Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 Keller Hall
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 17-013

Design of a Location-based Publish/Subscribe Service using a Graph-based Computing Model

Anand Tripathi, Henry Hoang

November 21, 2017

Design of a Location-based Publish/Subscribe Service using a Graph-based Computing Model

Anand Tripathi and Henry Hoang
Department of Computer Science & Engineering
University of Minnesota, Minneapolis, Minnesota, 55455 USA
Email: tripathi@umn.edu and hoang159@umn.edu

***Abstract*—We present here the initial results of our investigation of a system architecture for location-based publish/subscribe services utilizing a graph-based model for managing data and computations. This architecture is implemented on a cluster computer using the facilities and the computation model provided by the Beehive framework which supports a transactional model of parallel computing on dynamic graph data structures. We implemented a *Museum Visitor Service* as an example of a location-based publish/subscribe system to study and evaluate the performance this approach. This service includes features utilizing location-based publish/subscribe functions for supporting coordination and collaboration among members in a social group visiting the museum. We implemented a testbed system for this service and evaluated its performance on a cluster computer. Our work also illustrates that weaker consistency models for transactions can be utilized in such services to achieve higher performance and scalability.**

I. Introduction

In the last decade there has been a surge of interest in *location-based services* (LBS) [28], [33], [1]. Location-based services utilize location data of a person or object to augment and enhance the application functions provided by the service. A location-based service utilizes user location data and other information such as the activity context and user profile to provide appropriate information and functions. Exemplifying the notion of *geofencing* [26], [39], a broad class of applications and services have been envisioned, investigated, and developed for utilizing location data of users and mobile objects. Tour guide [11], [2] applications give information about the places to the visitors based on their current location. Several mobile social networking applications such as *find-a-friend* [34] have been developed in the recent years. Systems such as *PlaceMail* [23] and *GeoNotes* [25] allow users to create messages and memos to be delivered to specific people only when they are at certain locations. Such applications are now available on Apple iPhone and Android phones for place-based reminder systems. Location-based services can be combined with micro-blogging models such as Twitter for public dissemination of information to people in certain areas. For example, in a city or museum tour, visitors may post to share comments about various points of interest at a location. Other areas for LBS include workflow management in organizations that have mobile workers. For example, applications for coordinating and managing vehicles in fleet systems. Location-based services are also used for assisting people in planning their use of public transit systems [6]. M-commerce is another emerging area of LBS applications for businesses to communicate advertisements targeted to people in specific geographic areas. Emergency alert notifications related to traffic, weather, and public safety have emerged as important application domains for LBS to support targeted alerts to people in specific areas. Location-based applications have also been developed for geographic games such as treasure-hunts and geo-caching.

Location-based applications and services typically interact with users whose identity may not be known a priori and it is determined based on their locations and other attributes. Publish/subscribe systems are ideally suited for such a need of decoupling the providers of data from their clients. A publish/subscribe system acts as an intermediary or broker between the publisher of data and its subscribers. This model of communication is asynchronous and it decouples the publishers and subscribers in space and time. The publisher may not know the identity of the subscribers who receive the notification, and similarly the identity of the publisher can be hidden from the subscribers.

The primary focus of our work is on the investigation of a system architecture for location-based publish/subscribe services [13] using graph data structures as the underlying data management model. In our approach, a location-based publish/subscribe service executes on a cluster of computers, which can be hosted on a cloud platform. In this investigation we use the Beehive [35] framework to show that its transactional model of parallel computing on graph data structures can be easily utilized in building location-based services hosted on a cluster computer. We present the design of a museum visitor service to illustrate our approach of using graph data structures at the core of its architecture, implemented using the graph data management facilities provided by the Beehive framework. We evaluate the performance and scalability of this service on a cluster computer.

In the museum visitor service presented here, a visitor can register his/her interest for information pertaining to various topics, themes, art media, and artistes to get notifications for matching publications at the current location. It also provides functions to facilitate a group of visitors to collaborate in sharing their comments with others. For example, members in a group can track or communicate with others, based on their locations and interests. A person can post a note for others at some location or one can request a notification when a group member is present at a location. It also allows the recipients to add their comments to a location-based note. This service can also be used for posting location-based announcements for events happening at different locations in the museum.

In the next section we discuss the characteristics and requirements of location-based publish/subscribe systems and the related work in this area. Section III presents a framework for designing publish/subscribe system architectures. In Section IV we outline the features and functionalities of our museum visitor service. In Section V we present the graph-based model for managing data and computation for our museum visitor service. Section VI presents the design of the museum visitor service using Beehive as the implementation platform, focusing on how the graph data model is implemented and how the parallel computing model of Beehive is utilized in supporting the service functions. In Section VII we present the performance of this service on a cluster computer. The last section presents the conclusion of this work.

II. Characteristics of Location-based Building Publish/Subscribe Systems

Publish/subscribe models have been extensively studied and investigated in the past. Earlier systems supported topic-based (also called channel-based) or type-based publications and subscriptions [13]. The most general and expressive model is based on content-based subscription where a subscriber can specify certain predicates on event contents for selecting the published events to be notified. Events are structured as a collection of attribute-value pairs. Several systems, such as JEDI [9], Siena [8], Elvin [30], Gryphon [3], PADRES [18], and Meghdoot [15] have supported the content-based publish/subscribe model.

Most of the earlier publish/subscribe systems, including those noted above, are based on distributed broker architectures. In these architectures, brokers are connected by an overlay network over the Internet, and each of them performs all publish/subscribe functions in addition to performing routing and maintaining the overlay topology. A subscription registered at a broker is forwarded to all others. Routing paths are maintained for forwarding a publication to the registered subscribers. The Siena [8] system introduced the concept of *subscription subsumption* to reduce the network traffic. With the goal of elasticity and scalability, several cloud/cluster based architectures have been developed for content-based publish/subscribe services [4], [22], [38] as an alternative to broker-based wide-area network overlay architectures. The cluster-based approach eliminates the need of complex routing and overlay management protocols. Our work is also directed towards developing a cluster-based service architecture.

Several general characteristics of location-based publish/subscribe systems have been identified in the past [17], [27]. These characteristics need to be taken into consideration when designing such systems. The general characteristics are related to user interaction and information communication models, location models, user profile, privacy issues, and the need for large-scale data management and continuous query processing. Content-based models have certain limitations as information brokers do not maintain any state about the subscribers. In contrast, a location-based publish/subscribe system has to be stateful and it needs to be context-aware. In such systems, notifications to subscribers are required to be based on their current location, activity context, personal preferences, and group memberships. For example, traffic alerts in some area may be intended only for those who are driving in a particular direction on a certain segment of a road. A stateful model is also needed to keep some past history of notifications. For example, consider an M-commerce application which notifies a retail store's location-based advertisements to people in the vicinity of the store. Here one may want to make sure that the same ads are not communicated repeatedly to a person who happens to be in the store's vicinity again. Some applications may require the past history of notifications to affect the subscription policies. For example, in case of an LBS for tourist information, one may wish to deliver a different set of information items to a person visiting for the second time. The utility of stateful publish subscribe systems and the need of an expressive language for subscription policies are discussed in [16]. It is observed in [10] that the content-based model does not provide suitable mechanisms for the requirements in location-based applications.

An important characteristic of location-based publish/subscribe systems is the dynamically changing nature of a user's subscriptions based on his/her mobility [19], [7]. To eliminate the need of unregistering a subscription and then adding a new one, the concept of parameterized subscription is introduced in [19] where a subscription's filter condition is dependent on some application-defined dynamically changing variable. Additionally, support is required for continuous query executions to match updated subscriptions with publications when location updates are received from mobile users. Since updates can occur at high rates, support for parallel execution of such continuous query operations is required. The problem of dynamically changing subscriptions in a topic-based publish/subscribe system using a broker architecture is addressed in [40].

Recently, graph-based models have been investigated for supporting dynamically changing subscriptions in location-based publish/subscribe systems [7]. GraPS [7] has developed a graph-based model for publish/subscribe systems in the context of a multiplayer gaming application where players move in a virtual space. It supports subscriptions associated with a dynamically changing set of nodes based a player's current location. GraPS is implemented using the PADRES [18] content-based publish/subscribe system together with an external graph processing library/framework. Conceptually, our work is driven and guided by similar motivation in using graph based approach for representing dynamic relationships between user's subscriptions and locations. However, we use a single framework for both graph data management and continuous query execution for matching functions.

Several variants of R-tree based models have been investigated [21], [37] for managing geo-spatial data for associating publications, subscriptions, and users with geo-spaces in location-based publish/subscribe systems. In our current work we have used a simple model where a location corresponds to a symbolically denoted space. In our future work we plan to investigate support for more general representations of geo-spaces.

Different kinds of transactional consistency models in data management may be utilized in such systems for higher performance and scalability. Weaker consistency models can be used when a publication is not critical and an occasionally missed notification can be tolerated. For example, missing an advertisement when a user enters some area can be tolerated when the user-location updates are performed using a weaker consistency model. On the other hand, certain service functions may require strong consistency based data updates. For example, updates to a user's privacy related preferences should immediately become visible across the system. Furthermore, some applications may require causality to be preserved in event notifications. For example, in the case when a user enters some area and then immediately leaves, any notifications based on these events should also follow the same temporal order.

III. A Framework for Location-Based Publish/Subscribe System Architectures

The general characteristics and requirements noted in Section II need to be taken into consideration when designing a location-based publish/subscribe system. Several other aspects also need to be taken into consideration in designing such systems. These are related to location models, user interaction models, large-scale data management, support for continuous query processing, and user privacy issues. We discuss here a framework for designing and building such systems.

Location Models: Location-based services typically require different kinds of models for representing location information [14], [20], [31]. Location could be represented using some coordinate system (e.g., GPS), or symbolic names representing a locality, building, room, or civic naming schemes [29] (e.g., name of a country, state, city, locality, locality, or zipcode). Locations may also identified by their semantic attributes, e.g. a library or a shopping center.

User Interaction Models: In the *pull-based* model of interaction, the user sends query requests to a service, including its location data. For example, one may wish to find the nearest gas station or restaurant. In some situations, one may be interested in knowing if a particular person or mobile object has arrived at a certain location, or if the next bus at a bus-stop is going to arrive within the next 10 minutes. The query model is suitable for requests of the former kind, however, for the latter kinds of requests, which require checking certain dynamically changing conditions, the user may have to make repeated queries. For such cases the *push-based* model is viewed as more suitable. When certain events of interest occur, the user is notified. This requires mechanisms to monitor the environment for the specified events and to notify the user when they occur. For push notifications a wide variety of technologies such as SMS, email, Apple Push Notification Service (APNS), and Google Cloud Messaging (GCM) service are utilized. A system architecture may also adopt a hybrid model where the system triggers execution of continuous queries and matching functions on the occurrence of certain events. The notifications resulting for a user are delivered when the user connects to the system.

Large-scale Data Management: A location-based publish/subscribe system may need to manage large time-varying data sets in environments supporting a large set of users. It requires processing of location updates, triggering executions of appropriate continuous query functions, management of user context and profile, and management of publications, subscriptions, and other persistent data. Scalable models for data storage and access are required to be supported in such systems.

Continuous Query Processing: The push-based and hybrid interaction models require continuous execution of queries and matching functions. For this, the system architecture needs to provide mechanisms for the application developers to define continuous queries and trigger conditions for their execution. For scalability and performance, the architecture must support a parallel execution model for such functions.

User Privacy: Privacy concerns with a person's ability to control access of others to his/her personal data. A number of researchers have raised and addressed privacy related issues in location-based services [32], [5], [24], [12]. Protection of location data is an important aspect of the information privacy requirement. The architecture should provide support for a person to specify his/her privacy preferences, including specification of who can access the user's location data. We view a location-based publish/subscribe service acting as a trusted broker between the publishers and subscribers, fully conforming to their privacy preferences.

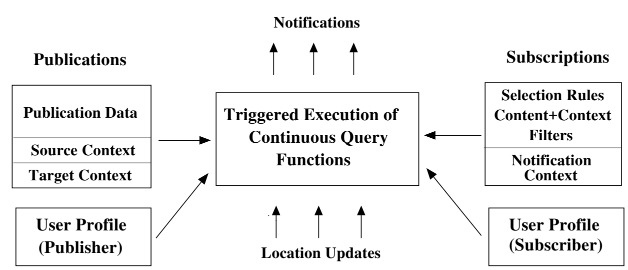

Fig. 1. Location-Based Publish/Subscribe Model

Figure 1 shows a model for executing continuous queries for matching publication and subscriptions. The publisher would specify a *target context* with a publication to identify the set of eligible subscribers who should be notified of the publication based on their current context. The target context may include location constraints to identify the subscribers in a particular location or area. The target context can specify certain mobility-based or activity-based attributes of a person, such as moving in a particular direction. Temporal constraints may also be specified. For example, notifications may be delivered only during certain time interval. Each publication may also have a *source context* containing information about the publisher, such as the publisher's id and location. The inclusion of this data would be subject to the publisher's privacy rules.

A subscription includes specification of certain topic-based or content-based filters for selecting publications that belong to certain categories or matching content. Content filters may also be specified on a publication's source context for making selections based on the publisher's id, location, group membership, or the time of publication. For example, a subscriber may specify that he/she is interested in advertisements only for particular products from a specific vendor. A subscription may also include a *notification context* whose primary function would be to indicate the situations when the subscription should be viewed as active. For example, one may specify temporal constraints on when the notifications could be given to the user. Similarly, location-based or activity-based constraints may be specified in the notification context. For example, a person may indicate that entertainment and shopping related notifications should not be delivered when he/she is at work.

Location updates and new publication events trigger execution of continuous query functions for matching publications with eligible subscriptions. In the event of a user's location update, such matching functions take into account the publications targeted to the user's new location along with the notification context and filtering rules specified in the subscriptions. The matching functions also need to take into account the publisher's and subscriber's profiles and preferences.

IV. A Museum Visitor Service based on the Publish/Subscribe Model

We developed a Museum Visitor Service as a case study to investigate the architecture of a publish/subscribe system using a graph-based model for data management and computing. In this study, we used the dataset[1] about the artifact collection in the Minneapolis Institute of Art to design and build an experimental testbed to provide a location-based publish/subscribe service for museum visitors. This dataset lists the artifacts present in different rooms of the museum. Each artifact has information about its title, the names of its creator artists, year of creation, and a brief description. This dataset also contains information about each of the artists. This dataset lists 129 rooms with a total of 3876 artifacts, created by 1368 artists.

The intended users of the museum visitor service presented here are the visitors and the staff. The goal of this service is to support location-based notifications to a person when visiting various rooms, based on the person's subscriptions. One can view the information about the artifacts present in a room as publications which are matched with a visitor's subscriptions when he/she enters that room. Some examples of location-based subscriptions by a visitor include interest in information about the artifacts created by specific artists, names of all the artists whose works are displayed in the current room, name of the artist with the most number of artifacts displayed in the room, and information about all artifacts created during a certain period. Other additional types of subscriptions can be easily defined in this system.

The testbed service provides functions to support collaboration among the members of a social/study group visiting the museum. A group member can publish notes in a museum room for other members in his/her group to receive as notifications when they visit the room. Such publications can be targeted to an individual, all members in a group, or the public. This service also supports location-context based subscriptions, using which a visitor can receive notifications when any member of a given group is present in a specified room, a group member enters or leaves that room, or the number of group members present in a room reaches some specified threshold. Such subscriptions may also be used to get notifications if an official tour-guide is present in some room. In our design, a simple model is supported for the users to specify the privacy preferences for their location information. A user can make this information visible to the public, all members in his/her group, or some specific individuals.

We distinguish between two types of publications: *permanent* and *ephemeral*. Permanent publications have no expiration time. For example, the data about artifacts and artists represent permanent publications in our testbed system. Ephemeral publications generally represent those which have some expiration time, and typically such publications are posted by the visitors or the staff. For example, the museum staff may post ephemeral publications regarding special events, lectures, or shows.

In our testbed system for the museum service, a visitor can register subscriptions of the following five types: *ArtistSubscription*, *ArtifactSubscription*, *RoomInfoSubscription*, *TopicSubscription*, and *LocationContextSubscription*. Each subscription provides several options for rules to match publications based on content and context.

A visitor can register multiple subscriptions of a type using different information matching rules. New filtering rules can be added to a subscription type. The currently implemented options in our testbed are outlined below. When a visitor enters a room, his/her registered subscriptions are matched against the publications associated with that room.

The *ArtistSubscription* type supports options to get notifications of the following kinds: (a) information about all artists from a given list of artists whose work is displayed in the current room, (b) the above matching rule can be further refined to obtain information about only those artists for whom the number of displayed works in the room is in a given range. For example, a visitor may want to get the list of the artists who have at least three artifacts displayed in the room.

The *ArtifactSubscription* type supports the following options to get notifications in the current room for: (a) all artifacts in the room which were created (collaboratively) by a specified set of artists, (b) all artifacts which were created within the specified range of years, and (c) all artifacts created collabo-

[1]Source: https://github.com/artsmia/collection

ratively such that for each artifact the number of collaborators is in a specified range.

The *RoomInfoSubscription* type allows one to specify interest in five kinds of information notifications in the current room: (a) the number of artists whose works are displayed in the room, (b) the number of artifacts in the room, (c) name of the artist who has the most number of artifacts in the room, (d) information about all artists whose work is displayed in the room, and (e) information about all artifacts displayed in the room. A subscription of this kind can include any number of these five filtering options, and these options do not require any parameters.

The *TopicSubscription* type is used by a visitor to obtain notifications about the ephemeral publications posted at a location on some topic. An ephemeral publication specifies the following items: (a) topic name, (a) *target* field specifying a list of users, or groups, or public who should be notified of it, (c) information text, (d) expiration time, and (e) *creator* field identifying the individual who published it. A user creates and registers a *TopicSubscription* specifying the topic name along with other options such as the creator id or the creator's group id.

A *LocationContextSubscription* is registered by a visitor to obtain context related information for a room. For example a visitor may request notifications for situations when: (a) a specific individual is present in that room, (b) any member of a given group is present in the room, and (c) the number of members from a specified group in the room goes above or below some threshold number. Such a subscription by a visitor is associated with a specific room. Even when that visitor is not be present in that room, he/she would be notified when any of the specified situations occurs.

Our testbed implementation of this service provides interfaces to add or remove users, and register or delete subscriptions of a user. In addition to obtaining information through notifications, users of this service can make ad hoc queries to obtain information. The current testbed system supports several such queries. These include queries for finding: (a) which rooms contain the works of a given artist, (b) which rooms contain works created during a given period, (c) which artists created work during a given period, (d) who created the most number of artifacts, and (e) who collaborated most number of times with other artists. Such queries can be used by a person for selecting the rooms to visit during a tour.

V. Graph-Based Computation Model for the Museum Visitor Service

Our work in designing the museum visitor service has been primarily driven by the goal of investigating a graph-based model for data management and computation structures required for its location-based publish/subscribe model. This graph model needs to support event-triggered execution of continuous query functions on user-location changes and occurrence of other events such as the addition of new publications or subscriptions. The system architecture for this service is required to manage a large and dynamic collection of different types of objects and their relationships. These objects represent and contain information about rooms, artifacts, artists, visitors touring the museum, subscriptions registered by each visitor, and location-based publications. In this section we describe the graph model we have developed for our testbed system. In the next section we describe how this graph-based model is implemented using the Beehive framework.

In the architectural design of the museum visitor service using the graph-based approach, nodes (vertices) in the graph represent rooms, artifacts, artists, users (visitors), publications, and location-context based subscriptions. These nodes are represented as data objects, with various fields depending on the type of the node. Figure 2 shows an example graph representation for a subset of the museum dataset. We use this example graph to illustrate the relationships maintained between the nodes. Each node is identified by a unique node-id, which is used as the key for accessing it in the storage system. For example, a room node contains a list called *usersPresent* containing the node-ids of all the users (i.e. visitors) present in the room. Each user node contains a field called *currentLocation* which is set to the node-id of the room where the user is present. When the relationship between two nodes is symmetric, as in the above example, we show this simply by a bidirectional edge in this figure, implying that each node is maintaining the id of the other node. When a user moves from one room to another, then both room nodes and the user node are required to be updated.

In the graph data model for the museum service, the relationship between an artifact and an artist is represented by a bidirectional edge. If an artifact was collaboratively created by multiple artists, then in this case it would have edges to multiple artist nodes. The relationship between an artifact and the room where it is currently located is represented by a bidirectional edge between them. Thus, a room node contains a list of ids for all artifacts present in that room. Each artifact node contains information about that work, such as its title, names of it creators, year of creation, any other additional information, and the node-id of the room where is it is located.

The example in Figure 2 shows four artists: Renoir, Van Gogh, Monet, and Matisse. Room G355 contains two works by Renoir, one by Van Gogh, and one by Monet. It also shows that three visitors (A, B, and C) are present in this room. Room G377 contains one work by Monet and two by Matisse. There are three visitors (D, E, and F) present in this room. A list of subscriptions is associated with each visitor, and these are maintained in the corresponding user node objects.

An ephemeral publication in this system is also represented as a node and it is associated with a room. This relationship is shown by a bidirectional edge between a room and a publication. A room may have multiple such publications associated with it. In Figure 2, *P*, *Q*, and *R* are ephemeral publications. Publications *P* and *Q* are associated with Room G355, and *R* is associated with Room G377. A *LocationContextSubscription* by a user is also represented by a node in this graph data model. Such subscriptions are shown by hexagonal nodes in the graph. Each such subscription is associated with one or

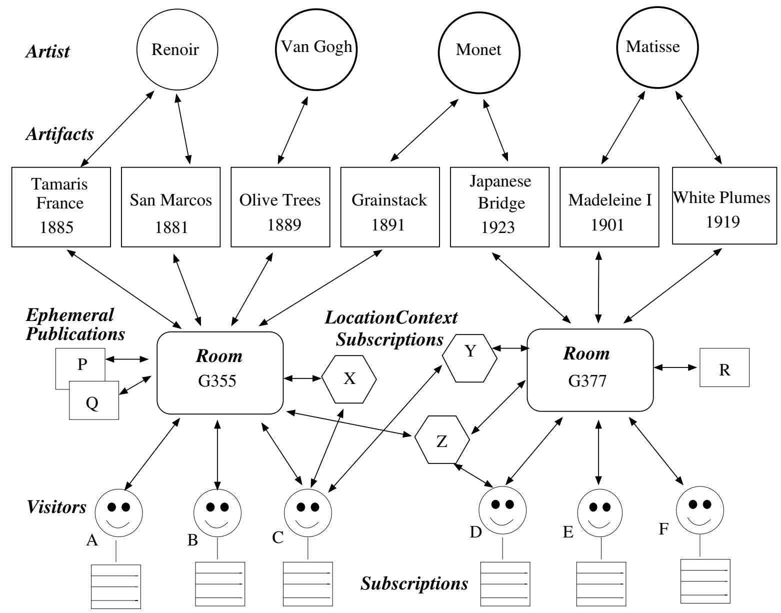

Fig. 2. Graph Data Model for Museum Visitor Service

more rooms and also with the user who created it. In this example, *X*, *Y*, and *Z* are such location-context subscriptions. Subscriptions *X* and *Y* are created by visitor C and they are associated with different rooms. Location-context subscription *Z* is created by visitor D and it is associated with two rooms to monitor the same context condition in both rooms.

Executions of continuous query processing tasks to match subscriptions with publications are triggered when the following kinds of events occur: user location change update, and addition of new publications and subscriptions by a user. User location change updates are the most common events in this environment. When a user moves from one room to another room, we need to first update the graph nodes representing the two rooms and the user. Next, the subscriptions currently registered for the user are matched with the publications (both permanent and ephemeral) associated with the room which the user has entered. Any notifications generated in this processing phase are recorded into the *pendingNotification* list maintained in the user node, to be sent to the user at a later point. We then check all *LocationContextSubscriptions* registered with the two rooms if the arrival/departure event of the user requires any notifications to be sent to the associated subscribers.

When a new subscription is registered by a user, the user node is updated. Additionally, the publications associated with the user's current room are matched with the new subscription. If the new subscription belongs to the *LocationContextSubscription* type, then the matching operation is performed for the target room with which that subscription is associated. Posting a new publication for a room requires checking the subscriptions for each user present in that room.

VI. Implementation of the Museum Service using the Beehive Platform

The Beehive [35] framework provides a transactional model of parallel computing on cluster computers for graph problems. Graph data is stored in a key-value based data store maintained in the RAM of the cluster computers. Each node in the graph is represented by a *Node* object in this data store. The base *Node* class contains a core set of data items such as the node-id and information about the edges to its neighbor nodes. An application can extend this class as needed. A node object is accessed or modified using its id as the access key.

Parallel computations in Beehive are performed as vertex-centric tasks. A task computation can read or update any nodes in the graph. A task can modify the graph structure at runtime to add/remove nodes and edges, or modify their attributes. A task may also create new tasks on its completion. The central concept is to execute the computation tasks as serializable transactions, satisfying the properties of *atomicity* and *isolation*. We refer to these computations as *transactional tasks*. Multiple tasks can be executed in parallel. On each Beehive cluster node, a pool of worker threads is maintained. A distributed task-pool in the system maintains a set of tasks to be executed. A worker thread picks a task from the task-pool and performs its computation as a transaction. A task is removed from the task-pool upon its successful execution.

The Beehive model for transactional task execution is based on optimistic concurrency control techniques. This is a lock-free model of execution, and a transaction reads only committed data. Under the optimistic execution model, tasks can be executed in parallel. The optimistic execution of a transactional task involves the following four phases: *read phase*, *compute phase*, *validation phase*, and *update phase*. In the compute phase a transactional task reads the required data items from the global storage into its local memory buffers. All updates are written to the private buffer memory. After the compute phase, a validation is performed for the transaction to check for conflicts. A transactional task is committed only if no other concurrently committed transaction has updated any items in its *read-set* or the *write-set*. If two concurrent transactional tasks conflict, then one of them is guaranteed to complete execution while the other is aborted. An aborted task is re-executed again later. On committing, the transactional task writes the buffered updates to the global storage and any new tasks created by it are added to the task-pool. A new task then executes at some later time when it is picked by some worker thread.

Beehive is a Java-based framework, and its core classes for representing nodes, edges, and tasks can be extended by an application. The implementation of the museum visitor service extends the *Node* class to represent different types of nodes, as shown in Figure 2, to represent artists, artifacts, rooms, users, publications, and subscriptions. In addition to the transactional task model, we included support in Beehive for the execution of in-line transactions which can be executed on any of the Beehive cluster computer running the museum visitor service to serve client requests. A client process executing on a user's interface (device) can initiate the following kinds of transactions:

• An *UpdateLocation* request is sent by a client to indicate the current location of its user. It indicates the new room where the user is present. This request is handled as a transaction which updates three nodes in the graph, the first representing the previous room, the second for the new room, and the third representing the user. This transaction then creates two new tasks. The first task performs matching of the user's subscriptions with the artifacts, artists, and publications associated with the new room. Any notifications generated by this task are added to the user's *pendingNotification* list. The service process handling this request waits until this task has completed before it forwards all pending notifications to the client. The second task performs checking of all *LocationContextSubscriptions* associated with the previous and new room. Any matching notifications generated by this task are added to the pending notification lists of the users who have made those subscriptions.

• A client can make an *UpdateSubscription* request to add or remove subscriptions for its user. In case a location-context based subscription is added, a new node of this type is created by this transaction. Moreover, it creates edges between this new node and the associated room and the user node. Adding new subscriptions results in creating a new task for matching artifacts, artists, and publications associated with the user's current room with the newly added subscriptions. For a location-context subscription, this task checks the state of the room with which that new subscription is associated. This task may result in adding new notifications to the user's *pendingNotification* list.

• A client posts a new ephemeral publication at a room by invoking the *Publish* transaction. This transaction creates and adds in the graph a new publication node and also updates the room node with which the new node is associated. A new task is created by this transaction to match this new publication with the subscriptions of the users present in that room. This task may result in creating and adding notifications for some of these users.

• A *PullNotification* request is made by a client to obtain all pending notifications for its user.

• The other transactions provided for the client interface support addition or removal of users.

We found that implementing all these transactions with strong consistency guarantee (i.e. ensuring serializability) can significantly reduce the performance and scalability of the system. For example, in our initial implementation of the museum visitor service, the *UpdateLocation* transactions encountered high abort rates. Each such transaction updated three nodes: two room nodes and one user node. The probability of two or more such concurrent transactions aborting increased with increasing number of visitors because multiple users entered or exited a room at the same time. This led us to rethink the implementation of this transaction using a weaker consistency model. In this implementation, when a user moves from one room to another, these two room nodes are updated by using a reflection-based remote object method execution model of Beehive. To support this, the room node class provides methods to *add/remove* a user in its *usersPresent* list. We also exploit the commutativity properties of these operations when validating a transaction to allow greater concurrency [36]. The location update transaction updates the two rooms using this mechanism. Thus, multiple concurrent transactions of this kind can update a common set of rooms without conflicting in the validation phase. The implication of this weaker model is that when a visitor enters a room, his/her presence may not be observed by some concurrent transactions. It may sometimes result in some missed notifications. Similar observation holds for a visitor leaving a room.

VII. Performance Evaluation

The primary objective of our experimental evaluations was centered on the performance of the museum visitor service under different load conditions and on various cluster sizes. For each cluster size configuration, we wanted to determine the maximum number of visitors which could be supported without significant performance degradation. The performance evaluation experiments were conducted on a cluster with nodes of different cpu power and memory. The more powerful nodes in this cluster had 16 cores of 1.2 GHz CPU and 65 GB memory. The less powerful nodes had 8 cores of 1.2 GHz CPU and 65 GB memory.

For our experiments we developed two visitor profiles, each with different kinds of subscriptions. We used a simple model of user mobility where a user visited all rooms in a random order. We conducted two sets of experiments, one for each of the following profiles.

• *Profile A:* A visitor with this profile subscribes to the list of all artifacts and all artists whose works are displayed in the visitor's current room.

- *Profile B:* A visitor with this profile has four subscriptions. An *ArtistSubscription* is used for specifying a set of artists to find those who have their works displayed in the current room. An *ArtifactSubscription* is used to find collaboratively created work in the current room such that the number of collaborators for a work is above a threshold. This threshold is set to 2 in our experiments. A *RoomInfoSubscription* type is used for getting the number of artifacts displayed in the room. A *LocationContextSubscription* is associated with a specific room to get notifications when two or more members of the visitor's group are present in the room.

In our experiments with these two profiles, we posted 10 ephemeral publications for each visitor group in randomly selected rooms. In the first set of experiments, which used *Profile A*, all visitors belonged to the same group. With *Profile A*, there were total 268 notifications generated for each visitor, and the average number of information items per notification was close to 20.

In the second set of experiments we used *Profile B* and created multiple groups of visitors, with each group containing 50 members. With *Profile B* we set the list of artists to contain six names. The average number of notifications received by a visitor was 189, and the average number of information items per notification was 2.35. The average number of location-context notifications received by a visitor was found to be close to 26 for a group size of 50.

In our experiments, we created a client thread to emulate a visitor. Thus, an experiment with 100 client threads emulated an environment with 100 visitors present in the museum at the same time. Each client thread registered subscriptions according to the given profile and then invoked the location-update transactions, visiting all rooms in a random order. After invoking a location-change update, the client thread waited to receive all notifications before making the next location-change update. In our benchmark, each client visited all museum rooms 10 times. The performance measure we examined was the completion time for the execution of the benchmark for a given set of clients, whose size was varied.

We configured the Beehive system to execute the key-value data storage service on a set of 16-core nodes of the cluster and varied the number of the nodes allocated for the data storage service. We used a set of 8-core nodes of the cluster as front-end to interface with the clients and to execute the transaction management functions. This separation of functionalities between the data storage service nodes and front-end nodes facilitates independent allocation of resources for data storage and transaction management functions for performance tuning. We also varied the number of these nodes for scaling out under different load conditions.

To improve the performance of this system we included support for object caching at the front-end cluster nodes executing the transactions. We observed that in the museum visitor service certain kinds of graph node objects are rarely updated. These include node objects representing artists and artifacts. In our experiments we cached these two types of objects at the front-end nodes. This significantly reduced network traffic between the front-end and data storage service nodes.

First we conducted some initial set of experiments using *Profile A* to find the peak capacity of a configuration using one cluster node serving as front-end for executing the transactions initiated by clients and one cluster node for the data storage service. These experiments showed that the system was able to support about 50 clients, and the limit was reached because of the capacity saturation of the front-end node. With that observation we used a simple rule of adding one additional front-end node per 50 clients.

Using *Profile A* we measured the benchmark execution time for different system configurations, varying the number of data storage service nodes and front-end nodes. In our experiments the number of data storage service nodes was varied from 1 to 3. The number of front-end nodes was set in proportion to the total number of clients, roughly one front-end node per 50 clients. Figure 3 shows the benchmark execution times for different number of clients and cluster sizes. The average values of transaction response times in these experiments are shown in Figure 4. For *Profile B*, Figure 5 shows the benchmark execution times for different number of clients and cluster sizes. The average values of transaction response times in these experiments are shown in Figure 6. The performance trends for both profiles are similar as the number of clients and the number of cluster nodes are varied. We observe that adding more data storage service nodes has benefits only when the number of clients exceeds some threshold.

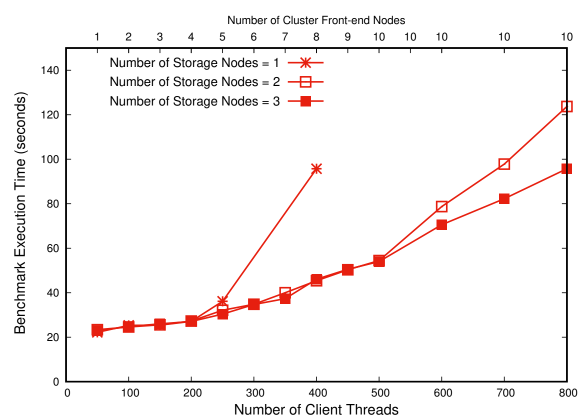

Fig. 3. Benchmark Execution Time using Profile A

Our performance evaluation experiments show that the architecture presented in this paper can be scaled to achieve high throughput for location update transactions. For example, our experiment using *Profile A* with 800 clients and executing on a cluster of 13 nodes performed 10,778 location update transactions per second. Each of these transactions also resulted in the execution of two additional transactional tasks for matching user subscriptions. In our experiments using *Profile B* with 800 clients and executing on a cluster of 13 nodes, the system attained throughput of 7619 location update transactions per second. Each of these transactions also resulted in execution

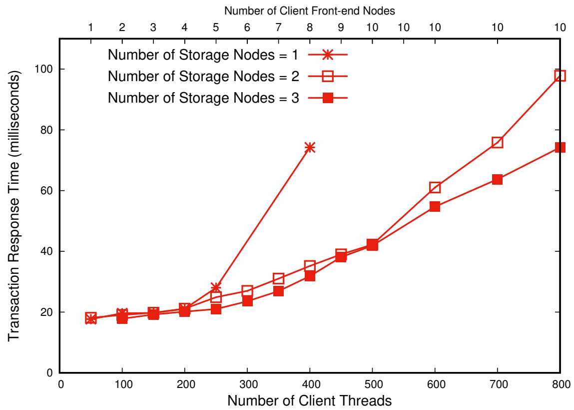

Fig. 4. Average Transaction Response Time with Profile A

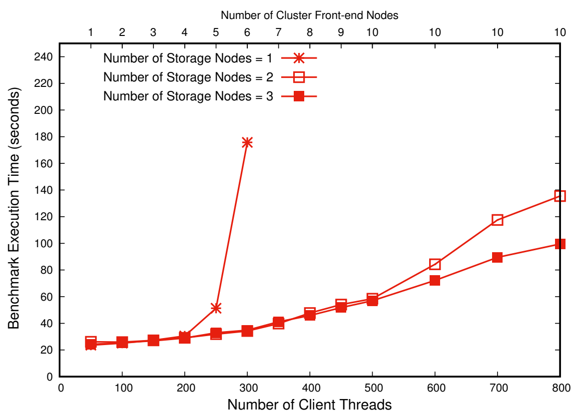

Fig. 5. Benchmark Execution Time using Profile B

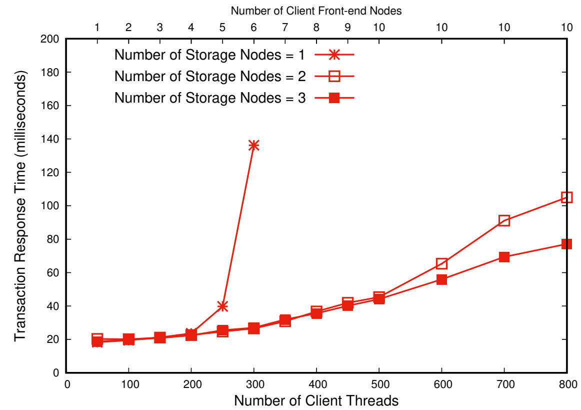

Fig. 6. Average Transaction Response Time with Profile B

of up to four additional transactional tasks.

VIII. Conclusion

In this work we have investigated a graph-based model for building location-based publish/subscribe systems. We developed a Museum Visitor Service as a case study, based on a dataset from the Minneapolis Institute of Art. In our investigation, for managing dynamic graph data structures we have utilized the Beehive framework, which supports parallel programming of graph problems on cluster computers. Our work leverages Beehive's transactional model of parallel computing for managing dynamic graph structures and supporting execution of continuous queries and matching functions. Specifically, the publish/subscribe service for the museum application is implemented using Beehive's key-value based graph data management facilities to manage various types objects, their interrelationships, and the computation tasks required in this application.

Our work shows that a graph-based approach for building location-based publish/subscribe services provides a conceptually clear model for representing dynamic relationships between various objects such as users, subscriptions, publications, and locations. We have shown how the transactional computing model of Beehive can be utilized for managing dynamic graph data structures in location-based publish/subscribe services hosted on cluster computers. An important observation from this work is that using strong consistency models for all transactions may lead to poor performance. Some transactions may be implemented using weaker consistency models to improve performance. The work reported here is the result of our initial investigation in this direction. There are several aspects of this work which demand more in-depth investigation, which we plan to pursue in the future. These include the use of more expressive models for specifying matching rules in subscriptions and support for different transactional consistency models for data management in such systems. Deployment of the museum service in the real environment to gain user-experience data would help us identify cases where weaker consistency models can be used for higher scalability and performance.

In the future, we want to investigate how the graph based model for location-based publish/subscribe services can be applied in other application domains such as public transportation systems. Exploration in other domains would require support for richer location models, such as geofencing and spatial models. This would require development of efficient mechanisms for representing geo-spaces and managing the associated publications/subscriptions in the context of the graph-based model presented here. Exploration into other application domains and location models would require us to investigate mechanisms to enhance the scalability this approach.

Acknowledgements: This work was supported by NSF Award 1319333 and computing resources were provided by NSF award 1512877 and the Minnesota Supercomputing Institute. Alexander Cina contributed in the initial development of the tools for loading MIA dataset on the Beehive system.

References

[1] IEEE Pervasive Computing - Special Issue on Location Based Services. January-March 2010.

[2] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton. Cyberguide: A Mobile Context-aware Tour Guide. *Wireless Networks*, 3(5):421–433, 1997.

[3] G. Banavar, T. Chandra, B. Mokherjee, J. Nagaraja, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *ICDCS '99: Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, page 262, Washington, DC, USA, 1999. IEEE Computer Society.

[4] R. Barazzutti, P. Felber, C. Fetzer, E. Onica, J.-F. Pineau, M. Pasin, E. Rivière, and S. Weigert. Streamhub: A massively parallel architecture for high-performance content-based publish/subscribe. In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, pages 63–74, New York, NY, USA, 2013. ACM.

[5] A. R. Beresford and F. Stajano. Location privacy in pervasive computing. *IEEE Pervasive Computing*, 2:46–55, January 2003.

[6] Brian Ferris and Kari Watkins and Alan Borning. Location-Aware Tools for Improving Public Transit. In A. Dey, J. Hightower, E. de Lara, and N. Davies, editors, *IEEE Pervasive Computing - Special Issue on Location Based Services*, pages 13–19. January-March 2010.

[7] C. Cañas, E. Pacheco, B. Kemme, J. Kienzle, and H.-A. Jacobsen. Graps: A graph publish/subscribe middleware. In *Proceedings of the 16th Annual Middleware Conference*, Middleware '15, New York, NY, USA, 2015. ACM.

[8] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *PODC '00: Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 219–227, New York, NY, USA, 2000. ACM.

[9] G. Cugola, E. Di Nitto, and A. Fuggetta. The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS. *IEEE Trans. Softw. Eng.*, 27(9):827–850, 2001.

[10] G. Cugola and A. Margara. Raced: an adaptive middleware for complex event detection. In *ARM '09: Proceedings of the 8th International Workshop on Adaptive and Reflective MIddleware*, pages 1–6, New York, NY, USA, 2009. ACM.

[11] N. Davies, K. Cheverst, K. Mitchell, and A. Efrat. Using and Determining Location in a Context-Sensitive Tour Guide. *IEEE Computer*, 34(8):35–41, August 2001.

[12] M. Duckham and L. Kulik. Location privacy and location-aware computing. In J. Drummond, R. Billen, E. Joo, , and D. Forrest, editors, *Dynamic and Mobile GIS: Investigating Change in Space and Time*. CRC Press, 2006.

[13] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[14] Graeme Stevenson and Simon Dobson and Paddy Nixon. LOC8: A Location Model and Extensible Framework for Programming with Location. In A. Dey, J. Hightower, E. de Lara, and N. Davies, editors, *IEEE Pervasive Computing - Special Issue on Location Based Services*, pages 28–37. January-March 2010.

[15] A. Gupta, O. D. Sahin, D. Agrawal, and A. El Abbadi. Meghdoot: Content-based publish/subscribe over p2p networks. In *Middleware 2004: ACM/IFIP/USENIX International Middleware Conference, Toronto, Canada, October 18-22, 2004. Proceedings*, pages 254–273. Springer Berlin Heidelberg, 2004.

[16] M. Ionescu and I. Marsic. Stateful publish-subscribe for mobile environments. In *WMASH '04: Proceedings of the 2nd ACM international workshop on Wireless mobile applications and services on WLAN hotspots*, pages 21–28, New York, NY, USA, 2004. ACM.

[17] H.-A. Jacobsen. Middleware for location-based services. In J. Schiller and A. Voisard, editors, *Location-Based Services*, pages 83–114. Morgan Kaufmann Publishers, San Mateo, CA, 2004.

[18] H.-A. Jacobsen, A. K. Y. Cheung, G. Li, B. Maniymaran, V. Muthusamy, and R. S. Kazemzadeh. The PADRES Publish/Subscribe System. In *Principles and Applications of Distributed Event-Based Systems*. IGI Global, 2010.

[19] K. R. Jayaram, P. Eugster, and C. Jayalath. Parametric content-based publish/subscribe. *ACM Trans. Comput. Syst.*, 31(2):4:1–4:52, May 2013.

[20] C. Kottman. Location-Based/Mobile Services. Available at: http://www.opengeospatial.org, May 2000.

[21] G. Li, Y. Wang, T. Wang, and J. Feng. Location-aware publish/subscribe. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 802–810, New York, NY, USA, 2013. ACM.

[22] M. Li, F. Ye, M. Kim, H. Chen, and H. Lei. A scalable and elastic publish/subscribe service. In *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium*, IPDPS '11, Washington, DC, USA, 2011. IEEE Computer Society.

[23] P. J. Ludford, D. Frankowski, K. Reily, K. Wilms, and L. Terveen. Because I carry my cell phone anyway: functional location-based reminder applications. In *CHI '06: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 889–898, New York, NY, USA, 2006. ACM.

[24] G. Myles, A. Friday, and N. Davies. Preserving privacy in environments with location-based applications. *IEEE Pervasive Computing*, 2:56–64, January 2003.

[25] P. Persson, F. Espinoza, and E. Cacciatore. Geonotes: social enhancement of physical space. In *CHI '01 extended abstracts on Human factors in Computing Systems*, CHI '01, pages 43–44. ACM, 2001.

[26] S. Rodriguez Garzon and B. Deva. Geofencing 2.0: Taking location-based notifications to the next level. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '14, pages 921–932, New York, NY, USA, 2014. ACM.

[27] A.-M. Roxin, C. Dumez, M. Wack, and J. Gaber. Middleware models for location-based services: a survey. In *AUPC '08: Proceedings of the 2nd International Workshop on Agent-oriented Software Engineering Challenges for Ubiquitous and Pervasive Computing*, pages 35–40, New York, NY, USA, 2008. ACM.

[28] J. Schiller and A. Voisard. *Location Based Services*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[29] H. Schulzrinne, H. Tschofenig, J. Morris, J. Cuellar, and J. Polk. Internet Draft: Geolocation Policy: A Document Format for Expressing Privacy Preferences for Location Information. Available at http://tools.ietf.org/html/draft-ietf-geopriv-policy-22, 2010 October.

[30] B. Segall and D. Arnold. Elvin Has Left the Building: A Publish/Subscribe Notification Service with Quenching. In *Proceedings of Queensland AUUG Summer Technical Conference*, Brisbane, Australia, 1997.

[31] S. Shekhar, R. R. Vatsavai, X. Maa, and J. S. Yoo. Navigation sysems: A spatial database perspective. In J. Schiller and A. Voisard, editors, *Location-Based Services*, pages 41–80. Morgan Kaufmann Publishers, San Mateo, CA, 2004.

[32] E. Snekkenes. Concepts for personal location privacy policies. In *Proceedings of the 3rd ACM conference on Electronic Commerce*, pages 48–57, 2001.

[33] S. Spiekermann. General aspects of location-based services. In J. Schiller and A. Voisard, editors, *Location-Based Services*, pages 9–26. Morgan Kaufmann Publishers, San Mateo, CA, 2004.

[34] M. Strassmana and C. Collier. Case study: Develoment of find friend application. In J. Schiller and A. Voisard, editors, *Location-Based Services*, pages 27–40. Morgan Kaufmann Publishers, San Mateo, CA, 2004.

[35] A. Tripathi, V. Padhye, T. S. Sunkara, J. Tucker, B. Thirunavukarasu, V. Pandey, and R. R. Sharma. A Transactional Model for Parallel Programming of Graph Applications on Computing Clusters. In *10th IEEE International Conference on Cloud Computing*, 2017.

[36] A. Tripathi and B. Thirunavukarasu. A Transaction Model for Management of Replicated Data with Multiple Consistency Levels. In *Proc. of IEEE Intl. Conference on Big Data*, 2015.

[37] X. Wang, Y. Zhang, W. Zhang, X. Lin, and W. Wang. Ap-tree: Efficiently support location-aware publish/subscribe. *The VLDB Journal*, 24(6):823–848, Dec. 2015.

[38] Y. Wang and X. Ma. A general scalable and elastic content-based publish/subscribe service. *IEEE Transactions on Parallel and Distributed Systems*, 26(8), August 2015.

[39] Y. Wang and M. A. Perez-Quinones. Beyond "geofencing": Specifying location in location-based reminder applications. In *Proceedings of the 33rd Annual ACM CHI EA '15*, pages 1767–1772, New York, NY, USA, 2015. ACM.

[40] Y. Zhao, K. Kim, and N. Venkatasubramanian. Dynatops: A dynamic topic-based publish/subscribe architecture. In *Proc of the 7th ACM DEBS*, pages 75–86, New York, NY, USA, 2013. ACM.