threshold for the maximum cluster size.

The first step in the clustering process consists of the server retrieving the current value from every OSG within the cluster for the following parameters: the mean-variance range bounds ($[\mu-\delta, \mu+\delta]$), the data value for the context attribute and a unique identifier assigned by the server to the OSG during registration. These values are stored in separate tables and are indexed using the unique identifier. The process of identifying the initial clusters is highlighted in Fig. 3.

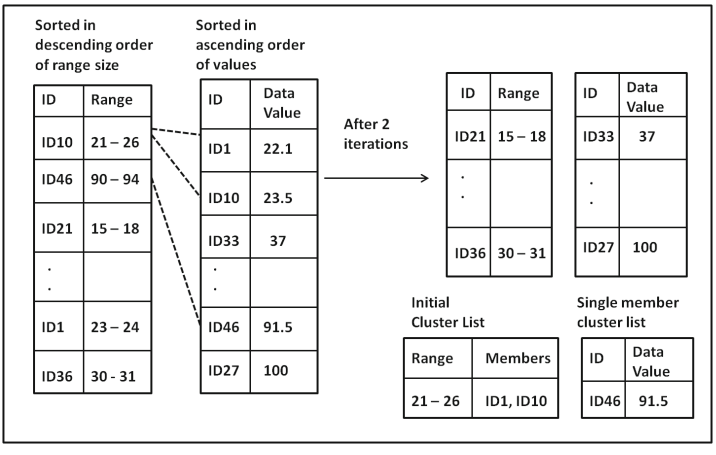


Fig. 3. The identification of the initial clusters

The clustering process aims to group OSGs with similar data change patterns. In order to achieve this, the mean-variance ranges are taken to be the cluster bounds and the data values falling within a particular mean-variance range are assigned to the same cluster. In order to facilitate this operation, the mean-variance range table is sorted in descending order of range size as the larger sized ranges will have a better chance of accommodating more data values and the data value table is sorted in ascending order. Once a data value is assigned to a cluster, the corresponding identifier is used to delete the corresponding entries from both the tables so that these entries are not considered further. If the only data value falling within a particular mean-variance range is found to be the one corresponding to the same OSG, the OSG is added to the single member cluster list. The cluster identification process is carried out until the range table is empty. The main operation of the algorithm involves the assigning of data values to the clusters. Since the data values are stored in a sorted table, an approach similar to binary search can be used to identify the subset of values to be considered for every cluster bound. With this assumption, the running time of the algorithm can be taken to be as O(nlogn) where n is the number of data values. However, since the size of both the tables is expected to decrease gradually due to the removal of the matching values, the running time of the algorithm in practice is expected to be faster.

The next step is the generation of the clusters that constitute the index. Since there can be a large number of small sized clusters generated during the initial phase, these clusters are merged to keep the total number of clusters low. After the initial clusters have been identified, they are arranged in order of their cluster bounds in preparation for merging. The final cluster generation process is described in Fig. 4.

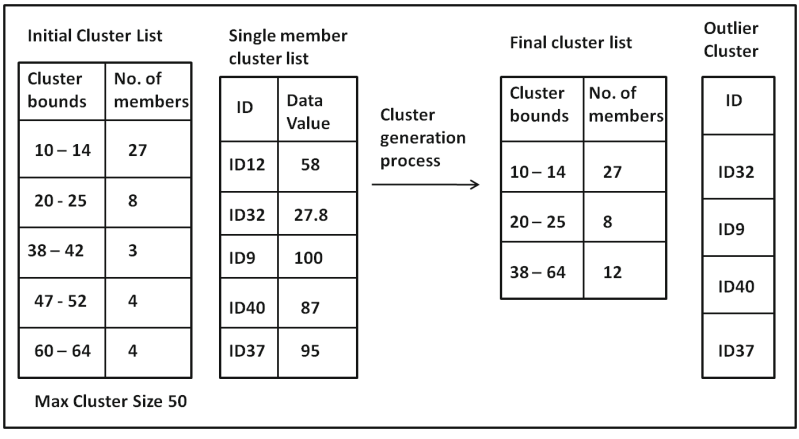


Fig. 4. The generation of the final clusters

A cluster is merged with its adjacent cluster only if the following two conditions are satisfied – both the size of the cluster being merged as well the size of the cluster after merging stay below 50% of the maximum cluster size. This heuristic is utilized to keep the clusters lightly loaded. After the final clusters are generated, the extent of some of the cluster bounds will increase due to merging. Consequently, some of the OSGs in the single member cluster list can now be allotted to one of the clusters. If the single member cluster list still contains some members after this operation, these indicate data values that are not part of any of the present clusters. In a typical clustering scenario, these values would be considered as outliers. However, they still represent valid values in our case and we cannot discard them. In order to handle these values, we designate a special cluster known as an *outlier cluster*. The outlier cluster is used to store the values that cannot be currently assigned to any of the existing clusters. After all the clusters have been finalized, the corresponding OSGs register with their allotted clusters. The semantic cluster now consists of a number of range clusters each of which forms a separate P2P network.

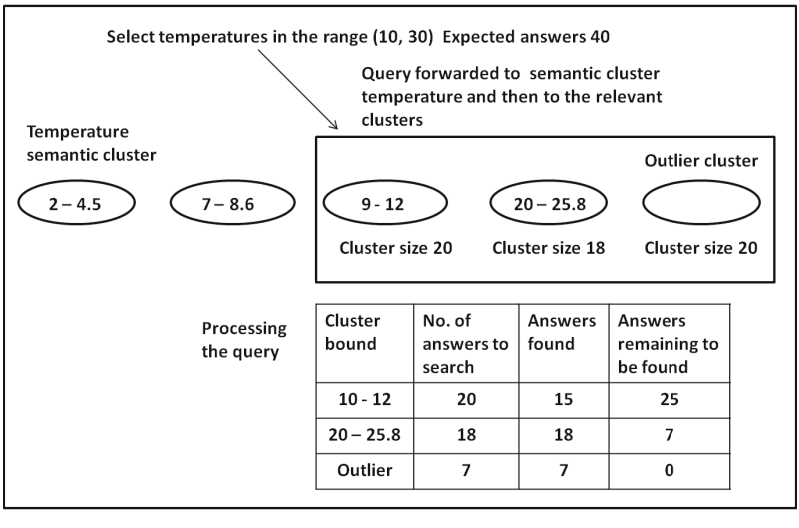
4.2 Maintenance of the Index Structure

After the initial clusters have been identified, subsequent OSGs are assigned to one of the clusters depending on their attribute values. The outlier cluster is used to accommodate the OSGs that cannot be currently assigned to any cluster. Whenever a cluster or an outlier cluster exceeds the maximum cluster size threshold, the clustering process discussed above is invoked within that cluster. Hence, even though the initial cluster bounds may not be very fine-grained, the clustering performance improves as

more data values are received. The index structure keeps evolving continuously as OSGs join and leave the system. After an OSG has been assigned to a cluster, it is possible for the OSG to move to a different cluster if its data pattern changes. A large shift in the variance value during the mean-variance calculation process can be taken as the indication to check if a cluster update operation is required. Alternatively, the OSG can periodically check its value against the cluster bounds and decide if it needs to move to a different cluster. This policy cannot be dictated by the server and it needs to be enforced within an OSG. The server periodically checks the cluster sizes in order to ensure that smaller size clusters are merged with adjacent clusters to keep the number of clusters low. Also, clusters with no members are removed from the system.

4.3 Context Lookup Using the Index

The context lookup in Coalition is carried out by issuing SQL style queries that specify the data acquisition conditions using range-search conditions. Since the query issuer is not expected to know the number of OSGs satisfying the query conditions, each application submits the number of answers expected for the query. The Coalition query processor redirects the query to the appropriate semantic cluster. The query parameters are compared against the range clusters to generate a set of candidate clusters that are expected to have the answer. The query processing operation is depicted in Fig. 5. In case the query parameters are not confined in a single cluster, the outlier cluster is included as part of the candidate cluster set. The processing of a query within a single cluster is controlled using the number of query answers received and a system-defined timeout that controls the duration for which a query is processed within the cluster.



Cluster bound	No. of answers to search	Answers found	Answers remaining to be found
10 - 12	20	15	25
20 – 25.8	18	18	7
Outlier	7	7	0

Fig. 5. The context lookup operation

The query is processed by flooding it in every cluster in the candidate set. For each cluster, the number of expected answers is compared with the cluster size and the number of answers received is compared with the lesser of the two values. This

ensures that the number of answers to be searched within a cluster takes the current cluster size into account. This operation is then repeated in the next cluster in the candidate cluster list with all the parameters suitably updated as observed in Fig. 5. This process ensures that the response time for a query is minimized by customizing the search operation in each range cluster. The context lookup operation terminates either when all the requested number of answers have been received or when the timeout value expires for the last cluster in the candidate cluster list.

5 Experimental Results

The proposed index structure was implemented and integrated with a prototype of the Coalition middleware. A Dell PowerEdge T300 server with four 2.83 Ghz quad-core Intel Xeon CPU was used as the server. Multiple operating spaces were simulated by running multiple instances of the OSG module on 6 desktop PCs where each PC had an Intel Core 2 Duo 2.83 GHz CPU and runs the Windows XP OS. The data set used for testing comprised of a combination of real world and simulated values to ensure randomness. Half of the values were drawn from the Intel lab data set [8]. This set has multiple sensor readings of different physical properties like temperature, humidity etc over a period of time. The test data was generated by taking samples of different lengths from the data stream associated with a sensor for different sensors. The mean and variance values were then calculated over these samples. The simulated data was obtained using a random number generator. A set of values randomly chosen from the set (0, 1000) were designated to be the mean values as well as the data values whereas the variance values were chosen randomly from the set (0, 1.5). Each OSG instance was assigned a randomly chosen unique data sample comprising of a mean, variance and data value from this test data set. The test environment reflected a stable system state wherein a fixed no of OSGs are present, no OSG enters or leaves the system and after an OSG is assigned to a cluster it does not move to another cluster. The experiments were conducted using a context space with a single semantic cluster.

5.1 Query Response Time

We first studied the query response times achieved using the proposed index structure for different network sizes and query window sizes. The response time was measured as the time duration between the issuing of the query and the reception of the answers. The results presented in this section were taken as the average of 400 query runs. Each query issued was a range query with range bounds (A, B) where the bounds were randomly chosen from the test data set for each run. The query window size or the number of valid answers for a query was varied by modifying the range bounds. We compared the query response times achieved using the proposed index structure to that achieved using a flooding based approach. The network size (the total number of OSGs in the system) was varied from 250 to 1500 and the query window size was varied from 10 to 150. The maximum cluster size was set to 50. We highlight the results achieved for the query response time comparison for the case with query window size as 150. The results are illustrated in Fig. 6.

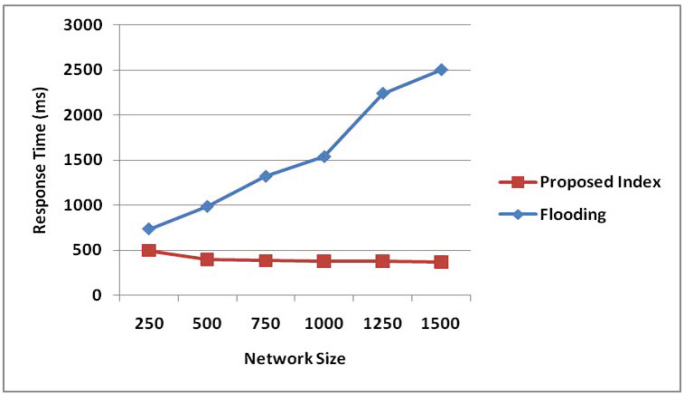


Fig. 6. Comparison of query response times

The results indicate that the proposed index performs quite better than the flooding based approach in terms of variations in query response times with an increase in network size. The increase curve for our proposed index is quite flat as compared to the steep increase for the flooding based approach. This indicates that the proposed approach is scalable with increase in network size. The minimal variation of response time in our approach can be attributed to the fact that the range clusters constituting the index efficiently reduce the search space for a query as compared to flooding.

We further examined the variation in the query response times by varying the network sizes for a particular query window size. The results are illustrated in Fig. 7.

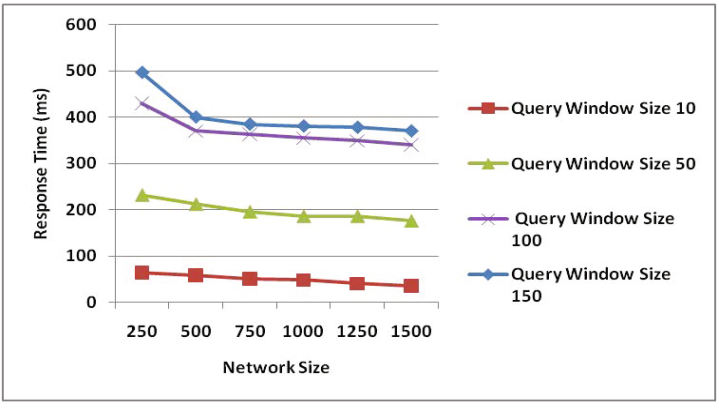


Fig. 7. Query response times with different query window sizes

The results achieved show that the response time increases with an increase in the query window sizes which is expected. It is also observed that for a given query window size, the response time decreases as the network size is increased. The ratio of query window size to network size gives the % of OSGs that contain the valid answers for a query. In Fig. 7, the query window size is kept fixed and the network size is varied. In this case, the ratio gradually decreases which indicates that the subset of the network containing the valid answers becomes smaller as the network

size increases. Intuitively, in the absence of any ordering among the OSGs, a larger network size should result in a larger response time which is observed from the results in Fig. 6 for the flooding based approach. However, the proposed index compensates for the increase in the network size by partitioning the OSGs into multiple clusters. Also, as discussed previously, the proposed index is expected to improve its clustering performance as more data values are received. As the range clusters become more fine-grained, the localization of the OSGs with a valid answer occurs faster thereby resulting in a faster response time. The results indicate the scalability of the system with respect to different network and query window sizes.

5.2 Query Accuracy and Time Breakdown for Clustering Process

We first studied the accuracy of the index structure with respect to the number of answers received for a query. In order to measure the accuracy, the network size was kept constant and the query window size was varied. For each query window size, the accuracy measurements were taken as the average of 200 query runs and the query range bounds were varied for each run. The results for a network size of 1500 are highlighted in Table 1.

Table 1. Accuracy results and timeout values

Accuracy ranges	% of answers received for an accuracy range for different query window sizes				Timeout values for different cluster sizes	
	10	50	100	150	Size	Timeout (ms)
90% - 100%	94	98.5	97.5	97	0 - 10	25
					10 - 20	35
80% - 90%	4	1.5	2.5	1.5	20 - 30	45
					30 – 40	55
70% - 80%	2			1.5	40 - 50	60

The results indicate that the accuracy performance of the proposed index structure is quite satisfactory as a majority of the answers received fall in the high accuracy range for different query window sizes. The cases where the accuracy falls below 90% can be attributed to the fact that the range clusters are P2P networks and are not always expected to return all the answers. Moreover, a timeout value is used as one of the parameters control the processing of a query within a range cluster as discussed previously. If this timeout value is very low, the accuracy can become low and a higher timeout can result in a larger response time. The timeout values used are expected to have a bearing on the query accuracy as well as the response time. We vary the timeout values according to the cluster size and timeouts used in the experiments are highlighted in Table 1.

We next analyzed the time breakdown for the operations involved in the clustering process when a single cluster is split into multiple range clusters. Table 2 shows the time required for the different steps in the clustering process. It is observed that the time required for the clustering process is dominated by the re-registration process that assigns all the OSGs to their respective range clusters. The re-registration

overhead will also be present when two clusters are merged. This indicates that unless the dynamism of context data and the membership of OSGs are very high, the frequency of the re-registration process will be low and the latency will not affect the overall system performance.

Table 2. Time breakdown for clustering process

Operation	Time Taken (ms)
Retrieval of data values from OSGs	167
Generating clusters	25
Re-registering OSGs to assigned clusters	560

6 Related Work

Dynamic data indexes need to focus on minimizing the index update overhead. We discuss some of these index structures in this section.

Research efforts in the area of data management for moving objects have led to the development of R-tree based index structures [9][10] that try to reduce the number of index updates. This is achieved by representing the motion of an object as a function and updating the index according to the variation in function parameters. Alternatively, the physical properties of moving objects like the motion trajectories are used to make indexing decisions. These indexing techniques are not generic as they are designed for a particular category of applications and use the special properties of the application class as part of their index design. Hence, these indexes lack applicability outside their target application domain. Sensor networks are also an important area where support for dynamic data management is required. Indexes for sensor networks [11][12] are usually query-driven and operate proactively or reactively according to the query frequency. In order to store frequently changing sensor data in databases, database systems are augmented with uncertainty management components [13]. These databases associate a data value with an interval and probability distributions functions and handle queries using probabilistic operators that define the answer quality. These indexes operate at the sensor level and directly interact with sensor data as compared to our scenario where we consider large scale context sources over wide areas where each context source encompasses multiple sensors.

There have been efforts to support range queries in p2p systems by extending the DHT based data lookup techniques [14][15]. Since DHT systems utilize hash functions to store and lookup data, the usage of these algorithms with dynamic context data will prove to be difficult as the hash value will need to be continuously updated. Also, the DHT model involving a set of nodes which store data according to the key-node mapping is structurally different from our Coalition middleware architecture as we consider multiple autonomous context sources. Since the context sources are autonomous, it will be difficult to impose a DHT based overlay structure in our system. These problems prevent the adoption of DHT based range query techniques as part of our context aware middleware.

7 Conclusion and Future Work

We have presented a mean-variance based index structure to support dynamic context lookup in a context-aware environment and provide support for range and point queries over context data. This index structure uses an incremental clustering approach to partition the OSGs within a semantic cluster into multiple range clusters based on their mean and variance values. The mean and variance values are used as they are relatively more stable than the actual data which changes frequently. The initial experimental results demonstrate that the proposed index structure achieves good response times and is scalable with respect to the increase in network size. The results also indicate that the performance of the index with respect to accuracy is good and the time required for the clustering process is reasonably small. As part of our future work, we plan to carry out more extensive testing of the index and study the impact of the frequent leave/join operations of OSGs on the index performance, provide support for join operations involving multiple attributes and extend the index structure to support string and composite attributes.

References

1. Baldauf, M., Dustdar, S., Rosenberg, F.: A survey on context-aware systems. IJAHUC, 263–277 (2007)
2. Kjær, K.E.: A survey of context-aware middleware. In: 25th Conference on IASTED International Multi-Conference (2007)
3. Dey, A.K.: Understanding and Using Context. Personal and Ubiquitous Computing, 4–7 (2001)
4. Context-Aware Middleware Services and Programming Support for Sentient Computing, `http://lucan.ddns.comp.nus.edu.sg:8080/PublicNSS/researchContextAware.aspx`
5. Xia, Y., Cheng, R., Prabhakar, S., Lei, S., Shah, R.: Indexing continuously changing data with mean-variance tree. IJHPCN, 263–272 (2008)
6. Xue, W., Pung, H.K., Ng, W.L., Gu, T.: Data Management for Context-Aware Computing. In: EUC (1), pp. 492–498 (2008)
7. Hartigan, J.A.: Clustering Algorithms. Wiley. New York (1975)
8. Intel Lab Data, `http://db.csail.mit.edu/labdata/labdata.html`
9. Silva, Y.N., Xiong, X., Aref, W.G.: The RUM-tree: supporting frequent updates in R-trees using memos. VLDB J, 719–738 (2009)
10. Saltenis, S., Jensen, C.S., Leutenegger, S.T., Lopez, M.A.: Indexing the Positions of Continuously Moving Objects. In: SIGMOD Conference, pp. 331–342 (2000)
11. Dyo, V., Mascolo, C.: Adaptive Distributed Indexing for Spatial Queries in Sensor Networks. In: DEXA Workshops, pp. 1103–1107 (2005)
12. Diao, Y., Ganesan, D., Mathur, G., Shenoy, P.J.: Rethinking Data Management for Storage-centric Sensor Networks. In: CIDR, pp. 22–31 (2007)
13. Cheng, R., Singh, S., Prabhakar, S.: U-DBMS: A Database System for Managing Constantly-Evolving Data. In: VLDB 1271–1274 (2005)
14. Gao, J., Steenkiste, P.: An Adaptive Protocol for Efficient Support of Range Queries in DHT-Based Systems. In: ICNP, pp. 239–250 (2004)
15. Li, D., Cao, J., Lu, X., Chen, K.: Efficient Range Query Processing in Peer-to-Peer Systems. IEEE Trans. Knowl. Data Eng, 78–91 (2009)