CONTEXT DATA MANAGEMENT FOR LARGE SCALE CONTEXT-AWARE UBIQUITOUS SYSTEMS

SHUBHABRATA SEN

Bachelor of Technology, Computer Science and Engineering

VIT University, India

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2013

DECLARATION

I hereby declare that the thesis is my original work and has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis. This thesis has also not been submitted for any degree in any university previously.

Shubhabrata Sen

Shubhabrata Sen

23 December 2013

ACKNOWLEDGEMENT

First and foremost, I would like to thank my supervisor Dr Pung Hung Keng for guiding me through the perilous journey of obtaining a PhD and providing constant encouragement during my moments of self-doubt and having faith in me. The valuable suggestions imparted by him concerning all aspects of research ranging from writing papers, giving presentations, conducting experiments as well as his ideas regarding the direction of my PhD project have been extremely helpful to me and have enabled me to become a better researcher. I would also like to thank Dr Xue Wenwei for his guidance and support during the beginning of my PhD. The discussions that I had with him during the initial phase of my study were instrumental in the formulation of my PhD project. I would like to thank my thesis committee members Dr Chan Mun Choon and Dr Teo Yong Meng for their valuable suggestions and comments for the improvement of the thesis work.

I would like to thank the Department of Computer Science, School of Computing, National University of Singapore for giving me the opportunity to pursue my PhD study. I would like to thank all the members of the Network System and Services Lab including Chen Penghe, Daniel Tang, Vikash Ranjan, Zhu Jian, Xue Mingqiang and Mohammad Oliya for all their support during the course of my PhD. In particular, I would like to thank Chen Penghe for the collaborative work that we carried out together. I would also like to thank our lab technicians Ms Lim Chew Eng and Mr Chan Chee Heng for providing all the necessary assistance to establish the experimental setup for testing my work.

I would like to thank all my friends in Singapore – Deepak, Amit, Rishita, Shreya, Divya, Abhilasha, Lavanya, Prachi, Shilpi, Nina, Sarada, Sangit and Jagadish who helped keep me sane and ensure that my life outside the lab was enjoyable. I would like to especially thank my friend and roommate Deepak who tolerated all my idiosyncrasies, lent a patient ear to my endless cribbing about PhD and offered wise counsel during the times I needed it the most. I really appreciate all the help he provided during my thesis writing phase when I was a total bundle of nerves.

Last but not least, I would like to thank my parents Tapas Kumar Sen and Maitrayee Sen for their continuous encouragement and emotional support during the entire duration of my PhD without which the completion of this journey would have been impossible.

TABLE OF CONTENTS

SUMMARY

The paradigm of context aware computing has been the focus of extensive research interest over the recent years. Context aware computing uses the concept of “context” to realize computing processes that can react and adapt to the changes in their environment. In order to facilitate the development of context aware applications, a number of context aware middleware systems have been proposed. The traditional deployment scope of such systems has been restricted to lab based deployments. However, there is an increasing demand for middleware systems that can efficiently manage context sources over wide area networks thereby making them suitable for real world deployments. Context aware applications need to retrieve context data from different context sources to drive their behavior. This is a challenging problem as context data is usually dynamic and distributed across multiple context sources that may be spread across a large scale area. Also, as applications may need to discover context sources during runtime as a result of changes in user requirements or the operating context, a standard and ubiquitous data discovery and acquisition method is required.

In this thesis, we address the problem of designing and developing a context data management system to manage context data as well as support lookups efficiently over context data. In the first part of the thesis, we propose a range clustering technique to partition the context sources into a set of clusters according to their data values to facilitate the context lookup process. This is a preliminary solution to establish an ordering among the context sources to reduce the search space for a context lookup. We then address the problem of dynamic context data management using a mean-variance based indexing technique which is an extension of the range clustering approach that utilizes the statistical properties of data to design an index that can handle the update overhead due to dynamic data. The next part of the thesis addresses the problem of designing an index structure for string based context data. Since the mean-variance indexing approach is restricted to numeric values, we propose the concept an incremental tree based index structure for string attributes using the concept of radix sort and ternary search trees. In the final part of the thesis, we present the detailed design structure of a hierarchical context data management system that can be used to support context lookup requests with different scopes.

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CDG	Context domain gateway
CSM	Context space manager
LCP	Longest common prefix
LCSM	Location specific context space manager
PSG	Physical space gateway
SC	Semantic cluster
TST	Ternary Search Tree

PUBLICATIONS

1. Sen, S., Xue, W., Pung, H. K., & Wong, W. C., “*Semantic P2P Overlay for Dynamic Context Lookup*”, Proceedings of the Fourth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2010), October 2010
2. Chen, P., Sen, S., Pung, H. K., Xue, W., & Wong, W. C., “*Context Data Management for Mobile Spaces”,* Proceedings of the Seventh Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous 2010)
3. Sen, S., & Pung, H. K., “*A Mean-Variance Based Index for Dynamic Context Data Lookup*”, Proceedings of the Eighth Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous 2011), December 2011
4. Xue, W., Pung, H. K., & Sen, S, “*Managing context data for diverse operating spaces*”, Pervasive and Mobile Computing, 9(1), 57-75, 2011
5. Xue, W., Pung, H. K., Sen, S., Zhu, J., & Zhang, D, “*Context gateway for physical spaces*” Journal of ambient intelligence and humanized computing, 3(3), 193-204, 2012
6. Chen, P., Sen, S., Pung, H. K., Xue, W., & Wong, W. C., “*A context management framework for context-aware applications in mobile spaces*” International Journal of Pervasive Computing and Communications, 8(2), 185-210, 2012
7. Chen, P., Sen, S., Pung, H. K., & Wong, W. C., “*Context Processing: A Distributed Approach*”, Proceedings of the Second International Conference on Intelligent Systems and Applications (INTELLI 2013**)**, April 2013
8. Chen, P., Sen, S., Pung, H. K., & Wong, W. C., “MPSG: a generic context management framework in mobile spaces”, Proceedings of the 8th International Conference on Body Area Networks (BodyNets 2013), 2013

CHAPTER 1

INTRODUCTION

1.1 Context-aware computing

The paradigm of ubiquitous computing has been the focus of extensive study and research over a significant period of time. The notion of ubiquitous computing strives to elevate the desktop based computing model to a more advanced scenario where computing can appear anywhere and everywhere. As per this idea, a computing process can occur in any location, using any available device and in any possible format. In other words, the process of computing becomes more pervasive. An important component of the ubiquitous computing paradigm is the context-aware computing model. This model adds the idea of 'context-awareness' to the traditional computing model thereby enabling computing processes to sense their environment, react to the changes in the environment and adapt their behavior according to these changes [1, 2]. While the notion of context was initially restricted to mean the user location, several definitions of context have since been put forward by the research community. In this thesis, we choose to use the following definition of context as proposed in [3]

"*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves*."

The term 'information' in this definition can either refer to either the physical environment that includes the location, infrastructural details and physical conditions or it can denote the human factors such as the user information, social environment and the tasks carried out by a user. This context information is usually retrieved from different context sources. These sources can be of two types – physical (sensors, actuators) and virtual (software tools, programs). Context-aware systems and applications are designed to utilize this information to enhance the end-user experience by delivering the most relevant information and services as dictated by the current context. In order to illustrate this idea, consider the example of a personal shopping application residing in the phone of a person that has the knowledge of the user's preferences. A shopping mall is also assumed to be equipped with an application that contains the information about the current deals in the mall. When the user walks into the mall, the application can use the preference information and match it with the context information provided by the mall to notify the user about the deals on the products he's interested in. Similarly, the idea of context-awareness can also be utilized to develop life-support applications as part of the healthcare sector. Such applications can be designed to monitor the different vital signs (heartbeat, blood pressure etc) of an individual using the appropriate body sensors and take appropriate action in case of any fluctuations in them. These actions can involve informing a medical personnel and making an emergency call to an ambulance service. These are just some of the examples that demonstrate the usability of the notion of context-awareness across a wide spectrum of application domains. Depending on the application requirements, context-aware systems may need to utilize both local as well as remote context information to dictate the application behavior. Considering the example of the healthcare application discussed earlier, one of the actions that the application can take is to call for an ambulance in case of an emergency. In the ideal scenario, an ambulance that is close to the victims' location should be summoned (local context). However, in case a nearby ambulance is not available, the application should be able to locate another ambulance from the pool of ambulances available in other locations (remote context). This simple example illustrates the fact that applications can require to use context information produced locally as well as available beyond its spatial-proximity.

A number of context-aware systems have been proposed in order to support context-aware application development. Earlier versions of such systems provided a tight coupling between the application logic and the underlying context sources thereby leading to a vertical software structure which was rigid and difficult to reuse. In order to alleviate this problem, the recent focus has been on developing context-aware middleware [4-7] that provide an abstraction between the context sources and application logic. In spite of being the focus of intensive research efforts for nearly two decades, the idea of context-awareness has not yet taken off in a big way. The main reason behind this can be attributed to the fact that as technology has progressed, a number of new design requirements have arisen that need to be met to popularize the use of context-aware systems. One of these requirements is to increase the operating scope of these systems. Early context-aware systems were usually deployed within a small experimental environment in a lab or a university. Currently, there is a growing requirement for context-awareness to be available everywhere and any-time due to increased user mobility, availability of wireless sensors and global network connectivity. Such systems should also be able to look-up and locate the desirable context sources from potentially large number of heterogeneous sources. This calls for a suitable organizational technique that can establish an ordering amongst a set of diverse context sources thereby facilitating the task of identifying the context sources with the required information. This problem serves as the key issue being addressed as part of this thesis.

There are a number of other requirements that pose a challenge to the development of context-aware systems. Even though those problems are beyond the scope of this thesis, we still discuss them here for the sake of completeness. One of these challenges includes the acquisition and processing of context dynamically. Context data related to the physical conditions are acquired through heterogeneous physical devices and a standardized representation of these devices is needed to ensure interoperability [8]. However, acquiring context data related to human factors can be challenging due to the in-precision nature of wireless sensors and the non-intrusive requirement of data acquisition. An important aspect of context pertaining to the human factor is detecting determining the tasks and activities carried out by a person (i.e. reasoning) and use them to drive application behavior [9]. Also, as applications can be driven by context data changes occurring in different places, the context reasoning process needs to be distributed across all the involved context sources at different spatial proximities to make an informed decision. Context-aware systems should also possess suitable security and privacy mechanisms [4]. As certain context data like the health records of an individual are deemed confidential, the security mechanism should ensure that the dissemination of such data is restricted according to the credentials of the requestor. Finally, context-aware systems should provide the necessary software engineering tools for developers. These requirements are essential for a context-aware system to make their transition from a lab based experimental setup to a wide scale real world setting.

In recent times, the paradigm of the Internet of Things (IoT) has been the subject of widespread attention in the research community. Initially proposed in 1998 [10], the IoT computing model envisions a world where different objects are connected to the internet and that can communicate and collaborate with each other. These objects can refer to any of the following - people connected to the internet via social networks, conventional computing devices like desktops and laptops, and “smart” version of everyday devices. These devices can be phones, cars, refrigerators or even smart houses and offices. The objective of the IoT paradigm is to use these interconnected objects to create a working environment where these objects are aware of the user requirements and preferences and they are able to fulfill these requirements without explicit instructions. Although this idea might have seemed like a distant dream when it was first proposed, the emergence of affordable smart phones with highly evolved sensing capabilities and significant processing power has already started paving the way for this vision to become a reality. The idea of interconnected and communicating objects leads to the applicability of IoT across different application domains ranging from industry (supply chain management, transport and logistics, aviation etc.), environment (disaster management, agriculture, environmental monitoring etc.) and society (healthcare, telecommunication etc.)[11-15]. The use of different enabling technologies has been proposed in order to realize the IoT vision. The two most important of these technologies are – sensor networks and middleware systems. Sensor networks are integral to IoT as they perform the essential task of collecting and processing the data that is needed to drive the decision making process. The presence of middleware systems for IoT is required to facilitate the development of IoT applications by exempting the developers from the underlying details that are not the primary focus of application development. According to the survey conducted in [16], middleware systems for IoT need to possess the following functionalities – effective device management, interoperation, platform portability, context awareness and security/privacy mechanisms. An assessment of the leading IoT middleware solutions against these parameters in [16] reveals that most of them do not provide the functionality of context awareness. The importance of context-awareness within the IoT paradigm is related to the IoT vision of an environment where there are large number of sensors and other data sources generating huge volumes of data. In order to make this data more useful, it needs to be analyzed, reasoned, interpreted and understood. As context-aware computing deals with this challenge in the pervasive and mobile computing paradigms, it is expected to tackle this issue successfully within the IoT scenario as well. As a result, context-aware computing is being envisioned as an important enabling technology for IoT [17]. The design principles required to adapt context-aware systems to an IoT scenario are mostly in line with the requirements for context-aware systems as discussed previously. Some of the additional requirements that are introduced due to the properties of IoT are increased support for mobile devices and handling disruptions due to mobility, resource optimization in large scale networks and an extended and comprehensive context modeling technique [14, 17]. This association of context-aware systems with the IoT scenario can prove to be instrumental for context-aware systems to make the transition from lab based deployments to a large scale real world setting.

1.2 Data management in context-aware systems

As the primary function of context-aware systems is to react to the changes in context, one of the main requirements of these systems is to manage context data and provide reliable access to the relevant data with minimal delays. Therefore, it is important for such systems to have an effective context data management system. The essential functions of a context data management system are – acquiring context data, processing the acquired data to generate higher order context information, storing the context data and supporting context lookup operations over the data. Context lookup can be defined as the process of identifying the context sources holding the required data and the retrieval of the data from these sources. This operation is usually carried out using queries containing the list of data items to be retrieved along with the constraints and conditions to filter the data. To cite an example, consider the example of the shopping application discussed earlier. A typical context lookup request for that application could be to find the set of shops stocking a certain type of product within a 5 km radius from the user's current location. The constraints on the requested context information (the shops) in this case are the type of products carried by the shops and the distance of the shops from the user. The usefulness of the application will depend on the timely delivery of the correct context information. Another strong justification for this requirement is the fact that context-awareness is being proposed as an enabling technology for a large scale system like IoT, the task of locating context data from the large number of sensors and other smart objects efficiently will become absolutely crucial for the functioning of the system.

An efficient context data management system is important in a context-aware system as it is an important prerequisite for most of the other system components. In the previous section we discussed some of the new system requirements that have emerged for context-aware systems. As part of our ongoing research project, we are developing the Coalition middleware [18-20] to develop a context-aware system to meet all these requirements satisfactorily. The research problems being addressed in this project are related to five main functional requirements of context-aware systems and each of them represents a different aspect of the system design. These requirements include the following:

1. **Context data management** – This component needs to manage the context data from the different sources and support the lookup operations over the context data.
2. **Context data aggregation** – As applications can specify conditions that require data in a summarized form, appropriate data aggregation mechanisms need to be put in place.
3. **Context reasoning** – The main aim of a reasoning module is to deduce relevant information that is of importance to users and applications by making use of the currently available context data
4. **Context data security and privacy** – The security and privacy subsystem provides a basic authentication and access control mechanism leaving applications free to tailor their own security requirements and inform it to the system.
5. **Programming support for developers** – As the Coalition middleware is intended to support context-aware application development, adequate programming support in the form of APIs and interfaces need to be developed.

We now examine the role played by the data management component in the aforementioned system design issues. The context reasoning component is dependent on the context lookup process as it relies on context data retrieved from one or more sources in order to infer knowledge about a situation. In case the context lookup process is erroneous, it can have a serious bearing on the reasoning process especially if they are associated with mission or life critical applications. Also, since the reasoning component may need to work with data distributed across multiple context sources, the context lookup must be able to locate the relevant context information from all the context sources. A similar argument holds true for the context data aggregation component as well as it needs to summarize context data belonging to different context sources. It is the responsibility of the context lookup process important to ensure that the data to be aggregated is delivered in time and accurate. The relationship of the context data management component with the security and privacy subsystem may not be too obvious. As per our middleware design scheme, only a basic security and authentication mechanism is provided by the system while allowing applications to design their own security/privacy mechanisms. The application specific security/privacy mechanisms usually make access control decisions based on the context information of the data requestor. For example, the healthcare records of a person should only be accessed by his authorized doctors. If a lookup request for this data is received, we need to retrieve the context information of the requestor to allow or deny the request. As far as the programming support aspect is concerned, there is no direct role of the data management component in it and is limited to redirecting the data retrieval requests to the data management system while designing the APIs. This discussion clearly highlights the importance of an efficient context data management system within a context-aware system. As we shall see in detail in Chapter 2, the design of a context data management system and especially the context lookup operation incorporates a lot of challenging issues. The key issues to be dealt with in this thesis are how to handle the look up of context data in large search spaces as well as to handle the dynamicity of the context data. We outline and elaborate the key issues as follows:

1. The system should ensure that the context lookup process is scalable with respect to the query response time and a classification mechanism like an index is present to partition the context sources.
2. As context data is usually dynamic in nature, the index should be capable of working with large volumes of dynamic data and handle the associated update overhead efficiently.
3. The system should be able to handle the variations in the query requirements depending on the data types. Also, the system should be able to support multiple query scopes to meet different application requirements.

1.3 Motivation

The previous section clearly establishes the fact that the design and development of a context data management sub-system is critical for the successful operations of context-aware systems and is a non-trivial task. As part of the research efforts focused towards developing context-aware systems, the problem of managing and supporting lookups over context data has been addressed using different strategies by the research community. We will discuss these systems and their associated features in detail as part of our literature survey in Chapter 2 but we briefly highlight the key features of the data management techniques of these systems and their shortcomings as follows. One of the initial approaches adopted for context lookup was the direct retrieval of the required data from the corresponding sensors [21-23]. This approach is easy to implement but as we shall observe in Section 2.2, the retrieval of data from multiple sensors for every lookup request may lead to higher delay due to transmission delay and complexity in processing of raw context data. Another class of context-aware systems relies on using relational database systems to create a context data repository that stores context information [24-26]. The lookups are now carried out by querying this repository. These systems assume that the problem of acquiring, storing, indexing and updating dynamic context data can be handled by databases but there are no experimental results provided to support this claim [27, 28]. Another class of context-aware systems disseminates lookup requests to only the context sources that exist within a fixed area to minimize the query scope [29-31]. The usefulness of this approach is restricted to applications that require data just from nearby sources. The usefulness of this approach is restricted to applications that require data just from sources of the designated area. For instance, anytime-anywhere applications that require context data from a wider spatial scope will not be able to benefit from this type of organizational technique. In order to enable relational databases to store context information, a number of techniques have been proposed that augment traditional databases with context-aware heuristics [27, 32, 33]. These techniques mostly focus on the formal specification of a context model and the associated query language. The low level details of actually acquiring context data for the database are not discussed in these techniques. Some of these techniques acknowledge the dynamicity of context data as part of their design requirements but the implication that this property can cause an index update overhead is not considered. Existing context data management strategies look at the context lookup operation from a higher level of abstraction in which the factors that constitute the query context are given more importance than the low level data handling aspect. This brief discussion highlights the fact that existing context-aware systems do not satisfactorily address all the issues associated with context data management and lookup. Especially, the critical issue of managing dynamic context data and minimizing the overhead due to the data dynamicity is not addressed by any of the existing context data management techniques. This serves as the primary motivation for our research project as described in this thesis.

1.4 Problem statement and research objectives

In this thesis, we address the problem of designing and developing a context data management system to manage context data as well as provide an indexing scheme to support lookups efficiently over pervasive context data. These works are developed as an extension of the data management component of the Coalition middleware system. We can formally define the goal of this thesis with the following problem statement:

To develop an efficient and reliable context data management system capable of managing and supporting lookups over different types of context information distributed across multiple spatial proximities (also known as physical spaces).

As part of our objectives to achieve this goal, we try to ensure that the three key design issues associated with managing context data as outlined in page 8 are handled satisfactorily as part of the proposed system. In brief, one of the main focuses of the thesis work is to develop an indexing scheme that can address the index update overhead problem associated with dynamic data as well as classify the context sources according to their data values. As we shall observe in Section 2.1, the use of conventional database indexes with dynamic data leads to frequent updating of the index structure leading to the unavailability of the index during the update periods. Further, as a context-aware system operates in a dynamic environment, we need to ensure that the index structure be able to adapt itself as context sources leave and join the system and adapt its structure accordingly to reflect the data distribution accurately.

Another important issue that we aim to address as part of this thesis is the scalability concerns arising due to a large number of context sources. In the absence of a suitable organizational structure, the response time required for retrieving the required context data can increase rapidly with the increase in the number of context data sources. Another challenge that we aim to address as part of this thesis is to make provisions for the different query requirements for different context data types (such as numeric and strings) and the need to tailor the indexing and lookup schemes accordingly. This issue will be discussed in detail in Section 2.1. The contributions of this thesis can be outlined as follows:

1. **Indexing context data using range clusters** – In the first part of the thesis, we propose a range clustering technique to partition the context sources into a set of clusters according to their data values. This is a preliminary solution to establish an ordering among the context sources. This range clustering technique is integrated with the Coalition middleware and experiments are conducted to compare the response times for processing queries using the proposed scheme compared to the flooding approach. The experimental results indicate that proposed index succeeds in minimizing the response time for queries by reducing the search space for a query as compared to the flooding. However, it is also observed that the index is not equipped to handle the problem of managing dynamic data satisfactorily and can lead to errors in the query processing operation.
2. **Mean-Variance based Indexing scheme for dynamic context data –** The second part of the thesis addresses the problem of dynamic data management using a mean-variance based indexing technique. This is an extension of the range clustering approach which utilizes the statistical properties of numeric data to design an index that can handle the update overhead due to the dynamicity of data. The dynamicity of data in this case refers to the fact that certain context attributes keep changing in value frequently. This indexing scheme is primarily designed for handling numeric data. A set of experiments is conducted to evaluate the index performance using a set of dynamic data values. These experiments assess the index performance based on different parameters that include the query response time, the number of answers received for a given query the number of index updates occurring during a given period as well as the variations in the system performance with the change in the dynamicity of data. The results indicate that the index performance is satisfactory especially with respect to the query response time as well as handling the update overhead due to dynamic data.
3. **An incremental tree based string index structure –** Since the use of the mean-variance index structure is restricted to numeric values, we propose an index structure for string attributes using the concept of radix sort and ternary

search trees. We also use the idea of longest common prefix to improve the indexing process by identifying a set of strings sharing a large common prefix. The index structure provides a grouping amongst the context sources according to the shared common prefixes of their string attribute values. The length of the shared prefix is not predefined and is varied according to the data being indexed. This index structure is designed to support exact matching, prefix search and range queries on string attributes. The performance of the index is evaluated through a set of experiments that evaluate the variation of query response time for different network sizes, the handling of dynamic strings and the variations in the index size with the number of strings. The experimental results indicated that the index structure was able to handle queries efficiently over different network sizes as well as small amounts of dynamism. Also, the variations in the index size were observed to be slow with respect to the number of strings being indexed as well as the length of these strings.

1.5 Thesis outline

The remainder of the thesis is organized as follows – In Chapter 2, we discuss the existing work in the field of context data management systems and assess them critically using the design issues identified previously. This is followed by an overview of the Coalition middleware system in Chapter 3 where we establish the need to provide an efficient data management system from the system perspective. Chapter 4 describes the initial range clustering based indexing scheme that is proposed to simplify the context lookup problem. Since the initial indexing scheme still has some drawbacks especially when the context data is dynamic, we discuss a mean-variance based indexing scheme in Chapter 5. This indexing scheme is designed to work with dynamic context data and provide a scalable lookup mechanism over such data. Since the indexing scheme discussed in Chapter 5 is primarily geared towards handling numeric data, we discuss an incremental tree based indexing approach for string context attributes in Chapter 6. Chapter 7 discusses the future direction of R&D for the proposed data management system and concludes this thesis.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, we discuss and critically review the current approaches adopted towards solving the problem of context data management. We assess the existing context data management strategies within the context of the functional requirements for such systems as identified in the Introduction section of Chapter 1. The chapter is organized as follows: Section 2.1 presents the design requirements for a context data management system. We discuss the current approaches of managing and querying context data and identify the gaps in the current methods in Section 2.2. We conclude this chapter in Section 2.3 by summarizing the observations from the survey of the related work and justify the motivation for the work carried out in this thesis.

2.1 Design requirements for context data management systems

Before we review the existing work carried out in the field of context data management, the design requirements for context data management systems need to be examined. As we shall see in Section 3.2, our proposed middleware delegates the tasks related to context data generation, processing and storage to the context sources. Hence, we focus on the context lookup aspect as our main design concern for a context data management system. The simplest form of a context source is the raw context data source, such as a smart sensor or a more complex smart object. However, the context sources referred in this thesis have a broader functional scope. As we shall observe in Section 3.2.1, we use the concept of a physical space gateway (PSG) to represent a set of related raw context sources as a single aggregated context data source. The processed outcome of context data is known as 'context information', which is a kind of 'higher level' context data. Unless otherwise stated, the use of the term 'context sources' refers to a collection of aggregated context sources and treat context information as context data from hereon. We focus on the context lookup aspect among large number of PSGs as our main design concern for a context data management system within the environment of the Coalition system.

An important requirement associated with the context lookup operation is to ensure that the variations in the query response time are minimal as the number of context sources is increased. The response time for a query depends on the capability of the system to identify the context sources relevant to a query among all the context sources. Clearly, the data management system needs to have a classification scheme like an index. However, contrary to the design requirements of conventional database indexes, an index for a context-aware system should be able to manage dynamic data as context information can be static or dynamic. Static context data usually indicates information that changes infrequently. Comparatively, dynamic context data changes asynchronously and frequently. There are several examples of dynamic context data including the current location of a moving object, the crowd level of a locality, atmospheric conditions at a given place and the vital physiological signs of a person.

As context aware applications operate by reacting to context changes, the management of dynamic context data is essential. The problem with using a traditional database index with dynamic data is the high update overhead involved. The update overhead depends on the number of indexed data items as well as the update frequency. Since the index will need to be updated and rebuilt each time there is a change in the indexed data, the overhead can become significant if all the indexed data items keep changing frequently. Since a database index is usually unavailable during the index rebuild, frequent updates can lead to delay in processing queries and become a major performance bottleneck. The frequent updates also contribute to large amounts of network traffic which can also cause a drop in the system performance. Hence, it is essential to ensure that an index structure designed for context data is able to handle the dynamicity of the data and the number of index update operations is minimized. Since a context-aware system is expected to operate in a dynamic environment where context sources can appear and leave randomly, this index should be built incrementally and adapt itself to the movement of the context sources.

As context-aware applications usually require that the data being retrieved is up to date, it is important to ensure that a context data management system delivers the most recent copy of the data to the applications. The consequences of not receiving fresh data can be severe especially for applications that are life or mission critical. This issue is related to the previously discussed problem of updating an index when the associated data is dynamic. If the change in the value of a context data item is not reflected promptly in the index, there might be inconsistencies between the received and current data value. The system should also ensure that the effect of the mobility of context data sources causes minimal disturbance to the data retrieval process. Context sources like houses, shops, offices are static i.e. they do not shift locations frequently and retrieving data from them is straightforward. However, with the advent of wearable body sensors, smart phones and smart cars, there is an increase in mobile context sources like transporters and robots, in addition to people. This mobility can cause a problem with context data retrieval as the network connections are now expected to be intermittent which can lead to incomplete data retrievals. The system should make adequate provisions for handling the mobility and resuming an interrupted data retrieval process when the network connectivity is available again.

As context-aware systems are expected to deal with a wide variety of context information, the data management system should be able to handle different data types and data representation formats. Also, as the query requirements can differ according to the data type, the system should be able to support a diverse set of queries. The basic query types for numeric context attributes are exact matching and range queries. As far as string attributes are concerned, the standard query types are – exact match, prefix/suffix match, wildcard match and range queries. The design of an indexing and organization scheme for context data should take these query requirements into account. Also, the notion of context required by an application can differ according to its specific requirements. For example, a personal shopping application might need to find all the shops in the vicinity of the user having a product within a certain price range. On the other hand, an application advertising the best deals in a city will require a list of all shops (irrespective of location) that have that same product with a particular price. In the first case, a location based classification of the context sources will be useful whereas a classification based on the product type as well as the price range will enable the processing of the query faster in the second case. This example illustrates that for the same set of context spaces (the shops in this case) different applications define the scope of their context data requirements using different perspectives. An effective context data management system should be able to support multiple query scopes as part of its lookup mechanism. The scope for a context data query request can either refer to a system wide scope (the queries are processed against all the context sources in the system to retrieve all the matching data) or a selective scope (the queries are processed against a subset of context sources) [8]. In the previously discussed example, the first scenario refers to a limited scope query (only shops close to the user are queried) whereas the second scenario refers to a query with a system wide scope. All further references to the term ‘query scope’ in this thesis shall refer to these two query models. These design requirements will be utilized as guidelines to assess the existing context data management strategies in Section 2.2.

In order to study the different approaches adopted towards designing context data management systems, we use the data storage architecture adopted by these approaches as a primary criteria to classify them. The main techniques used to store and retrieve context data include the use of a single centralized repository, a distributed storage model comprising of multiple repositories or storing the data at the individual context sources. Apart from the data storage model, another criteria that we use to classify these data management systems is based on the context data retrieval and indexing strategies used by them. These indexing strategies refer to the techniques utilized by the data management systems to establish an ordering amongst the context sources to facilitate the propagation of context queries. For example, context sources can be ordered according to the type as well as the value of the context data provided by them. Additionally, context sources can also be ordered according to their physical proximity relative to each other. The technique used to establish the ordering amongst the context sources by a given context data management system will also have an impact on the type of queries that can be supported by that system.

2.2 Review of data management in context-aware systems

In this section, we review some of the existing strategies adopted towards managing context data especially with regard to the context lookup process. The context lookup operation comprises of acquiring the context data from the relevant context sources for a query. In the initial part of the survey, we classify the context data management strategies based on their approach towards storing and querying context data. These include direct data access from the raw context sources as well as the middleware based approaches that utilize data repositories. The later part of the survey focusses on the different techniques used by the context-aware systems to establish an ordering among the context sources in order to process queries efficiently. We also discuss some of the techniques that discuss the augmentation of relational database systems with context-aware heuristics thereby enabling them to store and process queries on context data.

One of the approaches proposed to carry out the process of context lookup is the direct data retrieval strategy in which the relevant data are retrieved directly from the individual raw context sources. The Cooltown project [21] introduces a software layer to integrate the physical environment with the Web and uses web servers to directly access the data stored in sensors. The COSINE framework [22] also utilizes a web-service based architecture and provides separate web-services to access data from sensors, aggregate sensor data and manage the availability of context services. The RCSM middleware [34] handles data acquisition from both local as well as remote sensors. A context discovery protocol is utilized to search for remote sensors and all data is directly retrieved from the sensors. The problem of context lookup is handled using different strategies that include the publish/subscribe model, XML/XPath queries and web service based data retrieval.

The lookup strategies are usually query driven to minimize the power load on the sensors. The direct data retrieval strategy is straightforward and the freshness of the received data is always ensured. Also, since the sensor data generation and storage is handled locally, these schemes do not need to handle the problem of dynamic data management. However, since sensor data usually needs to be processed before use, a query requesting multiple attribute values can affect the query response time as multiple data items need to be retrieved from the individual sensors and processed. These schemes usually utilize an entity model to represent collections of different correlated sensor and the classification among the context sources is based on the different entity types modeled by the system and the types of information supplied by them. These schemes do not provide any ordering on the actual data values of the attributes which makes the processing of queries having constraints on the data values difficult.

As retrieving data from individual sensors can prove to be challenging to programmers, context-aware middleware usually try to keep the data acquisition process transparent to the applications. These systems usually abstract collections of related context sources as a single entity. For example, a set of sensors providing information about a house are visualized as a collective entity representing the house. We define these entities as context spaces. The CASS middleware [35] which is primarily designed to support context-awareness for mobile devices uses a centralized data repository to store the context information and uses database queries to carry out the context lookup. The Contory middleware [24] also utilizes a central repository to store and query context data retrieved from infrastructure based systems. The CAMUS middleware system [25] uses two separate data repositories to store the current context data and the historical context data. The rationale behind using two repositories is to improve the query processing performance by keeping the current context data separate from the historical data.

The C-CAST context management framework [36] uses the concept of context providers and consumers to differentiate between the context data sources and the applications interested in the data. The context providers derive context information from sensors and actuators and store them in a context repository. The interaction between the providers and consumers is mediated using a set of context brokers that maintain a directory of the context providers. The SOCAM middleware [18] also uses a centralized context database and an ontology based model to represent context entities in different domains and to model context data. The context data retrieval operation supports direct context queries as well as event subscriptions. The context lookup strategies for these systems involve the context data from different context sources being stored in the repository where it is available for retrieval by the interested applications/consumers. The context data providers periodically update the repository either directly or using an intermediary like a context broker. The classification of the context information is usually based on the value, type and scope of the context information. As opposed to the use of a centralized context data repository, the Nexus middleware system [37] uses a distributed data storage system where each data storage server stores a specific class of context data and caters to the specific requirements of that class. The different classes of context data are obtained by classifying context data items according to the following factors – the update rate of the data item and the importance of a data item as a selection criterion. As part of the system design, the following classes of data storage servers are identified – location data, static spatial data, indoor spatial data, embedded system data, data from a smart home and dynamic sensor data. The last class of storage server is intended to store and process queries on frequently changing sensor data values. Although the requirements for designing a storage system for dynamic data is discussed in detail that include the need to minimize the index update overhead, the actual implementation details as well as the mechanisms used to manage dynamic data are not mentioned.

Although the technique of using context data repositories to store and query context data as discussed in the previous systems is easy to implement, there are a number of problems associated with this approach. Since a context aware system will need to manage a large number of context sources each having multiple attributes, the database will need to store and manage an enormous amount of data. This cost can be deferred to a certain extent by using a collection of repositories instead of a centralized one [37]. Additionally, the storage and indexing of dynamic context data poses a significant challenge in terms of large update costs. This issue is not addressed by any of these techniques explicitly. Although the data management system discussed in [37] acknowledges the special requirements of having a separate storage system for dynamic data, the actual problem of designing such a system is not addressed in detail. The use of a data repository will also affect the network load as large amounts of data will need to be transferred in order to store them in the repositories. Also, the raw data will need to be processed before it can be stored in the databases which can again cause an overload at the central repository due to the large volumes of data that need to be preprocessed. The freshness of the context data being retrieved is also an important concern as these schemes rely on the periodic update of the repositories to refresh the data. This can lead to obsolete data being read in the middle of the update cycles. One of the main aims of this thesis is to devise an organization scheme that avoids these problems associated with managing large volumes of dynamic data in a centralized location.

Context-aware systems that do not use a centralized data repository utilize different techniques to organize the context spaces and define the scope for a context lookup request. This section of the survey discusses a set of context data management strategies classified according to their strategies for organizing the context sources. One of these approaches includes a classification based on the type of context information provided by a context source. The COPAL middleware [38] uses the concept of context types to represent the context information and a broker based approach similar to the one described in [36] to mediate the context lookup process where a context type corresponds to a unique name and a set of attributes. The context providers register with the broker along with their context types and the delivery of context data is done through events using the publish/subscribe model. The idea of using brokers to mediate the context lookup process is also discussed in [39]. This system uses an overlay network of distributed brokers to manage the context consumers and providers. The context providers register with the broker using an attribute-value and the context lookup operation is carried out using direct query invocations of queries as well as the publish/subscribe event notification model. The Solar middleware [40] provides a programming model with a set of operators that can be utilized by applications to customize the use of context data. It uses an overlay of hosts called Planets that manage these operators and handle the context data dissemination process. The context data lookup is done using a publish/subscribe model and the context data distribution is carried out using application-level multicast trees where each Planet refines data according to the application requirements. The Coalition middleware [18, 20, 41] classifies context sources belonging to a particular domain class together and uses p2p networks to connect context sources sharing a common attribute type within the particular domain. The context data is locally stored at each context source and the context lookup is done via SQL queries by flooding the p2p networks. The Coalition system forms the basis of the work carried out in this thesis and will be discussed in detail in Chapter 3.

An important observation that can be made here is that context-aware systems without a data repository don’t need to address the problem of indexing dynamic data as the data is stored and managed locally at the context sources. However, this same reason prevents the creation of an organizational structure on the attribute values. Since there is no value based classification, queries looking for context data within a certain range of values or having a particular value cannot be processed efficiently. Further, the context lookup is usually restricted to publish/subscribe based event notification model. Although event notification is an important aspect of context-aware applications, it usually requires an application to be aware of the context sources involved in an event. In this thesis, we are more interested in solving the problem of locating the context sources that have the relevant data for a context query as per the constraints mentioned in the query. Since these constraints are evaluated on the values of the context attributes and should be addressed in the design of the query processing language, we focus on developing an indexing scheme that can establish an ordering amongst the context sources according to their data values.

A location based organization is also a popular technique used by context-aware systems to limit the scope of context queries. The Pervaho middleware [29] uses a location based publish/subscribe service to organize the context sources and carry out the context lookup process. Each context publisher and subscriber is associated with a location area scope and an event is delivered from the publisher to a subscriber only if both their location scopes intersect. A similar approach to limit a query scope is used in the CORTEX system [42] as well. The SALES middleware [30] uses a combination of data repositories and a locality principle to distribute context data only to devices in a physical/logical vicinity, thereby attempting to reduce the context data traffic. The system has a hierarchical structure where the top level contains a centralized repository for historical context data storage and the lower level nodes comprise of the context sources and the manager modules. The EgoSpaces system [31] uses the concept of views to represent and query context data together with a locality based organization. The context data is stored in the form of tuples containing the information about a context attribute. Each context provider contains a set of tuples that describe its properties and a view is used to specify the query constraints. The context lookup is done through agents that disseminate the query among providers in its immediate physical locality. The use of the physical location to limit the query scope is also discussed with different variations in [43-51].

The use of a location scope to limit query scopes for context aware systems seems to be intuitive as context-aware applications are usually interested in events happening in their vicinity. However, this organization is only useful to applications requiring location aware information. As we observed in Section 2.1, depending on the application requirement, a query may need to be processed against the entire system scope in which case a location based organization will not be useful. Further, as location is the only classification constraint used in these systems, the processing of queries having constraints on the data values will be inefficient. As part of our initial thesis work, we consider the problem of optimizing the processing of queries with data value based constraints across a system wide scope. Since it is important for a middleware to support multiple query scopes, we discuss the design of a multi-level data organization scheme in the later part of the thesis to solve this problem.

A different class of context data management techniques proposes the augmentation of existing relational database systems with certain heuristics to make them “context-aware”. These context-aware databases can now be used to store context data. The X-RAY scheme [27] is based on the assumption that context data is stored using XML schemata. A generic mapping scheme is proposed that can map XML schemata to relational schemata and enable the storage of context data in a relational database. The problem of handling dynamic data updates is addressed as part of the requirements of managing context data. However, it is assumed that a database is equipped to handle concurrent and high frequency updates using transaction processing mechanisms but no experimental data is provided to support this claim. A context relational model (CR model) is discussed in [52] which uses the concept of a multi-facet entity that is an information entity having different facets when viewed from different perspectives. This idea is used to build a context-relation cube that forms the basic unit of storage in a context-relational database. The main focus is on the formal specification of a context data model and the indexing part is not addressed. A context query language designed as an extension of a SQL based language is described in [53] that proposes a set of predicates to add ‘context-awareness’ into the query language itself. A similar approach is discussed in [54] which uses the notion of dimensions to represent context and formulates a query language based on this idea. An alternative approach to devise a context-query language using ontology is discussed in [55] that identifies a set of requirements for querying context information and presents a query language designed to support them. The dynamicity of context data is described as part of the requirements and the implication of this issue is taken to be the fact that there will be a set of historical values that can be queried. The support for dynamic data retrieval is restricted to processing queries over historical data.

These techniques focus on the development of a formal query language and extending the conventional relational algebra model with context specific operators. They look at the problem of context data acquisition from the query issuers' perspective which is quite an important issue in its own right. However, these techniques do not consider the problems associated with the actual process of the data retrieval from the databases and consequently do not address the problems associated with managing dynamic data. The management of dynamic context data is one of the important issues that we aim to address as part of this thesis. An important point to be noted here is that although the existing context-aware systems do not handle the problem of managing dynamic data satisfactorily, this problem has been independently addressed as part of the research work in several other application domains that include dynamic documents, spatial data management systems, sensor networks etc. We review some of these techniques separately in Chapter 5 and evaluate their applicability for indexing context data while satisfying the design and organizational constraints of our context-aware middleware system.

2.3 Summary

As is clear from the preceding survey, the problem of context data management and providing appropriate context lookup mechanisms has emerged as a significant research challenge given the rising importance of the context-aware computing paradigm. We summarize the observations made in the previous section in Table 1 which evaluates the surveyed approaches using the following criterion – context data storage technique, context lookup technique, and support for dynamic data management.

Table 1. Summary of the surveyed approaches

System	Data Storage	Lookup technique	Dynamic data management
[21, 22, 34]	Data stored at context source and classified according to the data type	Direct access from sensors using web services and XML queries	Not applicable
[24, 35]	Centralized context data repository	Querying the repository using database queries	Not handled
[25]	Distributed set of context data repositories. Historical data stored separately	Querying the repository using database queries	Not handled
[36]	Centralized context data repository according to data type and context scope	Querying mediated by context broker, direct queries and event notifications	Not handled
[37]	Distributed data storage servers with each server tailored to a specific class of context data	Queries issued to storage servers and processed using access paths	Separate server provided for dynamic data storage. The implications of storing dynamic data not discussed
[38, 39]	Data stored at publishers as well as cached in a repository	Broker mediated publish/subscribe queries	Cached data updated periodically
[29, 40, 42-51]	Data stored at context provider	Location scope restricted publish/subscribe model	Not handled
[30]	Data stored locally as well as in a persistent centralized repository for rapid access	Location scope and multi-level organization to route queries, ad hoc networks used for query propagation	Not handled
[31]	Data stored at context provider in tuple form	Views used to restrict query scope along with physical locality	Not applicable

[41]	Data stored locally and represented using ontology model	Direct queries and event subscriptions	Not applicable
[18, 20]	Data stored at context provider and grouped using domain/attribute combination	Direct queries and event subscriptions	Not applicable

Based on the information in the table, we highlight the current trends and observations regarding the design of a context data management system as follows

1. The direct retrieval of the context data from the individual raw context sources ensures the freshness of the retrieved data. This technique has the drawback of being inefficient for queries requiring values from multiple context sources as each source needs to be individually contacted that can have an impact on the response time. Also, the preprocessing required for sensor data can result in the increase in response time for a query.
2. A popular technique to store and query context data is to use a centralized data repository for persistent data storage. However, the problem of managing the update overhead due to the dynamic nature of context data is not addressed satisfactorily by any of these schemes. Also, these schemes do not address the concern of data freshness as they rely on periodic updates to update the context data. This can result in obsolete data values being read by applications in the middle of update cycles.
3. Although the problem of managing dynamic data is discussed by some of the schemes as a design requirement [27, 55], it is either considered to be handled by the underlying database system or the implications are taken to be the support of queries over historical data. There are no experimental results provided to support the claims of the database being able to handle the dynamic data update overhead.
4. Context-aware systems that do not use a data repository do not need to handle the problem of managing dynamic data. On the other hand, these schemes

cannot provide a classification on the context data values which restricts the processing of queries having conditional constraints on the values of the data. Since context queries can be effectively translated to some kind of range or point queries, it is important to optimize the processing of these queries.

5. The context lookup operation in context-aware systems is carried out using different approaches that include directly invoking queries on the context sources, querying context information stored in repositories, event based publish/subscribe notification and location based filtering of queries. The type of queries that can be supported by a context-aware system depends on the technique used by that system to establish an ordering amongst the context sources. Since applications may need to use different query scopes based on their specific requirements, a context-aware system should be able to support multiple application scopes.

Based on these observations as well as the summary of the surveyed approaches in Table 1, we can conclude that the problem of handling the update overhead as well as the freshness concerns associated with managing and updating large volumes of dynamic context data is not addressed satisfactorily by any of the existing schemes. Further, existing systems usually provide a restricted querying model and are applicable to a particular class of applications. Clearly, there is a lack of a context data management system that is capable of managing large volumes of dynamic context data and handling different types of data and data representations formats. Also, as we observed in Section 2.1, the query requirements vary according to the data type and a context data management system should support different types of queries including exact/range queries for numeric attributes and prefix/suffix/keyword matching queries for strings. As part of the survey, it was also observed that existing context data management systems usually support a single query scope. The importance of supporting different query scopes in Section 2.1. The fact that the current context data management systems do not handle these requirements effectively serves as the primary motivation for the research work carried out in this thesis. As we shall see in the subsequent chapters, we will use the observations and conclusions gleaned as part of the literature survey to serve as the initial design guidelines for the development of our proposed context data management system.

CHAPTER 3

COALITION SYSTEM OVERVIEW

In this chapter, we present an overview of the Coalition system. Coalition is a context aware middleware that has been developed as part of our ongoing research project. Since the work carried out as in this thesis is developed as an extension of the Coalition system, it is important to have a clear understanding of the system. The chapter is organized as follows: Section 3.1 describes the design philosophy and guidelines followed in the development of the Coalition middleware system. We discuss the system architecture along with the details of the context data management layer in Section 3.2. Section 3.3 describes the details of the context data retrieval operation in Coalition together with its limitations and provides the motivation for the thesis work. The chapter is summarized in Section 3.4.

3.1 Design philosophy and guidelines

Coalition [18, 20] is a context-aware middleware being developed as part of our ongoing research project to manage and process context information pertaining to context sources distributed across large scale networks. We outlined the research problems being addressed as part of this project in Chapter 1. Prior to the discussion of the Coalition system architecture, we briefly highlight some of the important design guidelines utilized as part of the development process of the Coalition middleware as follows:

1. **Supporting scalable context data acquisition and processing** – Based on our discussion about the features of context-aware systems in Section 1.2, we aim to develop a system that can manage, process and disseminate context data across context sources distributed using a wide area network. Since the number of context sources can be large, these tasks must be highly scalable. Also, in order to prevent performance bottlenecks during the context processing (i.e. reasoning), the data processing functions should be distributed. The system should also make provisions for the mobility of context sources and the implications of the same on the data retrieval process. In order to make the data retrieval process uniform, a consistent high level naming of the different types of context information should be adopted. Since every context source can refer to its local context data using different terminologies, it is important to standardize the naming at the high level. As we shall see in the subsequent sections, the Coalition middleware utilizes a schema matching algorithm to achieve this requirement.
2. **Moving the context processing task closer to the context sources** – The majority of the processing operations are delegated to the individual aggregated context sources (PSGs in Coalition) as a way to decentralize the context processing operation. This approach spreads the processing load to distributed components of the middleware and eliminates the performance bottleneck of the centralized data depository and processing approach as discussed in Section 2.3. As mentioned before in design of Coalition, we use the concepts of a physical space (i.e. spatial proximity) and a physical space gateway (PSG) for each physical space to act as an aggregated context source

for acquiring, aggregating, storing and reasoning context data available within that space. Also, the processing outcomes of physical spaces may further be aggregated and/or processed by using a configurable context data aggregation method, that involves the desirable aggregated context sources interoperable to each other to further perform inter physical-space context exchange and processing.

3. **Service oriented design approach** – Since the Coalition middleware is intended to support the development of context-aware applications, we follow a service oriented approach to design the system. The different functionalities of the middleware viz. data retrieval, reasoning and aggregation are exposed as middleware services. Application developers can integrate these services into their application design.

In this thesis, we mainly focus on the problem of ensuring the efficient retrieval of context data from multiple aggregated context sources. As established in Section 1.2, the performance of the context data retrieval process has a direct bearing on the performance of the other data related operations like aggregation and reasoning. We discuss the system architecture of the Coalition in Section 3.2.1 and the details of the data management system in Section 3.2.2.

3.2 Coalition System Overview

3.2.1 System architecture

The high level system architecture of the Coalition system is illustrated in Figure 1. As is evident from the figure, Coalition has a layered architecture. The uppermost layer is the application layer that contains context aware applications. The application layer is followed by the service management layer. We follow a service oriented system design approach wherein the middleware functionalities are exposed as services. This layer handles the tasks related to service registration and discovery, as well as supporting orchestration of different services for application integration [56]. These services usually perform context-aware tasks involving context data retrieval, context processing, reasoning and aggregation. Also, a set of specialized services can be defined that provides the functionalities that are commonly required by a wide class of applications. An example of such services can be a location service that returns the current location of an object.

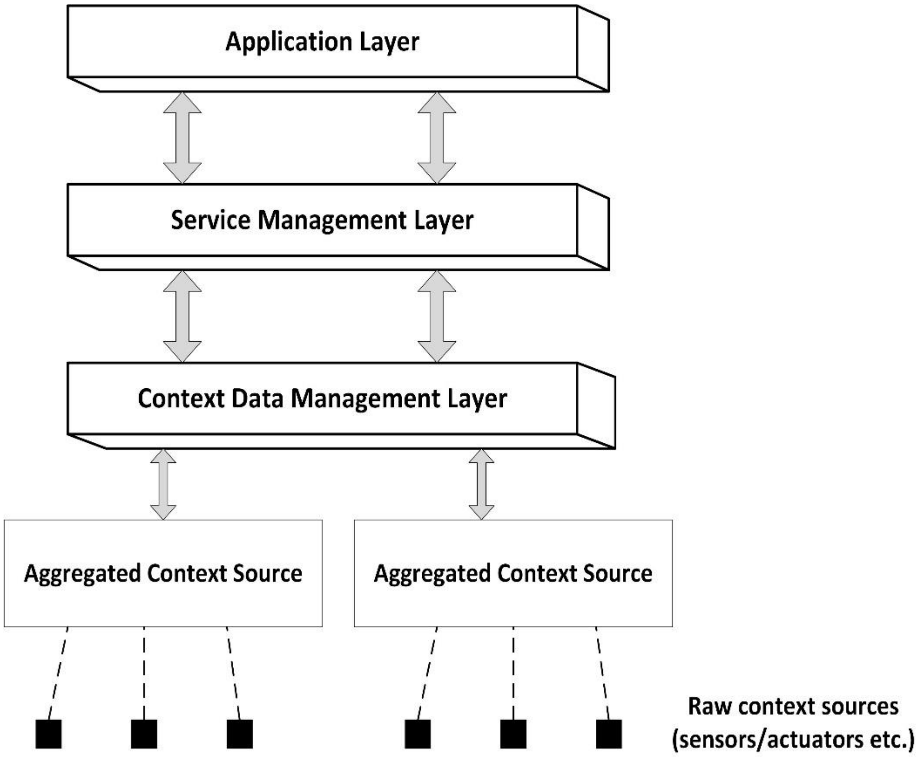


Figure 1. Coalition System architecture

The layer following the service management layer is the context data management layer which is responsible for managing all the operations related to the retrieval and processing of context data. The individual aggregated context sources interact with this layer and it provides an organizational scheme to group these context sources and the context information provided by them. In this thesis, we shall be focusing on improving the organizational scheme in the context data management layer to facilitate the faster lookup of the relevant context information.

3.2.2 Coalition – Context data management layer

Prior to the discussion about the organizational structure of the context data management layer, it is important to discuss the context model used in Coalition to represent and model the different context sources and the types of context information provided by them. As part of our system design process, we utilize the concepts of *physical spaces* and *context domains* to provide an abstract representation of context data. A physical space is defined as a person, place or object in the real or virtual world and can encompass multiple raw context sources that provide information that describes that particular space. For example, a physical space like a home can contain light, temperature and humidity sensors. Figure 2 provides an illustration of the concept of a physical space using the examples of a person and a house.

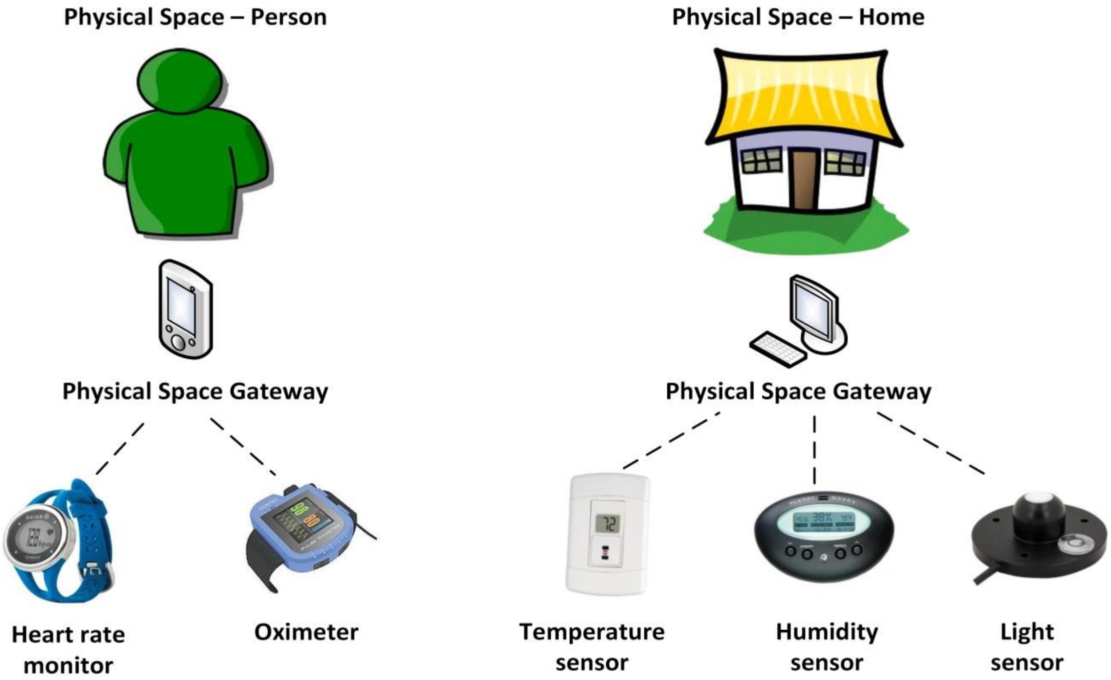


Figure 2. Illustration of the concept of physical space

The context data provided by a physical space can be termed as a *context attribute*. In order to manage the context data that is obtained from all of these raw data sources, each physical space contains a *physical space gateway (PSG)*. A PSG is a software module that manages all the data pertaining to a particular physical space [57, 58]. This involves the retrieval of data from raw context sources, aggregates and processes the data and providing the relevant data (including raw data if is desirable) to the interested parties. The PSG provides a single point of interaction between a physical space and external applications as well as other physical spaces. For example, the PSG for a person can reside in a smartphone whereas it can reside in a desktop computer for a home. Although the residency of PSG is location independent, it is preferred to be in the physical proximity of the corresponding physical space to minimize the network transmission overhead. In order to provide an ordering amongst the physical spaces, Coalition categorizes and clusters these spaces into multiple domain classes. These domain classes are defined as *context domains* and each of them represent a real world entity like shops, houses, people etc. Every context domain is associated with a set of attributes that provide a conceptual description of the domain and indicates the types of information that can be retrieved from that domain. In this way, each PSG can be mapped into one or less commonly more than one context domain (e.g. a home operating as an office). This forms the basis of the context model used in Coalition to represent the different context sources and context data values [59]. We now proceed to discuss the detailed structure of the context data management layer as illustrated in Figure 3.

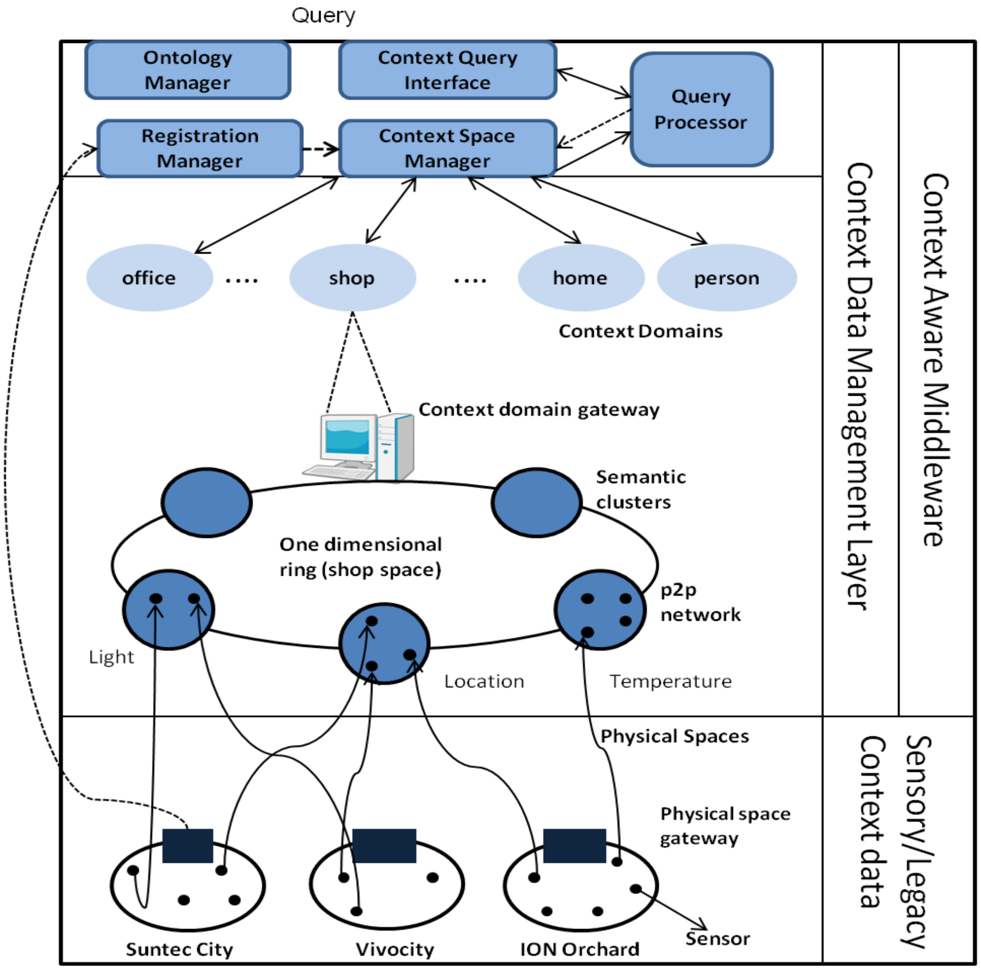


Figure 3. Overview of Coalition data management layer

A physical space belonging to a particular context domain may contain all or a subset of the attributes associated with that domain. In order to group these physical spaces according to the type of attributes provided by them, Coalition introduces the concept of *semantic clusters*. A semantic cluster represents a single attribute in a context domain and is modeled as a peer-to-peer network where the peers comprise of the physical spaces sharing the corresponding context-domain and attribute combination. All the semantic clusters belonging to a context domain are connected using a network topology such as a ring to form a semantic peer-to-peer overlay. As a context-aware environment is expected to be dynamic wherein physical spaces can leave and join semantic clusters, a peer-to-peer model is adopted to minimize the disruptions due to these movements. Each context domain is also equipped with a *context domain gateway* (CDG) that manages the semantic overlay and also acts as the entry point for data retrieval requests from the middleware server to the overlay.

A physical space can associate with Coalition by sending a registration request to the middleware through their physical space gateway. Figure 4 illustrates the process of a PSG joining the Coalition middleware system. The registration request contains the local context schema of the physical space that comprises the name of the relevant context domain and a list of attributes that the space wishes to make available through the middleware.

The context information at a physical space is modeled using a simple key-value model. A set of global context schemas are maintained at the middleware server where each schema corresponds to a particular context domain. These schemas are incrementally updated using the local context schemas submitted by the physical spaces during their registrations. The process of integrating the local schemas with the global schemas is carried out using a context schema matcher [60] developed as part of the middleware project. After the relevant context domain has been identified, the PSG of the physical space joins the peer-to-peer networks of the semantic clusters corresponding to the attributes in its context schema. An important point to be noted here is that the physical space can choose the set of attributes to be exposed to the middleware and is not obliged to reveal its entire attribute set. For example, a physical space corresponding to a person may choose to reveal only the name and personal preference attributes whereas keep confidential information like medical records private. As part of the network joining process, each PSG is assigned a random number of PSG neighbors from the semantic cluster.

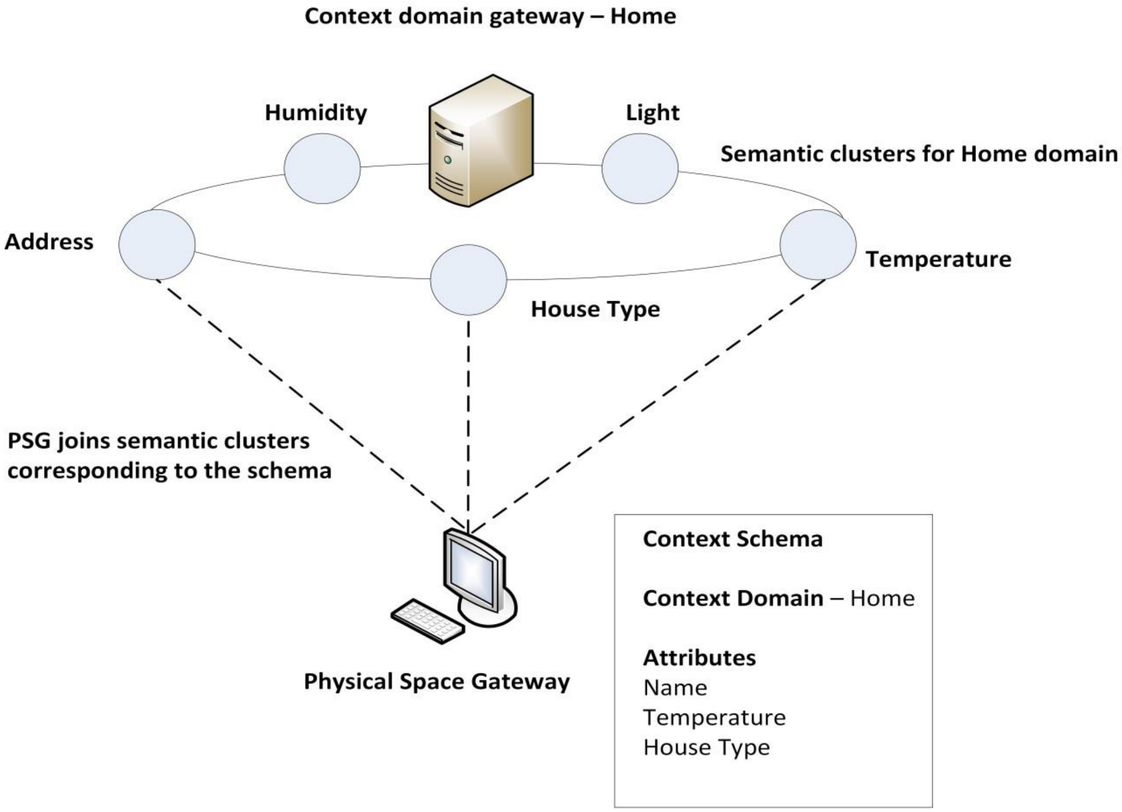


Figure 4. Registering a PSG with the Coalition middleware

The network connections between a PSG and its neighbors are not maintained persistently. Instead, the PSG stores the network information of all its neighbors and uses them to build a one-off connection as and when required. Similarly, a PSG leaving Coalition deletes all the neighbor tables corresponding to the semantic clusters it joined. The PSG also sends a message to all its neighbors informing them of its removal from the system following which the neighbors delete the PSG from their neighbor tables.

The Coalition system is intended to be generic and is not designed with a particular application domain in mind. The main purpose of Coalition is to provide an organizational structure that indicates the different types of information that can be retrieved from the physical spaces associated with it. The only restriction placed by Coalition is that applications are required to conform to the context domain and semantic cluster based data model. Also, Coalition does not store any of the context data belonging to the physical spaces associated with it and forwards all context data retrieval requests to the relevant PSGs which then evaluate these requests against their local context data. Since a data retrieval request is processed locally at a PSG, the most up to date value of a context attribute is used to process the request. This alleviates the concerns about the freshness of the retrieved data as discussed in the introduction. Also, this is in line with our design philosophy of delegating the tasks related to context data processing to the context sources. Further, as we observed in Section 2.3, context-aware systems that do not store context data do not face the overhead corresponding to updating large volumes of dynamic data. The decision of responding to a data retrieval request lies solely with the PSG and is not enforced by the middleware. This feature is important because it helps to address the concerns related to data privacy up to a certain extent. For example, consider a PSG of a person that has exposed the attribute indicating the person's name. A healthcare application can now search for this particular person using Coalition and request to see their medical records. When this request is forwarded to the PSG, the PSG can check the sender information and respond to the request accordingly. In this case, if the PSG deems that the sender application is trustworthy, it can proceed to send the requested information. This is a key aspect where the Coalition middleware differs from traditional database systems as well as the different context aware systems discussed in Chapter 2.

3.3 Context data retrieval in Coalition

Since Coalition does not store any context data in the middleware, the main task of the data management component in Coalition is to receive the data retrieval requests from applications and redirect them to the physical spaces that are most likely to provide the required answer(s) [59, 61]. Coalition provides a SQL-like declarative interface that can be used by applications to issue queries to the middleware. The basic query format supported by Coalition is as follows

SELECT < attribute > **FROM <** domain > **WHERE** (predicate) [**AND|OR** predicate ..]

The query processor component of Coalition parses the received queries to identify the required context domain/semantic cluster combination to which the query needs to be forwarded. The **FROM** clause identifies the context domain whereas the semantic cluster can be identified from either the **SELECT** or the **WHERE** clause. The query is then passed to the corresponding context domain gateway which then forwards the query to the relevant semantic cluster. The query is now disseminated in the p2p network representing the cluster using a controlled flooding operation. A random PSG peer from the cluster is selected as the starting point for the query. This peer evaluates the query against its local context data and forwards the query to its neighboring peers each of which repeats this process. A one-time network connection is setup between a peer and its neighbors for this purpose. Depending on the type of query, the PSGs can either report the answers directly to the query issuer or send them to the query processor where the answers are aggregated and reported to the issuer.

Although the flooding based technique of query dissemination is straightforward and can be easily implemented, it is restricted by the fact that it is not scalable when the semantic cluster size is very high. The semantic cluster size refers to the number of PSGs within the semantic cluster. A large semantic cluster size can result in an increased query response time using the flooding approach. Context-aware applications are usually interested in retrieving context data that satisfies certain conditions (such as data within a particular range of values, data from sources in a particular location etc). A flooding technique is not very efficient in this case especially in the case when only a small percentage of the PSGs contain the relevant data as a lot of time is now spent in querying PSGs that do not contain the required data.

Clearly, the current classification scheme using the context domain and semantic clusters is not sophisticated enough to handle the context lookup operation efficiently. A possible solution is to provide an additional level of classification that partitions the PSGs according to their data values. As we observed in Section 2.3, this problem is also present in context-aware systems that do not store the context data. This is the primary challenge we wish to address as part of this thesis – implementing a classification scheme to reduce the search space for a context lookup operation that satisfies the constraint of not having a persistent storage for context data. The effectiveness of the classification technique will depend on its ability to represent the data distribution pattern in a semantic cluster as uniformly as possible. Also, this classification scheme will need to adapt to the dynamics of the context data values as well as the movement of the PSGs as they join and leave the middleware.

3.4 Summary

In this chapter, we provided an overview of the Coalition middleware system wherein we highlighted the design principles behind the middleware. We also discussed the overall Coalition system architecture as well as the organizational scheme of the context data management layer which is the focus of this thesis work. As part of our study of the current data management and context data retrieval techniques followed in Coalition, we observed that the current organizational scheme comprising of context domains and semantic clusters is not sufficient to carry out lookups over large network sizes in a scalable fashion. This served as the motivation to augment the current organization by adding another layer of classification based on the data values of the attributes. This is a challenging problem due to the fact that the Coalition middleware does not store any context data and any classification technique used will need to satisfy this constraint. As part of our initial solution to this problem, we propose the idea of a range clustering technique to create an index structure on the context attribute values to facilitate the processing of context lookup queries.

CHAPTER 4

RANGE CLUSTERING BASED ORGANIZATION FOR CONTEXT LOOKUP

This chapter presents the detailed design of the range cluster based index structure that is introduced to facilitate the context lookup process as an initial alternative to the flooding based approach. The main issue addressed in this chapter is to improve the problem related to the scalability of the context lookup process. The chapter is organized as follows – Section 4.1 discusses the motivation behind the range cluster based indexing approach and an overview of the indexing scheme. We discuss the details of the indexing process and the associated maintenance operations in Section 4.2. Section 4.3 discusses the preliminary experimental results achieved using the proposed index structure. We summarize and conclude the chapter in Section 4.4.

4.1 Overview

As discussed in the previous chapter, the current classification followed in Coalition using context domains and semantic clusters is not sufficient to conduct lookups efficiently over a large number of PSGs. Considering the fact that context-aware applications usually require context data satisfying a set of conditions, the primary requirement is to have an additional classification scheme like an index that can process queries having conditional constraints effectively. Indexes in traditional database systems usually contain a single value or a range of values as the index key along with the associated addresses where the data corresponding to those values is stored. Since Coalition does not store any context data belonging to the PSGs, the direct adoption of the database index technique is not feasible. Also, the classification mechanism will need to take into account the fact that the data being indexed changes frequently and adapt to the changes as PSGs leave and join the system.

In view of the constraints placed by the organizational structure of the Coalition middleware system and the requirements for designing an index structure for context data, we propose the use of a range cluster based classification scheme to address the problem of context lookup [62]. As we saw in Chapter 2, the query requirements for context attributes vary according to their data type and a classification scheme needs to take these requirements into account as part of their design process. Since providing a single indexing scheme that can handle different data types is difficult, we adopt separate indexing mechanisms for numeric and string data types. The indexing schemes discussed in this chapter as well as Chapter 5 are designed to handle numeric context attributes. The problem of classifying string attributes is addressed at a later part in the thesis. The basic idea behind the range clustering based indexing scheme is to further classify the PSGs within a single semantic cluster according to their data values as illustrated in Figure 5.

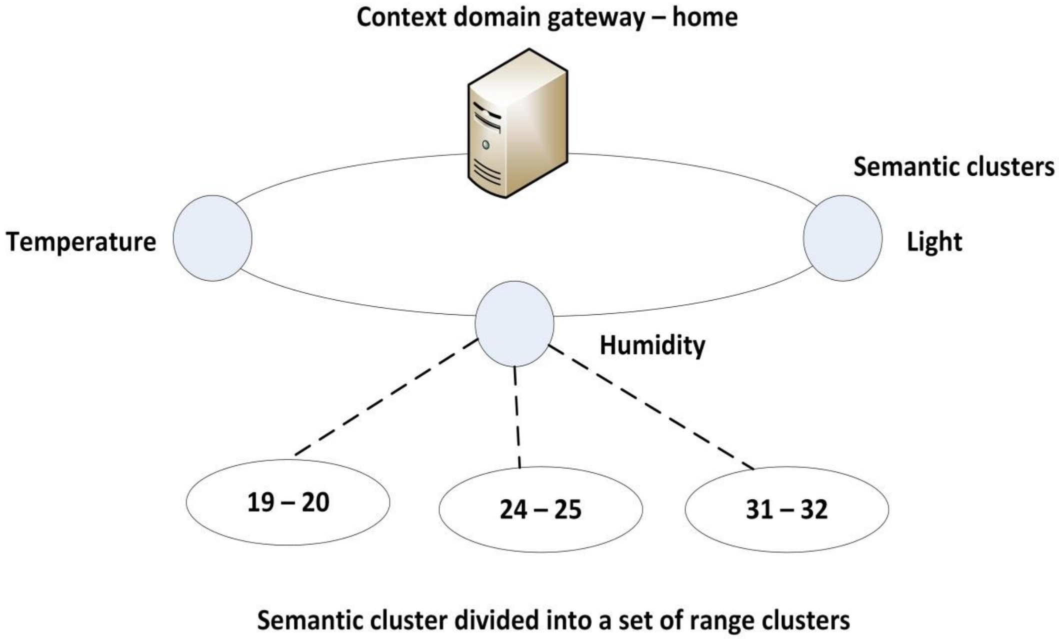

Context domain gateway – home
Semantic clusters
Temperature
Light
Humidity
19 – 20
24 – 25
31 – 32
Semantic cluster divided into a set of range clusters

Figure 5. The proposed range cluster based index structure

As per the new organizational scheme, each semantic cluster is now further subdivided into a series of range clusters. Each of these range clusters correspond to a range of data values and contain the corresponding PSGs that possess the data values belonging to that range. These range clusters now serve as the index structure for that semantic cluster and can be used to process context lookup requests. We discuss the details of the indexing process in the subsequent sections.

4.2 Range cluster based index structure for context data

4.2.1 Index structure generation using range clusters

The scope of the index structure discussed in this section is within a single semantic cluster belonging to a particular context domain. This structure can be replicated for different semantic cluster to manage multiple attributes. The main idea involved in creating the index structure is to use a clustering technique to partition the PSGs into multiple clusters. Since the index structure is required to work within a dynamic context environment, one of the primary requirements is that the clustering process should be incremental. One of the most well-known incremental clustering algorithms is the sequential leader clustering algorithm [63]. This algorithm compares each new value against the set of existing cluster leaders by using a distance metric to assign the value to a cluster. If the value cannot be assigned to any existing cluster, a new cluster is generated with the new value as its leader. One of the main drawbacks associated with this algorithm is that it can result in the generation of a large number of small sized clusters. We attempt to address this issue as part of our proposed clustering technique. In the initial phase of our index construction, all the PSGs are part of a single P2P network and are assigned to a single range cluster having no fixed bounds. The clustering process is initiated when the number of PSGs in this cluster exceeds a predefined system threshold for the maximum cluster size. The cluster size in this discussion refers to the number of PSG members in a cluster. This threshold value should be chosen with a view to ensure that the response time for processing a query in a single range cluster when it is at its maximum capacity is within a reasonable bound. It can be chosen by progressively varying the maximum threshold size and observing the corresponding fluctuations in the response time. The threshold can be chosen in accordance with the network conditions in the deployment environment.

Since each PSG contains the data value for the attribute corresponding to the semantic cluster, they are treated as the data points over which the clustering process is applied. Since no context data is stored in Coalition, the first step of the clustering process involves the retrieval of the data values of the relevant attribute from the PSGs. After the data values are received, they are inserted in a sorted list in increasing order of magnitude. In order to generate the clusters, the sorted list is then partitioned from the middle to obtain two sorted lists that are used to generate the range clusters. Figure 6 provides a detailed illustration of the process of generating the clusters. In the next step of the clustering process, the existing cluster is now replaced by two clusters each of which corresponds to one half of the initial sorted list. The minimum and maximum values of the two sorted lists represent the cluster bounds. The main idea involved in this step is to utilize the retrieved data values to create a set of clusters that provide an approximate representation of the data distribution within the original cluster. Once the clusters have been generated, each PSG is instructed to join the range cluster that corresponds to its current value.

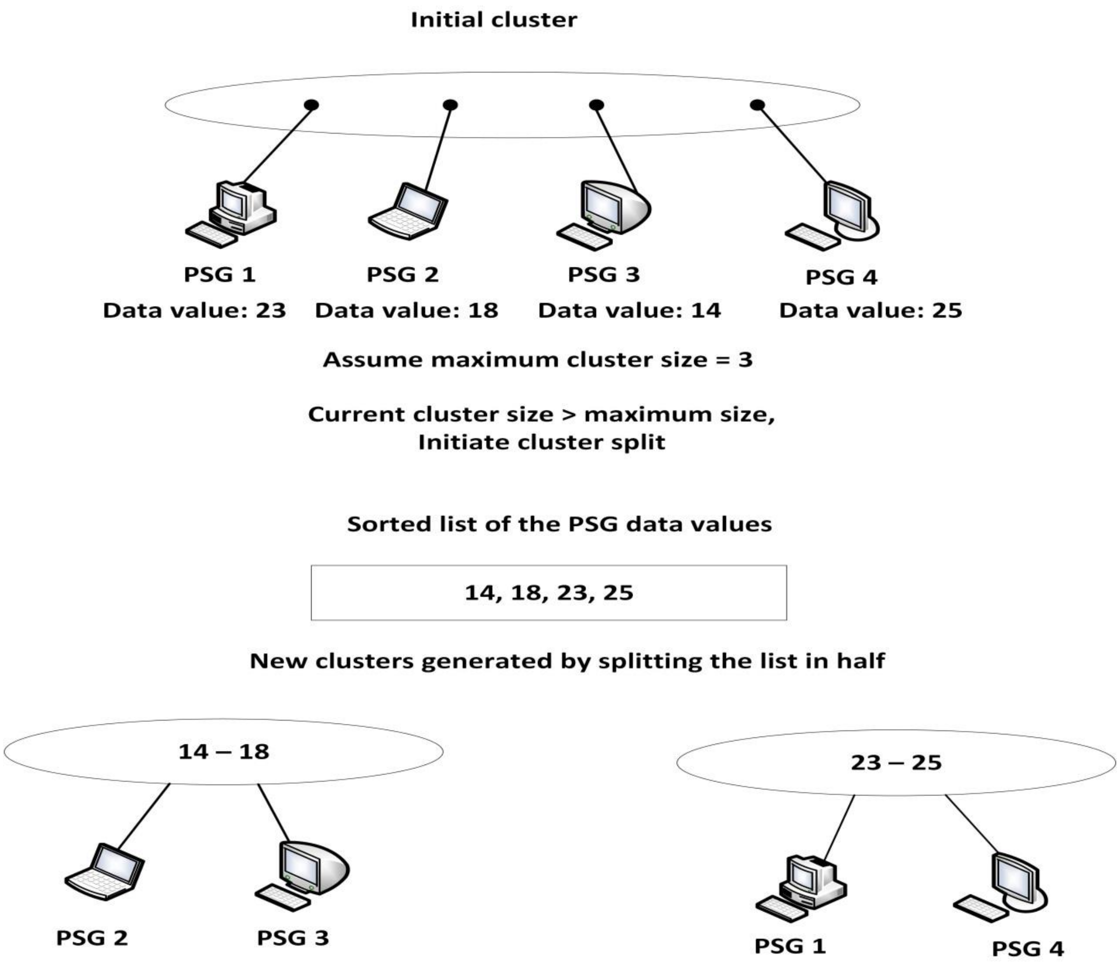


Figure 6. The cluster generation process

With the introduction of the range cluster based index structure, the single P2P network is now replaced by several P2P networks each of which corresponds to a range cluster. The P2P network corresponding to a range cluster is composed of the PSGs that possess an attribute with a data value in the range denoted by the cluster. As can be seen in the figure, the initial P2P network is composed of the four PSGs – PSG 1, PSG 2, PSG 3 and PSG 4. After the clusters have been formed, PSG 1 and PSG 4 form a P2P network whereas another separate network is formed by PSG 2 and PSG 3. This index structure satisfies the constraints of the Coalition middleware of not storing any context information as the index construction process only involves a one-time retrieval of the context data from the PSGs.

4.2.2 Index structure maintenance operations

After the initial clusters have been formed, subsequent PSGs registering with Coalition are assigned to the proper cluster based on the cluster bounds and the data value of the PSG at the time of registration. A PSG assigned to a range cluster becomes a part of the P2P network corresponding to that cluster. If the size of a range cluster exceeds the predefined threshold for the maximum cluster size, it is split using the previously described clustering process. In this way, the number of range clusters as well as their range bounds keeps evolving as PSGs join the system. The splitting of the clusters also keeps the range clusters balanced by preventing a single cluster from containing a large number of members. The clusters formed as a result of a cluster split are expected to have a range bound that is tighter than the original cluster bound.

As PSGs can leave as well as join the middleware, there can be a case where a range cluster contains either no members or very few members. In order to prevent this, the minimum number of members that need to be present in a cluster is defined using a threshold. When the size of a range cluster falls below this threshold, it is merged with an existing cluster to keep the total number of clusters low. The merging of two clusters results in a new cluster with a range bound obtained by merging the corresponding range bounds. It is important to ensure that the merging of two clusters does not disrupt the existing order of the range clusters. The detail of the cluster merging process is illustrated in Figure 7.

The first step of the merging process involves the ordering of the range clusters according to their cluster bounds. Following this, the position of the cluster selected for merging is identified. This cluster can be merged with any of its adjacent clusters as per the ordered list. As observed in the figure, there can be either a single adjacent cluster or two adjacent clusters. If there is only a single adjacent cluster, a new cluster is created by merging these two clusters. Also, the P2P networks corresponding to these two clusters are deconstructed and a single P2P network is created with the members of both the clusters. If the merging process creates a cluster with a size greater than the maximum cluster size, then a subsequent splitting operation is required. In order to minimize the chance of this event, a cluster having two adjacent clusters is merged with the cluster having the lower size.

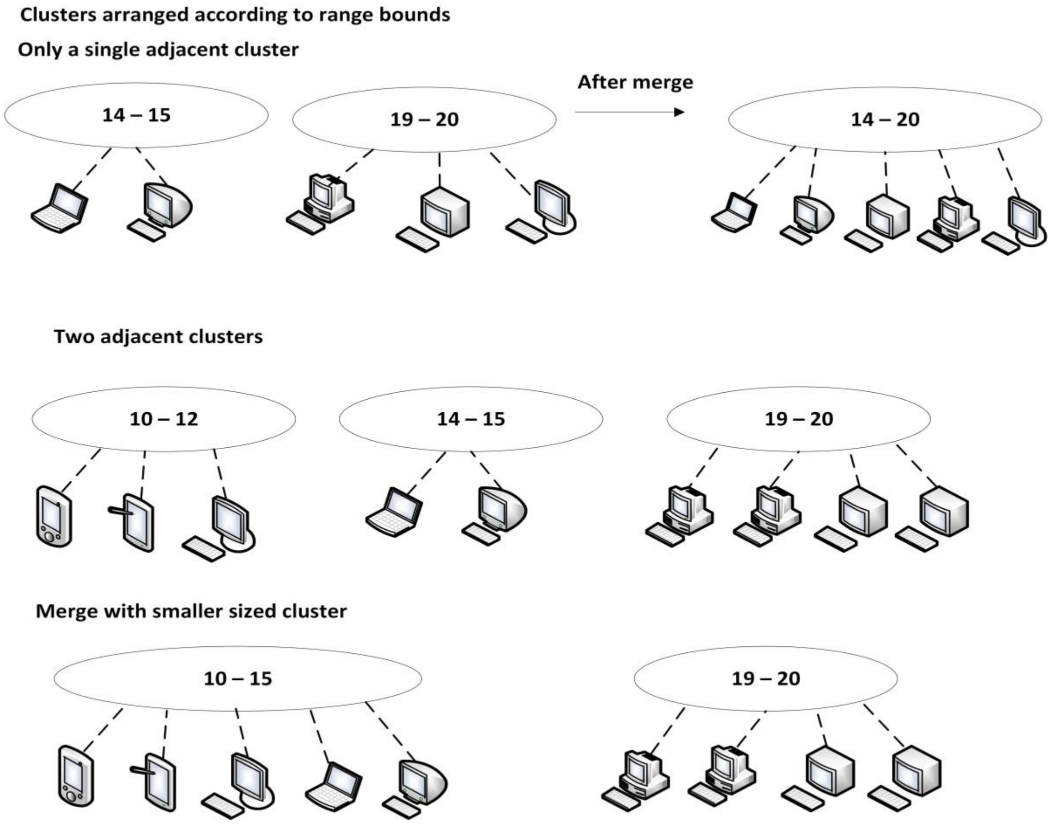

Clusters arranged according to range bounds
Only a single adjacent cluster
14 – 15
19 – 20
After merge
14 – 20
Two adjacent clusters
10 – 12
14 – 15
19 – 20
Merge with smaller sized cluster
10 – 15
19 – 20

Figure 7. The cluster merge process

A range cluster having no members is removed from the index. The split and merge operations play an important part in adapting the index structure according to the movement of the PSGs and ensure that the index reflects the continuously changing data distribution within the semantic cluster as accurately as possible. Apart from the leave and join operations, a PSG can also shift between the different range clusters due to the dynamic nature of context data. Since the initial assignment of a PSG to a range cluster is done on the basis of the value of the context attribute at that instant, subsequent changes in the attribute value need to be accounted for.

In order to handle this issue, each PSG periodically checks its data values against the bounds of the cluster which it has been assigned to. In case the new value is inconsistent with the current cluster bounds, the PSG leaves this cluster and joins a new one. The movement of PSGs between the range clusters can also result in a cluster split/merge operation. The frequency with which a PSG moves between different range clusters depends on the cluster bounds as well as the data change pattern of the context attribute within the PSG. The storage space requirements for the proposed index structure are minimal as the index does not store any of the data being indexed. The information required by Coalition to represent the index structure comprises of the cluster bounds of each of the range clusters and the list of members present in each range cluster.

4.2.3 Context lookup using the index structure

The context lookup operation involves the identification of the context sources having the relevant data for a query and the retrieval of the data from these sources. As per the previous scheme, a query was redirected to the appropriate context domain/semantic cluster combination following which the query was flooded to all the members of the semantic cluster. The range cluster based indexing scheme now introduces an additional layer of ordering within a semantic cluster. Context queries in Coalition are modeled using the SQL syntax and specify the data to be retrieved along with the necessary constraints. Figure 8 illustrates the details of the context lookup operation using the range clusters.

The constraints specified in the context queries are in the form of a range condition or a set of logical conditions on the value of a context attribute. As can be seen from the figure, the proposed index structure can be used to process these context queries by evaluating the constraints against the range cluster bounds and identifying the set of range clusters that satisfy those constraints. This reduces the search space for a query and forwards it to only the relevant subset of PSGs expected to have a valid answer for the query. Once the relevant set of range clusters have been identified for a given context query, the query is disseminated in each of these clusters using a controlled flooding based technique as discussed in section 3.3.

Although flooding is still used to process the query within a range cluster, the associated overhead is expected to be less as compared to a system wide flooding as the flooding scope is now reduced to a subset of the system. In other words, the number of PSGs involved in the flooding operation is reduced. Also, the introduction of the proposed index structure does not require any modifications in the context lookup operation from the perspective of a PSG. The primary function of an index is to provide an efficient and fast way to access data. The effectiveness of an index thus depends on its partitioning ability and the accuracy of its representation of the distribution of the indexed data.

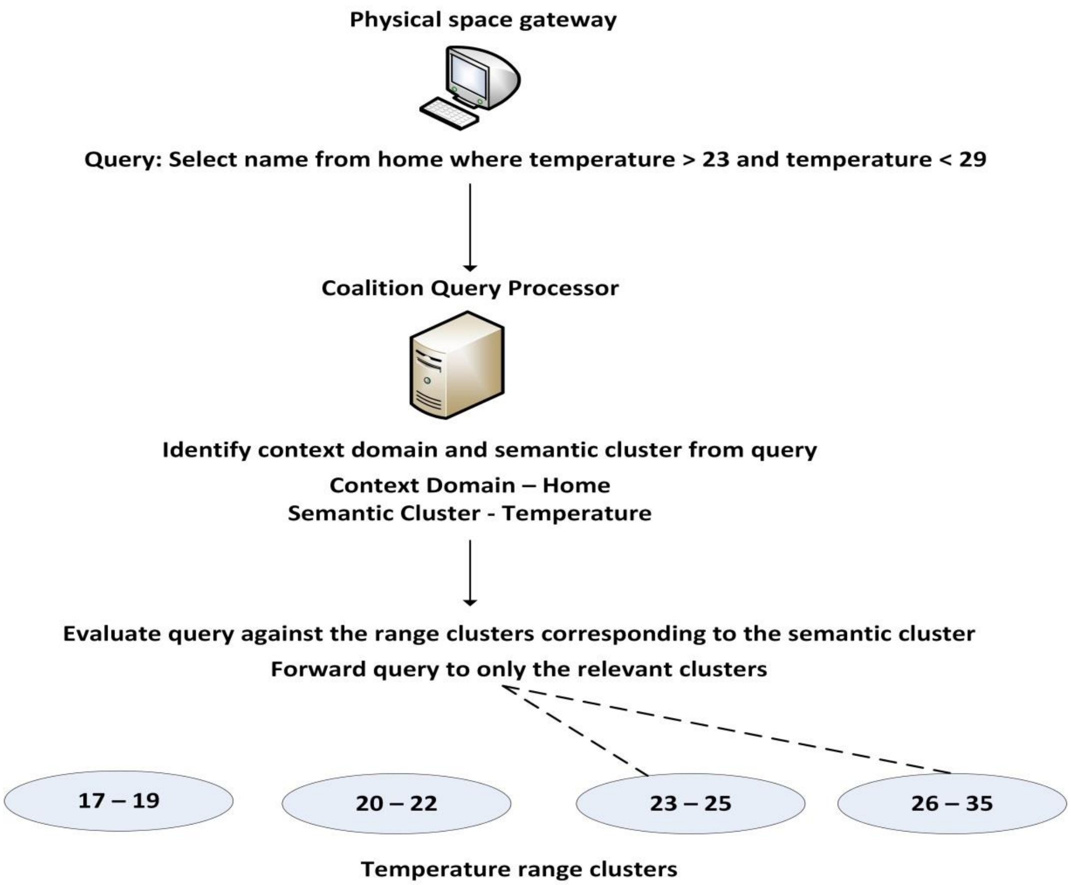

Physical space gateway
Query: Select name from home where temperature > 23 and temperature < 29
Coalition Query Processor
Identify context domain and semantic cluster from query
Context Domain – Home
Semantic Cluster - Temperature
Evaluate query against the range clusters corresponding to the semantic cluster
Forward query to only the relevant clusters
17 – 19
20 – 22
23 – 25
26 – 35
Temperature range clusters

Figure 8. Context lookup using the range clusters

Since the indexed data is dynamic, the proposed index can only provide an approximate representation of the data distribution. When a query is forwarded to a range cluster, the number of answers received depends on two factors – the number of PSGs with data satisfying the query conditions as well as the PSGs that respond to the query. Since the decision to respond to a query rests with a PSG, it may not respond to a query in spite of having a valid answer. This is one of the aspects in which Coalition as well as the proposed index differs from the conventional notion of indexes in database systems. The interval size of the cluster bounds of the range clusters also play an important part in determining the efficiency of the query processing operation. The interval size of a cluster refers to the difference between the upper and lower cluster bounds. To cite an example, a cluster bound of 10 – 11 represents a short interval as compared to that of a cluster bound of 20 – 30. If all the clusters have low interval sizes, the query processing performance can become more efficient as queries can be directed to the relevant set of data sources quite accurately. Also, this property can lead to the formation of a large number of clusters as each cluster can only accommodate values in a narrow range. Conversely, if the clusters have large interval sizes, the total number of range clusters is reduced as each cluster can now accommodate a larger range of values. However, this might affect the query performance due to the looseness of the index.

Since the cluster bounds are not predefined in our proposed indexing scheme, the interval sizes of the clusters keep changing as PSGs join and leave the system. When a cluster splits, the resulting clusters have a smaller interval size as compared to the original cluster. Similarly, the merge operation creates a cluster with a larger interval. The threshold values that dictate the minimum and maximum number of members in a single cluster try to alleviate the problems associated with both the types of interval sizes. The minimum size threshold tries to keep the total number of range clusters low by merging clusters. On the other hand, the maximum size threshold is used to address the issue of the degradation of query performance with larger interval sizes. Taking the example of the clusters in Figure 8, consider four queries each searching for one of the following values (30, 31, 32, 34). Assuming that none of these values are currently possessed by any PSG in the system, the query processor will still redirect all these queries to the last cluster that has a relatively larger interval size. The query will be disseminated among all the members of the cluster and a result set comprising of zero answers will be returned after all the members have been probed. This example is used to illustrate the impact that clusters with larger interval sizes can have on the query processing performance. If such a range cluster contains a large number of members, the time taken to report that a valid answer is not present can increase proportionately. We attempt to address this problem in our index structure by restricting the membership size of a range cluster using the minimum and maximum thresholds.

4.3 Experimental analysis

4.3.1 Experimental setup

The proposed index structure was implemented and integrated with the prototype of the Coalition middleware. The Coalition middleware was then deployed on a Dell PowerEdge T300 server with four 2.83 GHz quad-core Intel Xeon CPU and this machine was designated as the server. A set of desktop machines each running the Windows XP operating system and having an Intel Core 2 Duo 2.83 GHz CPU is used to simulate the PSGs. Since a PSG is designed as a software module, a single desktop can be used to simulate multiple instances of the PSG. The data values used for testing were derived from the Intel lab sensor data set [64]. This data set comprises of temperature, humidity, light and voltage values from 54 sensors collected over a period of time. Each sensor contains ~55,000 data values for each of the four aforementioned attributes. As discussed in Section 1.1, context can be defined as any information that can be used to describe an entity. This information can refer to the physical environment as well as the human factors. In this case, the sensor data set used to derive the testing data for the experimental results contains a set of environmental data values that provide information about a particular place (the deployment environment of the sensors). In other words, the data values represented by the Intel lab sensor data set can be designated as a valid set of real-world dynamic context information. This justifies the selection of the Intel lab sensor data set as a valid source to generate the testing data for our proposed indexing structure.

In order to generate the test data set, we first extracted all the data values corresponding to a particular sensor and attribute combination. This step resulted in a collection of 54 data files for a given attribute type. Since the proposed index is implemented within the scope of a single semantic cluster, we used different attribute values as per the requirement of the experiment. The set of data values generated at a particular PSG was simulated by taking a random sample comprising of a fixed number of values from one of the data files obtained previously. For example, to model a set of temperature values at a PSG, a data file corresponding to the temperature attribute was randomly chosen and a fixed number of values was selected from that file. Each of the simulated PSG instances was assigned a set of these data values. The process of a PSG periodically receiving a new value for a data attribute was simulated by reading these values one by one after a fixed time interval. The maximum and minimum cluster size thresholds are fixed at 50 and 10 respectively. All experimental results are represented using a 95% confidence interval of the results by assuming a normal distribution wherever applicable.

It needs to be noted here that all the experimental results in this chapter as well as the subsequent chapters evaluate the performance of the proposed context data management system as a component of the Coalition system and the performance comparisons with other data management schemes is minimal. Since the proposed data management system is developed as part of a context-aware middleware system, it is not feasible to evaluate the data management system as an isolated system and compare it with other data management techniques. Also, the fundamental constraint of Coalition that prevents any data from being stored in the system differentiates the proposed data management system from techniques that optimize the management of data values stored in memory. The only viable way to do a performance comparison of the proposed system is to compare it against the data management systems of other context-aware middleware systems. However, the lack of a standardized performance benchmark for context-aware systems combined with the fact that such middleware systems are usually proprietary software and not freely available make this type of comparisons infeasible. The only comparison that can be done in this case is to provide a feature-wise comparison of our proposed system with existing systems.

4.3.2 Query response time

Since the primary motivation for developing the proposed index structure is to facilitate the context lookup, the preliminary sets of experiments are conducted to assess the index with respect to the query response time. 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 the query result. The response time measurements were carried out in a stable test environment wherein a fixed no of PSGs are present, no PSG enters or leaves the system and after a PSG is assigned to a cluster it does not move to another cluster. In each round of this experiment, the number of PSGs having a valid answer for the query was fixed to be 50. We compared the query response time achieved by two P2P schemes: the normal flooding based scheme as well as the proposed index structure. Both the P2P schemes were implemented using the UDP protocol. The results are shown in Figure 9 which shows the variations in the query response time plotted against different network sizes.

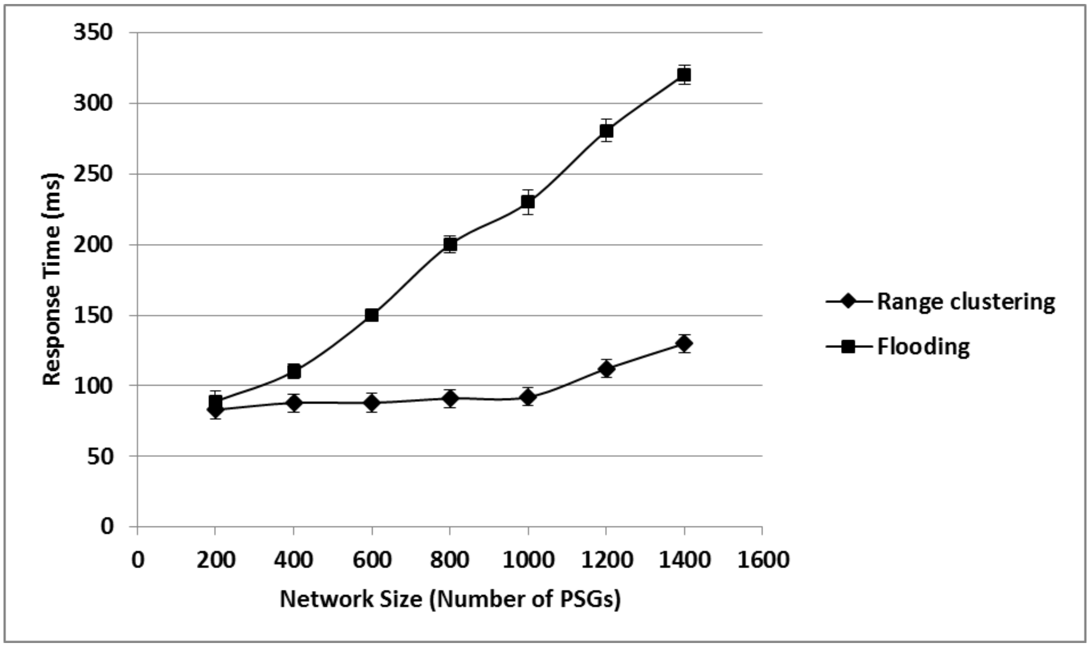


Figure 9. Query response time with different network sizes

The network size refers to the total number of PSGs registered with the middleware. The results indicate that the response time shows an increase as the network size increases for both the schemes. This observation is based on the fact that the increase in the number of PSGs in a system has an impact on the total search space for a query. We also observe that the performance of both the schemes is similar when the network size is small. However, the response times for the flooding based scheme increase sharply with the increase in the network size as opposed to the comparatively slower increase observed with the proposed indexing scheme. The slow increase in the response time can be attributed to the ordering established by the indexing scheme which helps to reduce the search space for a query. This is illustrative of the fact that the introduction of the proposed indexing scheme is an important first step towards realizing an efficient context lookup mechanism. An important point to be noted here is that all the experimental results discussed in this section as well as the subsequent chapters are carried out in a controlled test environment. When a real world deployment is considered, it is expected that the end to end network delays will contribute to the response time as well. While this is an important issue to be considered in its own right, we do not consider the impact of the network delay in these experimental results as we are more interested in optimizing the process of organizing the context sources to facilitate the retrieval of context data from these sources. Since the main focus of this thesis is to develop a context data management system and assess the performance of this system with respect to its ability to partition and query context data, we choose to emphasize this aspect of the system in our studies and do not consider the network latency. We further examined the response time of different schemes by varying the number of PSGs with the required results for a query in a semantic cluster. The network size of the semantic cluster was fixed to be 1000 PSGs in this experiment. The results are illustrated in Figure 10.

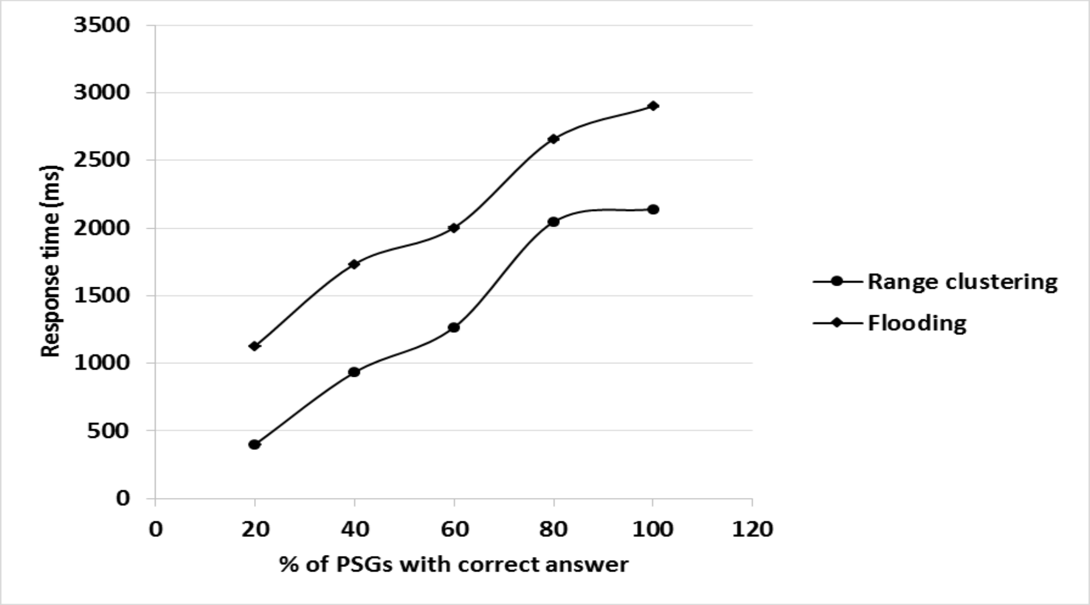


Figure 10. Query response time with different number of PSGs with valid answers

As can be observed, there is an increase in the query response time for both the schemes with a corresponding increase in the number of PSGs with valid answers which is expected. However, it can also be observed that the proposed index structure achieves a query response time that is faster by 35% - 45% than that achieved using the flooding based scheme especially when a smaller percentage of PSGs contain the valid answers. This can be explained by the fact that the proposed index structure enables a faster localization of PSGs with valid answers due to the value based partitioning imposed by it. Also, the results indicate that the performance of both the schemes is comparable when the percentage of the PSGs in the network containing the valid answers is quite large (80% and above). In this case, the presence of an index does not offer any additional advantage over the flooding approach. Based on these preliminary experimental results, we can conclude that the presence of the indexing scheme is effective in improving the context lookup process especially when the number of context sources is large and only a small percentage of them contain the required context information.

4.3.3 Index performance with dynamic context data

One of the primary design requirements of a context lookup mechanism is the ability to manage dynamic context data. We do not handle this problem directly as part of our index design and rely on the PSGs to periodically check their attribute values and update to a different cluster in case of an inconsistency. Although this approach is straightforward, it can affect the query accuracy. Since the consistency checking is done periodically by the PSG, a situation can arise wherein a PSG is a member of a cluster even though its local data value has moved out of the cluster bounds. This can interfere with query correctness as the query processor could redirect a query to a range cluster(s) containing PSGs that have data values that are inconsistent with the cluster bounds. In order to verify this observation, we conduct a series of experiments to check the inconsistencies between the local data of a PSG and the cluster bounds.

As discussed previously, a data file is associated with every PSG instance to simulate the set of attribute values generated at the PSG. In each run of the experiment, a PSG reads a new value from the file at intervals of one second thereby simulating the periodic generation of context data and conducts the consistency checking at intervals of 1 minute. We deliberately choose the frequency of data checking to be higher than the data generation frequency in order to observe the inconsistencies occurring as a reason of the same. The experimental setup is as follows – After a PSG is assigned to a range cluster, the compatibility of the new value and the cluster bounds is checked with every new value that is generated. In case of an inconsistency, the details of the PSG are recorded. We first identify the percentage of PSGs that exhibit this inconsistent behavior within a fixed network size during a time period of 1 hour. After the completion of the experiment, the list of unique PSGs that exhibited this inconsistency (one or more times) was compiled. Figure 11 illustrates the results of this experiment. The results demonstrate that for a given network size, a significantly high percentage of PSGs ranging from 50% to 70% are part of a range cluster whose bounds are incompatible with their current data value. Also, as each of these

PSGs would have exhibited this behavior multiple times, the risk of not receiving accurate answers for a query increases. Clearly, the proposed index is not equipped to handle the dynamic nature of context data.

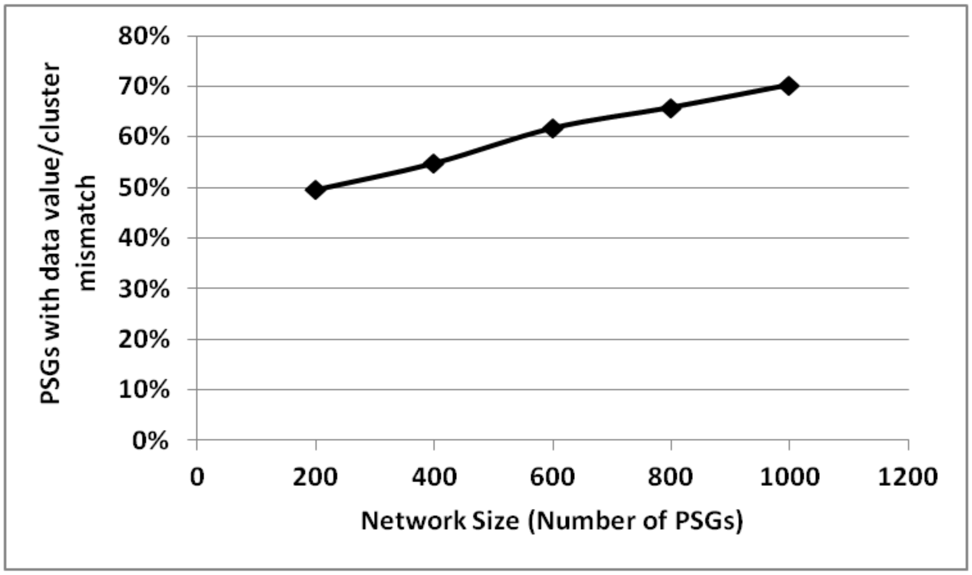


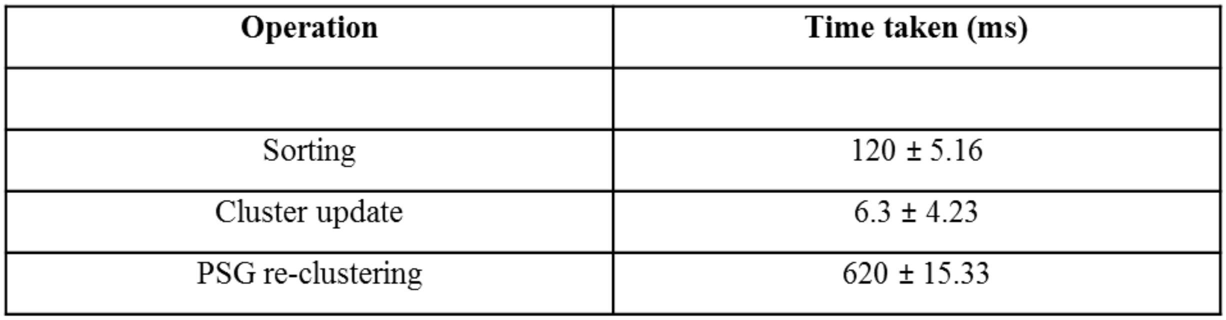
Figure 11. Identifying PSGs having data inconsistent with cluster bounds

There are two aspects where the design of the index should be modified in order to improve the dynamic data management. Firstly, the checking of the data values and the cluster bounds should not be carried out arbitrarily. The second aspect that needs to be improved is the classification process so that the clusters reflect the data distribution more accurately. The current practice of cluster generation of splitting the sorted list of the cluster members can lead to clusters being generated that reflect an unbalanced view of the data members in the semantic cluster. If we consider the following sorted list [20, 21, 21, 23, 24, 45, 46, 47], the clusters generated using this list will be [20 – 23] and [24 – 47]. In this case, a better distribution would be achieved by shifting the PSG corresponding to the value 24 to the first cluster. Also, the current clustering scheme always splits an existing cluster into two clusters which cannot always guarantee a fair data distribution. Ideally, the number of clusters a single cluster is split into should not be restricted to a fixed number and should be based on the data distribution of the PSGs. The implementation of these two features would help to ensure that the index structure can efficiently manage context data.

4.3.4 Time breakdown for cluster maintenance operations

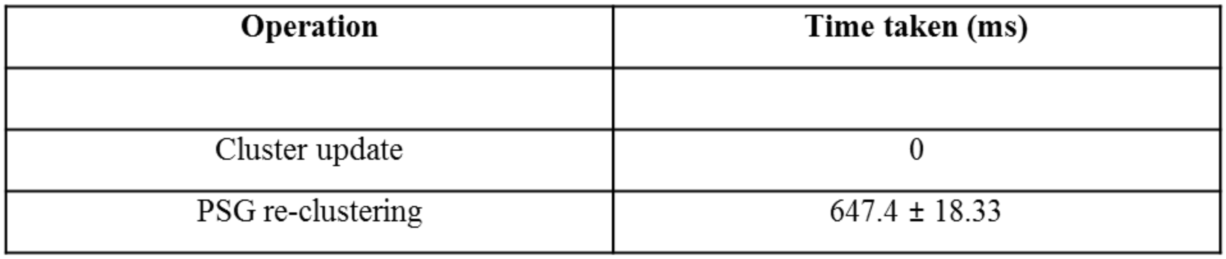
Since the cluster maintenance operations of splitting and merging play an important role in adapting the index structure according to the movement of the PSGs, we briefly discuss the time breakdown required for these operations in Tables 2-3. It is observed that the cluster splitting and merging operations require a similar maintenance overhead with the merging requiring slightly more.

Table 2. Time breakdown for cluster splitting



Operation	Time taken (ms)
Sorting	120 ± 5.16
Cluster update	6.3 ± 4.23
PSG re-clustering	620 ± 15.33

Table 3. Time breakdown for cluster merging



Operation	Time taken (ms)
Cluster update	0
PSG re-clustering	647.4 ± 18.33

The regrouping costs of PSGs in the newly merged/split clusters dominate the total time taken for these operations. Unless the dynamism of the context data and the memberships of PSGs are high, the frequency of merging/splitting will be low and hence the latency of the operation will not affect the overall system performance. Again, the inability of the index to cope with the dynamic data values can lead to repeated cluster splitting and merging operations as PSGs can rapidly join and leave the range clusters. Since this can have an impact on the system performance, it is another strong motivation to improve the system performance with respect to dynamic data.

4.4 Summary

We discussed our initial approach for developing a context data management system in this chapter in the form of a range cluster based indexing scheme. This scheme utilizes the data values contained within the context sources i.e. the PSGs to generate an approximate ordering over these PSGs and partition them into a set of range clusters to facilitate the context lookup process. A set of maintenance operations are also defined to keep the index structure balanced as well as adapt to the movements of the PSGs. The initial experimental results indicate that the proposed indexing scheme succeeds in minimizing the response time for context queries by reducing the search space for a query and offers a better alternative than the flooding based approach. Also, the time breakdown of the cluster maintenance operations suggest that frequent invocations of these operations can have an impact on the system performance. Further, it is also observed that the index is not equipped to handle the problem of managing dynamic data satisfactorily and can lead to errors in the query processing operation. Since this is an important design requirement for a context data management system, We address this issue in the subsequent chapter that utilizes the statistical data properties of context data to create an index structure that can efficiently handle the problem of managing dynamic context data and carrying out lookups over them.

CHAPTER 5

A MEAN-VARIANCE BASED INDEX FOR DYNAMIC CONTEXT DATA LOOKUP

This chapter discusses the design and implementation details of a mean-variance based indexing scheme to manage and carry out lookups over dynamic context data. The chapter is organized as follows – Section 5.1 gives an overview of the problem of dynamic data management. This is followed by a survey of the current techniques/approaches used to solve this problem in Section 5.2. The use of the statistical properties of data i.e. the mean and variance to create an index structure is discussed in Section 5.3. This section also gives the details of the mean-variance calculation process within a PSG using the dynamic queuing approach. The details of the index creation process using the mean-variance values are discussed in Section 5.4. Section 5.5 discusses the process of context lookup using the proposed indexing scheme as well the factors that influence the number of answers received. The detailed experimental results are discussed in Section 5.6. The chapter is summarized and concluded in Section 5.7.

5.1 Overview

We have already established the need to incorporate the efficient management of dynamic context data as an important design requirement for context data management systems. In the previous chapter, we observed that the range clustering based index was not able to satisfactorily handle the dynamicity of context data. The experimental results carried out previously also indicated that an inadequate handling of dynamic data can lead to query inaccuracies. In this chapter, we discuss an extension of the previously mentioned indexing scheme that aims to address these issues. This new scheme utilizes the statistical properties of the data values namely mean and variance to build an index that can handle the problem of frequent index updates occurring due to dynamic data.

5.2 Dynamic data management

We observed in Chapter 2 that the existing context data management strategies do not address the problem of managing dynamic data satisfactorily. However, this problem has been addressed as part of the research efforts in other application domains and we review some of these techniques in this section.

Inverted indexes are an important class of indexes designed for documents and store a mapping from content, such as words or numbers to their locations in a document or a database file. With the rapid growth of the Internet and web applications, the management of dynamic web documents has become a primary concern. One approach towards reducing the update cost by minimizing the memory accesses for the index rebuilding is to carry out this process selectively [65]. This is done by buffering new documents and sequentially integrating them into the main index. The concept of forward indexing and the use of landmarks to encode the positions of words in a document is proposed in [66] to handle the index update problem for dynamic documents. A hierarchical index for managing dynamic structured documents (like XML and SGML) is discussed in [67]. This index structure keeps the indexing information only at the lowest levels and the information at intermediate levels is obtained by accumulating the corresponding leaf node frequencies thereby minimizing the update overhead. Although these techniques are effective in managing the index updates occurring in dynamic documents, the design principles involved are tightly coupled with the application domain. In other words, it is difficult to extend these data management strategies for use in a context-aware system.

Another application domain that needs to manage large volumes of dynamic data involves spatial data systems and indexes for moving objects. Most of these techniques involve augmenting the R-tree index [68] to support dynamic data indexing. The RUM tree variation [69] attempts to reduce the update costs by allowing the updated entries to co-exist with the current entries and reduce the cost of an update operation to that of an insertion one. As moving objects over an area often share some information, the FT-Quadtree [70] attempts to minimize the redundant information to be indexed by using shared trajectory segments. These segments are used to build a reduced size index thereby making it more efficient for indexing large volumes of spatial data. The CNG index structure [71] uses a two level topography-aware grid structure as an in-memory index for moving objects. The update procedure is carried out with each object reporting their new positions to the cluster nodes performing the task of index updating by checking if the positions fall within their covering ranges. The CT-R-tree (change tolerant R-tree) [72] builds an index based on the observation that most indexes inherently tolerate some changes in the data being indexed. This tolerance is exploited by identifying quasi-static regions denoting the ranges of the domain enclosing a large number of updates and minimizing inter-region updates. The idea of minimizing index updates using the trajectory information to predict the path of moving objects is also discussed with different variations in [73-77]. Similar to the case observed with the inverted indexes, the indexing schemes for spatial databases and moving object systems utilize the special properties of the application domain like the motion trajectories as part of the index construction process. This dependence on the system properties makes these index structures unsuitable for indexing generic context data.

Sensor networks are also an important area where support for dynamic data management is required. Indexes designed for sensor networks [78-81] need to be distributed and energy efficient. These indexes attempt to minimize the update overhead by making the index query driven and fix the operating mode of the sensors to be either proactive or reactive depending on the query frequency. Alternatively, the sensor network is visualized as a database where a query is directed to specific sensors in accordance with the query issued. In order to store this dynamic sensor data in databases where the value might differ from the actual sensor value, database systems are augmented with uncertainty management components [82]. A data value is associated with an interval and probability distributions functions and queries are handled using probabilistic operators that define the answer quality. The sensor network indexes operate at the individual sensor level and are designed for applications interacting directly with sensors as compared to our case where we consider context sources encompassing multiple types of sensors. Also, the technique of storing the sensor data in databases is different from our middleware scenario where all the data is stored locally at the PSGs.

Since the frequent updating of data lends a temporal aspect to the static data, time based partitioning can be used to minimize the update overhead. Dynamic data can be visualized as data that is valid only for a certain amount of time. The use of traditional indexes in this case is difficult as new data needs to be inserted continuously whereas old data has to be deleted once its lifetime expires. The problem of indexing a time evolving set of data items is addressed in [83] and proposes a sliding window index that divides the window chronologically into a set of equal partitions (based on insertion or expiration times), each of which is then indexed by search key. The advantage of this approach is that only one sub-index is affected by an update at a given time as these sub-indices are partitioned on the insertion (or expiration) times and the updates fall within the range of a single sub-index. The prerequisite here is that each data item needs to be associated with a time duration denoting its validity period. The MVtree (mean-variance tree) [84] index structure is based on the idea that statistical attributes like the mean and variance are more resistant to change than the actual data value. This idea is used to create a modified R-tree structure that builds an index by visualizing data items as points on a 2-d mean-variance grid. Both these schemes are fairly generic and do not make any prior assumptions about the properties of the data.

Since we utilize P2P networks to organize the context sources and carry out lookups over them, we review some of the techniques used to process range queries over P2P networks. A popular class of distributed indexes designed for P2P networks is the DHT (distributed hash table) based data lookup techniques described in Chord, CAN, Pastry and Tapestry that use hash functions to assign data to nodes and lookup data [85-88]. Although the basic capability of DHTs is to support exact matching queries, there have been efforts to support range queries using DHTs [89-93]. The range query support for DHTs can either be provided as an independent functionality without modifying the existing overlay structure or can be incorporated as part of the overlay structure. A variety of techniques are employed in order to support the processing of range queries. Space filling curves can be utilized to map objects with multiple attributes to peers and then process range queries by searching the clusters generated by the curve. Alternatively, tree based index structures based on the binary values of the object keys can be constructed to facilitate single and multi-attribute range queries over DHTs. Apart from these, there exist a number of customized DHT techniques that make structural changes to existing DHTs to achieve delay-bounded range query response times. Since DHTs utilize hash functions to store and lookup data, the use of these algorithms with dynamic data will prove to be difficult as the hash value will need to be continuously updated. Also, the basic DHT model involving a set of nodes which store data according to the key-node mapping is structurally different from our middleware architecture where we consider multiple autonomous context sources that do not store any information about the type of data stored in any other context sources. Thus, there is a difficulty in adapting these techniques to create an index structure over context data.

Based on the design principles of the surveyed techniques, we identify the following two indexing schemes that are not associated with any particular application domain – the index built using the temporal aspect of data and the R-tree based index built using the statistical properties of data. Since the second scheme does not require any prior assumptions about the data being indexed and provides a partitioning based on the data values, we use it as the basis to refine and redesign the range clustering process in our middleware.

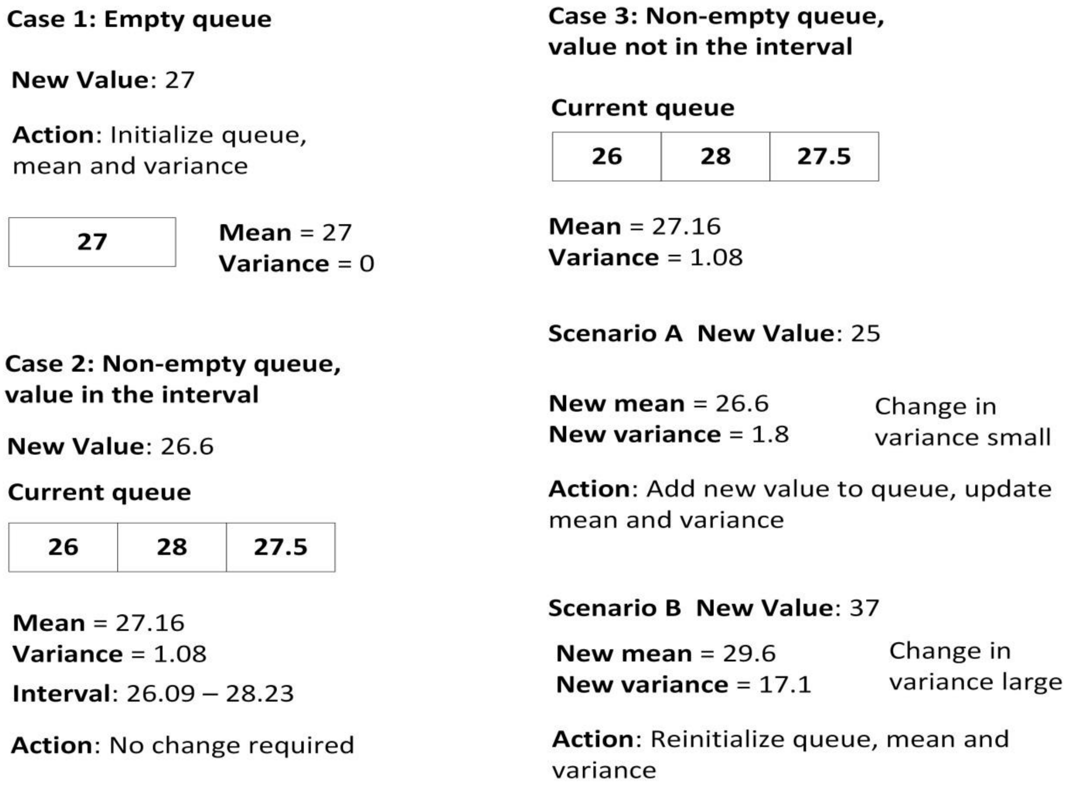
5.3 Using mean and variance to index dynamic data

In order to manage and carry out lookups over dynamic context data, we propose a mean-variance based index structure as a refined version of the previous index structure [94]. The principle behind this indexing scheme is partly based on the idea discussed in [84] which states that even though a data value 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. In other words, even though the data value keeps changing frequently, the magnitude of the change is low. Hence, an index constructed using these statistical properties are expected to have a lower index update cost. The index structure proposed in [84] is based on building an R-tree by visualizing the initial data set as points on a 2-d mean-variance grid and constructing MBRs around this data. One of the key assumptions in the R-tree construction process is the presence of a certain number of data points to facilitate the initial construction of the index structure following which the index is adapted as new data values are added to it. Clearly, this assumption is not feasible in our scenario as the index is expected to operate in a dynamic context-aware environment where context sources (PSGs) are expected to frequently leave and join the system. The use of an R-tree in this case will lead to frequent maintenance operations on the index thereby reducing its usefulness. Based on the requirements and constraints of the Coalition middleware system, we propose a refined version of the incremental clustering algorithm discussed in Section 4.2.1 that strives to use the mean-variance values in a novel way to establish a better ordering amongst the context data values within the PSGs and is also able to withstand the inherent dynamism of a context-aware environment.

Since the index construction is based on the mean and variance values, one of the key requirements is to keep track of these values for every attribute. Since a PSG is responsible for its own local data, these calculations must be carried out within the PSG. If a data attribute A is dynamic 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

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

Since a dynamic data attribute results in a continuous stream of values, attempting to calculate the mean and variance over the entire set of historical values can be expensive in terms of both processing and storage. In order to address this issue, a sliding window technique is used in [84] where a fixed size queue holds the values used to calculate the mean and variance. When the queue is full and a new value arrives, the oldest value in the queue is deleted and the new value is inserted. Since there are no fixed guidelines to determine an optimal queue size, we propose a dynamic queue structure that adapts according to the sequence of values. The modified mean-variance calculation process is illustrated in Figure 12. At any given time, the interaction between the queue and the new value can be described using three distinct states. In the first state, the queue is empty when a new value is generated. In this case, the queue is initialized with the new value as its member. The mean is initialized as the value itself and the variance is set to zero. In case the queue is non-empty and a new data value is received, this new value is checked with the current mean and variance. This basically involves checking whether the data value lies in the interval given by (μ – δ, μ + δ). This interval is used to approximately predict the range that the future data values are expected to fall into. If the new data value is found to be within this range, the mean and variance values are not changed and the new value is not added to the queue. The fact that the new data value lies within the interval is taken to be an indication that the data is still changing according to the expected pattern which is the range of values given by the mean-variance interval. This comprises the second state of the queue.



Case 1: Empty queue

New Value: 27

Action: Initialize queue, mean and variance

27

Mean = 27
Variance = 0

Case 2: Non-empty queue, value in the interval

New Value: 26.6

Current queue

26	28	27.5

Mean = 27.16
Variance = 1.08
Interval: 26.09 – 28.23

Action: No change required

Case 3: Non-empty queue, value not in the interval

Current queue

26	28	27.5

Mean = 27.16
Variance = 1.08

Scenario A New Value: 25

New mean = 26.6
New variance = 1.8

Change in variance small

Action: Add new value to queue, update mean and variance

Scenario B New Value: 37

New mean = 29.6
New variance = 17.1

Change in variance large

Action: Reinitialize queue, mean and variance

Figure 12. The mean-variance calculation process

Even though the latest data value is not added to the queue, it is still available for retrieval for any application that requires it. Finally, if the new data value is observed to lie outside the current mean-variance interval, the mean and variance values are recalculated taking the new data value into account. The change in the variance values are then compared against a pre-defined threshold to assert whether the data pattern is changing. A change in the data pattern in this context indicates a sharp increase or decrease in the magnitude of the generated data values. A large change in the variance is taken to be indicative of a shift in the data change pattern and results in the queue being reinitialized and restarting the mean-variance calculation process. In case the variance change is still within the threshold, the new data value is added to the queue (as it contributes to the new mean and variance calculations) and the mean-variance values are updated accordingly.

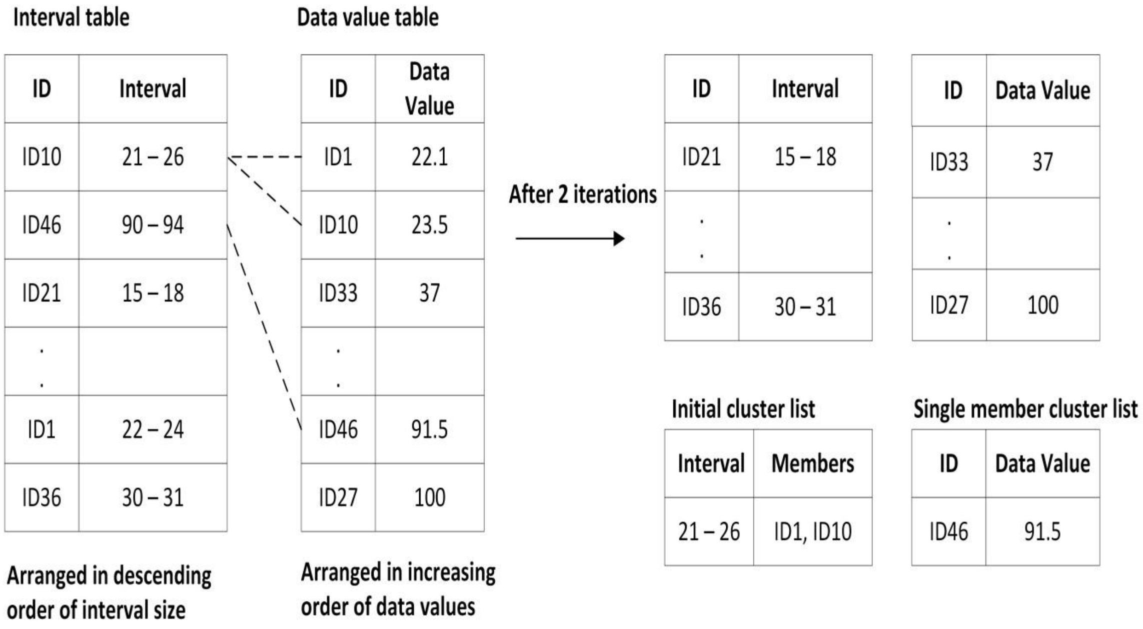
Since the mean-variance calculations are done on a subset of the data, the processing and storage load is reduced. The threshold value used to check the change in variance values needs to be chosen carefully. In case the threshold value is high, large changes in variance will be tolerated and consequently, the mean and variance values may not be truly reflective of the current data distribution pattern. Conversely, choosing a very low threshold value will result in frequent updating of the mean and variance. The threshold value can be customized according to the type of data attribute being indexed and needs to be consistent for all occurrences of that attribute. As we shall see in the subsequent sections, this mean-variance calculation process is used in the cluster construction process as well as by the PSGs to determine if an index update operation is required.

5.4 Constructing an index based on the mean and variance value

5.4.1 The index creation process

The proposed index structure utilizes the mean and variance values calculated within each PSG to partition the PSGs into multiple range clusters. The basic idea of the indexing process is similar to the previously discussed range clustering scheme. Again, the scope of this indexing scheme is within a single semantic cluster within a particular context domain. Initially, all the PSGs are assigned to a single cluster when no clusters have been defined. This process is continued till the size of the cluster exceeds a system-defined threshold for the maximum cluster size which is when the clustering algorithm is invoked. The first step in the clustering process consists of the middleware retrieving the current value from every PSG within the cluster for the following parameters: the mean-variance intervals ($[\mu-\delta, \mu+\delta]$), the data value for the context attribute and a unique identifier. This identifier is assigned by the middleware to the PSG during registration. These values are stored in separate tables and are indexed using the identifier. The first step in the clustering process involves the partitioning of the PSGs into a set of initial clusters. This step is illustrated in detail in Figure 13.

In order to identify the initial clusters, the mean-variance intervals are used to represent the cluster bounds and the data values falling within a particular mean-variance range are assigned to the same cluster. Since these intervals represent the pattern of data change within a PSG, they are used as the cluster bounds to ensure that the PSGs with similar data distribution patterns are clustered together. As we observed, the previous indexing scheme was not able to always reflect the data distribution within a semantic cluster accurately. We attempt to address this problem by choosing the mean-variance intervals to form the clusters instead of the data values. Also, we try to minimize the index updates occurring due to the dynamicity of data by clustering PSGs with similar data change patterns.



Interval table

ID	Interval
ID10	21 – 26
ID46	90 – 94
ID21	15 – 18
. .	
ID1	22 – 24
ID36	30 – 31

Arranged in descending order of interval size

Data value table

ID	Data Value
ID1	22.1
ID10	23.5
ID33	37
. .	
ID46	91.5
ID27	100

Arranged in increasing order of data values

ID	Interval
ID21	15 – 18
. .	
ID36	30 – 31

ID	Data Value
ID33	37
. .	
ID27	100

Initial cluster list

Interval	Members
21 – 26	ID1, ID10

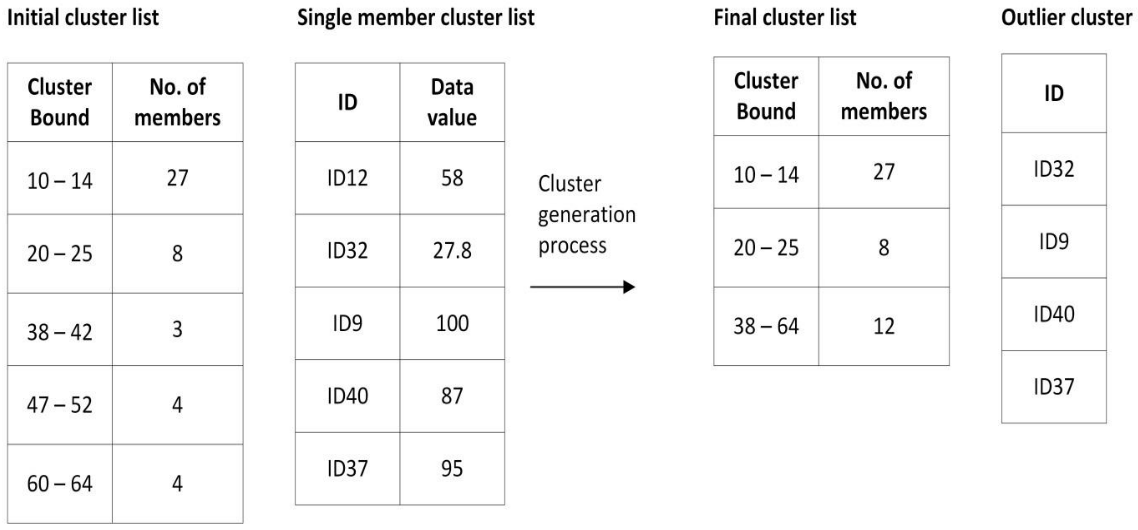
Single member cluster list

ID	Data Value
ID46	91.5

Figure 13. The identification of the initial clusters

The main step in the clustering process is as follows – a mean-variance interval is selected from the table and a cluster having a range bound equal to this interval is initialized. Subsequently, all the PSGs having a data value falling within this range are selected from the data value table and assigned to this cluster. In order to increase the efficiency of this operation, the interval table is sorted in decreasing order of the interval sizes. Since the larger sized intervals are expected to accommodate more values, the use of the sorted list can lead to a faster matching of the data values to the corresponding clusters. Similarly, the table containing the data values is sorted in increasing order so that the subset of the table that needs to be checked against a given interval can be easily identified. Once a data value is assigned to a cluster, the identifier is used to delete the corresponding entries from both the tables so that these entries are not considered during further iterations. This process is continued until both the tables are empty. If the only data value falling within a particular interval is found to be the one corresponding to the same PSG, the PSG is added to a separate list denoted as the single member cluster list. This case is illustrated in Figure 13 using the PSG with identifier 'ID46'.

After the initial clusters have been identified, the next step involves the generation of the final range clusters. Since there can be a large number of small sized clusters generated during the initial phase, some of 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. The final cluster generation process is described in Figure 14.



Initial cluster list

Cluster Bound	No. of members
10 – 14	27
20 – 25	8
38 – 42	3
47 – 52	4
60 – 64	4

Single member cluster list

ID	Data value
ID12	58
ID32	27.8
ID9	100
ID40	87
ID37	95

Final cluster list

Cluster Bound	No. of members
10 – 14	27
20 – 25	8
38 – 64	12

Outlier cluster

ID
ID32
ID9
ID40
ID37

Figure 14. The generation of the final clusters

The figure shows a sample set of initial clusters and single member clusters obtained from the first round of clustering. The first step in the cluster generation process is to identify the clusters for merging from the initial cluster list. A cluster is merged with its adjacent one 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 and minimize the chance of a subsequent cluster split. After the final clusters are generated, the extent of the merged cluster bounds will increase and some of the PSGs in the single member cluster list can be allotted to the generated clusters. If the single member cluster list still contains some members after this operation, these indicate data values that cannot be assigned to any of the clusters. In a typical clustering scenario, these values would be considered as outliers. However, as they still represent valid values in our case that cannot be discarded, we need to make provisions to account for these values as well. In order to handle these values, we designate a special cluster known as an outlier cluster which 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 PSGs register with their allotted clusters and each of these clusters forms a separate P2P network.

5.4.2 Analyzing the clustering process

The main step in the clustering process involves the assignment of the data values to the mean-variance ranges and generating the initial set of clusters. An iteration of the initial cluster identification process picks the first range from the mean-variance range table and assigns matching data values to it from the data value table until a mismatch occurs. The matching entries are removed from the corresponding tables and the next iteration is initiated. Thus, the size of the table keeps decreasing after each iteration. Since the data value table is sorted in increasing order, a variation of binary search can be used to find the starting point in the table to initiate the matching process. Since the performance of the algorithm depends on the number of matches happening in an iteration, we assume this parameter to be a fixed constant for our analysis purpose. We fix the number of values matched during each iteration as *n* and assume it to be a finite multiple of the total size of the data value table *N*. In other words, we fix N = k . n where k is an arbitrary constant. The running time for the first iteration can now be written as log N + n where the first term refers to the time to locate the starting point for the match and the second term refers to the comparison operations needed to assign the n values to the cluster. Since the table sizes change after each iteration the total running time can be written as

$$
\begin{aligned}
&(\log N + n) + (\log (N-n) + n) + (\log (N-2n) + n) + \ldots. + (\log (n) + n) \\
&= \log N + \log (N-n) + \ldots + \log n + k \cdot n \\
&= \log n + \log 2n + \log 3n + \ldots + \log kn + k \cdot n \text{ (Replacing N by k.n)} \\
&= \log (k! \cdot n^k) + k \cdot n \\
&= \log k! + k \log n + k \cdot n
\end{aligned}
$$

The final running time can be taken to be the order of log k! assuming k > n. Since log k! is known to have a tight bound of k log k, we can infer that the runtime complexity of the clustering process is θ ((N/n) log (N/n)) by replacing k with (N/n). In reality, the complexity depends on the values matched during each iteration. Although we considered this number to be a fixed constant for our analysis, this can vary during an actual execution of the clustering process. The worst case scenario for the clustering process occurs when each value in the mean-variance range table matches only a single value in the data value table where both these values correspond to the same PSG. In this case, the value of n is 1and the total running time can be written as log N + log (N – 1) + … + log1 which can be simplified to log (N!). Based on the previous analysis, the runtime complexity for the worst case scenario can be taken to be θ (N log N) and the running time in this case is dependent on the size of the data value table.

5.4.3 Index maintenance operations

In order to maintain the index structure, the following information needs to be stored in the middleware – a list of the current range clusters with their range bounds and the list of members present in each range cluster. This information is maintained and updated at the corresponding context domain gateways (CDG). After the initial clusters have been identified, subsequent PSGs are assigned to a cluster depending on their attribute values and the current range clusters. Since there can be an overlap between the cluster bounds, it may be possible to assign a PSG to multiple clusters. In this case, the PSG is assigned to the cluster with the lower membership size. If a PSG cannot be assigned to any existing cluster, it is assigned to the outlier cluster. The outlier cluster is similar to a normal range cluster but does not have any fixed range bounds.

Whenever the membership size of a cluster or an outlier cluster exceeds the maximum size threshold, the clustering process is invoked within that cluster and a new set of range clusters is generated. Similar to the indexing scheme discussed in Chapter 3, this process ensures that the index structure adapts itself as PSGs join and leave the system. The clustering process is slightly modified when invoked on the outlier cluster. Since the outlier cluster has no range bounds, PSGs assigned to an outlier cluster will remain there is no way to detect an inconsistency between their current data value and the cluster bounds. As context data is dynamic, there is a possibility that a member of the outlier cluster can be later assigned to an existing cluster. Hence, we perform this check prior to the initiation of the clustering process and assign a PSG to an existing cluster whenever possible. Even though the initial cluster bounds may not be very fine-grained, the clustering performance is expected to improve as more data values are received. Similar to the previous scheme, clusters with low membership sizes are merged to keep the number of clusters low and clusters with no members are removed from the system.

After a PSG has been assigned to a cluster, it is possible for it to move to a different cluster if its data pattern changes. The mean-variance calculation process is utilized to make the decision of switching to a different cluster instead of an arbitrary periodic checking. In case a large change in the variance value is detected, this is taken to be indicative of the fact that the data pattern is changing. Subsequently, a PSG checks if a cluster update operation is needed. If the data change pattern is gradual, such large shifts in variance will be infrequent. It needs to be noted here that depending on the range cluster bounds, even a large shift in the variance values may not necessitate the movement of a PSG to another cluster. As long as there are no updates required to the mean-variance values or the variance change is within the threshold, we assume that the PSG is still within the correct range cluster. This is a reasonable assumption as the clustering process is carried out based on the mean-variance values. This is again an improvement as compared to the previous range clustering scheme which could result in query inaccuracies due to a mismatch between the data value of a PSG and its range cluster bounds.

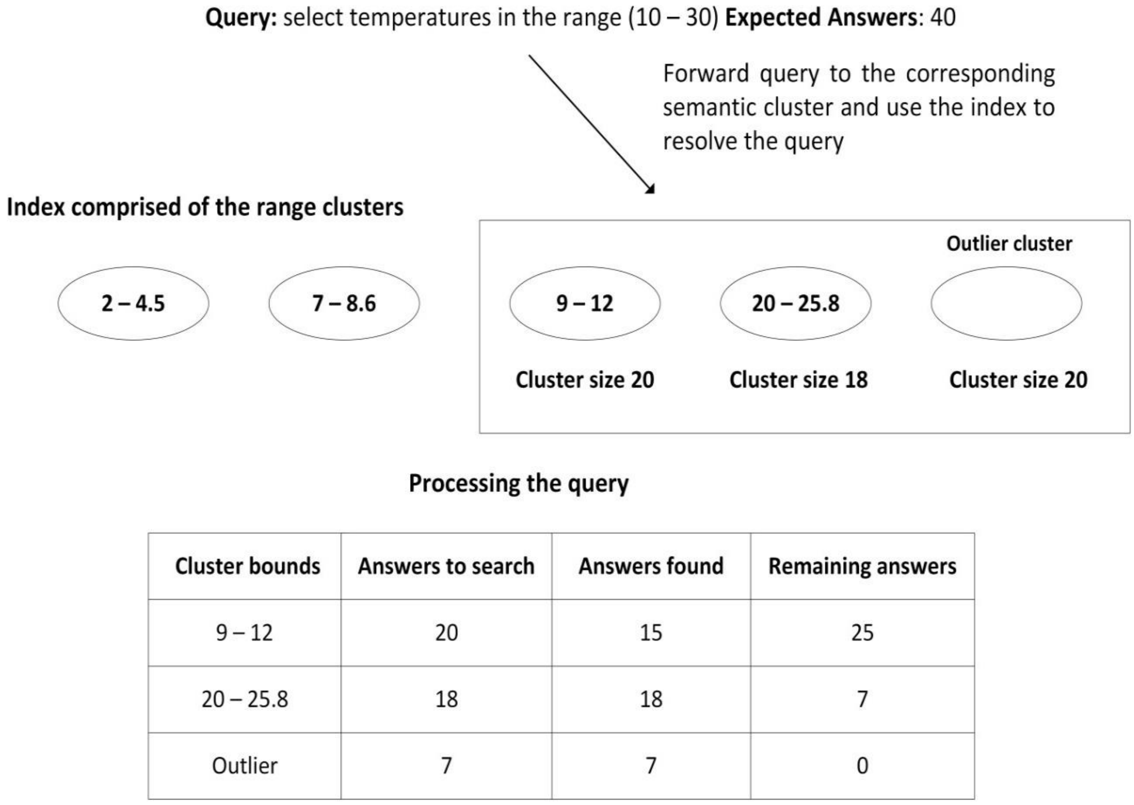
5.4.4 Handling the special cases during the cluster creation process

As we saw in the previous section, the first phase of the clustering process involves the identification of the initial clusters. The subsequent step of generating the final clusters is based on the assumption that the first phase generates at least two separate clusters. However, depending on the data values, this may not always be possible. One of the possible scenarios is that either all or a large percentage of the data values are assigned to a single cluster. Conversely, the second scenario refers to the case when the majority of the data values are part of the single member cluster list. We briefly discuss the process of handling these two special cases in this section. The first scenario occurs when the first mean-variance interval selected as part of the clustering process is capable of accommodating a large percentage of the data values. This can happen due to two reasons – either all the PSGs within the cluster have very similar data distribution patterns or the size of the selected interval is extremely large. In this case, we restart the clustering process and select a different interval with a lower interval size to begin the clustering. This new interval is usually selected from the middle of the interval table and promoted to be the first value of the interval table. The previously selected interval is assigned to the single member cluster list. The rest of the clustering process is carried out as usual. However, if the use of the new interval also leads to the formation of a single cluster, we utilize our previously discussed technique of partitioning the sorted list of data values to generate the final clusters.

Similarly, the second case occurs when the PSGs have rather varying data distribution patterns and most of them end up being part of the single member cluster list. In this case, we can either use the partitioning of the data values to generate the clusters or assign all the PSGs to the outlier cluster. If the partitioning technique is used, the interval size of the cluster bounds of the resulting clusters is expected to be large due to the variation in the data distribution. However, if we assign all the values to the outlier cluster, it can result in an immediate cluster split operation due to the increase in the membership size of the outlier cluster. In order to avoid this situation, we choose to use the partitioning technique to generate the new clusters.

5.5 Context lookup using the index structure

As discussed previously, the context lookup operation in Coalition is carried out by issuing SQL style queries that specify the data to be acquired using range-search conditions. Since the query issuer is not expected to know the number of PSGs satisfying the query condition, we allow each application to submit the number of answers expected by it for a query. When a context lookup query is received by the middleware, the query processor parses the query and redirects the query to the appropriate context domain and 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. In case the query is not confined in a single cluster, the outlier cluster is included as part of the candidate cluster set. As an outlier cluster does not have any defined range bounds, it is necessary to include it in the candidate cluster list for the sake of completeness. The query processing operation is depicted in detail in Figure 12.


Query: select temperatures in the range (10 – 30) **Expected Answers**: 40
Forward query to the corresponding semantic cluster and use the index to resolve the query
Index comprised of the range clusters
2 – 4.5
7 – 8.6
9 – 12
20 – 25.8
Outlier cluster
Cluster size 20
Cluster size 18
Cluster size 20
Processing the query

Cluster bounds	Answers to search	Answers found	Remaining answers
9 – 12	20	15	25
20 – 25.8	18	18	7
Outlier	7	7	0

Figure 15. Context lookup using the index

The processing of a query within a single range cluster is controlled using the number of answers received and a system-defined timeout that controls the duration for which a query is processed within the cluster. Since the query processing within a single cluster is carried out by flooding, the time taken to lookup a query will depend on the cluster size. The response time for a query (single attribute) will hence depend on both the number of clusters being queried as well as their size. 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 Figure 15. 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. The answers for a query can either be returned to the query issuer directly or to the query processor. Currently, we choose to report all the answers to the query processor component which aggregates all the responses and forwards to them the query issuer as per our current design.

As discussed in Section 4.2.3, the set of answers received for a context lookup query depends on the number of PSGs that have the valid answer and the PSGs among this set that choose to respond to the query. Additionally, the dynamicity of the context data can also affect the number of responses received by a query. For example, consider a PSG member of a range cluster that is a part of the candidate cluster list for a given query. If this PSG moves to another cluster while the query is being propagated in the original cluster, there is a possibility that the query will not be received by this PSG. Also, the P2P network corresponding to a range cluster will need to be updated when a PSG leaves or joins the cluster. In case the leave/join operation is very frequent, the PSGs will be occupied in the network updating operation which can also affect the propagation of a query within the cluster.

The frequent leave/join operation will usually be exhibited by PSGs whose data distribution pattern can be characterized as bursty or uneven wherein the change in the generated data values is not gradual. Rather, these changes happen abruptly and the magnitude of the change is also large. The mean-variance calculation process in such PSGs will be characterized by the frequent re-initialization of the queue and the mean/variance values. One approach that can be used to minimize the disruptions caused due to such PSGs is to assign them to the outlier cluster. Since the outlier cluster is always included as part of the candidate cluster set during query processing, the query correctness is maintained. In order to give applications an idea about the quality of the received answers, the query processor can return the size of the candidate cluster set along with the answers. The application can compare these two values to make an informed decision about the received answers and determine if the query should be reissued.

5.6 Experimental analysis

5.6.1 Experimental setup

The experimental setup followed here is similar to the one discussed in Section 4.3.1 which we reiterate here for clarity. The augmented version of the index structure is integrated with the prototype of the Coalition middleware which was deployed on a Dell PowerEdge T300 server with four 2.83 GHz quad-core Intel Xeon CPU. A set of desktop machines each running the Windows XP operating system and having an Intel Core 2 Duo 2.83 GHz CPU is used to simulate the PSGs. Since a PSG is designed as a software module, a single desktop can be used to simulate multiple instances of the PSG. The maximum and minimum cluster size thresholds are fixed at 50 and 10 respectively.

As discussed in Section 4.3.1, the data values used for testing were derived from the Intel lab sensor data set [64] and the same technique discussed therein was utilized to generate a set of data files that simulated the set of data values produced at a PSG. We briefly reiterate the data file generation process here. Since the test data set contained different types of sensor data from 54 sensors, the first step involved the generation of 54 data files for each type of data present in the set. The test data files were generated by taking a random sample comprising of a fixed number of values from one of the files obtained in the previous step.

For example, to model a set of temperature values at a PSG, a data file corresponding to the temperature attribute was randomly chosen and a fixed number of values was randomly selected from that file. The generation of the test data values was carried out in a random fashion to minimize the chances of data replication as well as reduce the impact of data correlation on the experimental results. Each of the simulated PSG instances was assigned a set of these data values. The process of a PSG periodically receiving a new value for a data attribute was simulated by reading these values one by one after a fixed time interval. The mean and variance calculations were carried out within each PSG on the set of its assigned data values. All the experimental results are represented using a 95% confidence interval wherever applicable.

5.6.2 Query response time

We conduct the initial set of experiments to measure the performance of the proposed index structure with respect to the query response time. The experiments for the query response times were carried by varying the network size as well as the valid answer set size. The network size refers to the total number of PSGs registered with the middleware whereas the valid answer set size refers to the number of PSGs having the valid answer for a particular query. The first set of results provide a comparison of the query response times achieved using the three organizational schemes – flooding, range clustering and mean-variance based indexing. The results are illustrated in Figure 16. This set of experiments was conducted in a stable system state where a set of PSGs were partitioned into a set of range clusters using the different techniques and there was no movement of the PSGs between the clusters. Also, the value used to assign the PSGs to the clusters remained constant. In order to simulate the mean and variance values required to build the index, a random sample of a sequence of data values was taken from the data file associated with the PSG. The mean and variance values were calculated over this sample set using the algorithm discussed earlier and the last value in the set was taken to be the current data value of the attribute.

Each PSG instance was assigned a triad of values in the form of [mean, variance, data value] which was used for the clustering process. The previously discussed range clustering scheme was constructed using the same set of data. The valid answer set size in this case was fixed to be 150 and the response time measurements were averaged over 300 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. 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 the results.

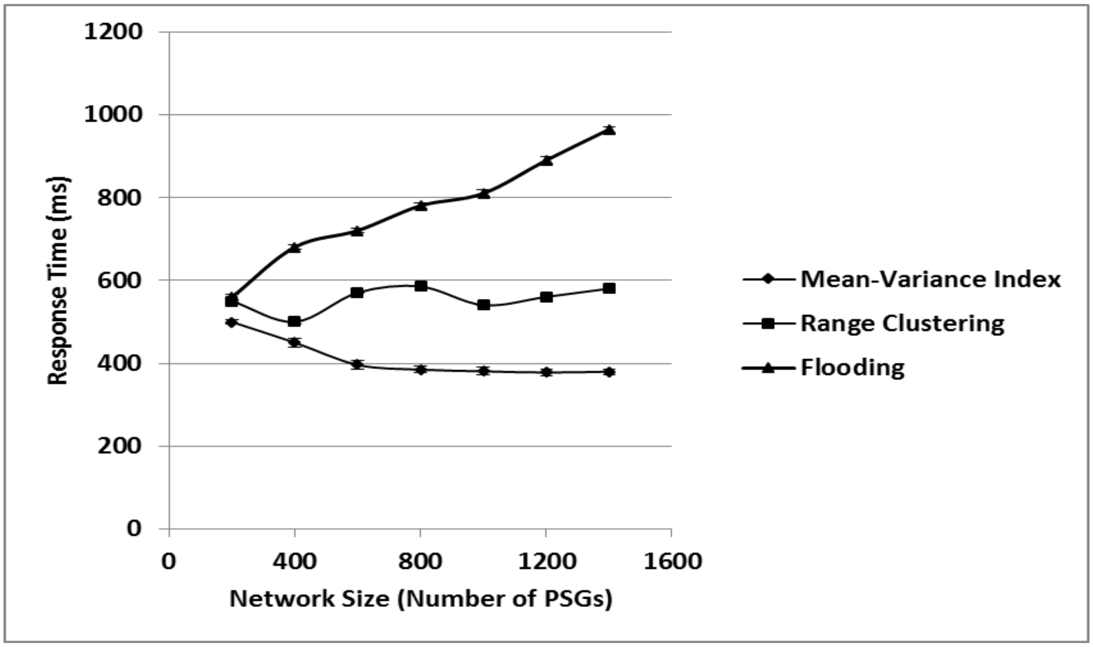


Figure 16. Comparison of query response time for the different schemes

As we can see from the response time measurements, the mean-variance based clustering scheme achieves a reduction of 40% to 60% in the response time as compared to the flooding approach which is an expected observation due to the classification imposed by the indexing scheme. Interestingly, it can also be observed that the mean-variance based index scheme fares better than the previously discussed range clustering approach by providing a reduction of 10% to 35% in the response time. As discussed in Section 4.3.3, one of the drawbacks of the range clustering scheme was that it was not always able to reflect the data distribution within a semantic cluster accurately. Since the mean-variance index uses the mean-variance range values to classify the PSGs, it can generate a set of clusters that portray the data distribution within the semantic cluster more accurately. This in turn enables a faster localization of queries thereby leading to a decrease in the query response time. 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 Figure 17.

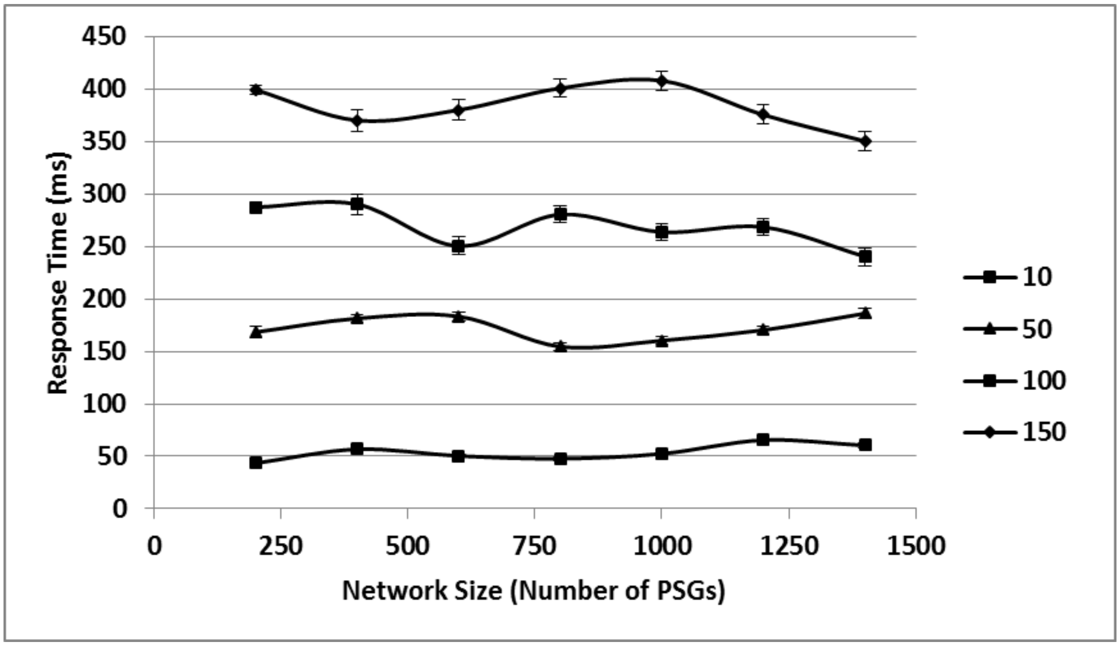
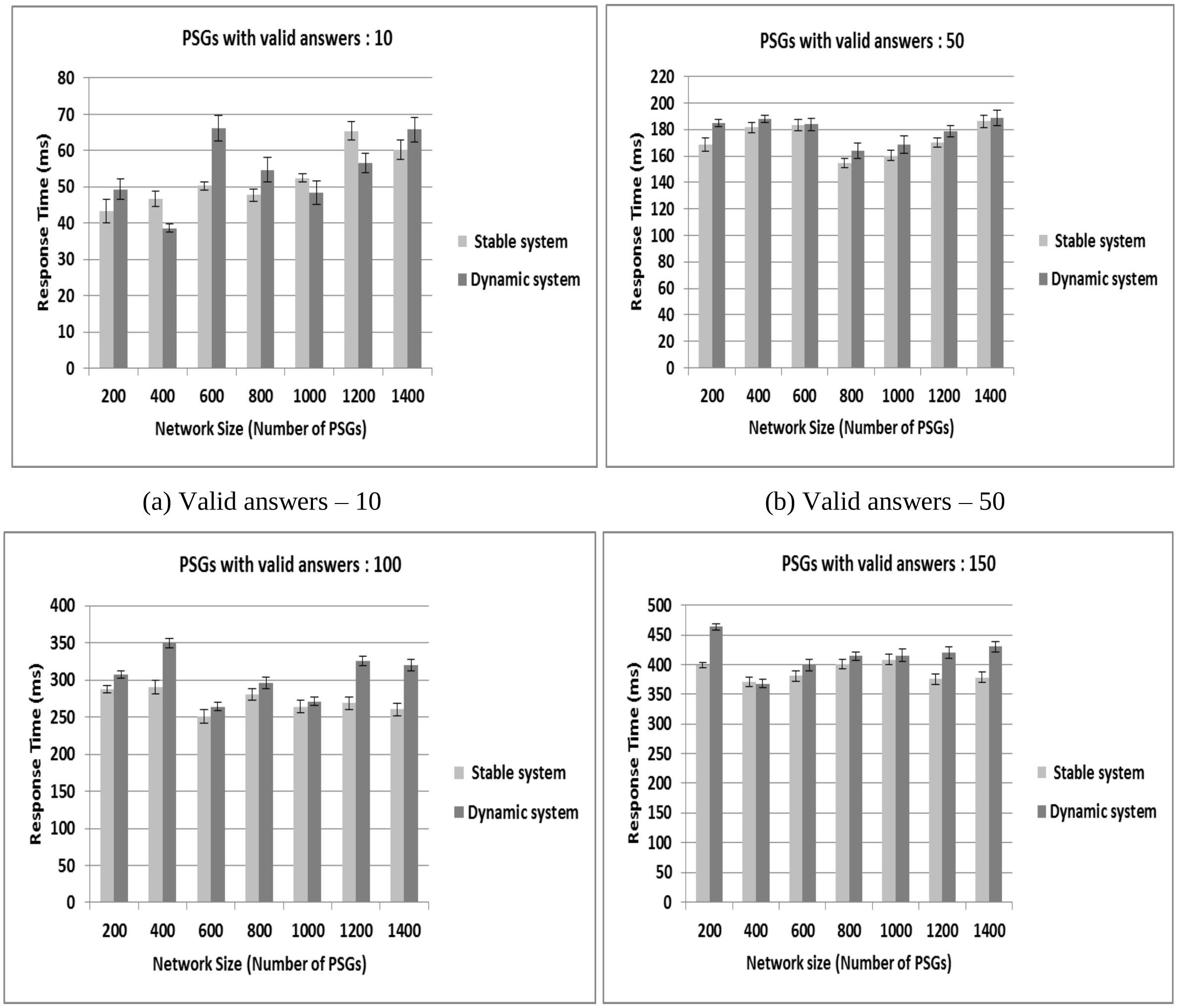


Figure 17. Comparison of query response time with different answer set sizes

The results achieved show that the response time increases with an increase in the query window sizes which is expected. Also, the variations observed in the response time to search for a fixed number of answers when the network size is increased are quite low. As we discussed earlier, the response time for a query depends on the number of range clusters to be looked up as well as the size of each of these range clusters. The minimal variations in the query response time indicate that the localization performance of the index remains consistent with the increase in the network size and it is able to redirect the context query to the relevant set of PSGs. These preliminary results highlight the fact that the mean-variance based index structure provides a better performance with respect to the query response time as compared to the previous range clustering scheme and the change in the response time is stable with an increase in the network size. We proceed to discuss the performance of the mean-variance index with dynamic data values in the subsequent sections.

5.6.3 Query response time with dynamic data

In this section, we examine the system performance when the data values corresponding to the PSGs are dynamic. The change in the data values at a PSG is simulated by reading a new data value periodically from the data files associated with each PSG instance. Each PSG carries out the mean-variance calculation process when a new data value is received and the time interval for the data generation is chosen to be 1 second. We initially study the effect of the dynamicity of the context data on the query response time. Figure 18 compares the response times achieved using a stable system state and a dynamic system environment. The results are averaged over 300 query runs where each query is a range query of the form (A, B). We vary the number of PSGs that have the valid answers as well as the network size.



(a) Valid answers – 10

(b) Valid answers – 50

(c) Valid answers – 100

(d) Valid answers – 150

Figure 18. Comparison of query response times for stable and dynamic system states

The experiment setup is similar to the one described previously with the only exception being that the data values within the PSG are dynamic and the PSGs can leave their current cluster and join a new cluster. As the results indicate, the response times achieved using both the schemes are mostly comparable and differ by a margin of 2% to 8%. In order to understand the implications of this observation, we need to examine the effects of the dynamicity of data on the index. Since the context data is not stored in the middleware, each change in the underlying data values does not need to be communicated to the index. Rather, the updating operation of the index is restricted to the leave/join operations of a PSG when its local data value is inconsistent with its assigned cluster bounds. The range cluster splitting and merging operations that occur due to the movement of the PSGs are also considered as part of the updating operation of the index.

The update operations of the proposed index structure can affect the query operation if they occur in a range cluster when a query is being processed in that cluster. For example, if some PSGs leave a range cluster during the query processing, it can result in a set of missed answers for the query. Also, the P2P network of the range cluster will need to be updated in case of a PSG leave/join which can also result in missed answers. As discussed in Section 5.5, we use the number of matching answers found in a particular range cluster as one of the factors to limit the query propagation. So, if a PSG(s) leave the cluster during query processing, the query processor can keep looking for matching answers based on the previous size of the range cluster. Also, depending on the query parameters and the set of clusters, a set of missed answers in a cluster can cause the query processor to look for answers in other clusters. Both these cases can increase the query response time. So, if the frequency of PSGs leaving/joining a range cluster as well as that of cluster split/merge operations is low, there will not be a significant impact on the query response time even when the underlying data is dynamic. We now study the effect of the variation of the data generation frequency in the PSGs on the response time. The data generation frequency refers to the periodic interval after which a new data value for an attribute is received by a PSG. In this set of experiments, we vary this frequency from 200 ms – 1 second and study the response time over different network sizes for a valid answer set size of 100. We also study the total number of cluster split/merge operations occurring for a given network size and data generation frequency combination. The obtained results are illustrated in Figure 19 and 20.

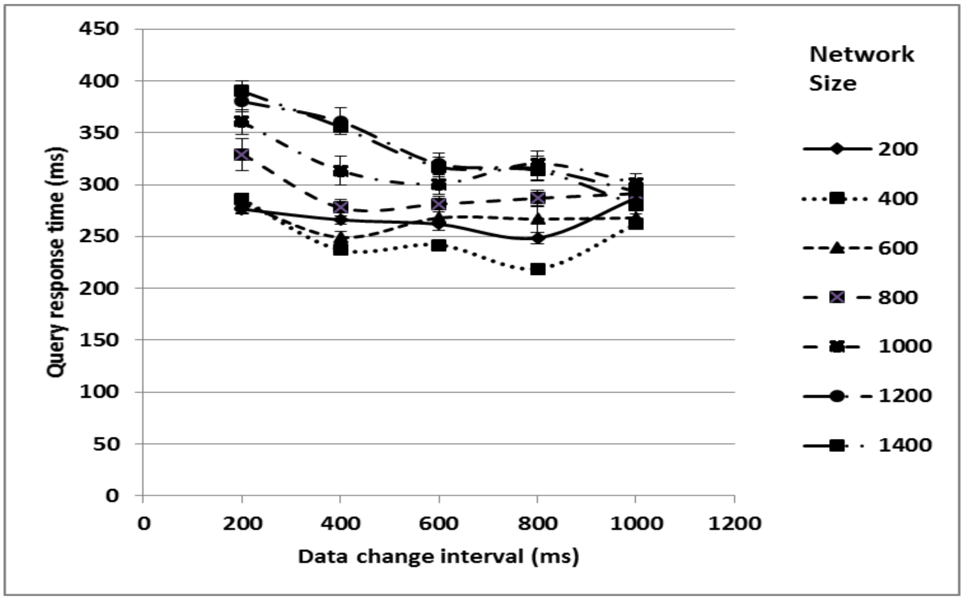


Figure 19. Variation of query response time with data change frequency

The main observation from the results is that the query response time is higher when the time interval between the generation of successive new data values is low. This variation is more noticeable in the larger network sizes. To cite an example, for the network size of 1000, the response time achieved with a data change frequency of 1 second is around 27% faster than with a frequency of 200 ms.

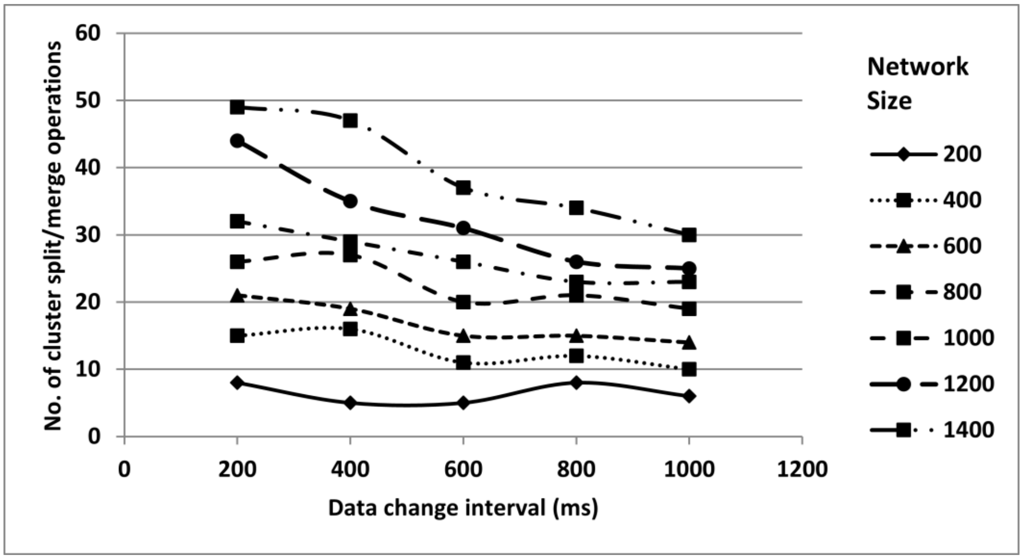


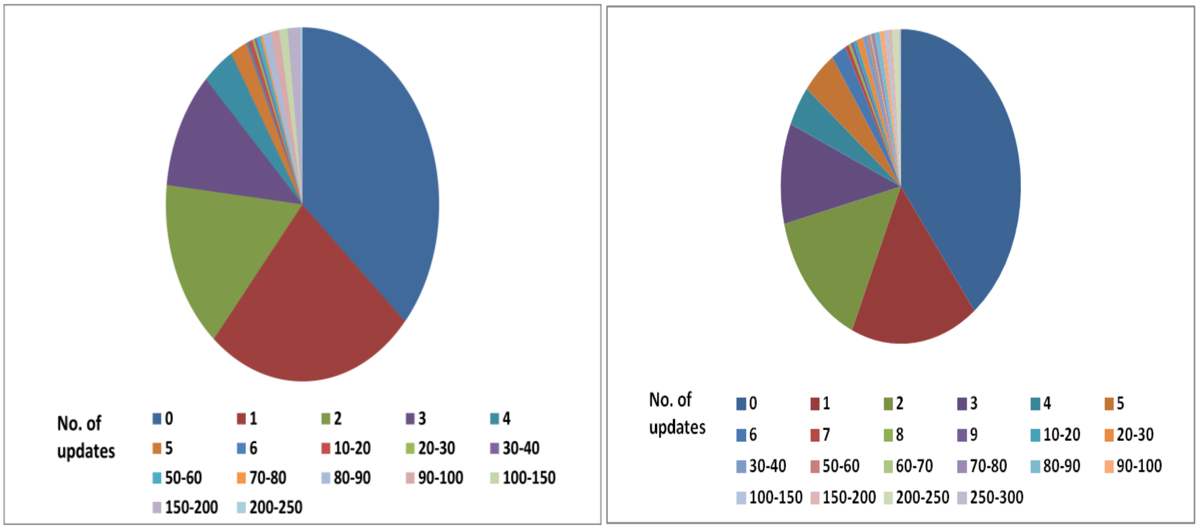
Figure 20. Variation of cluster splits/merges with data change frequency

Further, as we can observe in Figure 20, the variation in the number of cluster split/merge operations along with the data change interval also exhibit a change pattern similar to that observed for the query response times. This indicates that the number of cluster maintenance operations are higher when the frequency of data change is high (we only consider the time interval between the data value change and not the actual magnitude of the change) which affect the query response time. A high frequency of data change can also result in queries not receiving matching answers from certain PSGs (depending on the query parameters) as the local data values at a PSG become outdated very soon which can also cause an increase in the query response time.

Based on the experimental results in this section, we can conclude that the dynamicity of the underlying data does not have a significant effect on the index performance with respect to the query response time if the data change frequency as well as the number of cluster maintenance operations is low. Also, the variations observed in the response time with the increase in the network size are also consistent with the results achieved using a stable system state. The effect of a high data change frequency on the response time is noticeable especially for the larger network size. Since the occurrence of a cluster split/merge is linked to the data change happening within a PSG, we can solve this problem by using the PSG of a physical space to enforce a uniform and suitable data change frequency for the different context sources affiliated with it. The experimental results also indicate that the proposed indexing structure is fairly scalable with respect to the query response time for small to moderate network sizes even when the underlying data is dynamic. Based on the trends observed in the variations of the query response times, we can expect that the changes in the query response time when the network size is further increased will be minimal provided that the percentage of PSGs having uneven data distributions is low. However, if we consider extremely large network sizes associated with wide area network scale deployments, the proposed index structure would result in the formation of a large number of range clusters. In this case, the maintenance overhead of these clusters can outweigh the indexing benefits achieved due to the clustering process. This problem will occur when the indexing is applied across the system scope i.e. all PSGs are associated with a single index structure. We shall discuss the possible techniques to avoid this situation and impose additional indexing scopes in Chapter 7.

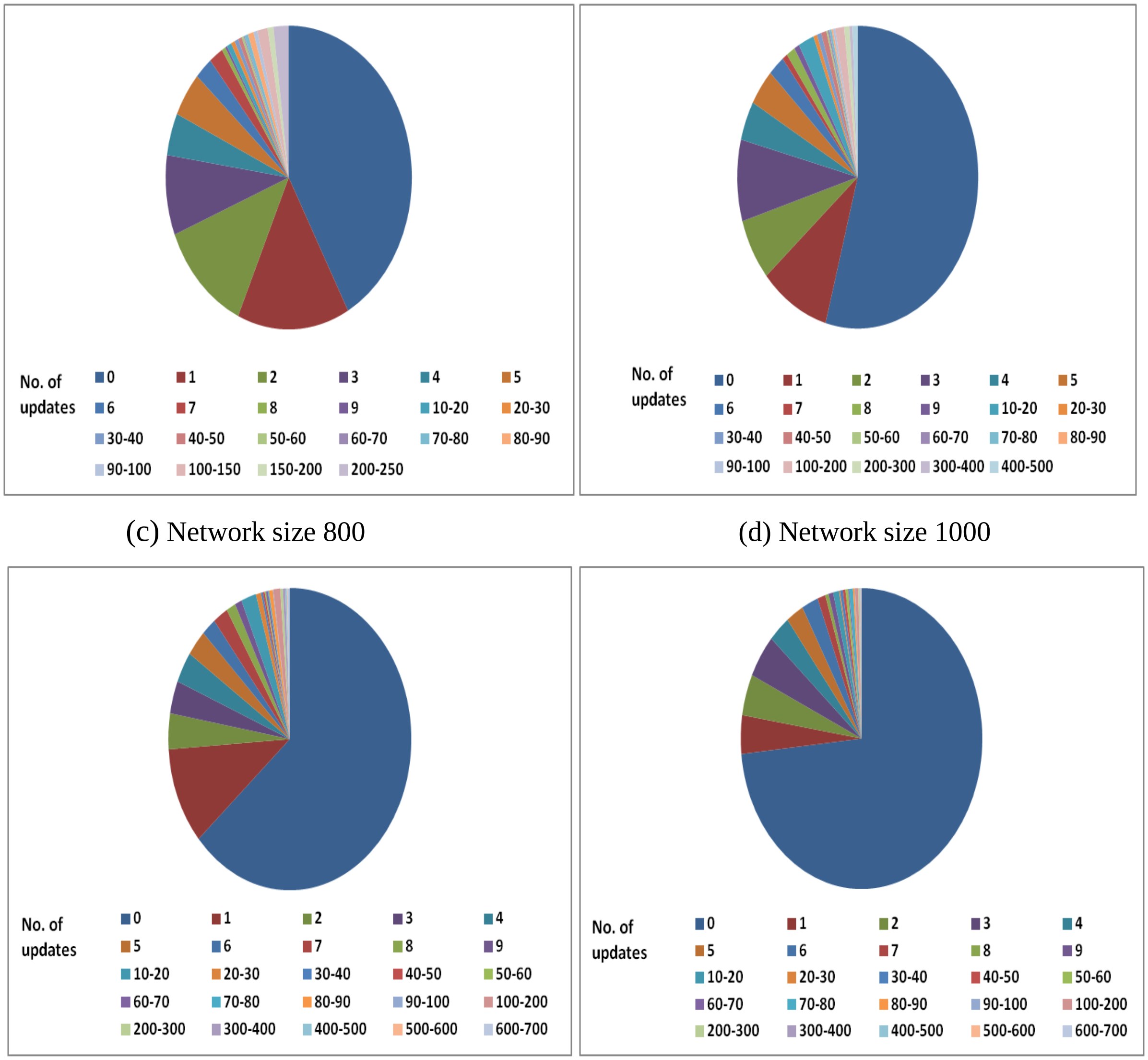
5.6.4 Index performance with respect to update operations

One of the primary constraints in designing an index structure for dynamic data is to minimize the number of update operations occurring due to the frequent changes in the data. The index update in our proposed index comprises of PSGs leaving their current range cluster and joining a new one due to the inconsistency of their data values with their assigned cluster bounds. In this section, we study the number of update operations occurring in our proposed index over a period of time. The experimental setup is as follows – a fixed number of PSGs were registered with the Coalition system and observed for the period of 1 hour. Each time a PSG leaves a range cluster and joins a new one, it is recorded as a single update operation along with the unique identifier of the PSG. We carry out the experiments using the temperature, humidity and light attribute values derived from the Intel lab test data set. Since the results obtained are comparable, we only illustrate the results corresponding to the temperature attribute here for the sake of brevity. In the first set of results we measure the number of times each PSG executes an update operation and then group the PSGs according to the number of update operations. The results are illustrated in Figure 21 for the different network sizes ranging from 400 – 1400 and plot the percentage of PSGs executing a certain number of update operations.



(a) Network size 400

(b) Network size 600



(c) Network size 800

(d) Network size 1000

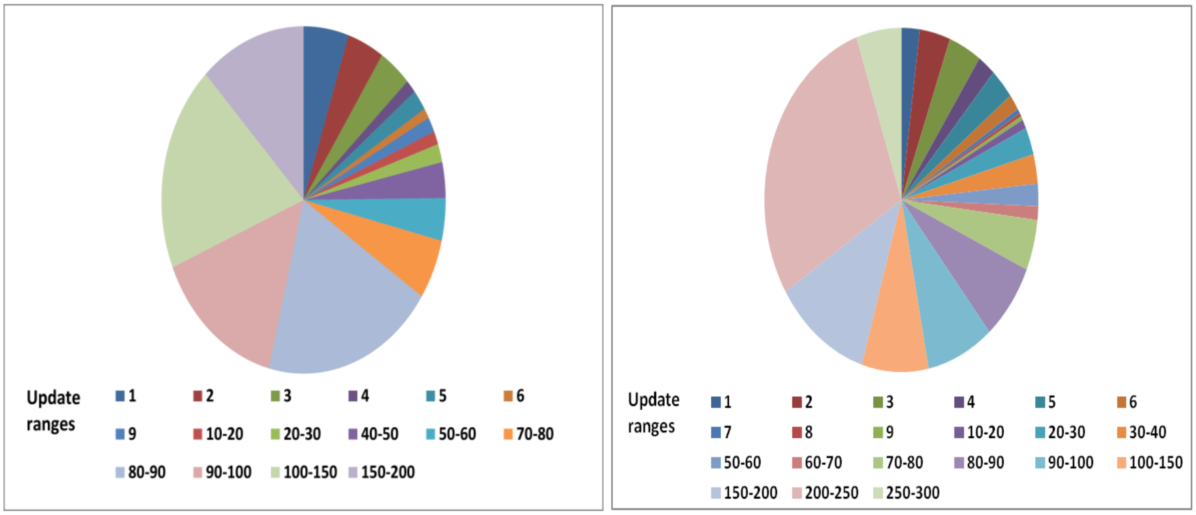
(e) Network size 1200

(f) Network size 1400

Figure 21. PSG update operations for different network sizes

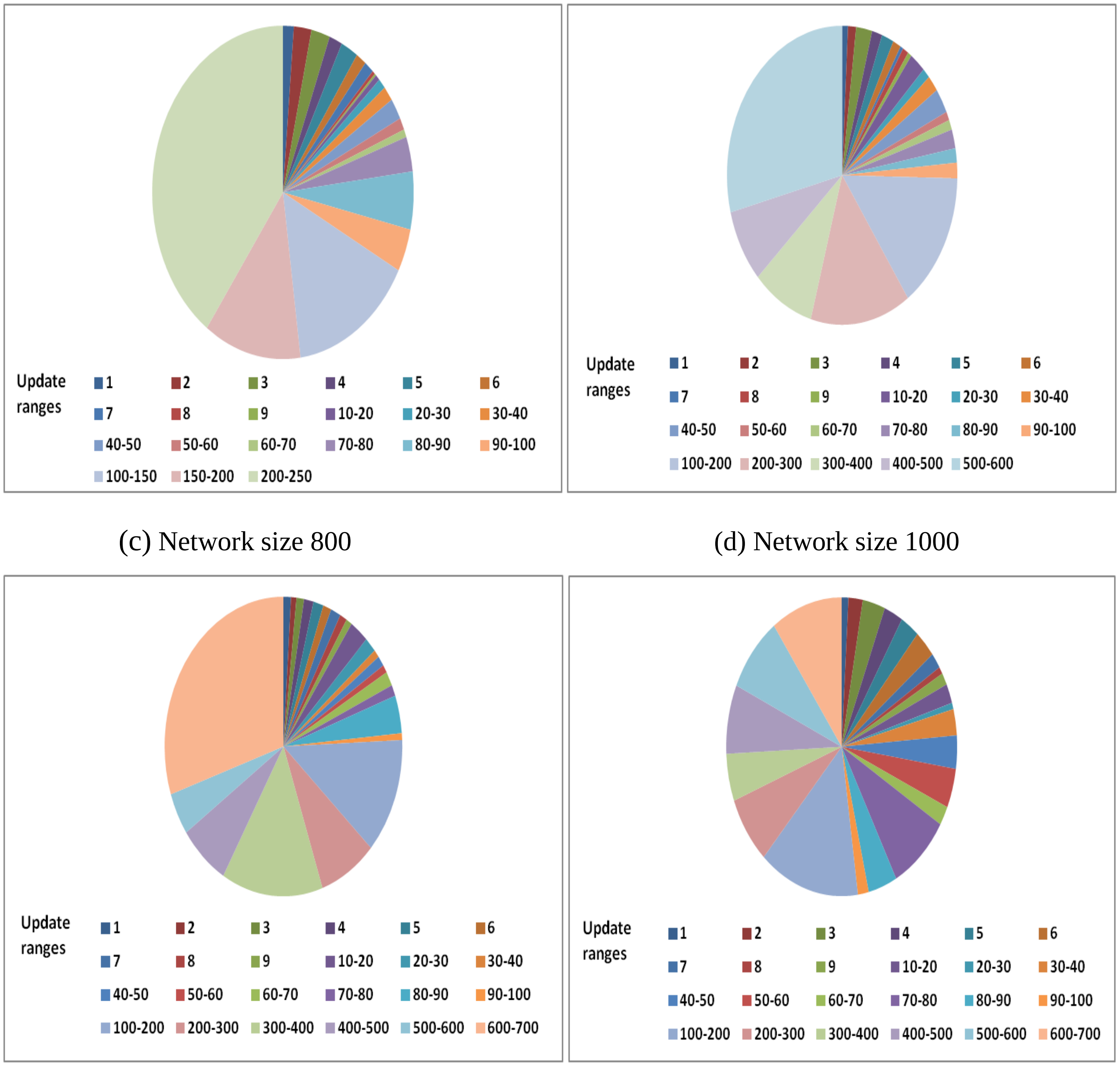
A clear observation from the results is that a large percentage of PSGs ranging from 35% to 60% of the total network size do not execute an update operation during the entire duration of the experiment and this observation is consistent across all the network sizes. This indicates that the index performance with respect to the update operations is fairly stable. It can also be observed that a sizeable percentage of PSGs carry out a small number (< 10) of update operations which is expected given the dynamic nature of the context data. However, since the total number of updates executed by a PSG is still low, it highlights the capability of the proposed index structure to manage dynamic data and minimize the update operations. The main reason behind this can be attributed to the fact that we utilize the mean-variance range values to build the index which enables PSGs with similar data change patterns to be grouped together. This in turn helps minimize the number of update operations as well. The number of update operations is also dependent on the actual pattern of data change within the individual PSGs. As can be seen from the figure, there is a very small percentage of PSGs that exhibit a very large number of update operations that range between 100 and 700. These indicate the PSGs which have a very uneven or bursty data distribution and the magnitude of data change is large. Since the data distribution pattern is uneven, such PSGs keep shifting between the range clusters. As long as such PSGs constitute a small percentage of the total network size, their presence will not affect the query processing operations with respect to the response time as well as the number of query answers received.

We next proceed to analyze the contributions of the cumulative values of the updates in a given update range to the total number of update operations observed for a given network size. The contribution of a given update range is given by the ratio of the cumulative values of the updates in that range to the total number of updates. The results are illustrated in Figure 22 and are based on the experimental results obtained for the analysis of the update operations.



(a) Network size 400

(b) Network size 600



(c) Network size 800

(d) Network size 1000

(e) Network size 1200

(f) Network size 1400

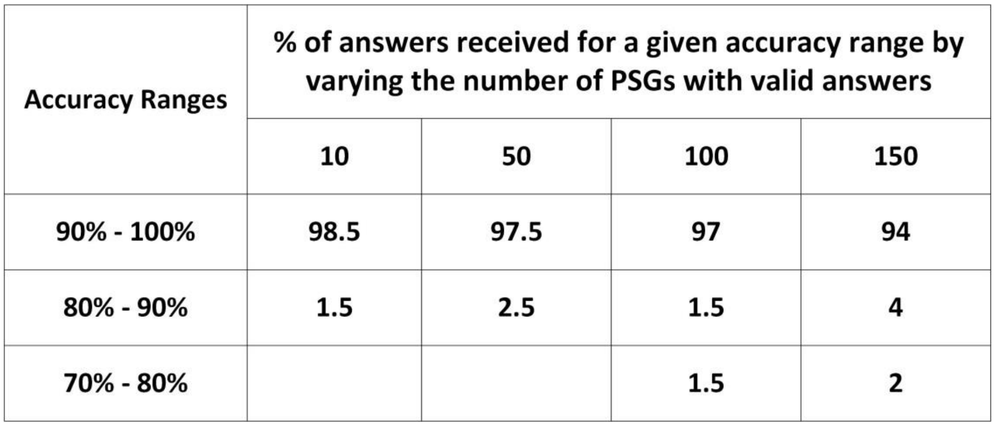
Figure 22. Contribution of cumulative updates in different ranges to the total updates

The results clearly indicate that the cumulative update of the higher valued update ranges have a larger contribution to the total number of updates whereas the lower valued update ranges have a proportionately lower contribution. As we observed in the previous set of results, the number of PSGs belonging to the higher update ranges constitutes a minor percentage of the total network size. So, even though the total number of update operations is high, the major part of those operations can be attributed to a small percentage of PSGs. We have already discussed the impact of the update operations on the query processing if the number of PSGs involved in the updates are high. In this case, since only a small percentage of PSGs are involved in the frequent update operations, the effect on the index performance will be minimal. The experimental results in this section indicate that the proposed index is stable with respect to the update operations of the PSGs and this behavior is consistent across network sizes. In a real world deployment, the index performance will depend on the dynamicity of the underlying data in each individual PSG. The test data used in this case corresponds to a set of environmental sensor data most of which changes rapidly but the magnitude of the change is small. If the majority of the data being indexed exhibits this kind of behavior, the number of index update operations will be low as is evident from the experimental results. This fact also forms the basic assumption behind the design of the proposed indexing structure. However, it was also observed from the experimental results that there are some PSGs that register a very large number of updates which in turn indicates a bursty data distribution pattern. The presence of a large number of PSGs with such bursty and uneven data distributions can lead to a degradation in the index performance. In the subsequent section, we evaluate the performance of the proposed index structure by varying the number of PSGs having uneven data distributions.

5.6.4 Query accuracy measurement with different PSG compositions

In this section, we evaluate the proposed index structure with respect to the query accuracy which is measured by the number of answers received for a query. For the initial part of evaluating the query accuracy, we issue a set of queries having a known number of valid answers and then compare the actual answers received. The results are obtained by issuing 300 random queries where each query is a range query of the form (A, B) and the number of PSGs having the valid answers as well as the network size (the total number of PSGs registered with the system) is varied. The number of answers received for each query is then measured against the expected answers. For the sake of brevity, we only highlight the results for a network size of 1400 in Table 4. The results indicate that the accuracy of the proposed index structure is quite satisfactory as a majority of the answers received fall in the high accuracy range. However, there are some instances where the query accuracy falls below 90% especially where the number of valid answers for a query increases.

Table 4. Query accuracy results



Accuracy Ranges	% of answers received for a given accuracy range by varying the number of PSGs with valid answers			
	10	50	100	150
90% - 100%	98.5	97.5	97	94
80% - 90%	1.5	2.5	1.5	4
70% - 80%			1.5	2

Since the query is propagated in a P2P network, there are a number of factors that can contribute to the missing out of valid answers that include routing errors, message delivery failures with UDP or due to query results being received after the timeout which is used to control the processing of a query. Also, the PSG update operation as well as the cluster maintenance operations can contribute to the missing of valid answers depending on the frequency with which they occur.

We now proceed to investigate the effect on the query accuracy with the variation of the number of PSGs having uneven or bursty data distributions for a given network size. As discussed in the previous section, the index performance is dependent on the data change patterns of the underlying PSGs. The main aim of this study is to introduce a fixed percentage of PSGs that exhibit frequent updates in a given network size and study the corresponding impact on the index performance especially with respect to the query accuracy.

In order to simulate this set of PSGs, we utilize the results achieved in Section 5.6.3 where we obtained a list of PSGs for each network size along with their number of update operations. We identify the PSGs from this list that register more than 80 updates in one hour as the ones having uneven data distribution and use the data files associated with them to generate PSGs with uneven data distributions. The creation process of these PSGs is as follows – we randomly select a PSG from the ones obtained in the previous step and retrieve its corresponding data file. The contents of this file are copied in random order to another data file which now represents the data values of a new PSG. The contents are copied in random order to prevent multiple PSGs from having the exact same sequence of data values. For a given network size, we use this technique to create a set of PSGs exhibiting frequent updates that vary from 10% to 100% of the total network size. For a given network size and percentage of PSGs with frequent update combination, we issue 300 queries with a valid answer set size of 100. For each issued query, we register the number of answers received and average it over the entire set of query runs. As part of our experiment, we try to focus the queries on the range clusters that are more involved in the leave/join operations. The results obtained are illustrated in Figure 23.

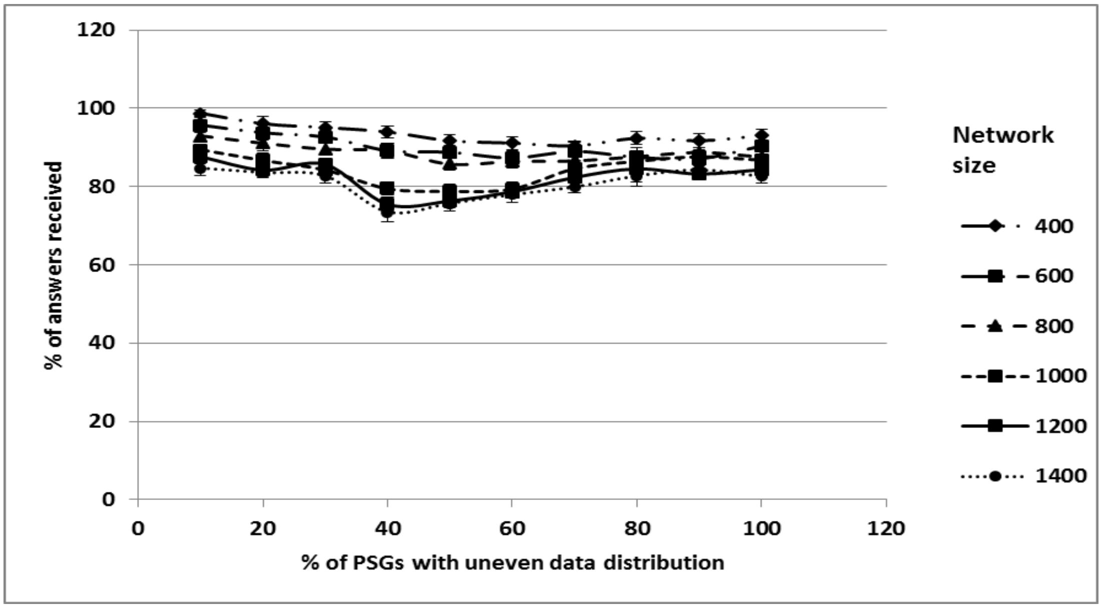
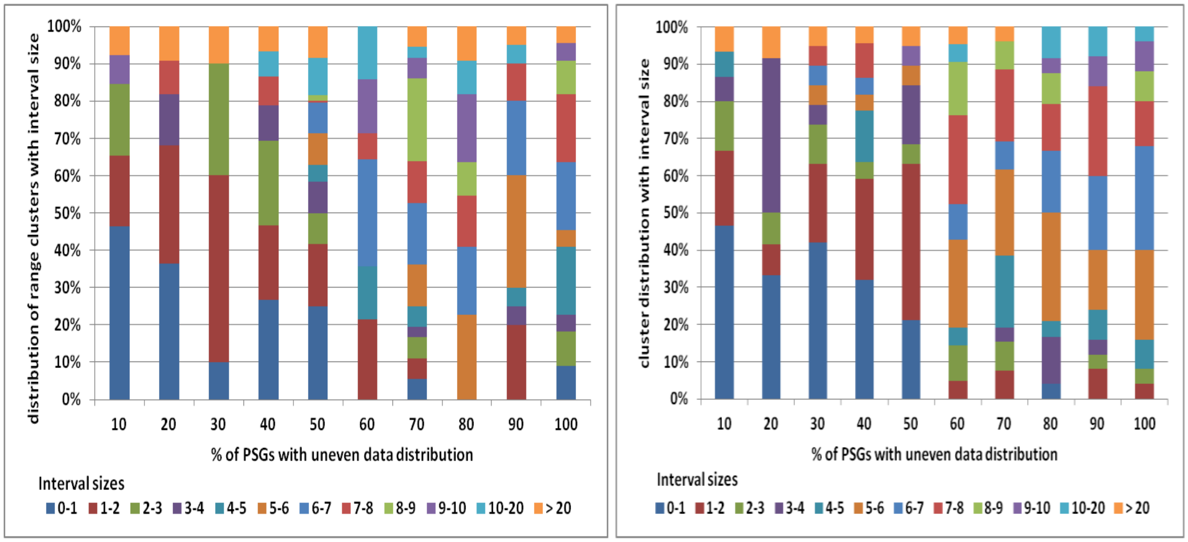


Figure 23. Variation of query accuracy with PSGs having uneven data distribution

We can see from the results that the introduction of PSGs that exhibit frequent update operations has a direct effect on the number of answers received for a query which is consistent with our observation that this operation affects the query processing operation. Also, the depreciation in the performance with respect to the number of answers received increases with an increase in the network size as the number of PSGs with uneven data distribution also increases proportionately. For a given network size, it is observed that the query accuracy performance initially deteriorates when the percentage of the PSGs with uneven data distribution is increased which is expected. However, it is interesting to note that the depreciation in the performance is not as drastic as expected with the increase in this percentage. On the contrary, the accuracy performance seems to marginally improve as compared to the initial cases where the percentage is low. This behavior seems to be counterintuitive as we expect the performance to worsen when the number of PSGs exhibiting frequent updates is increased. Clearly, the proposed index structure is providing some kind of adaptation technique to compensate for the increase in the number of PSGs.

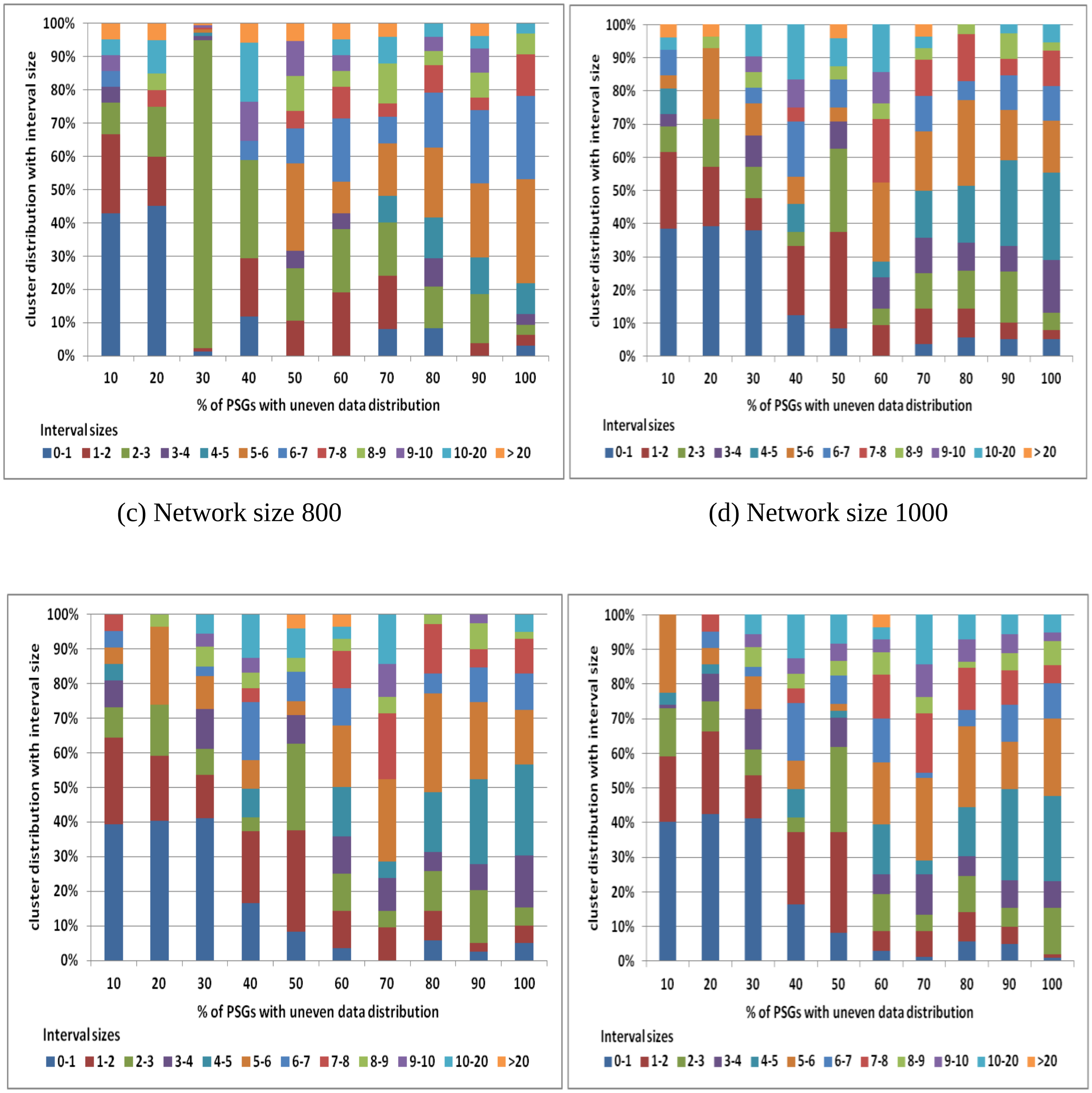
As we can recall from the index construction process in Section 5.4.1, the cluster bounds of the range clusters generated as part of the index are dependent on the data change patterns of the underlying PSGs. This observation regarding the range clusters and the cluster bounds offers a clue regarding the index performance behavior seen for the larger percentage of PSGs with uneven data distributions. In order to understand this phenomenon, we analyze the range cluster compositions for the different combinations of network size and PSGs with bursty data.

This study involves repeating the previous experimental setup PSGs, observing the clusters formed with regard to their interval sizes and grouping the range clusters according to the interval sizes. As discussed in Section 4.2.3, the interval size of a range cluster is the difference between the cluster bounds. Figure 24 highlights the variation of the interval sizes of the range cluster bounds for different network sizes.



(a) Network size 400

(b) Network size 600



(c) Network size 800

(d) Network size 1000

(e) Network size 1200

(f) Network size 1400

Figure 24. Variation of range cluster interval sizes for different network sizes

The results indicate that the range clusters with lower interval sizes dominate the total cluster composition when the percentage of the PSGs with uneven data distributions is low. This behavior is consistent across all the network sizes. However, when this percentage is higher, it can be seen that the clusters with the larger interval sizes constitute the bulk of the overall cluster composition. In other words, as the percentage of PSGs with uneven data distributions increases, there is a corresponding increase in the formation of range clusters with comparatively larger interval sizes. In order to understand the reason behind this occurrence, we need to reiterate the details of the range cluster formation. As we observed in Section 5.4.1, the cluster bounds of the range clusters are generated using the mean-variance ranges of the underlying PSGs and if there are a large number of clusters formed with small membership sizes, they are merged to keep the total number of clusters low. The merging of two clusters effectively increases the interval size of the resultant cluster.

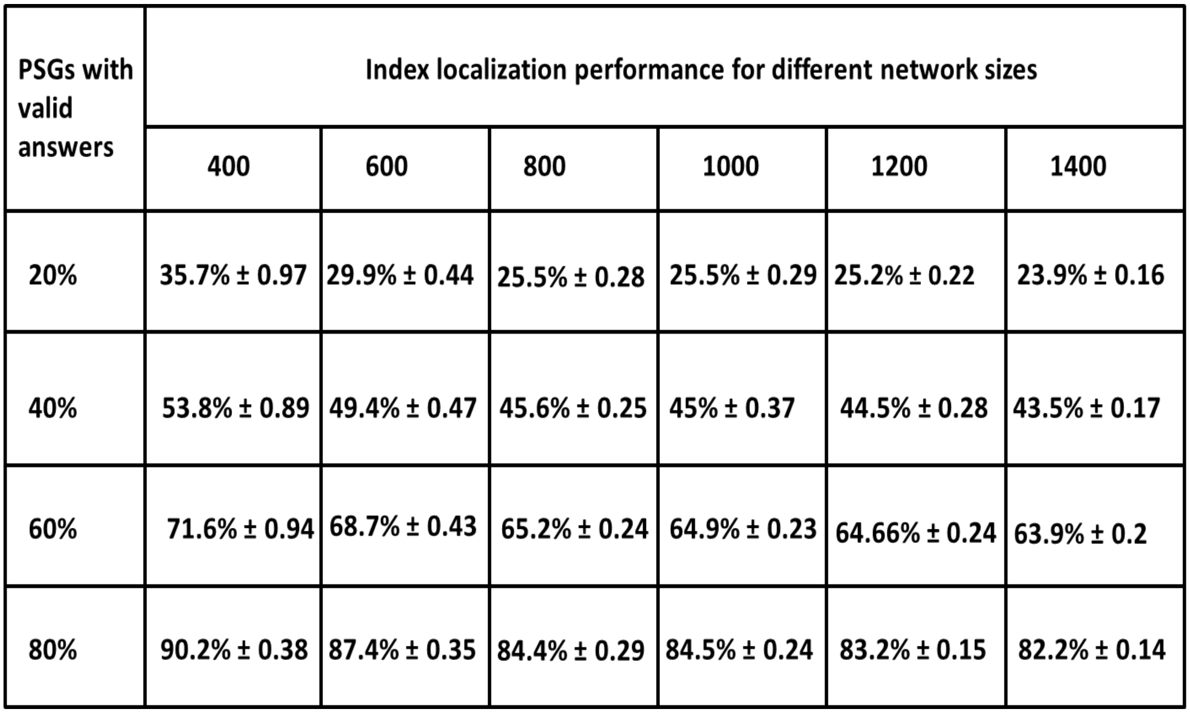
When the percentage of PSGs having uneven data distributions is large, the range cluster formation is mostly dependent on the mean-variance ranges of these PSGs. Since the mean-variance ranges of such PSGs will usually have a very low interval size, the cluster generation process will rely on the merging of the initial clusters formed to generate the final clusters which in turn will result in an increase in the interval size of the formed clusters. Consequently, the increase in the number of clusters having larger interval sizes will enable more values to be accommodated in them which compensates for the increase in the number of PSGs with uneven data distributions. However, the depreciation in the query accuracy performance is still present. This provides an explanation for the stabilization of the index accuracy performance for the larger percentage of the PSGs with uneven data. In reality, a case where all the PSGs associated with the middleware have very uneven data distributions is unlikely to occur. However, it is interesting to note that our proposed index structure is capable of handling such extreme cases and adapts itself according to the data change patterns of the underlying PSGs. This indicates that the proposed index structure is equipped to manage dynamic context data.

5.6.5 Index localization performance

As part of our evaluations of the index structure, we now analyze the localization performance of the index. The localization performance is measured by measuring the size of the candidate cluster set generated for a query that has a known set of answers. To cite an example, consider a query that has 30 valid answers. When the proposed index is used to evaluate this query, it identifies three clusters having 10, 20 and 25 members respectively that can have the possible answers. In this case, the total size of the candidate cluster set is 55. This metric gives an idea about the ability of the index to minimize the search space for a query. Table 5 gives an overview of the localization performance of the index for different combinations of the valid answers for a query and network sizes. Although the actual localization performance will vary in practice depending on the set of values and the ensuing range clusters, we highlight these results as an illustrative example. The results indicate that the localization performance of the index is reasonably good and the index can significantly reduce the search space even when the network size is large**.**

Since the candidate cluster set for a given query will always include the outlier cluster if the query range bounds are not contained within a single range cluster, it might cause the size of the candidate cluster set to be large as compared to the actual number of valid answers. However, the inclusion of the outlier cluster in the candidate cluster set is essential for the purpose of query completeness. Since the outlier cluster may or may not be queried depending upon the actual query parameters as well as the answers received, its inclusion is not always expected to have a significant bearing on the query response time.

Table 5. Index localization performance

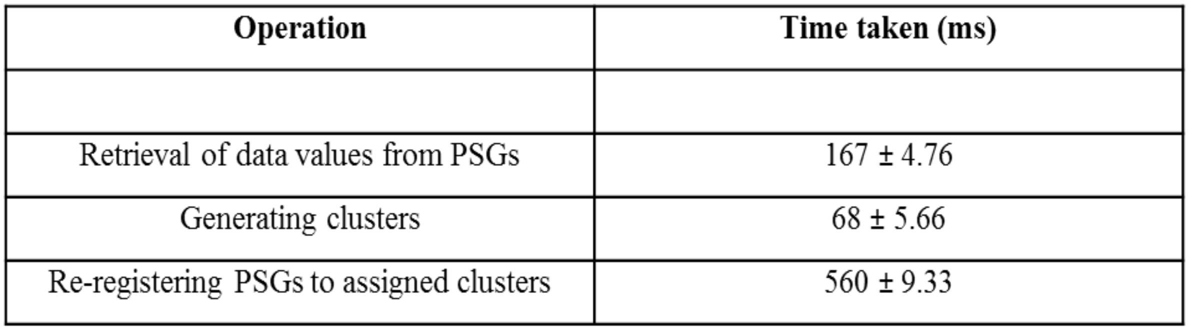


PSGs with valid answers	Index localization performance for different network sizes					
	400	600	800	1000	1200	1400
20%	35.7% ± 0.97	29.9% ± 0.44	25.5% ± 0.28	25.5% ± 0.29	25.2% ± 0.22	23.9% ± 0.16
40%	53.8% ± 0.89	49.4% ± 0.47	45.6% ± 0.25	45% ± 0.37	44.5% ± 0.28	43.5% ± 0.17
60%	71.6% ± 0.94	68.7% ± 0.43	65.2% ± 0.24	64.9% ± 0.23	64.66% ± 0.24	63.9% ± 0.2
80%	90.2% ± 0.38	87.4% ± 0.35	84.4% ± 0.29	84.5% ± 0.24	83.2% ± 0.15	82.2% ± 0.14

5.6.6 Time breakdown for clustering process and PSG leave/join operations

In this section we first analyze the time required to carry out the operations involved in the clustering process when a single cluster is split into multiple range clusters as this process is the main step required for our index generation. Table 5 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 PSGs to their respective range clusters as compared to the cluster generation process.

Table 6. Time breakdown for clustering process



Operation	Time taken (ms)
Retrieval of data values from PSGs	167 ± 4.76
Generating clusters	68 ± 5.66
Re-registering PSGs to assigned clusters	560 ± 9.33

Similar to the case observed with the range clustering scheme, the re- registration of PSGs dominates the overall time for the clustering process. If the frequency of the clustering process is low, the latency will not affect the overall system performance. However, this cannot always be guaranteed and depends on the actual deployment environment, the data change patterns of the underlying PSGs and the frequency with which the data changes in each registered PSG. For the next part of the evaluation of the performance of the proposed index structure, we study the impact of the variations of the data distribution patterns of the PSGs on the time taken by a single PSG to complete an update operation. As discussed previously, the update operation comprises of a PSG leaving a range cluster and joining a new one. Since the accuracy of the index in terms of the answers received for a query also depends on the efficiency of the update operation, it is important to study the variations in the same with different network sizes as well as the number of PSGs having uneven data distributions. In order to measure the variations in the PSG leave/join times, the experimental setup from Section 5.6.4 is repeated and the leave/join times are measured for a given network size and percentage of PSGs with uneven data. Since the results obtained across the different network sizes are comparable, we highlight the results achieved for a network size of 1400 in Figure 25.

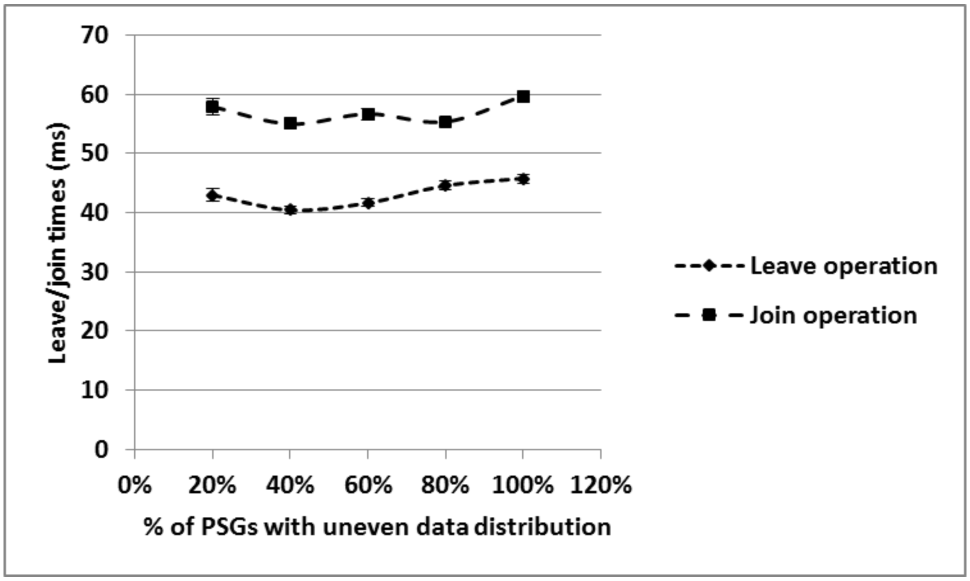


Figure 25. Variations of PSG leave/join operation times

The results indicate that the leave and join times for a PSG shows minimal variations with the changes in the data distributions of the underlying PSGs. From the perspective of a PSG, these operations just involve the removal of its old neighbors and updating its neighbor table with a new set of neighbors. Also, the leave operation happens faster than the join because the latter involves the location of a matching cluster and the retrieval of the new neighbor information. Based on these results, we can conclude that the time taken for the update operation for a single PSG is mostly independent of the data change patterns of the PSGs registered with the system. This also indicates that the update operation of a single PSG will not have a major impact on the index performance with respect to the query accuracy. However, as illustrated in Table 6, the re-registration of multiple PSGs to their respective clusters dominates the total time taken for the clustering process. This leads to the conclusion that the index performance will vary according to the total number of PSGs executing an update operation concurrently.

5.7 Summary

In this chapter, we proposed an improved version of the previously discussed range clustering scheme and utilized the mean and variance values of a data attribute to build an index structure that can manage dynamic data and support lookups over them. The experimental results indicate that the proposed index is able to achieve satisfactory query response times over different network sizes and localize queries efficiently. It is also observed to be stable with respect to the number of update operations and is able to adapt itself to the changes in the data distribution patterns of the underlying PSGs effectively even in the extreme cases. The current index structure is designed to handle single valued numeric attributes and needs to be suitably extended to support multidimensional data (e.g. location coordinates) attributes. Also, the effectiveness of the index structure to partition the data values depends on the data change patterns of the underlying PSGs which may vary according to attribute type and deployment environment and the use of this index will not be effective if all the PSGs have extremely similar data change patterns. Although this index can be used to index string attributes by hashing them to a numeric representation, the hashing results in the loss of the original string thereby restricting the type of queries that can be processed on them. Since context data attributes can be of either string or numeric data types, it is important to provide an organizational scheme for strings. This problem is addressed in the subsequent chapter that proposes an incremental indexing scheme for strings.

CHAPTER 6

AN INCREMENTAL TREE BASED INDEX STRUCTURE FOR STRING CONTEXT DATA

This chapter discusses the design and implementation details of our proposed indexing scheme for organizing and carrying out lookups over string context data. Since the query requirements for string attributes are different than that for numeric attributes, we develop a separate indexing mechanism for string attributes. The chapter is organized as follows – Section 6.1 discusses the general query requirements for strings as well as the index requirements. This section also highlights the constraints that need to be observed while designing a string index structure in our middleware system. Section 6.3 discusses the details of the construction and maintenance process of the string index structure. The experimental results are discussed in Section 6.4. The chapter is summarized in Section 6.5.

6.1 Overview

In Chapter 4 and 5, we discussed a range clustering based index structure and a refined version of the same using the mean-variance values that was able to handle the problem of managing dynamic context data and supporting lookup requests over it satisfactorily. The proposed indexing techniques were primarily designed for numeric attributes as they utilized the statistical properties of data to create the index structure. However, since context attributes can be of both numeric and string data types, it is important to provide a similar organizational scheme for string attributes as well. As we observed in Section 2.1, the query requirements vary according to the type of the data being queried and the standard query types for strings include exact match, prefix/suffix match, wildcard match and range queries. The existing index structure can be used to index strings by converting the strings to a numeric representation using a hash function and constructing the index on these numbers. In order to preserve the string ordering, lexicographic hash functions can be used as these functions create a hashed representation that is consistent with the alphabetical ordering of the strings.

Although this technique is straightforward, the use of a hashing technique can restrict the types of queries that can be performed on the strings. Since the hashing process replaces the original string value with a numeric representation, only exact string matching queries can be supported. In order to address this limitation of the hashing based approach, we need to devise an alternative indexing scheme that can support a wider variety of queries on string attributes. One of the key challenges involved in the index design is its ability to support the different types of query requirements associated with strings effectively. Also, since there is no prior knowledge about the set of strings to be indexed, the index will need to adapt itself as strings are received and removed corresponding to the movement of the PSGs. We attempt to address these issues as effectively as possible as part of our index structure for string attributes.

6.2 String indexing in Coalition – Requirements and constraints

The problem of indexing and querying a set of strings has been widely studied in the research community and a wide variety of techniques have been proposed to handle this problem in different application domains like speech detection, text analysis, image comparison, signal processing, data mining, DNA sequence alignment, pattern recognition etc [95-101]. One class of these techniques focuses on finding common and recurring patterns between strings in a document according to a particular kind of similarity metric and grouping strings that are close according to this metric. The similarity metric is chosen in accordance with the application domain for which the string classification technique is proposed. Variations of this technique are also utilized in text processing applications to search for spelling errors as well as to extrapolate strings from damaged and imperfect strings. The idea of the similarity metric is also used for the technique of approximate string matching. A separate class of approaches focus on the problem of creating an index structure to organize a set of strings stored in a memory space and optimizing the index with respect to the storage required for the index as well as the memory accesses needed to query the strings. These techniques are usually developed for database systems where the index as well as the strings is stored in a disk based memory system. The optimization of the space and memory access requirements becomes essential in this case to improve the system performance.

We now examine the problem of building an index structure for strings from the perspective of Coalition along with the associated constraints. Similar to the case with numeric attributes, the primary constraint for the index design is the fact that the strings themselves are not stored in Coalition. Consequently, the management of the issues regarding the storage of the strings in memory and the optimization of the disk access times is not very important in this case as compared to a database system. As was the case with the index structure for numeric attributes, a string indexing mechanism for organizing the strings in Coalition should provide an approximate ordering amongst these strings so that queries can be redirected to the relevant PSGs. The final step of the query processing will still be carried out in each individual PSG. This requirement again varies from the standard database based approach of using a tree structure for organizing strings stored in memory and storing the memory locations in the tree nodes. Further, as the Coalition middleware is expected to handle a wide variety of strings including names, addresses, locations, preferences etc. it is important that the indexing scheme utilized is fairly generic and makes minimal assumptions about the type of strings being indexed.

An important decision that needs to be made prior to the design of the index structure is the choice of the similarity metric to be used in order to partition the strings into groups. As discussed earlier, the choice of this metric is dependent on the types of strings being indexed as well as the type of queries that need to be processed. The main choice between the similarity metrics is the selection of either an exact matching or an approximate matching technique. Approximate matching metrics are utilized in applications where the requirement is to find strings that may not match exactly but rather match a given pattern approximately. We have discussed some of these applications previously that include DNA sequence matching, searching for spelling mistakes in a document, identifying imperfect strings from a string collection etc. Approximate string matching techniques also require all the strings to be present in the memory during the querying process which violates the data storage constraints of Coalition. Further, the query requirements for strings in Coalition is slightly different than these applications. Since the string attributes are generated and maintained at the individual PSGs, it is reasonable to assume that these strings will be mostly error free and properly formatted. Thus, the key requirement for a string index structure for the Coalition middleware system is to reduce the search space for applications looking for a particular string or a set of strings using exact matching queries. These queries can be invoked by applications searching for a particular name or a set of names that begin with a given prefix, address or preference information contained within a PSG. With this requirement in mind, an obvious technique would be to group all PSGs with the same string information in a cluster and representing the index as a collection of such clusters. However, as there is no prior information about the strings to be indexed, the grouping of strings based on an exact matching may not be feasible as it can give rise to a large number of clusters. The other alternative in this case is to cluster the strings based on their shared common prefixes as this would enable more strings to be grouped together. The use of the prefix to group the strings is preferred as it would enable both string and exact matching queries. Since the strings are not stored in memory, the length of the prefix that is used to group the strings will need to be varied according to the current set of strings being indexed. Also, since PSGs can join and leave the Coalition system, the common prefix length will need to be changed accordingly to accurately reflect the set of strings contained in the PSGs. In light of these observations, we propose an incremental tree based indexing structure constructed using the ideas of radix sort and ternary search trees to generate an approximate prefix based ordering amongst the strings. The proposed index structure differs from existing string indexing structures in one key aspect – the strings to be indexed are not stored in persistent storage and strings can be added as well as removed frequently.

6.3 Indexing strings incrementally using radix sort and ternary search trees

6.3.1 Radix sort and Ternary Search Trees

Since our proposed index structure for strings is designed using radix sort and ternary search trees, we provide a brief overview of these two concepts. Radix sort is a sorting algorithm primarily used for sorting data with integer keys by grouping keys by the individual digits sharing the same significant position and value. The initiation of the radix sort procedure is done from the least significant digit position for integer values. Radix sort can also be used to sort strings in the lexicographic order by using a most significant digit (MSD) version of radix sort. This variation partitions the total set of strings into a set of initial buckets according to their first character and then recursively sorts the contents of each bucket on the subsequent characters. The final sorted list is generated by listing the bucket contents in order. The advantages of the MSD radix sort include the fact that depending on the composition of the strings to be sorted, all the key values need not be examined and it can be utilized for variable length strings.

A ternary search tree is a type of prefix tree where the nodes are arranged according to a binary search tree arrangement. Each node in a ternary search tree contains a single key character and pointers to three child nodes. While inserting a string in an empty ternary search tree, all the string characters are inserted in separate nodes and each node is linked to its predecessor using the middle node pointer. Subsequent string insertions are done character by character and the insertion process is initiated by comparing the first character with the root node. If this character is less than the character denoted by the root node, it is inserted as a left child of the root and correspondingly, it is inserted as the right child if it is greater. In case the characters are the same, the tree is traversed along the middle child and the previous process is initiated with the next character of the string. This process is carried out until the entire string is inserted. The lookup of a string in the tree can also be carried out similarly. Figure 26 provides an example of a ternary search tree constructed from a set of strings.

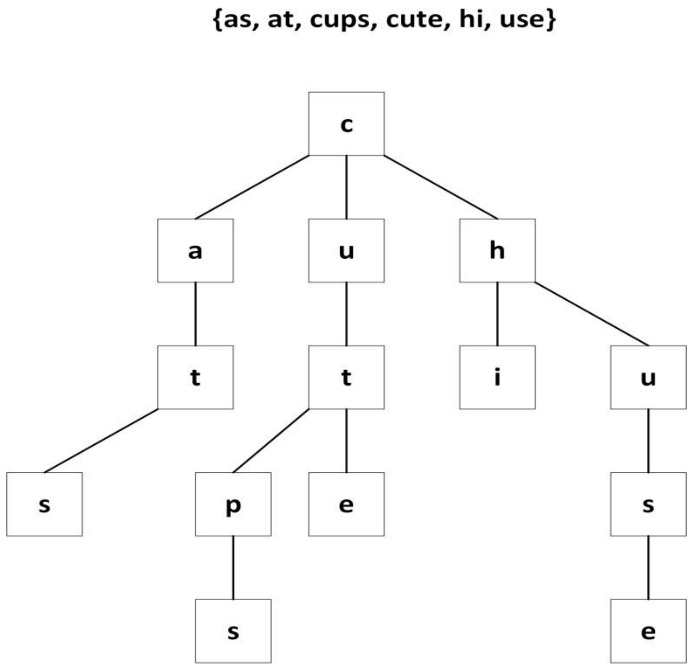
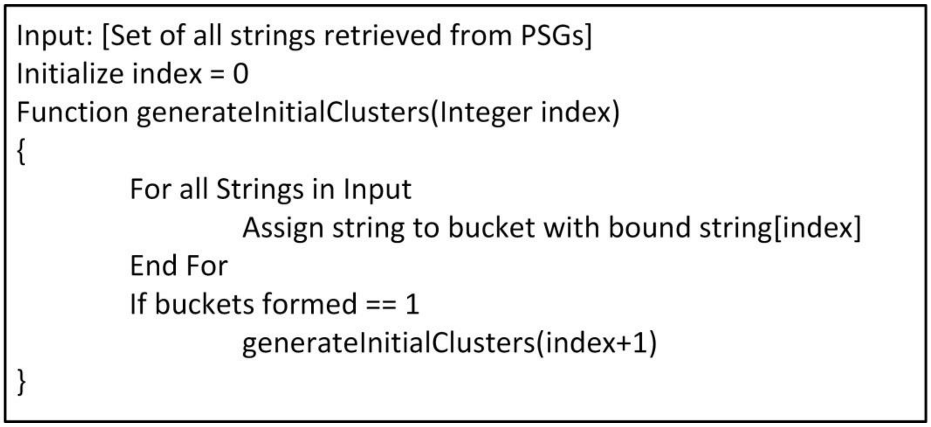


Figure 26. Example of ternary search tree

The average case running times of the insertion and the lookup operations of the ternary tree run in O (log n) where n is the total tree size. However, if the strings are inserted in lexicographic order, the tree is constructed in only one direction thereby giving rise to a worst case running time of O (n). Although the concepts of the radix sort and ternary search trees are well known and have been used in different application scenarios, we use a combination of these two concepts in a novel way to provide an ordering among string attributes in our middleware system.

6.3.2 Creating an index structure for strings

As discussed previously, the main aim of the proposed index structure is to provide an approximate classification of the strings stored in the PSGs that can be used to redirect queries to the relevant PSGs. In the discussion in Section 6.2, we identified a prefix based grouping of the strings as a suitable indexing technique for strings in Coalition that satisfied the constraints and requirements of the Coalition system. In light of this observation, we use the idea of the MSD radix sort to group strings based on their shared prefixes. However, the prefix length used to group the strings is not predefined and is adaptively varied according to the composition of the strings. Initially, all the PSGs are assigned to a single cluster until the cluster size exceeds the maximum cluster size threshold following which the indexing process is initiated. Since no strings are stored in the middleware, the first step involves the retrieval of the strings from the corresponding PSGs. Figure 27 illustrates the pseudocode for the first step of the indexing process.

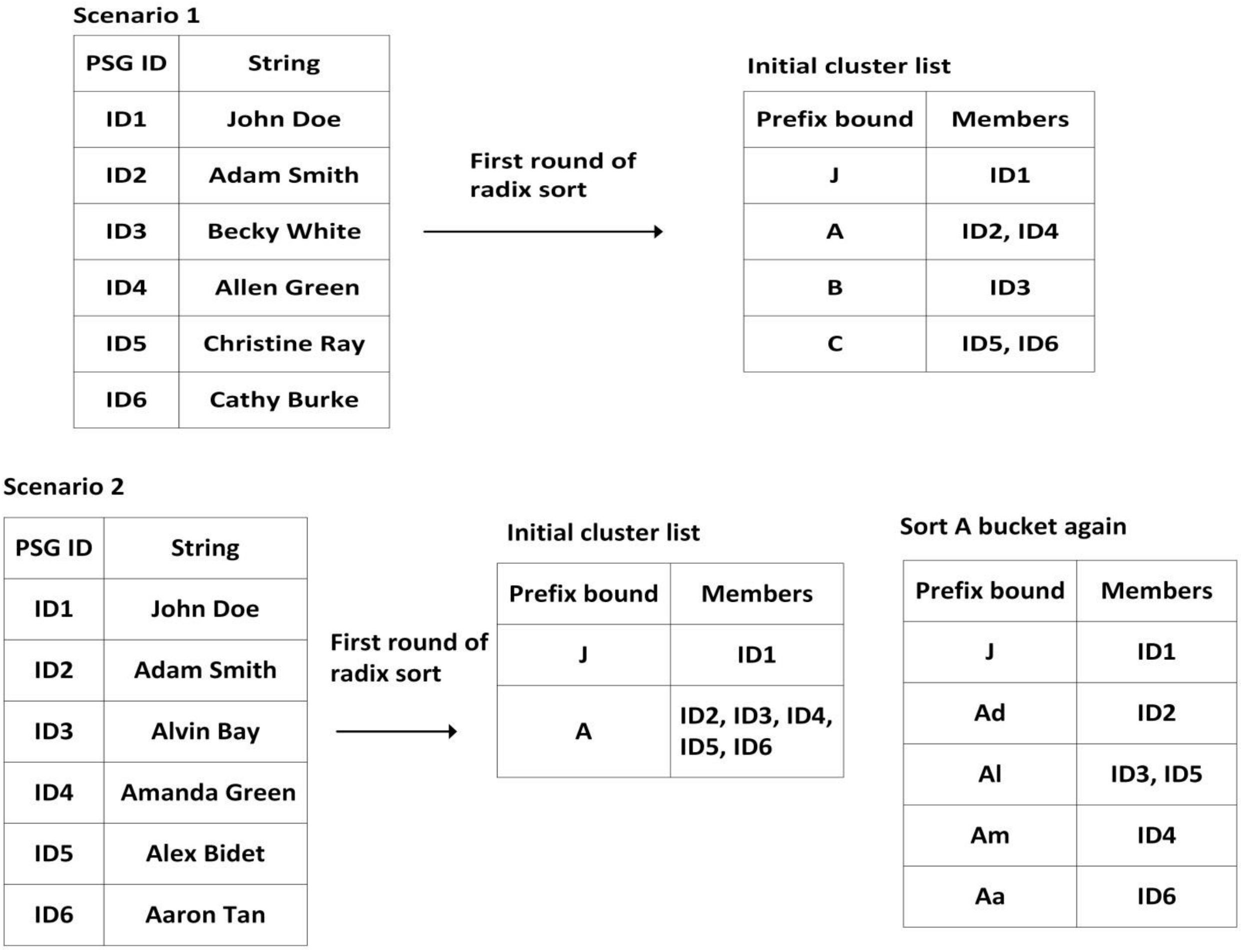


```
Input: [Set of all strings retrieved from PSGs]
Initialize index = 0
Function generateInitialClusters(Integer index)
{
        For all Strings in Input
                Assign string to bucket with bound string[index]
        End For
        If buckets formed == 1
                generateInitialClusters(index+1)
}
```

Figure 27. Initial indexing step pseudocode

The initial indexing step involves carrying out the MSD radix sort on the set of strings retrieved from the corresponding PSGs until we get at least two distinct clusters. Compared to the usual radix sort, the objective in our case is to provide a preliminary classification of the strings based on the shortest prefix length needed to achieve this classification. This step results in a set of clusters each of which represents a set of PSGs having strings sharing a common prefix. The common prefixes are taken to be the cluster bounds. A radix sort for strings runs in O (nk) time where n is the number of keys and k is the average key length. Since we use the radix sort to provide a preliminary classification, we may not need to compare all the string characters. In the best case scenario, a single pass of the sorting process would suffice to provide this classification which would indicate a running time of O (n). The average case running time would depend on the composition of the strings being indexed.

In case a single cluster contains more than 90% of the size of the total string set, we carry out another iteration of the radix sort to further divide the members of this cluster. This process is continued until a single cluster does not contain more than 90% of the total string set size. If the end of a string is encountered during this indexing step, that string is removed from the string set and the corresponding PSG is assigned to a cluster whose cluster bound is this particular string. The process of identifying the initial clusters is illustrated in Figure 28 for two different types of data sets.



Scenario 1

PSG ID	String
ID1	John Doe
ID2	Adam Smith
ID3	Becky White
ID4	Allen Green
ID5	Christine Ray
ID6	Cathy Burke

Initial cluster list

Prefix bound	Members
J	ID1
A	ID2, ID4
B	ID3
C	ID5, ID6

Scenario 2

PSG ID	String
ID1	John Doe
ID2	Adam Smith
ID3	Alvin Bay
ID4	Amanda Green
ID5	Alex Bidet
ID6	Aaron Tan

Initial cluster list

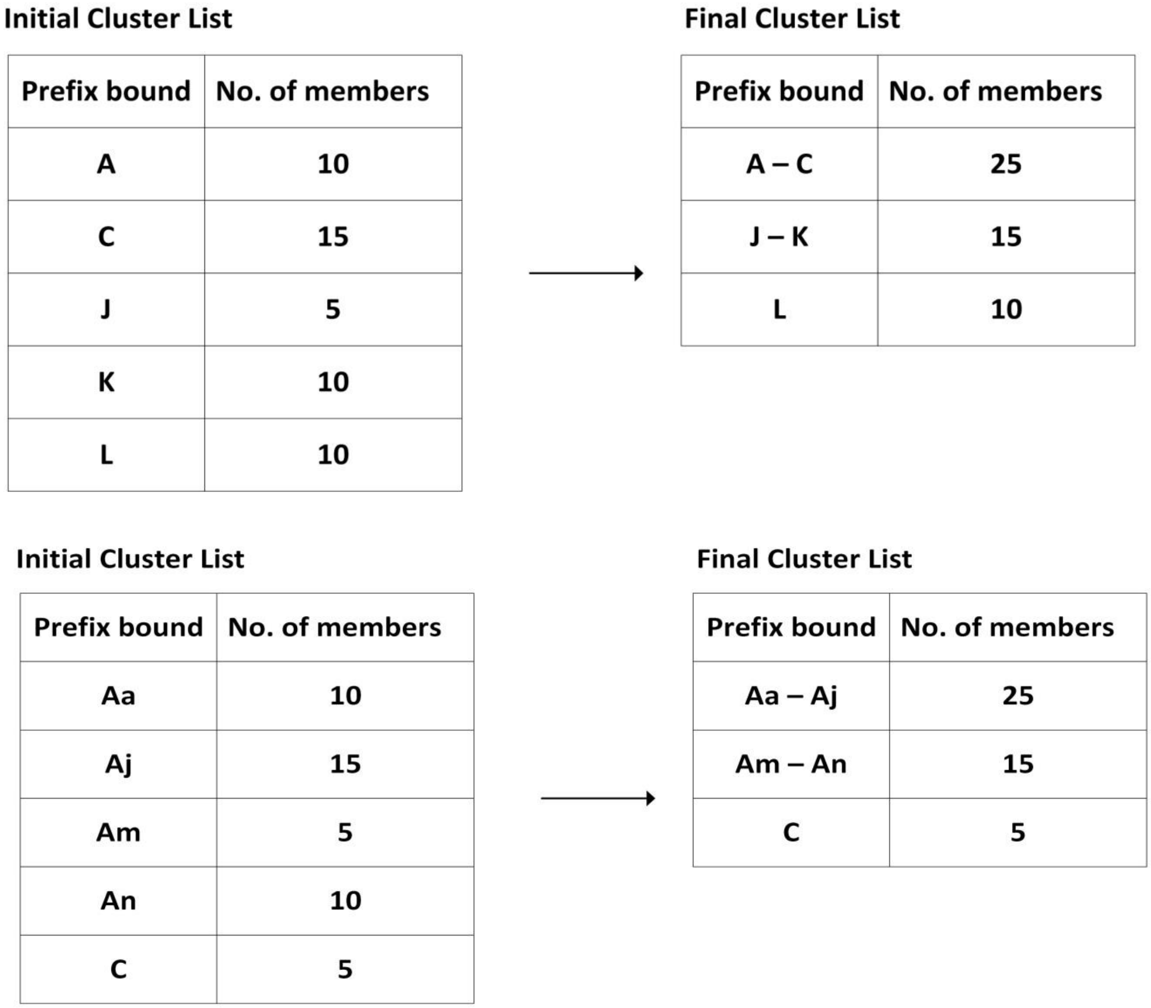
Prefix bound	Members
J	ID1
A	ID2, ID3, ID4, ID5, ID6

Sort A bucket again

Prefix bound	Members
J	ID1
Ad	ID2
Al	ID3, ID5
Am	ID4
Aa	ID6

Figure 28. Identifying the initial string clusters

After the initial clusters have been identified, the set of clusters representing the index structure need to be generated. Similar to the indexing process discussed for numeric attributes, we arrange the initial clusters according to the lexicographic order of their cluster bounds. This is followed by the cluster merging process to minimize the total number of clusters. Two clusters are merged if and only if their corresponding cluster bounds differ only in the last character. This constraint is introduced in order to facilitate the storage of the cluster bounds using the ternary tree. Figure 29 illustrates the process of the generation of the final clusters using two different sample set of initial string clusters.



Initial Cluster List

Prefix bound	No. of members
A	10
C	15
J	5
K	10
L	10

Final Cluster List

Prefix bound	No. of members
A – C	25
J – K	15
L	10

Initial Cluster List

Prefix bound	No. of members
Aa	10
Aj	15
Am	5
An	10
C	5

Final Cluster List

Prefix bound	No. of members
Aa – Aj	25
Am – An	15
C	5

Figure 29. The string cluster generation process

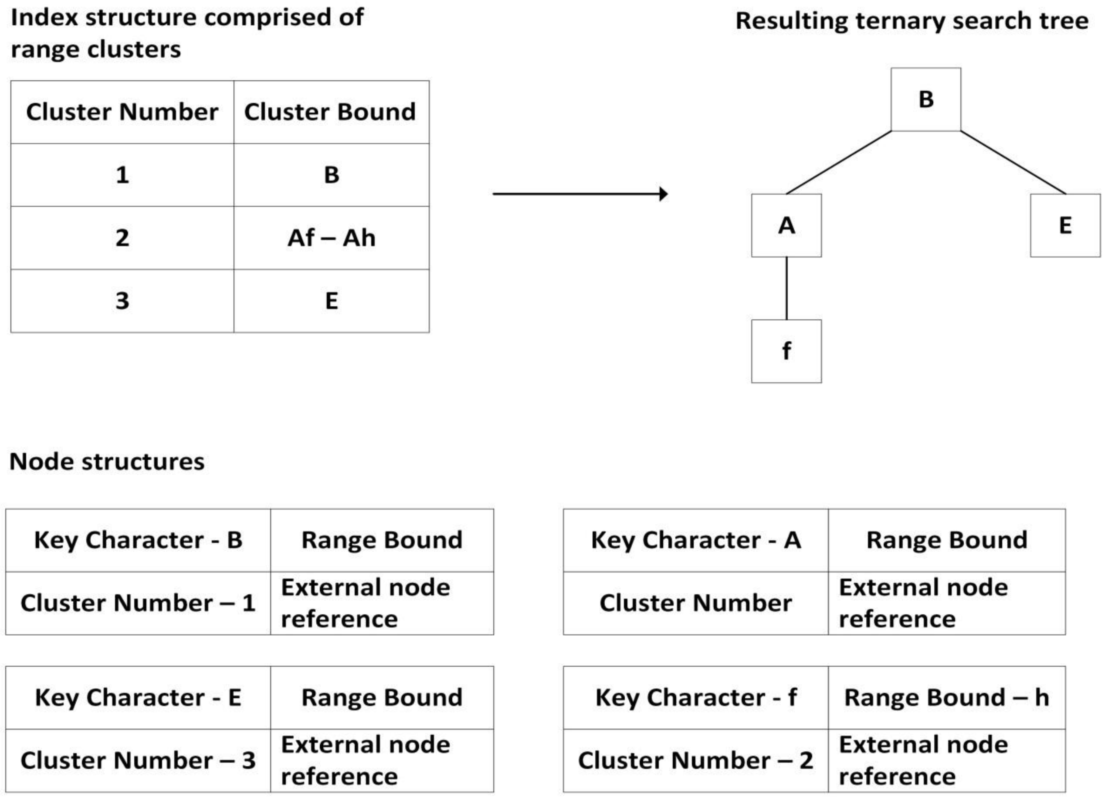
The index now comprises of a set of range clusters with the cluster bounds being either a single prefix or a prefix range. A cluster having a prefix range as its cluster bounds denotes a set of strings whose prefix lies lexicographically between the bounds. For example, a cluster with bounds [Ab, Ae] can accommodate strings starting with the following prefixes [Ab, Ac, Ad, Ae]. The PSGs corresponding to the strings register with their allotted clusters and form the corresponding P2P networks. The proposed index construction technique achieves our aim of providing an approximate ordering among the strings while satisfying the constraint pertaining to the storage of the attribute values. Also, we do not make any assumptions regarding the length of the prefix used to group the strings. After the initial set of clusters has been formed, an organizational structure for the cluster bounds is needed so that subsequent insertions and lookups can be carried out efficiently. The cluster bounds are organized using a modified version of the ternary search tree that allows both prefixes as well as prefix ranges to be stored. A TST node usually contains the key character represented by it and the three child pointers. One of the key augmentations that we propose in our version of the TST is the ability to store prefix ranges. Figure 30 illustrates our augmented version of the standard TST node. The child node pointers are omitted in the figure for clarity.

Key Character	**Range Bound**
Cluster Number	**External node reference**

Figure 30. TST node structure

We introduce an additional field denoted by *cluster bound* that stores the key character denoting the upper bound of a prefix. Since the prefixes that are part of a prefix range differ only in the last character, this field stores the corresponding upper bound. We also introduce two additional fields *cluster number* and *external node reference*. The first field is used to indicate the number of the cluster corresponding to the prefix represented by that node and uniquely identify that cluster number. We shall discuss the use of the second field in the subsequent sections.

After the range cluster bounds have been identified, they need to be inserted into the TST to generate the index structure. In order to minimize the chances of the tree becoming unbalanced due to the insertion of the cluster bounds in sorted order, we insert the cluster bounds into the TST in random order. Figure 31 illustrates an example of a TST constructed from a sample set of cluster bounds. When the last character corresponding to a cluster bound is inserted, the cluster number field is suitably updated. Similarly, the insertion of a cluster bound in the form of a prefix range entails the updating of the cluster bound field of the last character node. For example, the insertion of the range cluster [Af, Ah] results in the cluster bound field of the node corresponding to f being populated with h. As PSGs join and leave the system, this index structure will be updated accordingly. Subsequent PSGs are assigned to the correct range cluster according to their local data attribute value using the insertion process discussed in Section 6.4.1. This clustering operation is invoked whenever the membership size of a range cluster exceeds the maximum size threshold which entails the updating of the index structure. We shall discuss the details of this operation in Section 6.4.2.



Index structure comprised of range clusters

Cluster Number	Cluster Bound
1	B
2	Af – Ah
3	E

Node structures

Key Character - B	Range Bound
Cluster Number – 1	External node reference

Key Character - A	Range Bound
Cluster Number	External node reference

Key Character - E	Range Bound
Cluster Number – 3	External node reference

Key Character - f	Range Bound – h
Cluster Number – 2	External node reference

Figure 31. Creating a TST to organize the cluster bounds

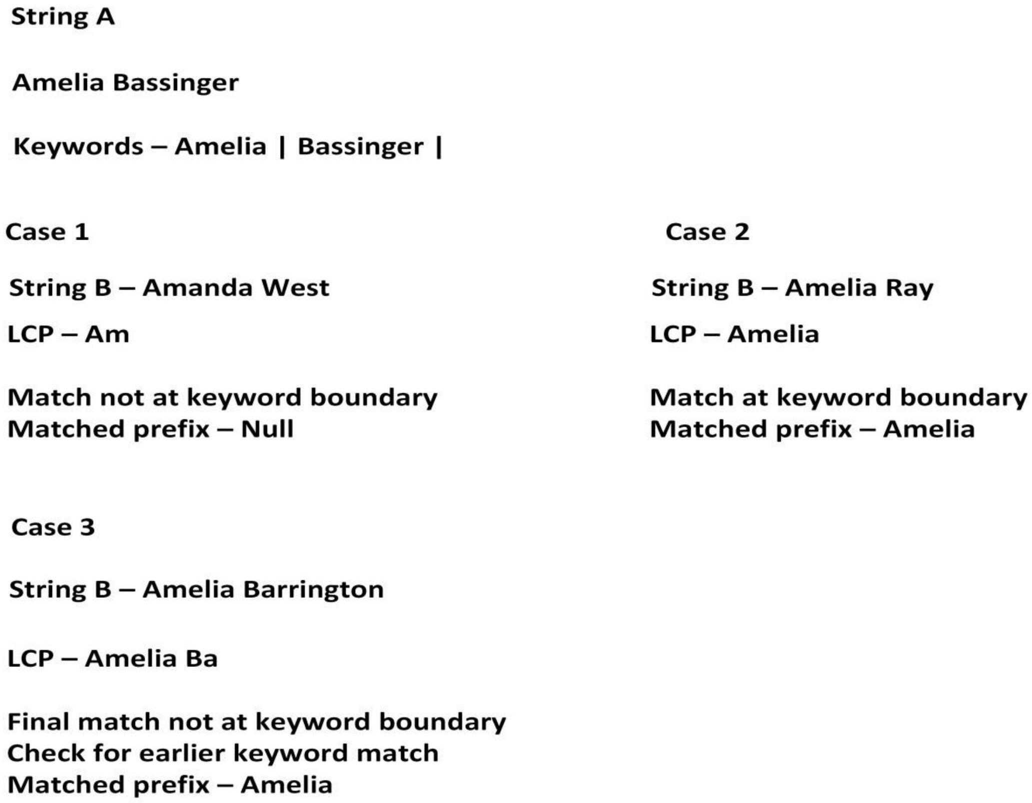
Since the indexing step uses the MSD radix sort, we need to ensure that string attributes composed of multiple strings are aligned in the correct order prior to the start of indexing. For example, name attributes can either be arranged either in a first name-last name or last name-first name format but this ordering has to be consistent throughout all the PSGs sharing this attribute. This ordering can be enforced during the PSG registration process by having PSGs provide the attribute value in the desired format. Alternatively, the middleware can carry out the ordering process after the retrieval of the string attributes from the PSGs prior to the indexing process. The same argument holds true for other string attributes composed of multiple strings like location information, address etc.

6.3.3 Identifying keywords based on longest common prefix

As we observed in the previous section, the total number of passes of the radix sort depends on the string set composition. In the event that the string set comprises of strings sharing a long prefix, the sort process will need to make multiple passes to generate the clusters. This phenomenon can occur when the string set comprises of a set of names with a common first name or a set of addresses in a particular location. The normal usage of the radix sort in this case will result in a long prefix being inserted into the TST and subsequent queries for strings with that prefix will be evaluated against this prefix chain. In order to avoid this issue and prevent the size of the main TST index from becoming too large, we propose a longest common prefix (LCP) based keyword identification technique to provide a separate organization structure for such strings.

The usual application of the LCP technique may not always be suitable for our intended purpose. For example, considering a set of strings [Amelia, Amanda, Amanda, Amanda], the normal LCP of these strings is [Am] which does not achieve our desired aim to group all the strings corresponding to [Amanda] separately. As part of our efforts to address this issue, we need to devise a technique to adjust the matched prefixes for the best match. The idea that we propose for the same is to visualize a string attribute value as a collection of individual keywords. To cite an example, a name attribute can be taken as a combination of a first name and last name keyword. Similarly, an address/location attribute can be defined as a combination of area, locality and street number keywords. Each string attribute can be associated with a set of boundary markers (usually in terms of the string length) that indicate the end position of each individual keyword within that string

We can now utilize this idea to determine if a matched prefix satisfies our requirement and change the usual LCP calculation technique. Figure 32 illustrates the main idea behind our modified LCP calculation technique between two strings A and B. As can be observed, we only qualify a LCP match as a valid match if the matching occurs at a keyword boundary. Again, the consistency of the ordering of the individual keywords within a given string attribute is an important prerequisite that needs to be fulfilled in order for this technique to work. Further, we currently only support exact string matches while evaluating the LCP.



String A

Amelia Bassinger

Keywords – Amelia | Bassinger |

Case 1

String B – Amanda West

LCP – Am

Match not at keyword boundary
Matched prefix – Null

Case 2

String B – Amelia Ray

LCP – Amelia

Match at keyword boundary
Matched prefix – Amelia

Case 3

String B – Amelia Barrington

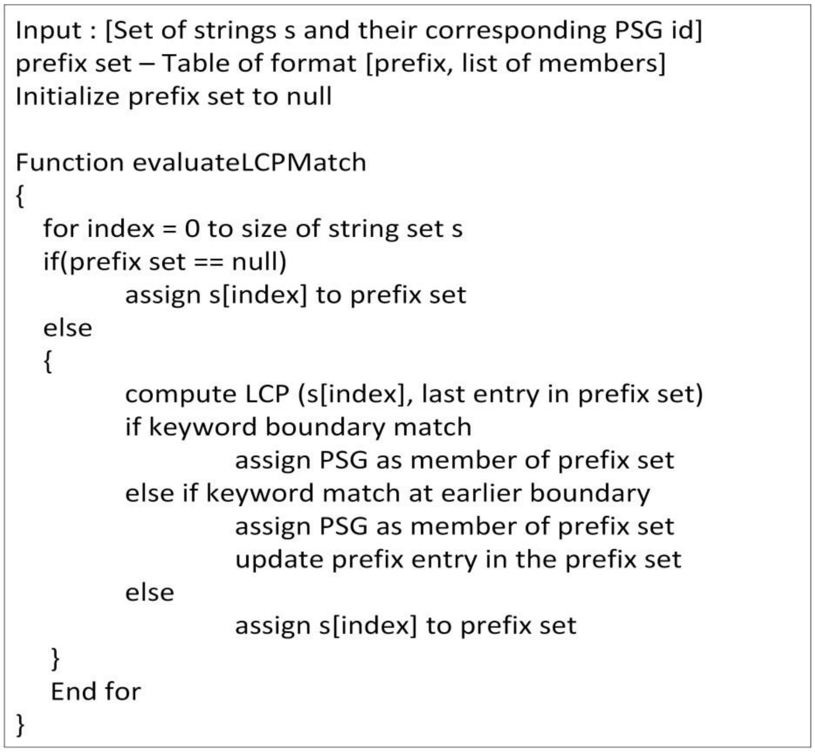
LCP – Amelia Ba

Final match not at keyword boundary
Check for earlier keyword match
Matched prefix – Amelia

Figure 32. Modified LCP matching process

The clustering process is now modified to incorporate this technique to identify groups of PSGs having strings with long common prefixes. After the first round of radix sort is completed, the sizes of the buckets formed are checked. In case a single bucket is found to contain more than 90% of the total string set size, the LCP based prefix matching process is initiated with the contents of this bucket. The strings in the bucket are sorted in alphabetical order prior to the initiation of the matching process. The details of the LCP matching process is illustrated in Figure 33.

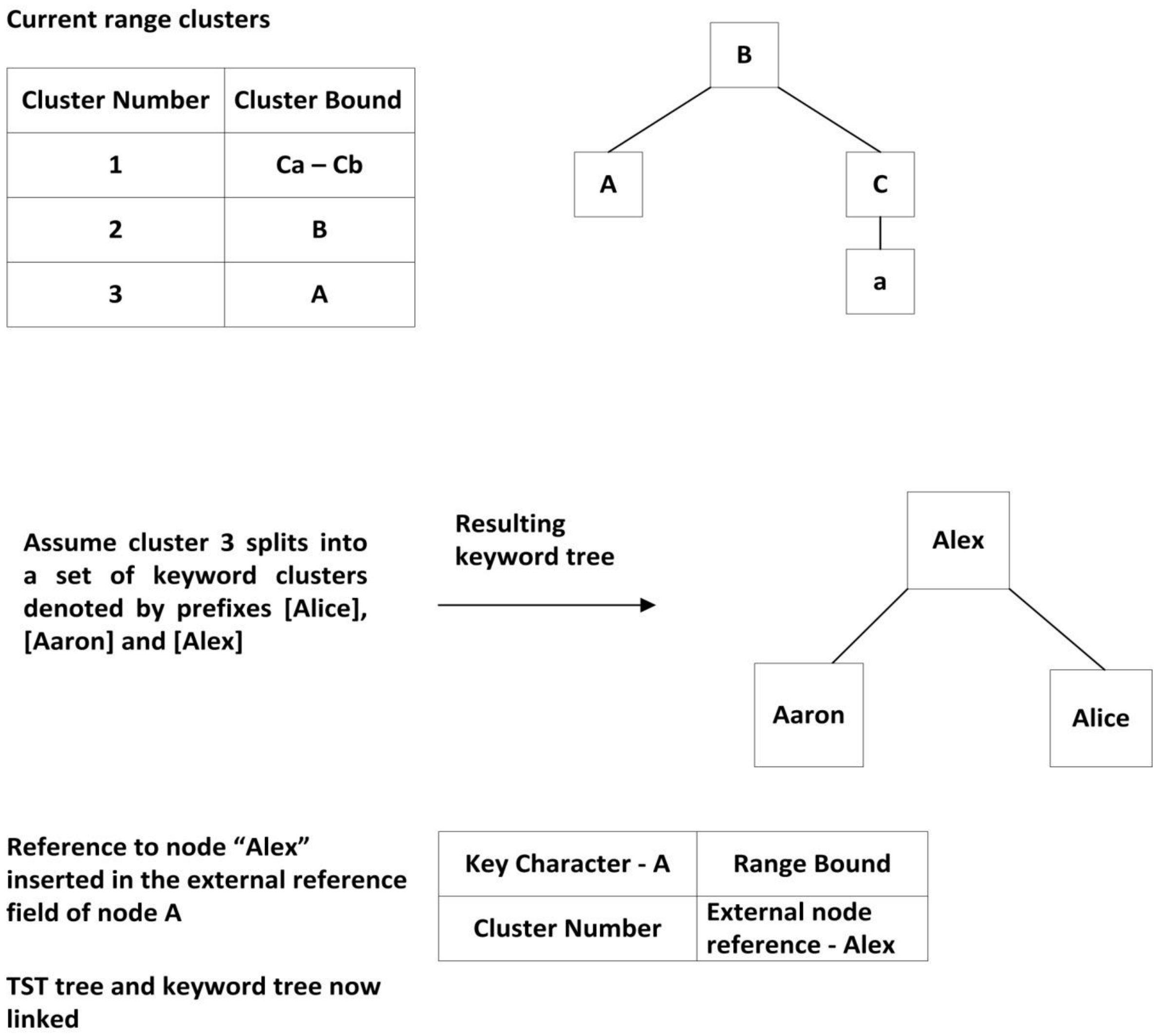
The basic idea of the matching process involves comparing the strings to identify the LCP as per our modified technique and assigning the corresponding PSGs to the clusters thus identified. Since the strings are pre-sorted, each string just needs to be compared to the most recently inserted LCP bound. In case a matching like case 3 in Figure 32 occurs, the prefix entry is updated to the newly obtained prefix. The running time of the matching process is linear and proportional to the total number of strings being compared. After the process is completed, the number of PSGs associated with each of the resulting prefixes is checked and a prefix is promoted to a cluster if its size is greater than the minimum cluster size threshold. Any strings that remain unassigned after this process are classified according to the normal clustering process discussed earlier.



```
Input : [Set of strings s and their corresponding PSG id]
prefix set – Table of format [prefix, list of members]
Initialize prefix set to null

Function evaluateLCPMatch
{
   for index = 0 to size of string set s
   if(prefix set == null)
            assign s[index] to prefix set
   else
   {
            compute LCP (s[index], last entry in prefix set)
            if keyword boundary match
                    assign PSG as member of prefix set
            else if keyword match at earlier boundary
                    assign PSG as member of prefix set
                    update prefix entry in the prefix set
            else
                    assign s[index] to prefix set
    }
    End for
}
```

Figure 33. Clustering PSGs based on modified LCP technique

The outcome of the LCP matching process is a list of long prefix bounds that denote a cluster as well as the PSGs associated with each of them. We use a secondary tree structure to organize these bounds that we denote as the *keyword tree*. This tree is a binary search tree where each node corresponds to one of the prefixes obtained in the previous step. Each node in this tree is assigned a unique cluster number and they follow the usual rules associated with the normal string range clusters. Subsequent PSGs that have string attributes with a prefix equal to any of the keyword tree nodes are assigned to the corresponding cluster. Figure 34 illustrates the organization of the keyword tree. In order to link the keyword tree with the main TST index, we utilize the external node reference field of the TST nodes. This link is useful as we can explore the two trees simultaneously while carrying out insertion and lookup operations.



Current range clusters

Cluster Number	Cluster Bound
1	Ca – Cb
2	B
3	A

Assume cluster 3 splits into a set of keyword clusters denoted by prefixes [Alice], [Aaron] and [Alex]

Resulting keyword tree

Reference to node "Alex" inserted in the external reference field of node A

Key Character - A	Range Bound
Cluster Number	External node reference - Alex

TST tree and keyword tree now linked

Figure 34. Generating the keyword tree

Whenever a cluster split operation results in the formation of such keyword clusters, this field of the node corresponding to the bounds of the cluster being split and add a reference to the first. When the size of a cluster corresponding to the keyword tree exceeds the maximum size threshold, it is split according to our usual radix sort based clustering process. The only difference in this case is since the strings are already known to share a common prefix; we start the matching process on the substring of the string beginning after the prefix. The resulting TST is attached to the keyword tree node that serves as the root node for that TST. Figure 35 highlights the result of a split operation occurring in a keyword tree node.

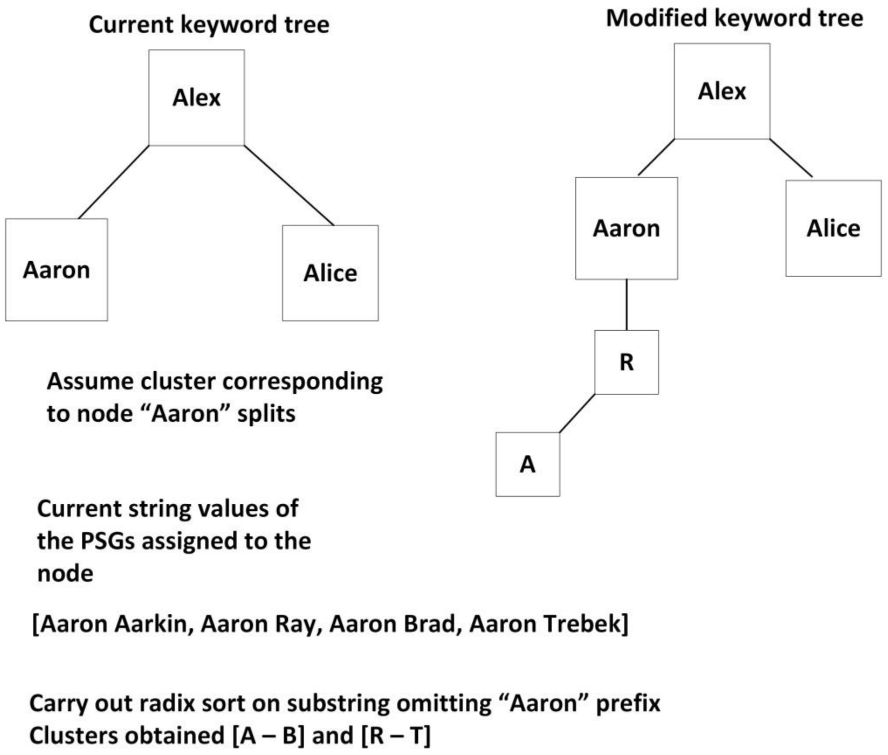

Current keyword tree
Alex
Aaron
Alice
Modified keyword tree
Alex
Aaron
Alice
R
A
Assume cluster corresponding to node “Aaron” splits
Current string values of the PSGs assigned to the node
[Aaron Aarkin, Aaron Ray, Aaron Brad, Aaron Trebek]
Carry out radix sort on substring omitting “Aaron” prefix
Clusters obtained [A – B] and [R – T]

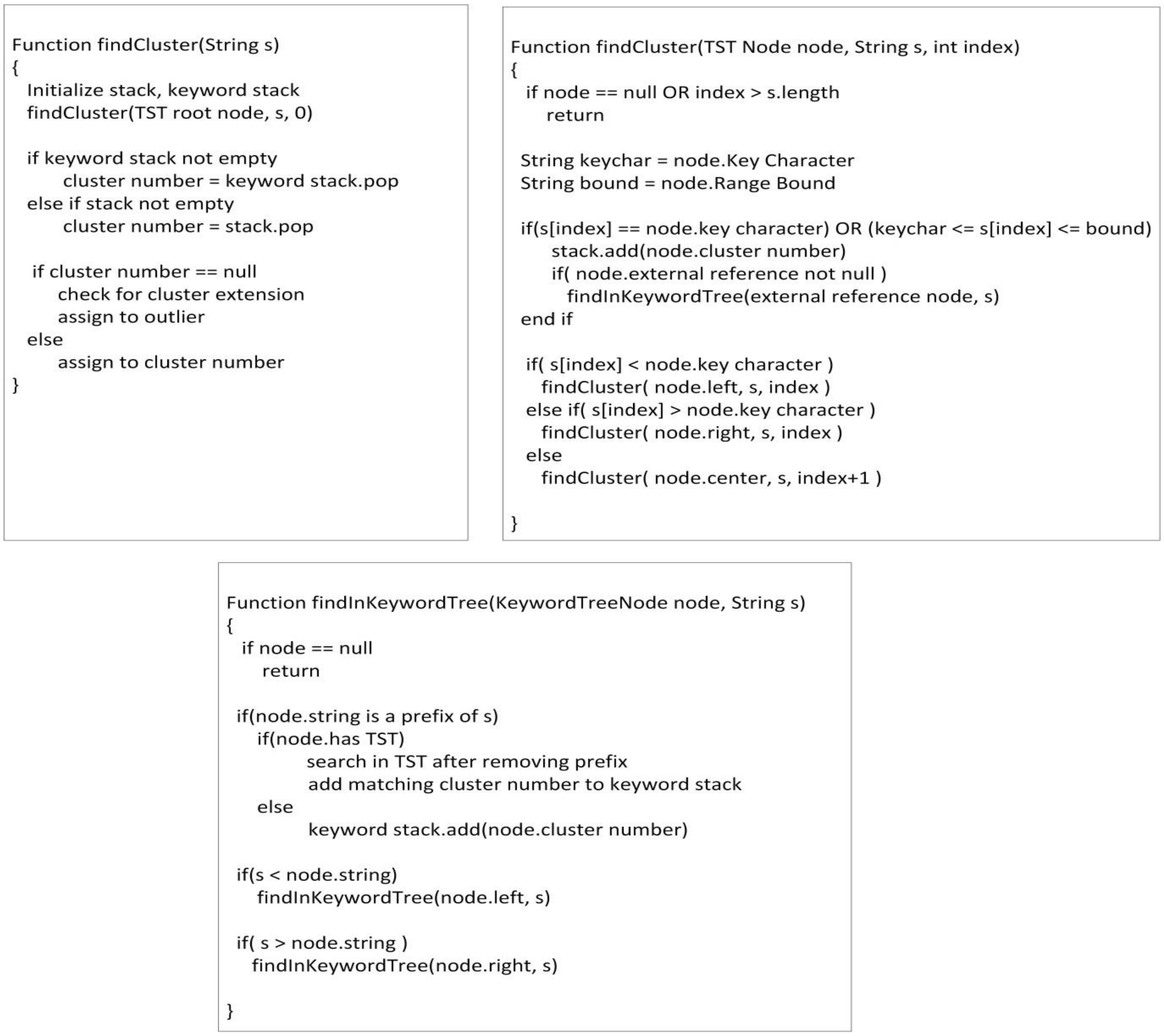
Figure 35. Splitting of a keyword tree node

This completes the discussion of the proposed index structure for strings. This index uses the standard techniques of radix sort as well as ternary search trees in a novel way to provide an adaptive index structure that provides an approximate representation of the string composition in a given semantic cluster. The radix sort is used to group the strings according to their shared common prefixes using the standard MST sort. However, the sort process is not carried out on the entire length of the strings. Rather, the sorting is carried out until the minimum prefix length that can be used to group the strings is achieved. This prefix length is not predefined and varies according to the set of strings being indexed. Also, in order to keep the total number of clusters manageable, we extend the idea of range clusters to string attributes as well by merging small sized clusters. The ternary search tree is designed to organize the cluster bounds to facilitate the lookup as well as the insertions of subsequent cluster bounds. The basic structure of the tree is kept intact but we make certain modifications to the tree node structure to represent the range cluster bounds as well as make the insertion/deletion of cluster bounds efficient. We also introduce the idea of keyword trees to minimize the size of the main tree and link the two trees in order to process string queries efficiently. As discussed previously, this index structure is different from the existing string indexing techniques due to the unique constraints and requirements imposed by the Coalition system.

6.4 Index maintenance operations

6.4.1 Assigning a PSG to a range cluster

When a PSG registers a string attribute with Coalition, it needs to be assigned to the appropriate cluster. The pseudocode for identifying the range cluster for a PSG with a particular string attribute is illustrated in Figure 36.



```
Function findCluster(String s)
{
  Initialize stack, keyword stack
  findCluster(TST root node, s, 0)

  if keyword stack not empty
       cluster number = keyword stack.pop
  else if stack not empty
       cluster number = stack.pop

   if cluster number == null
      check for cluster extension
      assign to outlier
  else
      assign to cluster number
}
```

```
Function findCluster(TST Node node, String s, int index)
{
  if node == null OR index > s.length
     return

 String keychar = node.Key Character
 String bound = node.Range Bound

 if(s[index] == node.key character) OR (keychar <= s[index] <= bound)
      stack.add(node.cluster number)
      if( node.external reference not null )
        findInKeywordTree(external reference node, s)
 end if

  if( s[index] < node.key character )
    findCluster( node.left, s, index )
  else if( s[index] > node.key character )
    findCluster( node.right, s, index )
  else
    findCluster( node.center, s, index+1 )

}
```

```
Function findInKeywordTree(KeywordTreeNode node, String s)
{
  if node == null
     return

 if(node.string is a prefix of s)
    if(node.has TST)
           search in TST after removing prefix
           add matching cluster number to keyword stack
    else
           keyword stack.add(node.cluster number)

 if(s < node.string)
    findInKeywordTree(node.left, s)

 if( s > node.string )
   findInKeywordTree(node.right, s)

}
```

Figure 36. Identifying the range cluster for a given string value

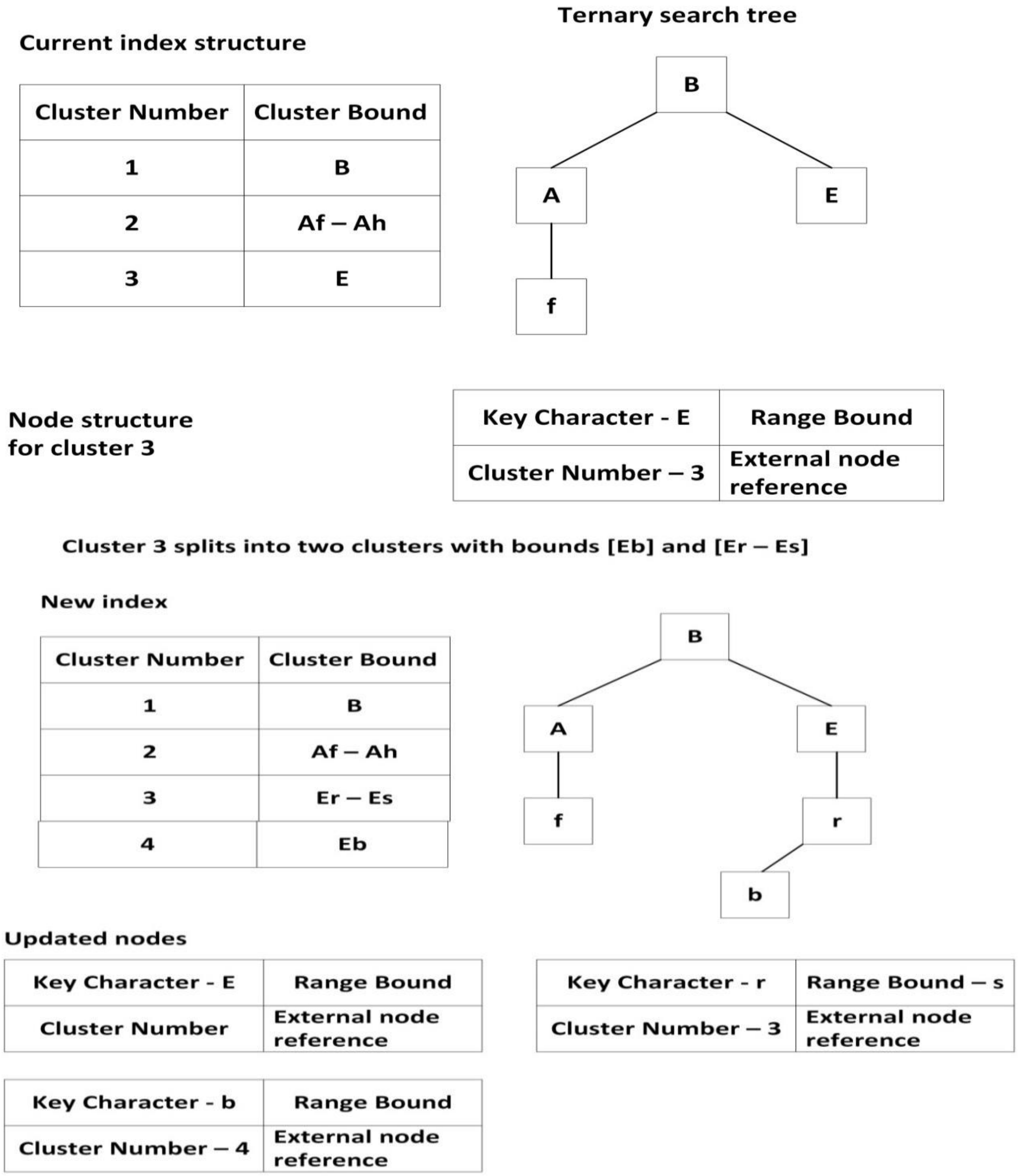
The algorithm is based on the standard TST search procedure and involves comparing the input string with the existing tree index until there are no more nodes to continue the comparison or the string is finished. The comparison is carried out using the normal string comparison procedures. Whenever a character is found to match against the key character of a node (direct or range), the cluster number (if valid) of the node is added to a stack. If a node contains a reference to a keyword tree node, the string is forwarded to that node where it is evaluated in the keyword tree. The PSG is either assigned to a normal range cluster or a keyword cluster depending on the matching clusters found.

Depending on the cluster composition, the string value of a PSG can match multiple clusters. In this case, the PSG is assigned to the cluster which has the longest prefix matching its string attribute value. The running time of the range cluster identification process depends on the size of the TST as well as the keyword tree. The average case time to identify a matching cluster is log N + log n where N and n correspond to the size of the TST and the keyword tree respectively. In case there is no matching cluster found, we can utilize the previously discussed concept of using an outlier cluster having no fixed range bounds to accommodate this value. Similar to the case with numeric attributes, the outlier cluster does not have any cluster bounds and is thereby not incorporated into the main index structure. The PSGs belonging to the outlier cluster just form a separate P2P network and there is no additional overhead required to maintain the same. After the relevant cluster number has been identified, the PSG registers with the corresponding P2P network.

6.4.2 Cluster splitting and merging operations

The cluster splitting operation is invoked whenever the membership size of a cluster exceeds the maximum threshold size and involves the execution of the indexing process described in Section 6.3.2 within the members of that cluster. The rationale behind the selection of the maximum threshold is same as discussed in Section 4.2.1. This applies to an outlier cluster as well. The splitting of a cluster can result in either a set of normal range clusters, a set of keyword clusters or a combination of both. After the new clusters have been generated, we need to update the index structure accordingly. When a cluster is split, the resulting cluster bounds are a more refined version of the original cluster bounds. In light of this observation, we do not delete the nodes corresponding to the original cluster bounds from the tree. Rather, we remove the cluster information from those nodes by resetting their cluster numbers.

The newly formed clusters are now assigned a unique cluster number. This operation is followed by inserting the new cluster bounds and updating the cluster numbers for all the affected cluster bounds. Figure 37 illustrates the updating operation carried out on the TST as a result of a cluster split.



Current index structure

Cluster Number	Cluster Bound
1	B
2	Af – Ah
3	E

Node structure for cluster 3

Key Character - E	Range Bound
Cluster Number – 3	External node reference

Cluster 3 splits into two clusters with bounds [Eb] and [Er – Es]

New index

Cluster Number	Cluster Bound
1	B
2	Af – Ah
3	Er – Es
4	Eb

Updated nodes

Key Character - E	Range Bound
Cluster Number	External node reference

Key Character - r	Range Bound – s
Cluster Number – 3	External node reference

Key Character - b	Range Bound
Cluster Number – 4	External node reference

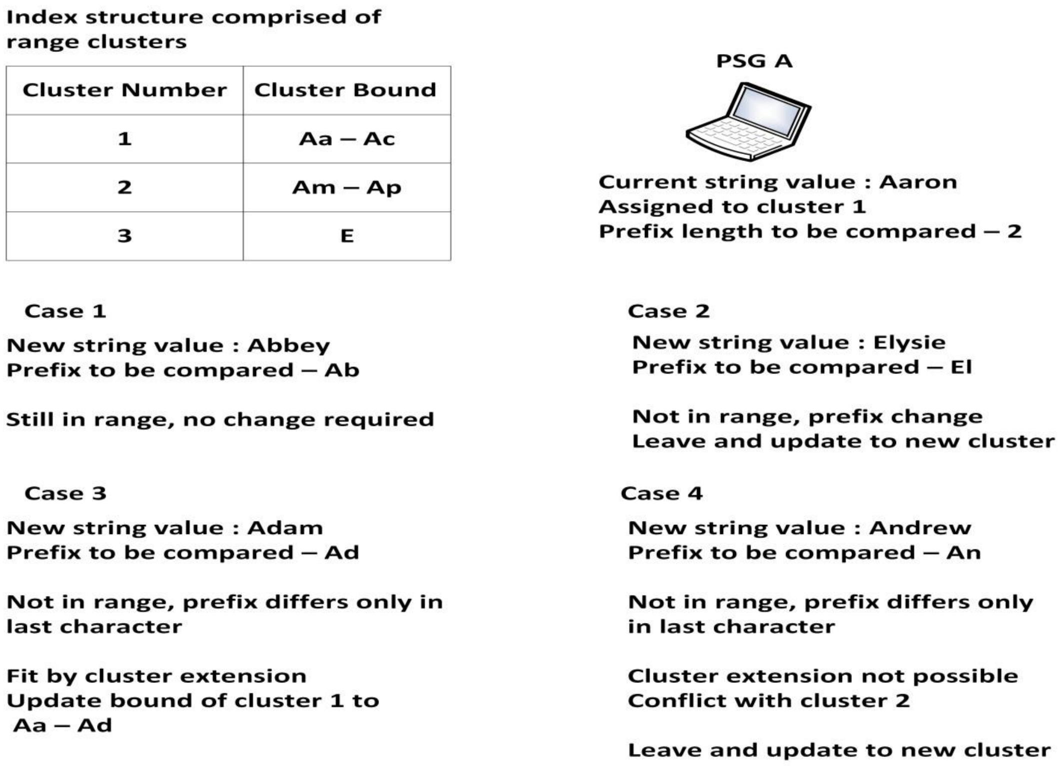
Figure 37. String cluster split operation

In case the splitting of a cluster results in the formation of one or more keyword clusters, these clusters are inserted in the keyword tree as described in Section 6.3.3. The cluster information corresponding to the node being split is removed from the TST and the other cluster numbers are updated accordingly. The TST and the keyword trees are then linked using the external reference field of the TST node. We have already discussed the splitting operation in a keyword tree node.

The merge operation is invoked when the size of a particular range cluster falls below the minimum cluster size threshold. In this case, the cluster bounds are sorted alphabetically to identify the candidate clusters for merging. The usual constraints for the merging process discussed in Section 6.3.2 apply. In case a cluster cannot be merged, the members are assigned to the outlier cluster and the cluster information is removed from the TST nodes. Since a keyword tree is organized as a binary search tree, the members of a node with a low membership size are directly assigned to the outlier cluster. In order to update the tree, the corresponding node from the keyword tree needs to be deleted. Since the deleting of the node will require the keyword tree to be updated, we choose to defer the deletion temporarily and instead reset the cluster number of the node to preserve the correctness of the index. The keyword tree can be periodically rebuilt to remove the nodes with no members.

6.4.3 Index update in case of string value change

Compared to numeric attributes, the handling of changes in string values is different as there is no set distribution pattern followed for the changes. The concept of edit distances can be used to determine the degree of string change but they are only useful when the whole strings are considered as opposed to the prefix based classification followed in our approach. Since the index structure uses the prefixes to organize strings into clusters, we can use this information to deem whether a PSG needs to move to a different range cluster. The details of the index update operation for string attributes is illustrated in Figure 38. When a PSG having a string value is associated with a cluster, a copy of the cluster bounds are passed to the PSG. Since the PSGs are grouped according to their shared prefixes, subsequent changes in the string values need to be checked only against the necessary prefix length. Consequently, each PSG maintains a copy of the prefix of length *l* of its local string value where *l* is the length of its assigned cluster bounds.



Index structure comprised of range clusters

Cluster Number	Cluster Bound
1	Aa – Ac
2	Am – Ap
3	E

PSG A

Current string value : Aaron
Assigned to cluster 1
Prefix length to be compared – 2

Case 1

New string value : Abbey
Prefix to be compared – Ab

Still in range, no change required

Case 2

New string value : Elysie
Prefix to be compared – El

Not in range, prefix change
Leave and update to new cluster

Case 3

New string value : Adam
Prefix to be compared – Ad

Not in range, prefix differs only in last character

Fit by cluster extension
Update bound of cluster 1 to
Aa – Ad

Case 4

New string value : Andrew
Prefix to be compared – An

Not in range, prefix differs only in last character

Cluster extension not possible
Conflict with cluster 2

Leave and update to new cluster

Figure 38. Cluster update operation for string attributes

As can be observed in Figure 38, all further changes in the string value are compared against this prefix. Any changes that affect only the characters beyond the designated prefix length need not be considered. The different scenarios that can occur as part of the index update operation for string values are illustrated in detail in Figure 38. As can be observed from case 3 in the figure, one possibility of deferring an index update operation is to check if the PSG can be accommodated in its current cluster by extending the cluster bounds. Although the cluster bound extension can result in a degradation of the query localization performance of the index, the tradeoff achieved is the minimization of the update overhead associated with joining and leaving the P2P networks.

6.5 Processing string queries using the index structure

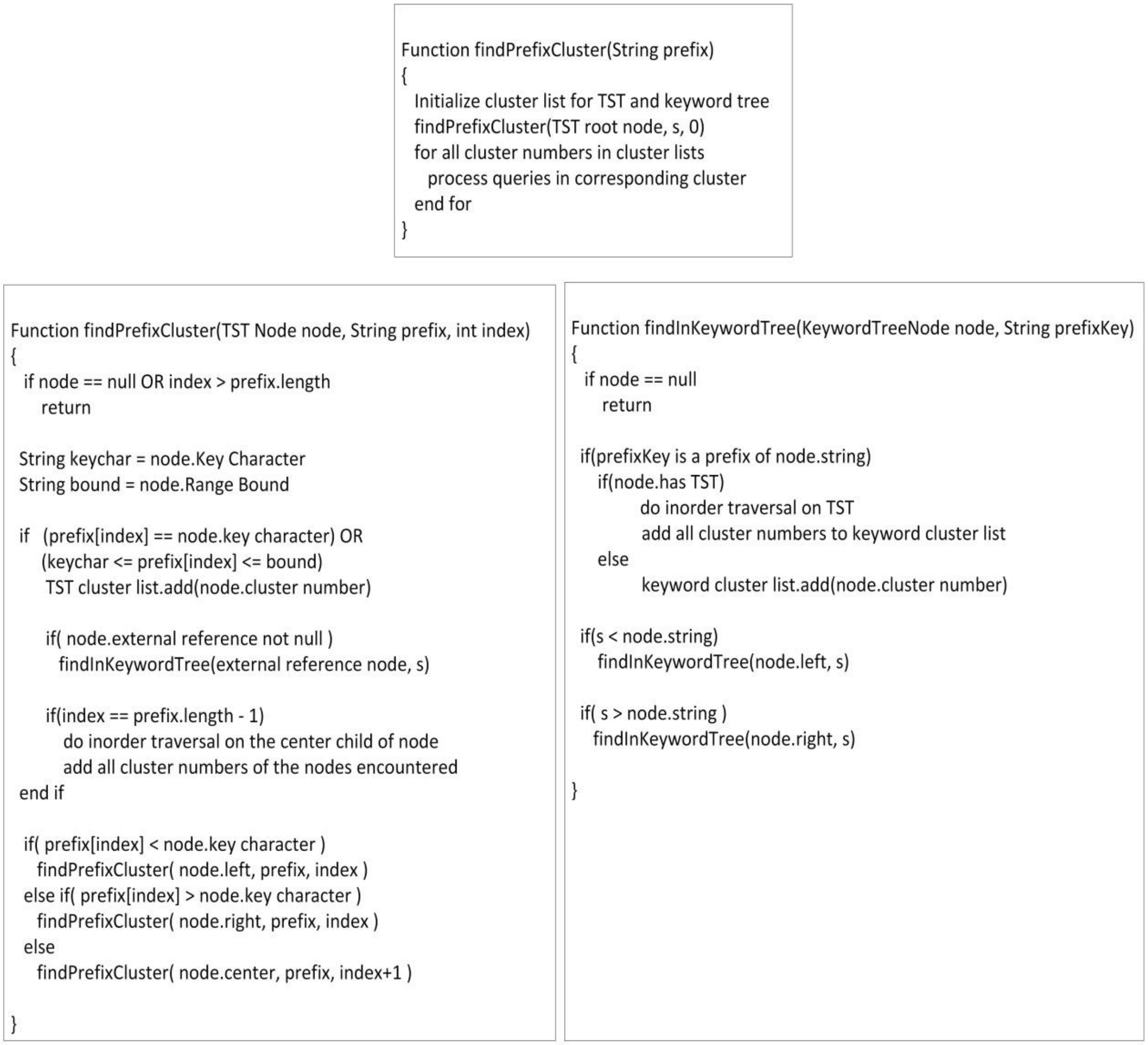
6.5.1 Exact and prefix matching queries

Since the proposed index structure organizes the strings using prefixes, the following types of queries can be accurately supported – prefix matching, exact matching and range queries. The organizational structure of the index prevents it from addressing wildcard matching queries where no prefix information is present which is a limitation of our index structure. In order to satisfy wildcard queries, it is necessary to maintain the full strings in the memory which violates one of the primary data storage constraints of the Coalition system.

The processing of an exact string matching query follows the same procedure used to assign a PSG to a range cluster as discussed in Section 6.4.1. Since the index provides only an approximate ordering of the string attributes, the query processor locates the range cluster corresponding to the longest prefix matching the search string. In case no matching cluster is found, the query is redirected to the outlier cluster. The query is now flooded to all the PSG members of the cluster and locally evaluated at the PSGs. Although we cannot guarantee that a matching answer will be found, we can ensure that if a matching answer(s) exists, it will be found in the range cluster(s) located using the search process.

The second type of queries that can be supported by the index is to find strings beginning with a particular prefix. The pseudocode for identifying the candidate clusters for a prefix search is illustrated in Figure 39. Based on the composition of the index structure as well as the prefix to search for, there can be two distinct cases that can arise during the processing of such queries. In the first case, a range cluster (or a keyword cluster) exists with a cluster bound that exactly matches the prefix to be searched or has a cluster bound with a length less than that of the searched prefix. The second case occurs when the index contains range clusters with cluster bounds having a length greater than the prefix to be searched. In this case, all the matching clusters need to be added to the candidate list.

To understand the implications of the second case, consider the following set of range clusters in the index [Ab], [Alf] and [Ale] and a prefix search using “Al” as the search key. Since there is no range cluster that exactly matches the search key, a simple search would yield no matching answers. In order to handle this case, we perform the checking of the prefix length and the current character being compared. When the character being compared is the last character of the search key and it matches the key character of a node, we carry out an inorder traversal of the tree rooted at the middle child of the node and add the cluster numbers of all the nodes thus encountered. In the previous example, this check would happen when the search reaches the node corresponding to “Al”.



```
Function findPrefixCluster(String prefix)
{
  Initialize cluster list for TST and keyword tree
  findPrefixCluster(TST root node, s, 0)
  for all cluster numbers in cluster lists
    process queries in corresponding cluster
  end for
}
```

```
Function findPrefixCluster(TST Node node, String prefix, int index)
{
  if node == null OR index > prefix.length
     return

 String keychar = node.Key Character
 String bound = node.Range Bound

 if   (prefix[index] == node.key character) OR
     (keychar <= prefix[index] <= bound)
      TST cluster list.add(node.cluster number)

      if( node.external reference not null )
        findInKeywordTree(external reference node, s)

      if(index == prefix.length - 1)
         do inorder traversal on the center child of node
         add all cluster numbers of the nodes encountered
 end if

  if( prefix[index] < node.key character )
    findPrefixCluster( node.left, prefix, index )
  else if( prefix[index] > node.key character )
    findPrefixCluster( node.right, prefix, index )
  else
    findPrefixCluster( node.center, prefix, index+1 )

}
```

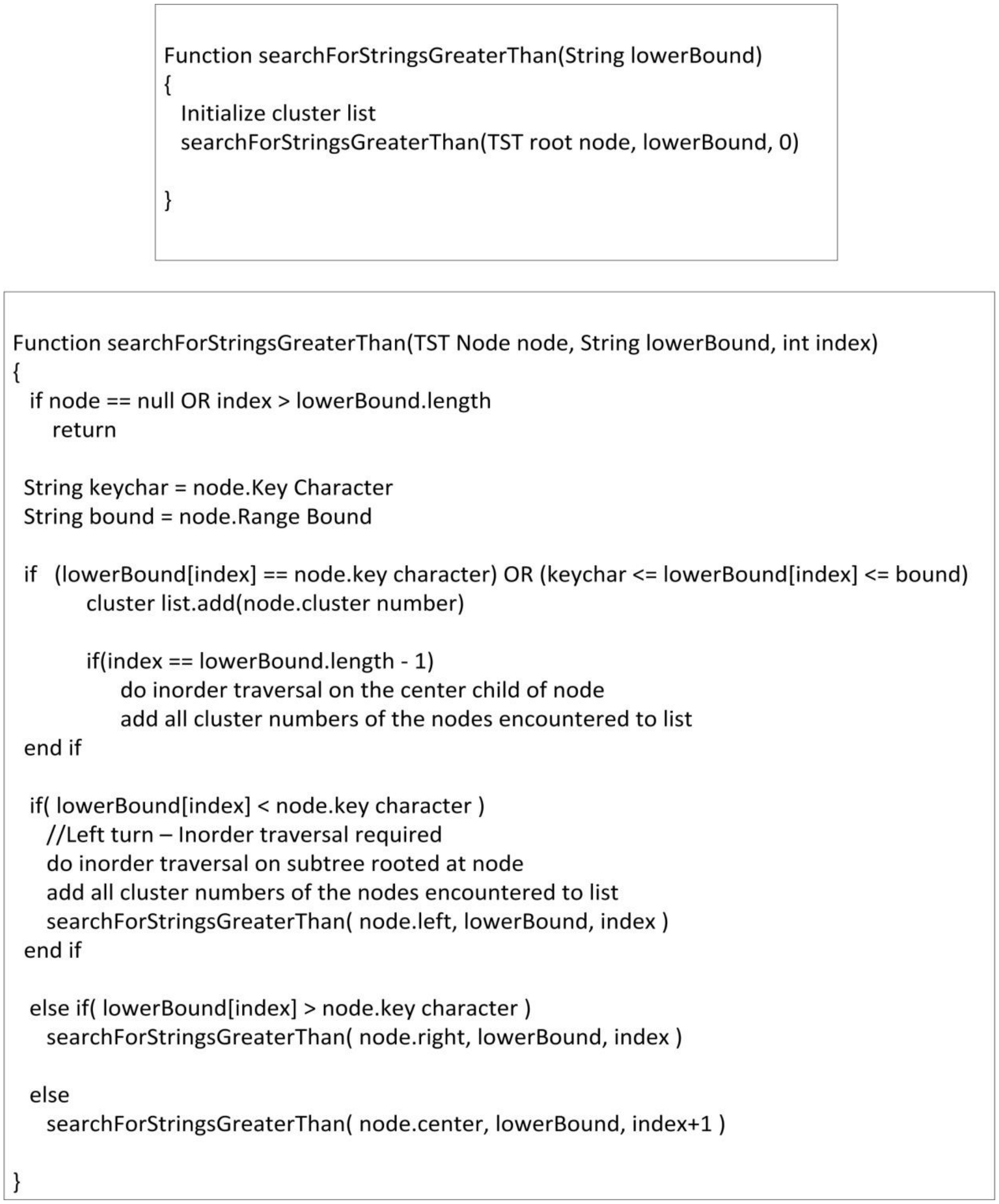
```
Function findInKeywordTree(KeywordTreeNode node, String prefixKey)
{
  if node == null
     return

 if(prefixKey is a prefix of node.string)
    if(node.has TST)
          do inorder traversal on TST
          add all cluster numbers to keyword cluster list
    else
          keyword cluster list.add(node.cluster number)

 if(s < node.string)
    findInKeywordTree(node.left, s)

 if( s > node.string )
   findInKeywordTree(node.right, s)

}
```

Figure 39. Prefix search process

The inorder traversal at the middle child would result in the addition of the cluster numbers corresponding to [Alf] and [Ale] which satisfies the query conditions. The outlier cluster is also included in the cluster list for the sake of completeness. When multiple clusters are involved in the query, the query processing is carried out using the technique discussed in Section 5.5 that involves the use of the expected answers values and the membership size of the range clusters to minimize the response time.

6.5.2 Range queries

The third type of queries that can be supported by the index structure comprise of range queries. The range query format for strings is usually [Range A, Range B] and the objective of the query is to look for all matching strings that lie lexicographically between these two strings. For example, consider a string set [Aaron, Amelia, Bobby, Nicole, Nancy] and a range query bound of [Al, Nc]. The matching strings in this case would be [Amelia, Bobby, Nicole]. Compared to numeric attributes, the processing of range queries for strings is slightly more complicated as it depends on the lengths of the query bounds as well as the index composition. The pseudocode for this process is illustrated in Figure 40.



```
Function searchForStringsGreaterThan(String lowerBound)
{
   Initialize cluster list
   searchForStringsGreaterThan(TST root node, lowerBound, 0)

}
```

```
Function searchForStringsGreaterThan(TST Node node, String lowerBound, int index)
{
   if node == null OR index > lowerBound.length
       return

  String keychar = node.Key Character
  String bound = node.Range Bound

  if   (lowerBound[index] == node.key character) OR (keychar <= lowerBound[index] <= bound)
           cluster list.add(node.cluster number)

           if(index == lowerBound.length - 1)
                do inorder traversal on the center child of node
                add all cluster numbers of the nodes encountered to list
  end if

   if( lowerBound[index] < node.key character )
     //Left turn – Inorder traversal required
     do inorder traversal on subtree rooted at node
     add all cluster numbers of the nodes encountered to list
     searchForStringsGreaterThan( node.left, lowerBound, index )
  end if

   else if( lowerBound[index] > node.key character )
     searchForStringsGreaterThan( node.right, lowerBound, index )

   else
     searchForStringsGreaterThan( node.center, lowerBound, index+1 )

}
```

Figure 40. Searching for strings greater than a given string

The initial step in the range query process involves searching for all the range clusters corresponding to strings that are greater than the lower range bound. The search process basically involves searching the TST for the lower range bound and doing an inorder traversal on the sub-tree rooted at a node from which the query is forwarded to the left child node. Due to the organizational structure of the TST, this step ensures that the range clusters corresponding to strings greater than the range bound are added to the cluster list. After the completion of this step, the range clusters and their cluster bounds thus obtained need to be checked against the upper range bound to generate the final cluster list. The straightforward way to carry out this operation is to process each range bound individually against the upper bound and remove all the non-qualifying entries.

Alternatively, we can use a symmetric version of the pseudocode in Figure 40 to generate a list of the range clusters corresponding to strings that are less than the upper bound. An intersection of the two lists can now be carried out to generate the final cluster list. The processing of range queries in the keyword tree is carried out in a single pass by performing an inorder traversal of the tree and adding all the matching nodes. The processing of the range query in the keyword tree has a running time proportional to the size of the keyword tree. After the final cluster list has been obtained, the query is forwarded to all the PSGs that are the members of the clusters and is processed according to our standard procedure discussed in Section 5.5.

The processing of the range query is the most complicated among the different types of queries discussed here as it involves making two passes over the index (search for lower bound and upper bound). The running time of a single pass is O (log N) + O (n) where the log N factor corresponds to locating the range bound in the TST whereas the inorder traversals happening as a result of the search contribute to the linear time where n is the size of the sub-tree traversed due to the search. In the worst case scenario, n is equal to the TST size (N) when there is a range query that encompasses the entire set of range clusters present in the index. Since our proposed index is constructed on a set of shared prefixes and prefix ranges, we expect the size of the resulting TST to be smaller than that of one constructed over the entire set of strings. This is expected to minimize the impact of the tree size on the overall running time.

6.6 Experimental results

6.6.1 Index performance with respect to query response time

The experimental setup to evaluate the string index is similar to the one discussed in Section 5.6.1. The proposed index for strings was developed and integrated with the Coalition system. For the first part of our evaluation, we study the index performance with respect to the query response time. All the results in this section refer to the query processing in a single semantic cluster and in a stable system state. The maximum and minimum cluster size thresholds were fixed at 50 and 10 respectively. All the experimental results are represented using a 95% confidence interval wherever applicable. We use a random set of names retrieved from the DBLP data set [102] as the input data for the first phase of testing and assign each name attribute to a single PSG instance. Similar to our previous experiments, the response time is measured as the time interval between the issuing of a query and the reception of the answers. Figure 41 illustrates the variation in the response times along with the network size achieved for an exact matching query where only a single PSG contains the valid answer. The string to be searched for was randomly selected from the data set. The response times are taken as the average of 300 query runs.

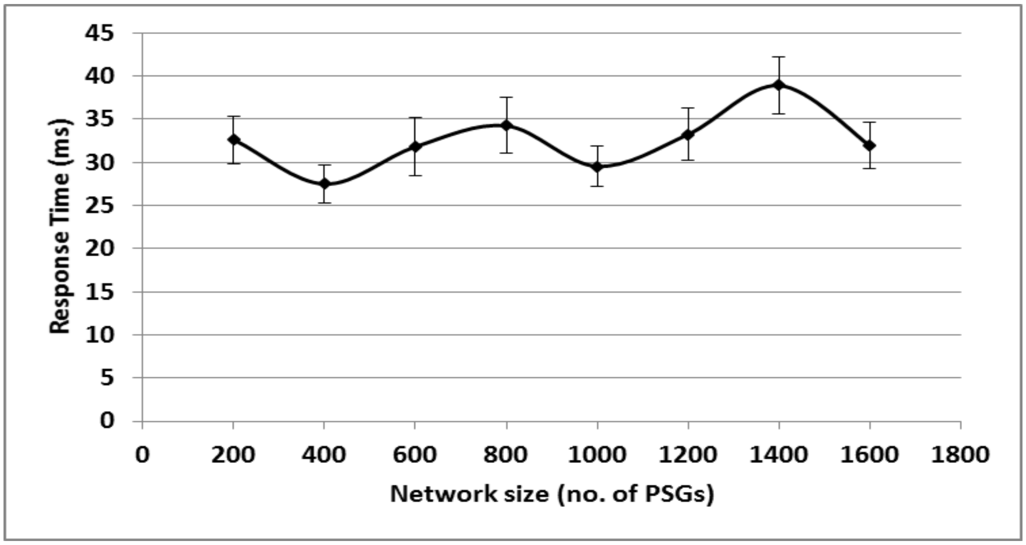


Figure 41. Query response time for exact string match

As can be observed from the results, the variations in the response time with the increase in the network size are minimal which indicates that the index is able to localize queries effectively even when the total number of PSGs is large. Since this is a simulation based study, the actual response times may vary when a real-world deployment is concerned. In that case, there can be an increase in the response time due to the end to end network communication delays between the PSGs and the query processor. However, the presence of the index will still be useful as the network delays will be confined to a subset of all the PSGs present in the system.

We next examine the index performance for range queries. As discussed in Section 6.5.2, the processing of range queries involves making multiple passes over the index structure and the running time depends on the size of the TST as well as the keyword tree. In order to incorporate the keyword tree as part of the experimental run, we incorporate a set of names sharing a common first name as part of the data set. The range query evaluation is carried out by varying the network size as well as the number of valid answers for a given query. The range query bounds are randomly selected from the set of input strings. Figure 42 illustrates the variation in the query response times for different network sizes and number of PSGs with valid answers.

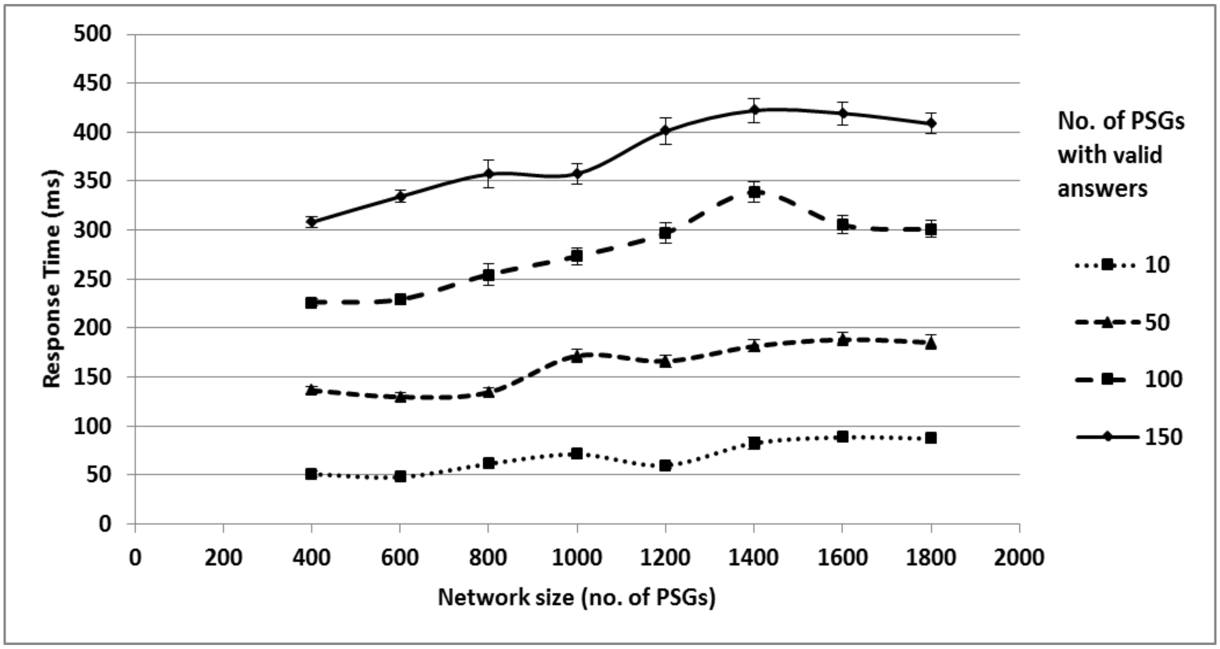


Figure 42. Query response times for range queries

The results clearly indicate that for a given number of PSGs with valid answers, there is an increase in the response time when the network size is increased. This behavior is expected as the size of the index increases with the increase in network size. The effect of the index size on the processing of range queries has been discussed in Section 6.5.2.However, the increase curve is observed to be smooth as opposed to a steep increase. This indicates that the index is still able to localize queries effectively even with the increase in the network size. As the index is constructed using a set of prefixes rather than the entire set of strings, the increase in the index size is expected to be gradual which compensates for the increase in the overall network size.

The initial results indicate that the index performance is satisfactory with respect to the query response time when a stable system state is considered. The variations in the query response time along with the increase in the network size are also observed to be gradual. Since the formation of the index structure is dependent on the actual set of strings being indexed as well as their sequence of insertion, the set of results discussed here are representative and intended to highlight the index performance with an average set of test data values.

6.6.2 Index performance with dynamic string data

As observed in Section 6.4.3, the problem of dealing with dynamic string values is challenging as there is no well-defined way to predict the patterns of string data change. Nevertheless, we have a provision in our index structure to handle the updates of string values. In this section, we evaluate the index with respect to its capability to handle dynamic string values. We use the data file based technique discussed in Section 5.6.3 to simulate the set of string values generated at a PSG. As part of the evaluation of the index performance with dynamic data, two types of data files need to be generated – one representing static data and the other representing dynamic data. In order to represent the static data files, we randomly select a name from the DBLP data set discussed earlier and populate a file with multiple instances (3600) of this name. This data file is associated with a PSG instance and represents a PSG with static data. The generation of the dynamic data files is a bit more complicated. As we saw in Section 6.4.3, an update operation by a PSG occurs only if the string change affects its prefix of length l where l is the length of its currently assigned cluster bounds. This observation is utilized while generating the dynamic data files. For the first phase of testing, the dynamicity is introduced by generating a set of strings that share the same starting character but may differ in the rest of the string. For example, consider a static data file corresponding to the name “Alan”. In order to add dynamicity to this file, we replace 40% of the total number of entries in this file with other distinct names starting with “A”. These names are derived from the DBLP data set as well. The dynamicity is simulated by inserting these names into the data file at random positions. This data file can now be associated with a PSG instance to represent a PSG with dynamic data. We generate multiple such files with different name values and associate them with multiple PSG instances to create a randomized data set for testing.

The setup for the experiment is as follows – for a given network size, we introduce a fixed percentage of PSGs having dynamic data values whereas the remaining have static data. The percentage is varied from 10% - 50%. We then proceed to evaluate the query response times achieved for a range query with 100 valid answers and compare it with that the results achieved for a stable system state. The network size is varied from 800 – 1800 and we read a new data value from the file at intervals of 1 second to simulate the change of dynamic data. The results are illustrated in Figure 43. The entries corresponding to the 0% of dynamic PSGs refer to a stable system state.

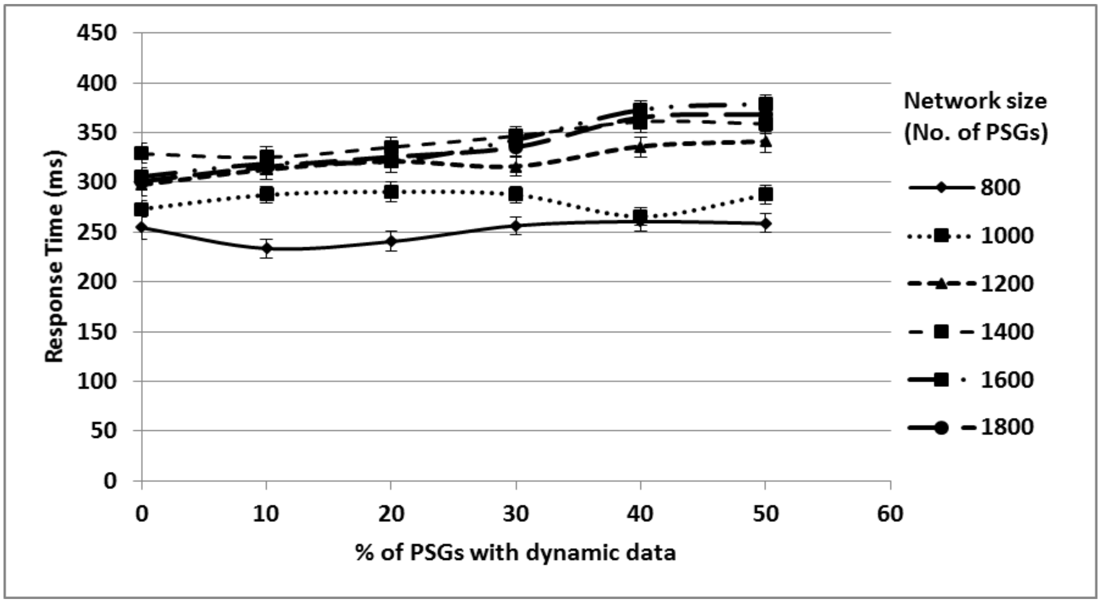


Figure 43. Query response time with dynamic string data – Case 1

The results indicate that for a given network size, the variations in the query response time with the increase in the percentage of PSGs having dynamic data is minimal especially for the lower network sizes as well as when the number of PSGs with dynamic data is low. We have already discussed the implications of the dynamicity of data on the query response time in detail in Section 5.6.3 where we observed that the frequent leave/join operations as well as the occurrence of cluster split/merge operations can cause an increase in the query response time. Based on the results obtained, we can infer that the dynamicity of data does not cause a major impact on the query response time if the change in data values only affects part of the prefix without changing the starting character of the string value.

For example, if the current classification level for strings starting with "A" is a cluster with bounds [A – B], any changes in the string after the first character will not cause an update. Further, as observed in Section 6.4.3, updates for string values that differ only in the last character with their current cluster bounds can be deferred by extending the cluster bounds. This technique is useful when the network size increases and the prefix length used to classify strings increases beyond a single character and plays a part in minimizing the effect of the data dynamicity on the query response time.

For the second part of the experiment, we follow the same experimental setup and execution process with a different approach in generating the dynamic data files. In this case, the dynamicity is introduced by randomly replacing 40% of the total entries in a static data files by any randomly selected string. There is no constraint imposed wherein the chosen strings should share the same starting character. In other words, each data file representing a set of dynamic string data now comprises of truly random string values. The results achieved with this data set are illustrated in Figure 44. As can be observed, there is a marked increase in the query response time for a given network size when the percentage of PSGs with dynamic data increases beyond 30%. This increase is especially noticeable for larger network sizes where the corresponding number of PSGs with dynamic data is high. If we consider the network sizes of 1400 – 1800, there is a 49% - 52% increase in the response time when the percentage of PSGs with dynamic data is increased from 10% to 50%.

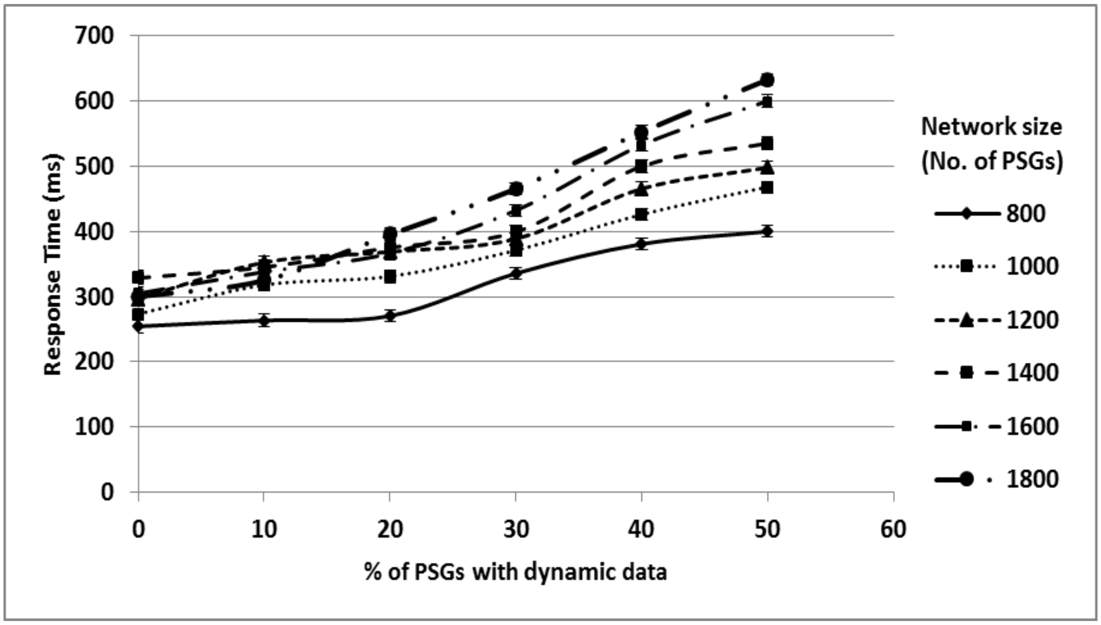


Figure 44. Query response time with dynamic string data – Case 2

Since the data change pattern is completely random now and cannot be deferred using the techniques discussed in Section 6.4.3, the PSGs exhibit frequent leave/join operations which increases the index update overhead thereby affecting the query response time. This indicates that the proposed string index structure falls short of handling a large number of string values that exhibit very rapid and random changes. Although the chance of occurrence of this kind of behavior is rare in reality, we need to provide a fail-safe in case this issue arises. A possible approach to handle this issue would be to have each PSG keep track of its update frequency i.e. the number of update operations executed by it over the period of time elapsed since its initial assignment to a range cluster. If this frequency is observed to be very high, the PSG can be shifted to the outlier cluster which can contain the updates due to its property of having no fixed cluster bounds. The implementation of this technique to handle the update overhead issue is found to resolve the problem for the lower network sizes. However, in order to handle this problem for the larger network sizes, we observe that the maximum cluster size threshold for the outlier cluster needs to be extended as the number of PSGs with random data change patterns increases proportionately and they need to be accommodated as part of the outlier cluster.

6.6.3 Evaluation of index size and construction times

In the final part of the evaluation of our proposed index structure, we examine the variation in the size of the index along with the increase in network size. The size of our proposed index involves the total number of nodes in the TST as well as the keyword tree. As we discussed in Section 6.5.2, the processing of range queries is dependent on the size of the tree. We highlight the change pattern of the tree size growth using our proposed index structure vs a standard TST constructed using the same set of string values used to register to Coalition by the PSGs. The strings are derived from the test data set discussed in Section 6.6.1. The results are illustrated in Figure 45.

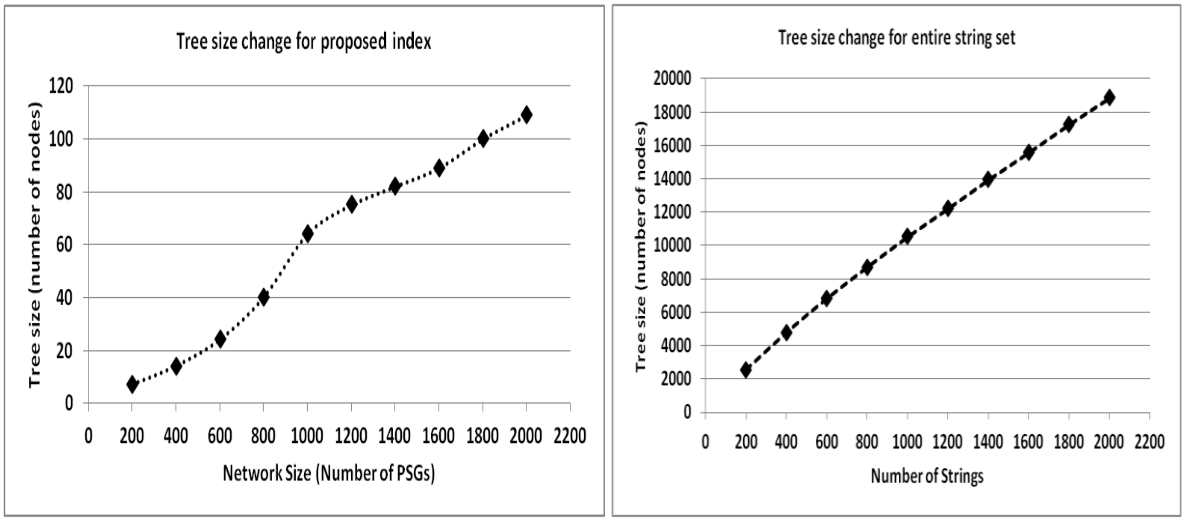
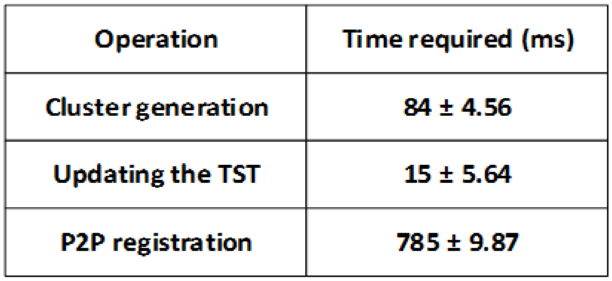


Figure 45. Variations of tree size with increase in network size

The figures clearly illustrate that the tree size change using our proposed index structure is much more gradual as compared to that of the tree constructed over the entire set of strings. The main reason behind this is the fact that we utilize the shortest prefix length that can impose an approximate ordering amongst the strings to construct our index. Also, the size of our index structure is significantly smaller when the total number of strings being indexed is considered. This reinforces our argument that the tree size will not have a significant impact on the running time. Again, the tree size depends on multiple factors including the set of strings being indexed as well as their insertion order in the tree. We believe that unless the string set comprises of exactly similar strings and possesses a reasonable amount of randomness, the index will exhibit a similar string size growth patterns as observed here.

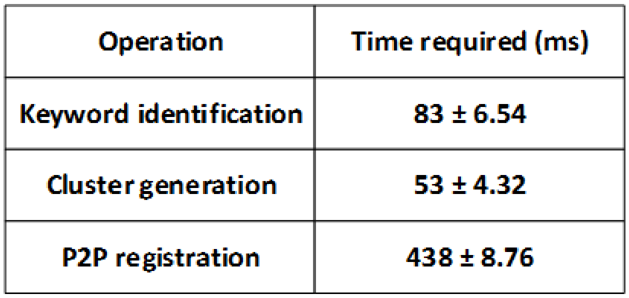
We now proceed to highlight the time measurement breakdown required for the index maintenance operations. These include the measurements for the cluster splitting operation as well as the keyword identification process. The results are highlighted in Tables 7 – 8. The results are consistent with the previously observed time measurement schemes in Chapter 4 and Chapter 5 wherein the registration of PSGs to their respective range clusters dominates the overall time for the clustering process. Again, the frequency of occurrence of the cluster maintenance operations will depend on the actual data being indexed, the sequence of insertion of the data values as well as the dynamicity of the data. If the indexed data is stable and does not exhibit a high degree of dynamism, these operations will not affect the overall index performance.

Table 7. Time breakdown for cluster splitting



Operation	Time required (ms)
Cluster generation	84 ± 4.56
Updating the TST	15 ± 5.64
P2P registration	785 ± 9.87

Table 8. Time breakdown for keyword cluster generation



Operation	Time required (ms)
Keyword identification	83 ± 6.54
Cluster generation	53 ± 4.32
P2P registration	438 ± 8.76

6.7 Summary

In this chapter, we discussed the design of an incremental tree based indexing structure for string context attributes. The index was designed using the concepts of radix sort and ternary search trees and provided a prefix based classification of the string attributes. The main idea was to use the radix sort technique to classify the strings according to the prefixes and organizing these prefixes through an augmented version of a ternary search tree. Although the use of a prefix based classification technique restricts the type of queries that can be supported on string attributes, we choose to use this technique as it satisfies our requirement of providing an approximate ordering on string attributes as well as provide a generic technique to query different types of context string attributes. The length of the prefix used to classify the strings is not predetermined and is chosen according to the string compositions. The index structure was designed to be adaptive and modify itself according to the movement of the PSGs. We also proposed a LCP based keyword identification technique to identify strings having shared long prefixes and provided a separate organizational structure for them to keep the size of the main index low. The experimental results indicated that the index structure was able to handle queries efficiently over different network sizes as well as small amounts of dynamism. Also, the variations in the tree size were observed to be slow with respect to the number of strings being indexed. However, it was also observed that the index performance suffers when subjected to extremely random dynamic data change patterns and we provided a preliminary technique to address the same. Similar to the numeric attribute index discussed in Chapter 5, the effectiveness of this index depends on the composition of the string data being indexed. As part of our future work in this direction, we would like to improve the LCP based keyword matching process to work with imperfect strings and reduce the reliance on the existing exact string matching technique.

CHAPTER 7

FUTURE WORK AND CONCLUSION

In this chapter, we present a future vision for our proposed data management system and propose the initial design idea of a multi-level indexing scheme capable of supporting queries over different scopes. The chapter is organized as follows – Section 7.1 highlights the problems and restrictions with regards to the context lookup operation in the current index structure. We also discuss the different options that can be used to augment the existing data management system. The initial system design as well as the proposed changes in the organizational scheme is discussed in Section 7.2. We also briefly highlight the implications on the context lookup operation due to the changes in the index organization. We summarize the contributions of this thesis and conclude it in Section 7.3.

7.1 Limitations of the proposed context data management system

In the previous chapters, we have proposed a context data management system that provides an incremental indexing scheme for numeric and string attributes as a preliminary solution to the problem of carrying out lookups efficiently over context data distributed across multiple context spaces (PSGs). These indexing schemes include – a range clustering approach to index numeric data, a mean-variance based indexing technique that improves upon the range clustering approach to provide an indexing scheme that can manage dynamic context data and an incremental radix sort and ternary tree based indexing scheme for string context attributes. The basic structure of these index structures comprises of a set of range clusters that provide an approximate ordering among the PSGs according to their data values. The proposed index structures are capable of handling different types of queries and are adaptive to the movements of the PSGs due to the leave/join operations as well as the updates occurring due to dynamic data changes. The experimental results also indicate that the index structures are able to provide access to the desired context information with reasonable query access times and can manage dynamic context data as well (primarily for numeric attributes). However, if we consider extremely large scale deployments such as beyond the size of a metropolis, the proposed indexing scheme may not be sufficient since it processes all queries with a larger search space (to be elaborated in the next paragraph) at a centralized query processor Further, the number of range clusters will also increase with an increase in the number of PSGs and the maintenance overhead of these clusters can outweigh the indexing benefits achieved due to the clusters.

We discussed the importance of a context-aware system to have the provision to support different query scopes in Section 2.1. The survey of the existing context data management systems in Section 2.2 revealed the fact that most of them support only a single query scope as part of their organizational scheme. Our proposed data management system currently supports a context search according to the domain/attribute/range of organization. This organizational scheme may not be sufficient (a) when the size of the domain is very large, such as the ‘home’ domain on the global scale, and (b) if we wish to support an extensive set of context lookup options that can be used by different types of applications.

7.2 Selecting additional indexing levels

Prior to the discussion about the design of the extended version of data management scheme, we need to explore the different alternatives that can be used to impose an additional ordering amongst the context sources based on the current organizational structure of Coalition. As discussed in Section 2.2, using the location of the context sources is a popular technique to enforce a selective query scope [29, 45, 48]. For example, a personal shopping application might need to retrieve a list of nearby shops that have a certain crowd level. In this case, a querying utility that allows filtering by location as well as attribute values will not only be useful but more efficient in the presence of a location-based indexing of the context attributes Hence, a location based ordering can be considered as one of the potential options for an additional indexing level. Hence, a location based ordering can be considered as one of the potential options for an additional indexing level.

Another approach towards providing an additional indexing level by further partitioning the context sources of a context domain into sub-domains as follow. For example, a shop can be classified into multiple categories like malls, bookstores, grocery shops and a home can further be classified as private apartments, landed houses, service apartments and so on. A detailed classification of the context domains would enable queries requiring information from a particular category of domains to be redirected to the relevant and yet smaller set of context sources easily.

The classification process of a high-level context domain into multiple sub-domains is challenging as there can be different classification parameters. For example, shops can be classified either on their type (stand-alone, part of shopping center etc) or based on the products stocked (bookstores, flower shops etc). These are just two of the possible classification techniques. Even if a generic technique to impose an ordering amongst the different context domains can be devised, it can result in a large number of sub-domains that need to be managed individually. Comparatively, a location based ordering can be carried out in a generic manner that is independent of the context domains. However, the imposition of a location based ordering may prove restrictive to support system scope queries. As part of our future work on the context data management system, we propose an organizational scheme that can support selective as well as system scope of queries.

7.3 Extending the current data management system

7.3.1 Overview of the proposed system architecture

Based on the discussion in the previous section, we propose an extension of our indexing scheme that imposes a location based ordering amongst the PSGs along with the current indexing technique. The initial system architecture is illustrated in Figure 46. As compared to the original Coalition architecture where a single context space manager was used to manage all the context domains and their associated PSGs, we use a collection of location specific context space manager (LCSM) that perform the same task at a location level(s).

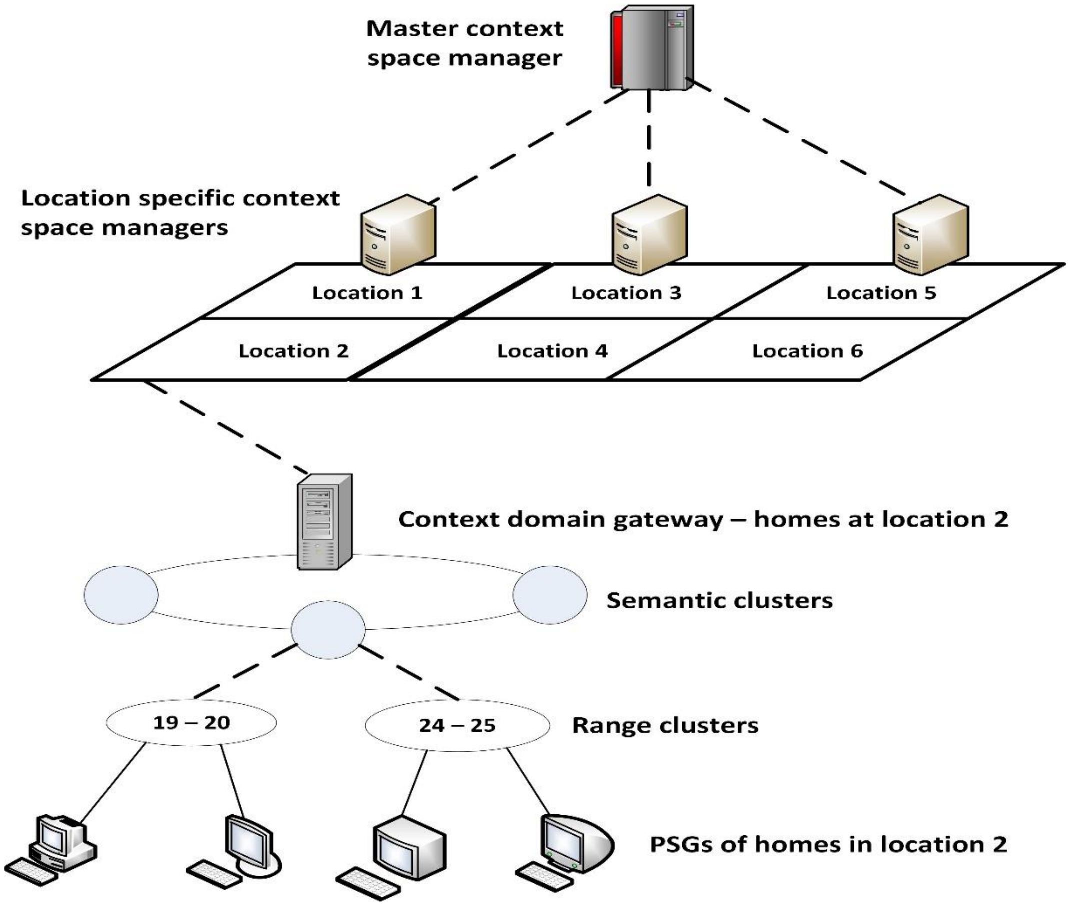

Master context space manager
Location specific context space managers
Location 1
Location 3
Location 5
Location 2
Location 4
Location 6
Context domain gateway – homes at location 2
Semantic clusters
19 – 20
24 – 25
Range clusters
PSGs of homes in location 2

Figure 46. Overview of the proposed system architecture

Each of these LCSMs is associated with a set of locations and is responsible for managing all the PSGs in those locations. Basically, a LCSM performs the same tasks as the context space manager but over a limited location based subset of the system. These LCSMs are managed by a master context space manager that handles the PSG registrations at the highest level. This master context space manager also contains a list of all the context domains together with the information of the LCSMs associated with them. For example, the list can contain an entry for the shop context domain and the contact information of the LCSMs having a shop domain associated with them. To facilitate the location-wise grouping of the PSGs, we can partition the deployment area into a grid comprising of individual location units. The organization of the location units can be carried out using a space filling curve technique like the Hilbert curve [103, 104]. The scale of the location units can be chosen in accordance with the deployment area and each LCSM is now associated with a location unit. A PSG registering with the middleware now submits its location information along with its context schema which is used to redirect the PSG to the corresponding LCSM where it is registered with the system in the usual manner.

. This scheme is useful when an extremely large scale deployment is considered as it avoids the management of all the PSGs at a single place. Further, since the set of range clusters corresponding to a semantic cluster is now built over PSGs in close physical proximity, the link maintenance of the P2P network will be more efficient. Consequently, the flooding operation within a single range cluster is also expected to be faster. However, this location based organization will affect mobile PSGs as they will need to update their P2P links as they move between location areas belonging to different LCSMs. This is an important issue that will need to be addressed as part of the design of this scheme.

7.3.2 Supporting multiple query scopes

As part of the proposed organizational scheme, the cluster domain/semantic cluster/range clusters based classification scheme will now be ported to every LCSM. While this approach will be useful in handling location based queries, it will be difficult to handle queries that are interested in a range of values irrespective of their location, such as ‘the number of homes that are vacant’. Also, as the answers to such queries are now spread across the location grid, the query response time can increase. In order to support these queries, we need to provide a location independent view of the different value ranges. This objective can be partly achieved by using a secondary range indexing consists of interval trees to organize the range cluster values from the different locations into a single place. Figure 47 illustrates the basic concept behind this idea. Each LCSM keeps track of its range cluster bounds and can update this information with the master context space manager that inserts this information into the global interval tree (a separate tree is present for a given context domain/semantic cluster combination). Each tree node is augmented to contain the contact information of the LCSM being represented by the interval. The indexing for string attributes can be similarly supported by maintaining a global ternary search tree composed from the corresponding ternary search trees present in each LCSM.

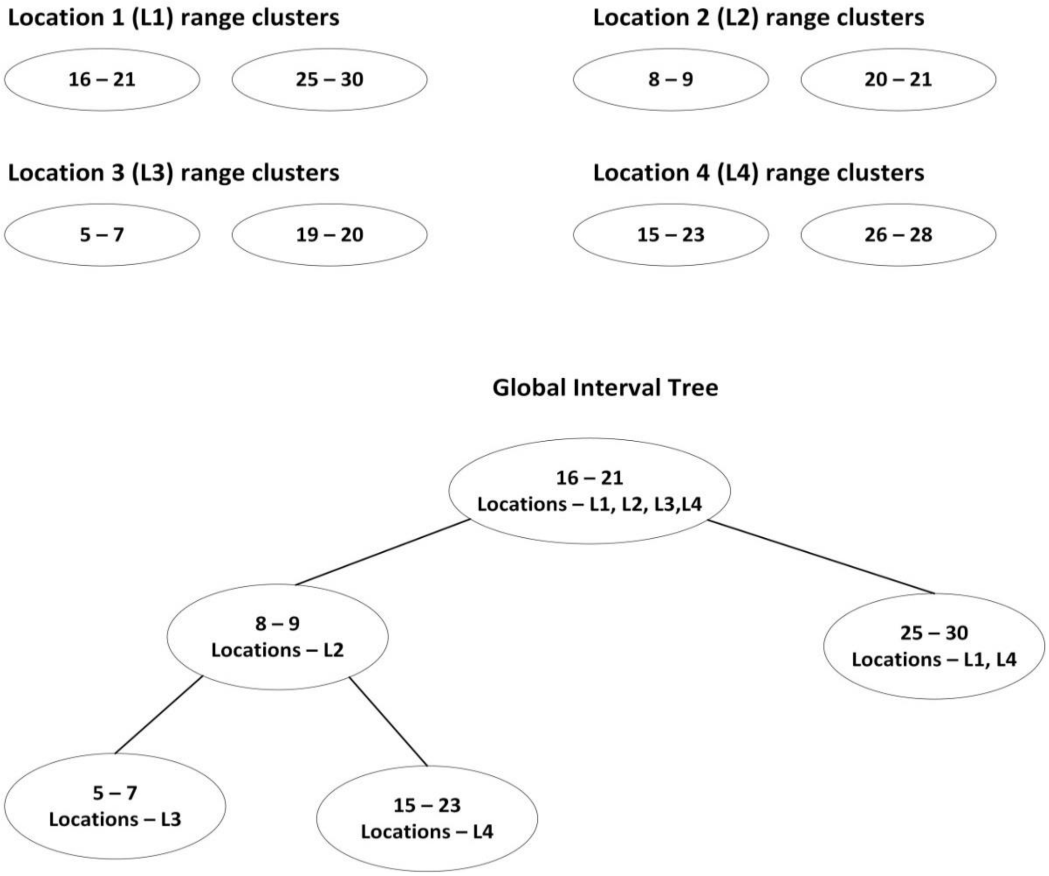


Figure 47. Using interval trees to support multiple query scopes

This global tree can now be utilized to process location-independent queries. Since this tree is composed of the cluster bounds distributed across multiple locations, it will need to be updated periodically. Since the range cluster bounds keep changing as PSGs join/leave the system, the notification of every update to the global tree may not be feasible due to the overhead. Some of the update operations can be deferred to minimize this overhead. The update corresponding to an existing range cluster split can be deferred as the resulting cluster bounds are a fine grained version of the original bounds. Although this results in a loss of the localization performance of the global tree, the notification of this update is not essential. On the other hand, whenever a cluster split occurs to define the initial clusters or an outlier cluster splits, the resulting cluster bounds must be compulsorily updated with the global tree. Since these two operations represent the formation of range clusters having cluster bounds that were not previously part of the index, they need to be updated with the global tree to preserve the correctness of the range cluster composition in a location. A potential issue that will need to be resolved is accounting for the outlier clusters. Since outlier clusters do not have any fixed range cluster bounds, it will be difficult to account for them in the global tree.

The query processing operation also needs to be modified with the inclusion of the location based organization scheme. As queries can now specify the location constraints along with the usual value based constraints, the query plan will need to be modified accordingly. If a query does not specify a location constraint, the global interval tree can be used to process the query. An interval in the global tree can be checked for its compatibility with the specified value constraints and the query can be forwarded to the corresponding LCSM for further processing. Location dependent queries can either be directly forwarded to the relevant LCSMs or preprocessed using the global tree to identify if there is a matching interval in the locations specified. It is intuitive that location independent queries will take a longer response time than location specific queries as the former case will entail answers to be aggregated from all the location areas and then reported to the query issuer. The current query processing model uses the query processor to aggregate all the matching answers. In order to speed up the response time for location independent queries, we need to use appropriate context data aggregation techniques that can distribute the aggregation process and avoid the performance bottleneck of a central aggregation point.

7.3.3 Directions for future work

We briefly discussed our vision and initial design plans for the enhanced version of the context data management system discussed in Section 7.3.2. We identified the use of location as an additional indexing level and proposed the use of interval trees to support multiple query scopes. One of the key issues that will need to be addressed as part of the introduction of the location based organization is defining the scale of the location grid and ensure that the scale is manageable from a processing standpoint. The selection of the scale of the location grid as well as the scale of the individual location units is an important issue as they will dictate the density of the PSGs in each location unit. The location grid can be either be pre-divided into a set of location units or can be incrementally partitioned depending on the number of PSGs in each location. The imposition of a single location grid structure to classify PSGs belonging to multiple context domains may not be a very efficient solution. For example, considering the deployment area as a city, the distribution of official and residential spaces may be quite different from each other and the use of a single location grid may lead to a particular domain being unevenly distributed. All these factors should be taken into account while designing the location grid based organization. An important concern that needs to be taken into account when a large scale deployment is considered is to ensure that the system is scalable globally as well as locally [105]. As applications may need to work with local as well as remote context information, it becomes essential to ensure that the response time incurred in retrieving data from remote PSGs is minimized. The context management system discussed in [105] uses a replication and caching based technique to store the data of interest near the relevant application. The direct adoption of this strategy may not be feasible in our scenario as there is no context data stored in our system. However, a variation of this technique can be utilized in this case wherein ranges of values along with the set of PSGs possessing them are replicated and stored in the different LCSMs. This would give an approximation of the data values of the PSGs and be used to process some of the queries locally instead of propagating them to the remote LCSMs. Further, as the location based partitioning may cause mobile PSGs to switch rapidly from one LCSM to another, we need to ensure that this handover is smooth and causes minimal disruption to the data retrieval process. This problem is similar to what the mobile IP protocol is designed to solve. However, the main difference here is that a mobile PSG is associated with other mobile PSGs in a P2P network corresponding to its participation in the semantic clusters. The movement of a mobile PSG between LCSMs will necessitate in the updating of these neighbor links which can cause a significant overhead when this updating is frequent. The handling of mobile PSGs is an important problem that will need to be addressed by the future work. As part of our initial work in this direction, we have developed a callback based mechanism to handle the disruptions occurring in the data transfer process due to mobility [106, 107]. The main idea involved in this technique is to represent mobile PSGs using a separate model known as the mobile space that distinguish them from other PSGs. Further, the concept of callbacks to provide an idea about the availability of a mobile PSG is introduced as a part of this technique. Applications can now use these callbacks to monitor the availability of the mobile PSGs and handle the disruptions in the data delivery process accordingly.

Another major part of the future work will be concerned with the improvement of the querying functionalities at the lower levels of data acquisition. In this thesis, we have proposed the design of an indexing scheme that can handle the management and lookup of single valued attributes. However, we need to extend these indexing schemes to be able to support the indexing of multi-dimensional attributes. A common example of a multi-dimensional attribute is the use of (x, y) coordinate pairs to represent the location of a moving object. Since these coordinate values cannot be indexed separately from each other, we need to provide a technique to organize these attributes. Also, the proposed string indexing technique relies on the ability of the PSGs to provide correct and orderly data. As part of the future work, this reliance needs to be reduced and the string matching process needs to be enhanced to work with imperfect data. Also, the proposed indexing structure needs to be augmented with regards to processing multi-attribute queries. The current index is designed to handle exact matching and range queries on a single attribute. A straightforward way to extend this system to handle multi-attribute queries would be to select a single attribute from the set of attributes in the query and forward the query to the PSGs that satisfy the constraint specified for that particular attribute. The complete query can now be processed individually at each of these PSGs. In order for this technique to work efficiently, prior knowledge of the number of PSGs that satisfy the query constraints for each attribute value is required. This information can now be used to select the attribute that has the minimal number of matching values and process the query against that attribute to minimize the processing time. Since it is difficult to maintain this information in a dynamic context-aware system, it is important to consider other sophisticated techniques to resolve this issue. This is an important research problem in its own right and is being independently addressed as part of our middleware design. As part of the initial research efforts in this direction, a distributed binary tree based query processing structure has been proposed [108].

The main idea behind this technique is to convert the query clause into a binary tree wherein the root and intermediate nodes refer to the AND/OR operations and the leaf nodes refer to the different predicates or attributes. This is followed by translating the query into a binary tree of simple queries based on the tree obtained in the first step. The query is now processed in this tree in a bottom up manner where the leaf nodes process the query against the individual attributes and the results are aggregated at the intermediate nodes until the final result is available at the root node. This query processing approach for complex queries also improves the data aggregation process by distributing the processing and aggregation tasks across multiple nodes instead of the single point centralized aggregation used in the current indexing approach. This approach can also be extended to support the processing of queries involving PSGs in multiple locations. This also forms an important part of the future work to improve the performance of the proposed context data management system.

7.4 Conclusion

The paradigm of context-aware computing has been the focus of concentrated research efforts for over two decades. The basic idea of context-aware computing to make computing processes available anywhere and everywhere might have seemed like a distant dream when it was initially proposed. However, the recent emergence of handheld devices with powerful processing capabilities as well as the proliferation of internet availability is all set to transform this vision into a reality. Consequently, the system requirements to realize an efficient context-aware system have also seen a large change over the passage of time. The initial small scale lab based deployment model for context-aware systems has given way to wide area deployments in real world settings that place additional constraints on the system design issues. Further, the identification of context-awareness as an important enabling technology for the IoT (Internet of Things) scenario is indicative of the fact that the paradigm of context-aware computing is going to be extremely important in the near future.

In order to provide an abstraction between the context aware applications and the underlying context sources, a context-aware middleware is required to mediate the application-context source interactions. The essential design requirements for such middleware systems include the following – acquire, process and query context information from multiple context sources, provide context reasoning and aggregation mechanisms, ensure the security and privacy of the context information and provide the necessary programming tools for application developers. Since context-aware applications primarily operate by reacting to the changes in context information, the issues related to the management and retrieval of context data become essential. Further, the association of context-awareness with technologies like the IoT makes the collection and querying of context information from multiple sources absolutely essential. The problem of acquiring, processing and querying context data is compounded due to the following facts – context data can be distributed across multiple context sources, context sources can be mobile, the data can have different representation formats, context data is usually dynamic in nature and changes frequently and the scope and requirement of a context query may vary according to application requirements. The Coalition context-aware middleware system which is being developed as part of our ongoing research project is intended to support large scale context-aware operations and forms the basis for the work carried out in this thesis.

As managing and querying context data is an important prerequisite for operations like context reasoning and aggregation, the presence of an efficient context data management system is essential in a context-aware middleware. The main focus of this thesis is to design and develop a context data management system capable of managing and supporting lookups efficiently over different types of context data. This data management system is developed as a component of the Coalition context-aware middleware system. The technical contributions of this thesis can be summarized as follows:

1. **Range cluster based indexing to facilitate context lookup** – One of the primary constraints in designing a lookup mechanism for context data in the Coalition system is the fact that no context data is stored in the middleware. In order to design an indexing technique that satisfies this constraint, we propose a range clustering technique that provides an approximate partitioning of the context sources according to their data values at a particular instant of time. The basic idea involved in the creation of this index structure is to replace the single p2p network used to connect the context sources together by a series of p2p networks where each network corresponds to a range of data values. This set of range clusters constitute the index structure which can then be used to facilitate the context lookup process by redirecting a query to the set of relevant clusters. The index structure is equipped with a set of maintenance

operations that perform the task of adapting the index according to the leave/join operations of the context sources. The experimental analysis carried out on the index structure indicates that the lookup time achieved using this scheme is significantly better than that achieved using a flooding based approach. The variations in the response time observed with the change in the network size are also found to be stable. However, the evaluation of the index with dynamic data values reveals the fact that it is not well equipped to handle this property of data and can lead to a situation where context sources are assigned to range clusters that are inconsistent with their current data values.

2. **Indexing dynamic context data using a mean-variance index** – The range cluster based indexing technique served as the basic blueprint for designing an index structure for a system not storing any context information. Since this index was not equipped to handle the dynamic nature of context data, we developed an augmentation of this index structure by using the mean and variance values to construct an index instead of the actual data values. This index structure is primarily designed for single value numeric context attributes. The main advantage of using the mean and variance values to build an index is that they are relatively stable as compared to the actual data value that keeps changing. This index structure is also able to represent the data distribution more accurately as well as handle the dynamicity of context data and minimize the number of update operations. The experimental analysis indicates that the index performance is satisfactory with respect to the query response time, query localization, accuracy as well as stability towards the index update operations. The effect of the dynamicity of the underlying data on the query response time is also observed to be minimal. Also, the index is observed to be able to adapt itself and manage PSGs with extremely uneven data change patterns. This indicates the usability of the index to manage and lookup dynamic context data distributed across multiple context sources.

3. **Incremental tree based index structure for string context data** –The mean-variance based index structure was observed to handle the problem of managing dynamic data satisfactorily. However, the use of the index was restricted to numeric attributes. In order to facilitate the lookup of string context attributes, we proposed the construction of an index structure based on the concepts of radix sort and ternary search trees. This index structure provided an approximate partitioning on a set of strings based on their shared prefixes. The length of the prefix to be matched was not fixed and was chosen based on the current composition of the strings in a semantic cluster. In order to organize the index structure, a modified version of the ternary search tree was proposed. Additionally, a longest prefix matching based keyword identification technique was also proposed to identify semantics based grouping among strings. The experimental analysis indicated that the index structure performs well with respect to the query response time for different types of string related queries and is able to provide an ordering among the strings effectively. It was also observed that the growth rate of the index structure i.e. the change in the size of the ternary search tree was significantly less than that observed in the case when all the strings were inserted directly into the tree. The performance of the index structure with dynamic string data was also examined with respect to two different types of data change patterns – keeping the starting character unchanged and the occurrence of random strings. The impact on the performance of the index in the first case was observed to be minimal. However, the impact was more severe in the second case. Although the presence of such a volatile set of dynamic strings is unlikely in a real world scenario, we proposed an outlier cluster based technique to handle this eventuality. The time required for the maintenance operations of the index were also observed to be minimal. This indicated that the proposed index structure is able to handle the problem of establishing an ordering among string context attributes effectively.

The technical contributions of this thesis partly fulfilled our goal of realizing a context data management system that is capable of managing context data and supporting lookups over them effectively. However, there are some research issues that need to be resolved in order to further improve the performance of the proposed data management system especially with respect to the support for different types of query scopes. Through this thesis, we believe that we have been able to make an important contribution to the field of context-aware research especially in the field of context data management and provided a future roadmap for the work to be carried out in order to augment the existing data management system that will help us realize our long term research objective.

Bibliography

1. Weiser, M., *The computer for the 21st century.* Scientific american, 1991. **265**(3): p. 94-104.
2. Schilit, B.N. and M.M. Theimer, *Disseminating active map information to mobile hosts.* Network, IEEE, 1994. **8**(5): p. 22-32.
3. Dey, A.K., *Understanding and Using Context.* Personal Ubiquitous Comput., 2001. **5**(1): p. 4-7.
4. Kjær, K.E., *A survey of context-aware middleware*, in *Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering*. 2007, ACTA Press: Innsbruck, Austria. p. 148-155.
5. Baldauf, M., S. Dustdar, and F. Rosenberg, *A survey on context aware systems.* Int. J. Ad Hoc Ubiquitous Comput., 2007. **2**(4): p. 263-277.
6. Saeed, A. and T. Waheed. *An extensive survey of context-aware middleware architectures*. in *IEEE International Conference on Electro/Information Technology (EIT), 2010*. IEEE.
7. Hong, J.-y., E.-h. Suh, and S.-J. Kim, *Context-aware systems: A literature review and classification.* Expert Systems with Applications, 2009. **36**(4): p. 8509-8522.
8. Bellavista, P., et al., *A survey of context data distribution for mobile ubiquitous systems.* ACM Comput. Surv., 2012. **44**(4): p. 1-45.
9. Bettini, C., et al., *A survey of context modelling and reasoning techniques.* Pervasive and Mobile Computing, 2010. **6**(2): p. 161-180.
10. Ashton, K. *That 'internet of things' thing in the real world, things matter more than ideas,".* June 2009 2012-07-30]; Available from: http://www.rfidjournal.com/article/print/4986
11. Chaves, L.W.F. and C. Decker. *A survey on organic smart labels for the Internet-of-Things*. in *Seventh International Conference on Networked Sensing Systems (INSS),*. 2010.

12. Chen, Y., J. Guo, and X. Hu. *The Research of Internet of Things' Supporting Technologies Which Face the Logistics Industry*. in *International Conference on Computational Intelligence and Security (CIS),*. 2010.

13. Yan-Wei, W., Y. Hui-Li, and L. Ya. *Internet of things technology applied in medical information*. in *International Conference on Consumer Electronics, Communications and Networks (CECNet),*. 2011.

14. Atzori, L., A. Iera, and G. Morabito, *The Internet of Things: A survey.* Computer Networks, 2010. **54**(15): p. 2787-2805.

15. Gao, C., Z. Ling, and Y. Yuan. *The research and implement of smart home system based on Internet of Things*. in *International Conference on Electronics, Communications and Control (ICECC),*. 2011.

16. Bandyopadhyay, S., et al., *Role of middleware for Internet of things: a study.* International Journal of Computer Science & Engineering Survey (IJCSES) Vol, 2011. **2**.

17. Perera, C., et al., *Context Aware Computing for The Internet of Things: A Survey.* 2013.

18. Zhu, J., et al., *Coalition: A Platform for Context-Aware Mobile Application Development.*

19. *Context-Aware Middleware Services and Programming Support for Sentient Computing*. 03/06/2010; Available from: http://lucan.ddns.comp.nus.edu.sg:8080/PublicNSS/researchContextAware.aspx.

20. Pung, H.K., et al., *Context-aware middleware for pervasive elderly homecare.* IEEE Journal on Selected Areas in Communications, 2009. **27**(4): p. 510-524.

21. Debaty, P., P. Goddi, and A. Vorbau, *Integrating the physical world with the web to enable context-enhanced mobile services.* Mob. Netw. Appl., 2005. **10**(4): p. 385-394.

22. Juszczyk, L., et al., *Adaptive Query Routing on Distributed Context - The COSINE Framework*, in *Proceedings of the 2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware*. 2009, IEEE Computer Society. p. 588-593.

23. Conan, D., R. Rouvoy, and L. Seinturier, *Scalable processing of context information with COSMOS*, in *Proceedings of the 7th IFIP WG 6.1 international conference on*

Distributed applications and interoperable systems. 2007, Springer-Verlag: Paphos, Cyprus. p. 210-224.

24. Riva, O. and C.d. Flora, *Contory: A Smart Phone Middleware Supporting Multiple Context Provisioning Strategies*, in *Proceedings of the 26th IEEE International ConferenceWorkshops on Distributed Computing Systems*. 2006, IEEE Computer Society. p. 68.

25. Hyun, K., C. Young-Jo, and O. Sang-Rok. *CAMUS: a middleware supporting context-aware services for network-based robots*. in *IEEE Workshop on Advanced Robotics and its Social Impacts, 2005.*

26. Pham, K.A.N., Y.K. Lee, and S.Y. Lee, *Middleware architecture for context knowledge discovery in ubiquitous computing*, in *Proceedings of the 2005 international conference on Embedded and Ubiquitous Computing*, 2005, Springer-Verlag: Nagasaki, Japan. p. 1177-1188.

27. Kapsammer, E., Schwinger, W., Retschitzegger, W. *Bridging Relational Databases to Context-Aware Services*. in *CAiSE Workshop Ubiquitous Mobile Information and Collaboration Systems (UMICS2005)*. 2005. Porto, Portugal.

28. Reichle, R., et al. *A Context Query Language for Pervasive Computing Environments*. in *Sixth Annual IEEE International Conference on Pervasive Computing and Communications, (PerCom 2008),*. 2008.

29. Eugster, P., et al., *Design and Implementation of the Pervaho Middleware for Mobile Context-Aware Applications*, in *Proceedings of the 2008 International MCETECH Conference on e-Technologies*. 2008, IEEE Computer Society. p. 125-135.

30. Corradi, A., M. Fanelli, and L. Foschini. *Implementing a scalable context-aware middleware*. in *IEEE Symposium on Computers and Communications, ISCC 2009.* 2009.

31. Julien, C. and G.C. Roman, *EgoSpaces: facilitating rapid development of context-aware mobile applications.* IEEE Transactions on Software Engineering, 2006. **32**(5): p. 281-298.

32. Roussos, Y., Y. Stavrakas, and V. Pavlaki. *Towards a context-aware relational model*. in *the proceedings of the International Workshop on Context Representation and Reasoning (CRR'05)*. 2005.

33. Martinenghi, D. and R. Torlone, *Querying context-aware databases*, in *Flexible Query Answering Systems*. 2009, Springer. p. 76-87.

34. Yau, S.S., et al., *Reconfigurable context-sensitive middleware for pervasive computing.* Pervasive Computing, IEEE, 2002. **1**(3): p. 33-40.

35. Patrick, F. and C. Siobhan, *CASS – a middleware for mobile context-aware applications*. 2004.

36. Knappmeyer, M., et al., *A context provisioning framework to support pervasive and ubiquitous applications*, in *Proceedings of the 4th European conference on Smart sensing and context*. 2009, Springer-Verlag: Guildford, UK. p. 93-106.

37. Grossmann, M., et al. *Efficiently Managing Context Information for Large-Scale Scenarios*. in *Third IEEE International Conference on Pervasive Computing and Communications (PerCom 2005),*. 2005.

38. Fei, L., S. Sehic, and S. Dustdar. *COPAL: An adaptive approach to context provisioning*. in *IEEE 6th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob),*. 2010.

39. Kiani, S.L., et al. *Large-scale context-aware system in smart spaces: Issues and challenges*. in *Baltic Congress on Future Internet Communications (BCFIC Riga),*. 2011.

40. Chen, G., M. Li, and D. Kotz, *Data-centric middleware for context-aware pervasive computing.* Pervasive Mob. Comput., 2008. **4**(2): p. 216-253.

41. Gu, T., H.K. Pung, and D.Q. Zhang, *A service-oriented middleware for building context-aware services.* J. Netw. Comput. Appl., 2005. **28**(1): p. 1-18.

42. Duran-Limon, H.A., et al. *A resource and QoS management framework for a real-time event system in mobile ad hoc environments*. in *Proceedings of Ninth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, 2003.*

43. Macedo, D., et al., *A distributed information repository for autonomic context-aware MANETs.* IEEE Transactions on Network and Service Management, 2009. **6**(1): p. 45-55.

44. Mashhadi, A.J., S. Ben Mokhtar, and L. Capra. *Habit: Leveraging human mobility and social network for efficient content dissemination in Delay Tolerant Networks*. in

IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks & Workshops (WoWMoM 2009).. 2009.

45. Ranganathan, A., et al., *MiddleWhere: a middleware for location awareness in ubiquitous computing applications*, in *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*. 2004, Springer-Verlag New York, Inc.: Toronto, Canada. p. 397-416.

46. Borcea, C., et al., *The MobiSoC middleware for mobile social computing: challenges, design, and early experiences*, in *Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*. 2007, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering): Innsbruck, Austria. p. 1-8.

47. Shiva, C., et al. *Mobile Gaia: a middleware for ad-hoc pervasive computing*. in *Consumer Communications and Networking Conference, CCNC. 2005*.

48. Yu, K., et al. *A Location-Based Peer-to-Peer Network for Context-Aware Services in The 2005 Symposium on a Ubiquitous Environment*. in *Applications and the Internet Workshops, 2005. Saint Workshops 2005.*. 2005.

49. Kosaka, Y., et al. *A P2P Network Construction Method Based on Cycle of Sensing in Ubiquitous Sensor Environment*. in *2010 IEEE 24th International Conference on.Advanced Information Networking and Applications Workshops (WAINA)*, 2010.

50. Boldrini, C., et al. *HiBOp: a History Based Routing Protocol for Opportunistic Networks*. in *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2007).*. 2007

51. Van Kranenburg, H., et al., *A context management framework for supporting context-aware distributed applications.* Communications Magazine, IEEE, 2006. **44**(8): p. 67-74.

52. Yannis, R., S. Yannis, and P. Vassia, *Towards a Context-Aware Relational Model*. 2005.

53. Ling, F. *Supporting context-aware database querying in an Ambient Intelligent environment*. in *3rd IEEE International Conference on.Ubi-media Computing (U-Media),* 2010.

54. Martinenghi, D. and R. Torlone, *Querying Context-Aware Databases*, in *Proceedings of the 8th International Conference on Flexible Query Answering Systems*. 2009, Springer-Verlag: Roskilde, Denmark. p. 76-87.

55. Reichle, R., et al., *A Context Query Language for Pervasive Computing Environments*, in *Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications*. 2008, IEEE Computer Society. p. 434-440.

56. Zhu, J., *Provision, discovery and development of ubiquitous services and applications*. 2011, National University of Singapore: Singapore.

57. Wenwei, X., et al. *Gateways of physical spaces in context-aware computing*. in *International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP 2008).*. 2008.

58. Xue, W., et al., *Context gateway for physical spaces.* Journal of Ambient Intelligence and Humanized Computing, 2012. **3**(3): p. 193-204.

59. Xue, W., et al., *Data Management for Context-Aware Computing*, in *Proceedings of the 2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing - Volume 01*. 2008, IEEE Computer Society. p. 492-498.

60. Xue, W., et al., *Schema matching for context-aware computing*, in *Proceedings of the 10th international conference on Ubiquitous computing*. 2008, ACM: Seoul, Korea. p. 292-301.

61. Xue, W., H.K. Pung, and S. Sen, *Managing context data for diverse operating spaces.* Pervasive and Mobile Computing, 2011.

62. Shubhabrata Sen, W.X., Hung Keng Pung, Wai Choong Wong. *Semantic P2P Overlay for Dynamic Context Lookup*. in *The Fourth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2010),*. 2010. Florence, Italy: IARIA.

63. Hartigan, J.A., *Clustering Algorithms*. 1975, New York: Wiley.

64. *Intel Lab Data*. 02/06/2004; Available from: http://db.csail.mit.edu/labdata/labdata.html.

65. Tomasic, A., et al., *Incremental updates of inverted lists for text document retrieval.* SIGMOD Rec., 1994. **23**(2): p. 289-300.

66. Lim, L., et al., *Efficient Update of Indexes for Dynamically Changing Web Documents.* World Wide Web, 2007. **10**(1): p. 37-69.

67. Jang, H., Y. Kim, and D. Shin, *An effective mechanism for index update in structured documents*, in *Proceedings of the eighth international conference on Information and knowledge management*. 1999, ACM: Kansas City, Missouri, United States. p. 383-390.

68. Guttman, A., *R-trees: a dynamic index structure for spatial searching.* SIGMOD Rec., 1984. **14**(2): p. 47-57.

69. Silva, Y.N., X. Xiong, and W.G. Aref, *The RUM-tree: supporting frequent updates in R-trees using memos.* The VLDB Journal, 2009. **18**(3): p. 719-738.

70. Ding, R., X. Meng, and Y. Bai, *Efficient Index Update for Moving Objects with Future Trajectories*, in *Proceedings of the Eighth International Conference on Database Systems for Advanced Applications*, 2003, IEEE Computer Society. p. 183.

71. Yuni, X. and S. Prabhakar. *Efficient CNG indexing in location-aware services*. in *Proceedings. 23rd International Conference on.Distributed Computing Systems Workshops, 2003.*

72. Cheng, R., et al. *Change tolerant indexing for constantly evolving data*. in *Proceedings of 21st International Conference on.Data Engineering (ICDE 2005),*. 2005.

73. Park, S.-Y., et al., *CLUR-Tree for supporting frequent updates of data stream over sensor networks*, in *Proceedings of the 7th international conference on Distributed Computing*. 2005, Springer-Verlag: Kharagpur, India. p. 87-92.

74. Biveinis, L., et al., *Main-memory operation buffering for efficient R-tree update,* in *Proceedings of the 33rd international conference on Very large data bases*2007, VLDB Endowment: Vienna, Austria. p. 591-602.

75. S.Saltenis, et al., *Indexing the positions of continuously moving objects.* SIGMOD Rec., 2000. **29**(2): p. 331-342.

76. Liao, W., et al., *Vtpr-tree: An efficient indexing method for moving objects with frequent updates*, in *Advances in Conceptual Modeling-Theory and Practice*. 2006, Springer. p. 120-129.

77. Tao, Y., D. Papadias, and J. Sun. *The TPR*-tree: an optimized spatio-temporal access method for predictive queries*. in *Proceedings of the 29th international conference on Very large data bases-Volume 29*. 2003. VLDB Endowment.

78. Dyo, V. and C. Mascolo. *Adaptive distributed indexing for spatial queries in sensor networks*. in *Database and Expert Systems Applications, 2005. Proceedings. Sixteenth International Workshop on*. 2005.

79. Diao, Y., et al. *Rethinking Data Management for Storage-centric Sensor Networks*. in *CIDR*. 2007.

80. Greenstein, B., et al., *DIFS: A distributed index for features in sensor networks.* Ad Hoc Networks, 2003. **1**(2): p. 333-349.

81. Li, X., et al. *Multi-dimensional range queries in sensor networks*. in *Proceedings of the 1st international conference on Embedded networked sensor systems*. 2003. ACM.

82. Cheng, R., S. Singh, and S. Prabhakar, *U-DBMS: a database system for managing constantly-evolving data*, in *Proceedings of the 31st international conference on Very large data bases*. 2005, VLDB Endowment: Trondheim, Norway. p. 1271-1274.

83. Golab, L., P. Prahladka, and M.T. Ozsu. *Indexing Time-Evolving Data With Variable Lifetimes*. in *18th International Conference on Scientific and Statistical Database Management,. 2006.*

84. Xia, Y., et al., *Indexing continuously changing data with mean-variance tree.* International journal of high performance computing and networking, 2008. **5**(4): p. 263-272.

85. Stoica, I., et al. *Chord: A scalable peer-to-peer lookup service for internet applications*. in *ACM SIGCOMM Computer Communication Review*. 2001. ACM.

86. Ratnasamy, S., et al., *A scalable content-addressable network*. Vol. 31. 2001: ACM.

87. Rowstron, A. and P. Druschel. *Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems*. in *Middleware 2001*. 2001. Springer.

88. Zhao, B.Y., J. Kubiatowicz, and A.D. Joseph, *Tapestry: An infrastructure for fault-tolerant wide-area location and routing.* 2001.

89. Gao, J. and P. Steenkiste, *An Adaptive Protocol for Efficient Support of Range Queries in DHT-Based Systems*, in *Proceedings of the 12th IEEE International Conference on Network Protocols*. 2004, IEEE Computer Society. p. 239-250.

90. Li, D., et al., *Efficient Range Query Processing in Peer-to-Peer Systems.* IEEE Trans. on Knowl. and Data Eng., 2009. **21**(1): p. 78-91.

91. Wang, D. and M. Li. *A Range Query Model Based on DHT in P2P System*. in *International Conference on.Networks Security, Wireless Communications and Trusted Computing (NSWCTC '09),*. 2009.

92. Carfi, D., et al. *DDT: A distributed data structure for the support of P2P range query*. in *5th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2009),*. 2009.

93. Andrzejak, A. and X. Zhichen. *Scalable, efficient range queries for grid information services*. in *Proceedings of the Second International Conference on Peer-to-Peer Computing, (P2P 2002).*. 2002.

94. Sen, S. and H.K. Pung, *A Mean-Variance Based Index for Dynamic Context Data Lookup, Mobile and Ubiquitous Systems: Computing, Networking, and Services,* A. Puiatti and T. Gu, Editors. 2012, Springer Berlin Heidelberg. p. 101-112.

95. Navarro, G. and R. Baeza-Yates. *A new indexing method for approximate string matching*. in *Combinatorial Pattern Matching*. 1999. Springer.

96. Bunke, H., *String matching for structural pattern recognition.* Syntactic and Structural Pattern Recognition. Theory and Applications, H. Bunke and A. Sanfelui (Eds.), Series in Computer Science, World Scientific, Singapore, 1990.

97. Behm, A., et al. *Space-constrained gram-based indexing for efficient approximate string search*. in *IEEE 25th International Conference on.Data Engineering, (ICDE'09),*. 2009.

98. Kazimianec, M. and A. Mazeika. *Clustering of Short Strings in Large Databases*. in *20th International Workshop on Database and Expert Systems Application (DEXA 2009),*. 2009.

99. Zhang, Z., et al. *Bed-tree: an all-purpose index structure for string similarity search based on edit distance*. in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. 2010. ACM.

100. Kahveci, T. and A.K. Singh. *An efficient index structure for string databases*. in *VLDB*. 2001.

101. Ferragina, P. and R. Grossi, *The string B-tree: a new data structure for string search in external memory and its applications.* Journal of the ACM (JACM), 1999. **46**(2): p. 236-280.

102. *The DBLP Computer Science Bibliography*. 2013-12-16; Available from: http://www.informatik.uni-trier.de/~ley/db/.

103. Sagan, H., *Space-filling curves*. Vol. 18. 1994: Springer-Verlag New York.

104. Moon, B., et al., *Analysis of the clustering properties of the Hilbert space-filling curve.* IEEE Transactions on Knowledge and Data Engineering, , 2001. **13**(1): p. 124-141.

105. Buchholz, T. and C. Linnhoff-Popien, *Towards realizing global scalability in context-aware systems*, in *Proceedings of the First international conference on Location- and Context-Awareness*. 2005, Springer-Verlag: Oberpfaffenhofen, Germany. p. 26-39.

106. Chen, P., et al., *Context data management for mobile spaces*, in *Mobile and Ubiquitous Systems: Computing, Networking, and Services*. 2012, Springer. p. 340-341.

107. Chen, P., et al., *A context management framework for context-aware applications in mobile spaces.* International Journal of Pervasive Computing and Communications, 2012. **8**(2): p. 185-210.

108. Chen, P., et al. *Context Processing: A Distributed Approach*. in *The Second International Conference on Intelligent Systems and Applications (INTELLI 2013),*. 2013.