2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing

Towards Scalable Processing for a Large-Scale Ride Sharing Service

Beihong Jin, Jiafeng Hu
Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
Graduate University of Chinese Academy of Sciences, Beijing 100190, China

***Abstract*—Ride sharing is a promising way to realize a convenient, economic and low-carbon travel. After analyzing and refining the requirements of a ride sharing service, the paper models the trajectory matching therein and discusses the implementation of a large-scale ride sharing service with the aim of improving the efficiency and scalability.**

Keywords- Pub/Sub; trajectory matching; ride sharing service; scalable processing

I. INTRODUCTION

Although protecting environments and conducting sustainable development are world-wide consensuses, the current actual states still remain not optimistic. In many large and medium-sized cities, the number of vehicles increases rapidly. For example, statistics released in February 2012 show there are more than 5 million vehicles in Beijing. Meanwhile, traffic congestion becomes a common phenomenon in our daily life. The traffic congestion not only impacts on people's normal work and life, but also causes a lot of economic losses. Moreover, vehicle exhaust pouring into the air will lead to the obvious decrease of air quality and also damage the health of the individuals. Currently, vehicle exhaust has become an important factor of city air pollution. Therefore, a comprehensive approach is needed to address the above situations with the goal of creating the environments suitable for human habitation and working. Some cities (such as Beijing) carry out the even-odd license plate policy, which limits the dates that a vehicle can move on the road so that the amount of vehicles can be decreased and congestion will be alleviated. Some projects introduce station-car systems, which permit a user to pick up a car at any station and then leave it at another station. Ride sharing lets the travelers with similar itineraries and time schedules to share the journeys so that more than one person travels in a car. In 2012, the policy of taxi sharing, a kind of ride sharing, comes out in Beijing. Undoubtedly, ride sharing is an effective measure to lessen the traffic jam and improve ambient air quality. Besides the societal and environmental benefits, ride sharing can reduce the costs of both drivers and passengers by sharing the trip costs (including fuel expense). So far, many cities have actively carried out the ride sharing.

To support ride sharing, a ride sharing service needs to be developed. In terms of functionality, such a ride sharing service should receive the requests of passengers who have the desires for ride sharing and then automatically search for the drivers who are willing to share the vacant seats in their cars. Since the drivers and passengers can be decoupled in time, space, and control flow, the ride sharing service can be regarded as an application of Pub/Sub middleware, i.e., the car-sharing requests submitted by the passengers are subscriptions, the drivers' car-sharing intentions are events, and a Pub/Sub middleware will perform the matching subscriptions with events and deliver the matching results to both passengers and drivers. In general, it provides the calculation of the trip-related costs and the detailed method of payment. In additional, a ride sharing service may provide the client software which can be deployed on mobile phones for user convenience, and make a connection with any reputation system or social network tool to ease the fear of potential security threats. However, we think the greatest challenge in a ride sharing service comes from the dynamics of loads. The car-sharing subscriptions and events are subject to change frequently and unpredictably. Facing the dynamics in a ride sharing service, the cloud infrastructure on cluster computers, featured as the elastic resource provision and a pay-as-you-go manner, is appropriate for hosting a ride sharing service. But in order to utilize the cloud infrastructure to the utmost, the designers are required to present a scalable matching solution for ride sharing.

The paper introduces our ongoing research work aiming at the scalable processing for a large-scale ride sharing service. The rest of this paper is organized as follows. Section II elicits the requirements for a ride sharing service and models the trajectory matching problem, where the trajectory stands for a vehicle routing. Next, Section III discusses a potential solution for trajectory matching. Then, the related work is given in Section IV. Finally, the paper is concluded in the last section.

II. MODELLING

2.1 Requirement Analyses

In general, the ridesharing desires published by drivers include the following three types of information: (1) Fundamental factors: a license plate number of vehicle, driver's gender, estimated departure and arrival time (both are in terms of a time interval), a travel trajectory (alternatively, a starting point, an ending point, and the points that need passing should be given, the last one is set to null by default), the number of vacant seats (one or more), the required gender of passengers. (2) Economic factors: the saving expenses in terms of a numerical value or a percentage. (3) Social factors: driver's various preferences. For example, some drivers only accept people who come from the same community or colleagues as passengers; or some drivers only accept non-smokers.

From the point of view of passengers, they have to declare their identities, gender, expected departure time, expected arrival time, an origin, and a destination. Similarly, they can declare their requirements in economic aspect and the social aspect.

978-0-7695-4843-2/12 $26.00 © 2012 IEEE
DOI 10.1109/UIC-ATC.2012.152

The matching for ride sharing in a ride sharing service is to match the ridesharing desires of drivers with that of passengers.

Although the ride sharing can obtain social benefits including reduction of traffic congestion and air pollution, it has to be easy, safe, flexible, efficient, and economical so that it can be widely adopted and becomes popular in practice. In particular, it is expected to achieve the following optimization goals: (1) maximize the success rate of matching for ride sharing, (2) minimize the travel time from a system-wide perspective or from individual driver-passenger pairs, (3) minimize the vehicle-miles from a system-wide perspective or following the principle of Pareto optimum for a driver-passenger pair.

We note that optimizing system-wide metrics does not mean that the metrics of a driver-passenger pair can also reach the optimal. On the other hand, in terms of system-wide vehicle-miles, the ride sharing is feasible if total vehicle-miles of a ridesharing trip are less than the sum of vehicle-miles of individual trips of its participants. Some possible ridesharing situations maybe like this: A driver takes a detour to the passenger's location and picks him up, or, a driver takes a detour to a passenger's destination and then drives up to his own destination. Permitting the above situations can increase the success probability of ride sharing, but such situations may lead to detour, i.e., changing the driver's original route.

The following example is used to illustrate the differences among metrics. Fig. 1 shows a road network, where the numbers on the edges represent the distances between two nodes. We assume that two drivers (D1, D2) and two passengers (P1, P2) have their origins and destinations below.

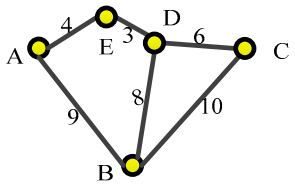

Figure 1. a road network

D1: E→B
D2: A→B
P1: C→B
P2: D→B

There are two ride sharing solutions. Solution 1 consists of two pairs: (D1, P1) pair whose routing is E→D→C→B, (D2, P2) pair with a routing of A→E→D→B. Solution 2 includes only a pair, (D1, P2) pair with E→D→B. Solution 1 is superior to solution 2 in terms of the number of successful matching (a system-wide metric). But in solution 2, (D1, P2) pair is optimal in terms of vehicle-miles. At the same time, in order to let D1, D2, P1 and P2 arrive at their destinations, solution 1 needs 34 vehicle-miles in total and solution 2 is 30 from a system-wide perspective.

We think providing a solution for optimizing the metrics for driver-passenger pairs without changing the driver's original routes is more important while giving first place to pursuing better user experience.

We observe that the ride sharing desires can be expressed by a logical combination of predicates in a form of "type attribute operator value". Meanwhile, all the factors except the driver's trajectory can be described by the values of primitive types (such as integer, string). However, the diversity of trajectories enables it to have the maximal degree of discrimination. Therefore, we divide the matching of ride sharing into two parts: the matching of trajectories and matching of the other predicates, where the matching of trajectories is to decide whether the passenger's origin and destination are on the planning route of a driver. Conducting the matching of trajectories as a first step can greatly reduce the scope of searching and relieve the burden of matching of the other predicates, so the efficient trajectory matching is the key in ride sharing.

2.2 Modelling the trajectory matching

If a driver only gives an origin and a destination rather than a trajectory in his ride sharing desire, then the trajectory from the origin and the destination can be generated from a digital navigation map. If the locations that need passing are given besides the origin and the destination, then the whole trajectory can be obtained by two steps: (1) generating the sub-trajectories by the digital navigation map, taking as its input the two locations passed through in turn; (2) concatenating the sub-trajectories to form the whole trajectory.

The essence of the trajectory matching is to find the ones which pass the two designated points (i.e. original point and destination point, denoted by OPoint, DPoint) from a set of trajectories. Obviously, every trajectory can be divided into a set of connected segments. If OPoint is on one segment (denoted by SegA), DPoint is on another segment (denoted by SegB), and SegA and SegB belong to the same trajectory, then the matching succeeds.

III. SOLUTIONS

3.1 A Basic Solution

To deal with the trajectory matching, the road network needs to be partitioned into several disjoint regional subnets so that every road only belongs to a regional subnet. Since the network partition is known as NP-complete, there are two ways to partition a road network. One is based on network semantics. For example, a city government may divide the whole city into multiple districts as basic administrative units; thereby the roads in the city can be divided by the districts, and the roads in a district form a regional subnet. Another is to adopt existing heuristic algorithms. For example, we can adopt the network partitioning approach in [1], which clusters roads into partitions based on spatial proximity [2].

Next, an R tree (called by segment R-tree) is built for regional subnets, where every leaf node stores a minimal rectangle which bounds a regional subnet, and the road segments in the regional subnet. As thus, a trajectory can be stored in the leaf nodes in the form of road segments, so long as a trajectory ID is assigned to the segments which belong to the same trajectory. On the other hand, OPoint or DPoint also can be stored in a leaf node whose MBR (minimum bounding rectangle) covers the location of OPoint or DPoint.

For example, a part of road network in Fig. 2 (a) can be divided into four disjoint regions R1-R4 shown by dotted lines. If a segment R-tree is created for the road network,

then MBRs in the leaf nodes of the R tree will be four red solid rectangles in Fig. 2 (b).

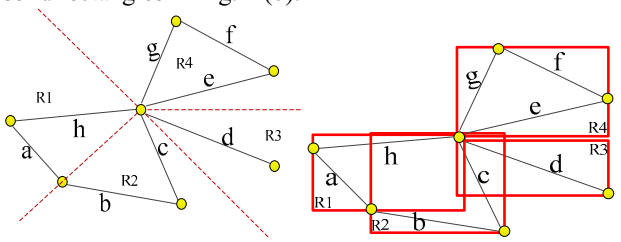

Figure 2. (a) a sample road network (b) MBRs

We note in many existing researches, building an index for spatial data is serving for mobile objects, i.e., it is used to support the queries like this: Given an ID of a moving object, its location or the other objects (and their locations) related with the object can be obtained by traversing the index. But unlike the above access method to trajectories, the trajectory matching in a ride sharing service is to find the trajectories on which a passenger's location is, and then to find the owners of the trajectories. So, we still use the R-tree or its variants to organize the trajectories but need to develop a new algorithm for the trajectory matching.

The new algorithm should include two parts: (1) it takes a trajectory as input. For every segment (denoted by SegX) which consists of the trajectory, it first locates the leaf node that SegX belongs to, and then matches SegX with the stored OPoints or DPoints at the leaf node. If the trajectory is found that it passes the stored OPoint and DPoint which are with same owners at the same time, then the trajectory matching succeeds, else the trajectory will be saved in several leaf nodes. (2) it takes OPoint and DPoint as input, and judges whether they are on some trajectory by traversing the R tree. If OPoint and DPoint are on the same trajectory, then the trajectory matching succeeds. Otherwise, it will save OPoint and DPoint on the proper leaf nodes by their locations.

3.2 A Scalable Solution

We assume that a driver's request has a trajectory predicate which has a set of segments {SegX} as its value besides the other predicates, and a passenger's request gives the values of OPoint and DPoint as well as the other predicate constraints.

If a driver's request permits to share multiple seats (e.g. *m* seats), then this request will be treated as *m* events, where each event only permits one seat to be shared.

To obtain efficient and scalable matching, we design three kinds of servers: an access server X, several working servers Yi and a finalized server Z, and deploy all these servers in VMs in a cloud infrastructure (e.g. XenServer). Here, X maintains an R-tree to store all the segments in the network of roads, and in every leaf node, a Y is designated to deal with the requests within its administrative region, i.e., the MBR of the leaf node. At service startup, only one Y is deployed, so it is responsible for dealing with all the requests. As time goes by, the number of Y (denoted by *n*) is adjusted automatically by the method in next subsection. Further, Yi (1<=i<=*n*) in the leaf nodes is also changed, and the Yi may have one or multiple administrative regions. Yi stores the requests of passengers whose OPoints or DPoints are in administrative region(s) of Yi, in other words, Yi classifies the requests of passengers in terms of the segments, and then organizes each group of requests of passengers into a tree STree according to covering relations among the contents of requests of passengers. Moreover, Yi stores in a list named DList the unmatched drivers' requests whose trajectories pass by the administrative region(s) of Yi. Finally, Z maintains a list named SemiMatchedList which consists of ReqDId, ReqD, the type of point (original point or destination point), ReqP, and ReqPId.

Assuming that there are only one Y and four requests from passengers as follows,

ReqP1: Opoint=o1, Dpoint=d1 //o1 is on Seg_h, d1 is on Seg_g
Start time: 2012-7-7 9:00am-9:30am, Arrival time: 2012-7-7 10:00am-10:20am
ReqP2: Opoint=o2, Dpoint=d2 //o2 is on Seg_h, d2 is on Seg_e
Start time: 2012-7-7 9:05am-9:15am, Arrival time: 2012-7-7 10:20am-10:30am
ReqP3: Opoint=o3, Dpoint=d3 //o3 is on Seg_h, d3 is on Seg_c
Start time: 2012-7-7 2:10pm-2:25pm, Arrival time: 2012-7-7 3:00pm-3:15pm
ReqP4: Opoint=o4, Dpoint=d4 //o4 is on Seg_h, d4 is on Seg_c
Start time: 2012-7-7 9:10am-9:25am, Arrival time: 2012-7-7 10:00am-10:15am

the above requests are organized as shown in Fig. 3.

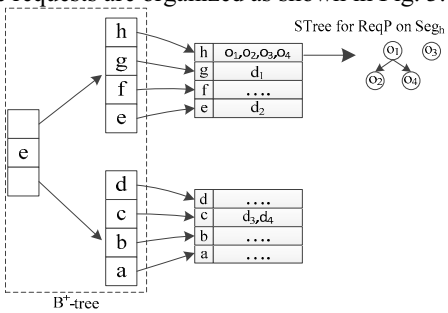

Figure 3. passenger requests on Y

The whole procedure of processing a driver's request is as follows:

(1) While a driver's request ReqD arrives at X, X assigns a unique identifier ReqDId to ReqD. Next, for every segment SegX in the ReqD, X finds in the R tree the leaf node (identified by LeafID) which stores SegX and then sends the ReqD, ReqDId, LeafID and SegX to the Yi whose administrative region covers SegX.

(2) While Yi receives the above message from X, it searches the STree corresponding to SegX. If OPoint or DPoint (MatchedPoint hereinafter) in a stored request of passenger ReqP is on SegX and ReqD satisfy the other predicate constraints in ReqP, then it packages ReqP, ReqPID, ReqD, ReqDId and type of MatchedPoint (original point or destination point) into a message and sends it to Z, otherwise it will store the driver's request in DList.

(3) Upon receiving the message from Yi, Z searches SemiMatchedList. If it finds an entry whose ReqDId is the same as that of the message and the type of node is opposite to that of the message, which means that a successful ridesharing pair (ReqD, ReqP) is found, then it sends a successful matching message piggybacking ReqPId to Yi, and a notification to the passenger and the driver, else it saves the message into the SemiMatchedList and sends a non-matched message in passing ReqDId to Yi.

(4) While Yi receives the successful matching message from Z, it will delete the stored information related with ReqPId. If Yi receives the non-matched message, it will store the unmatched driver' request in DList.

The whole procedure of processing a passenger's request is as follows:

(1) While a passenger's request ReqP arrives at X, X assigns a unique identifier ReqPId to ReqP. Next, for OPoint and DPoint in the ReqP, X finds in the R tree the leaf node (identified by LeafID) whose MBR covers OPoint or DPoint, and then find the SegX on which OPoint or DPoint is. After that, it sends the ReqP, ReqPId, LeafID and SegX to the Yi whose administrative region covers SegX.

(2) While Yi receives the above message from X, it searches the DList to find the ReqD whose segment covers SegX and other predicates can satisfy ReqP. If existed, then it packages ReqP, ReqPID, ReqD, ReqDId and type of MatchedPoint (original point or destination point) into a message and sends it to Z, otherwise it will store the passenger's request in STree.

(3) Upon receiving the message from Yi, Z searches SemiMatchedList. If it finds an entry whose ReqPId is the same as that of the message and the type of node is opposite to that of the message, which means that a successful ridesharing pair (ReqD, ReqP) is found, then it sends a successful matching message piggybacking ReqDId to Yi, and a notification to the passenger and the driver, else it saves the message into the SemiMatchedList and sends a non-matched message in passing ReqPId to Yi.

(4) While Yi receives the successful matching message from Z, it will delete the stored information related with ReqDId in DList. If Yi receives the non-matched message, it will store the unmatched passenger's request in some STree.

3.3 Self-adjusting the number of Y

Let $|e|$ denote the number of requests concerning segment e, and E_Y denote the set of all segments which Y is charge of. We use $|e|$ as the load of segment e, and $|E_Y|=\sum_{e\in E}|e|$ as the load of Y.

We record the loads of all Ys at X. When X finds Y is over-loaded, i.e., $|E_Y| > \Delta max$, then X decides to add a new Y (Ynew). In detail, X needs to split the leaf nodes Y manages into two sets setL1 and setL2 so that the difference between the total load of segments in L1and the total load of segments in setL2 is minimal. This can be turned into a classical set partitioning problem which can be solved by the method in [3]. The time complexity of the method in [3] is *O(nlogn)* in most cases, where *n* is the number of the elements in the set. After obtaining setL1 and setL2, X will update the information stored in leaf nodes of the segment R-tree. We assume that Y will manage the leaf nodes in setL1 and Ynew setL2. Then, Y will send to Ynew the passenger's requests whose LeafID belong to setL2 (in terms of STree). In addition, Y will divide DList into two parts Dlist-L1 and Dlist-L2 according to the set which the LeafID of an element of Dlist belongs to (i.e., setL1 or setL2), and sent Dlist-L2 to Ynew.

On the other hand, when X finds that Y is low-loaded, i.e., $|E_Y| < \Delta min$, then X will search the server who can hold the Y's load. X chooses the server whose load is minimal (except Y) (named Ymin), and transfers the load of Y to Ymin if the sum of load of Y and Ymin is less than Δmax. Finally, Y can be shut down.

The following gives an example to show the effect of adjustment. We assume that four leaf nodes are managed by Y1 so the load on Y1 consists of four parts {(R1,30),(R2,16),(R3,73),(R4,56)}, after applying set partitioning algorithm, we can get two sets as follows:

set1={(R1,30), (R3,56)};
set2={(R2,16), (R4,73)};
where ||set1|-|set2||=3

As a result, as shown in 4(a), Y1 is assigned to deal with the requests concerning R1 and R3 which are shown as red dotted rectangles in Fig. 3(b) respectively, and Y2 is for requests concerning R2 and R4 (blue solid rectangles in Fig. 4(b)).

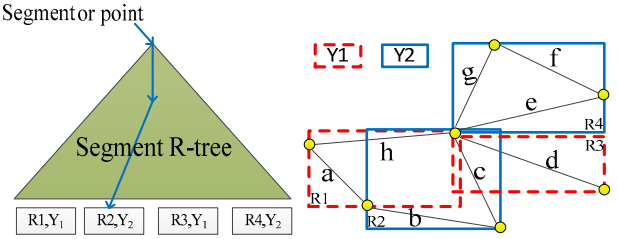

Figure 4(a). a segment R-tree at X (b) the regions managed by Yi(i=1,2)

Every Y periodically makes the unmatched requests persistent on a separate storage server, and records the updates about the requests happened between two persistence jobs. As thus, while Ynew is activated, the most of data Ynew needs can come from the cloud storage instead of the heavy-loaded Y.

IV. Related Work

In recent years, multiple applications including Carticipate (http://www.carticipate.com/), Piggyback (http://www.piggybackmobile.com/?page_id=10), Zimride (http://www.zimride.com/), Avego (http://www.avego.com/), GoLoco (http://goloco.org/greetings/guest) and Wodache (http://wodache.com/) offer ride sharing service. But there are no reports related with the scalability of these applications.

[4] is a report which is prepared for U.S. Department of Transportation. It not only surveys the state of the art of dynamic ridesharing projects in U.S. but also presents a formal model for ridesharing matching problem. [5] systematically outlines the optimization challenges that arise at the time of developing techniques to support ridesharing.

Our target system is a kind of location based service whose users interact with each other in a Pub/Sub way; meanwhile it is expected to be built on the cloud infrastructure. Therefore, our research on a ride sharing service is related to index technology for spatial data from the field of database, distributed processing of trajectory data

and Pub/Sub middleware from the field of distributed systems, and migrating existing systems to cloud infrastructure from cloud computing.

Since the original R-tree was proposed for spatial data by Guttman [6], there have been many variants [7], such as 3D R-tree for spatiotemporal data, TB-tree (Trajectory Bundle tree) and SETI (Scalable and Efficient Trajectory Index) for trajectory data, FNR-tree (Fixed Network R-tree) for spatial data in a fixed spatial network. However, all of them are built on a single server. In regards to distributed processing of trajectory data, most of systems classified under moving objects databases [8], streaming processing systems [9], or location management systems [10] adopt the space partitioning approach, i.e., every server is designated to be responsible for a region in advance, and then it stores the segments which belong to the region, or execute the queries related to the region or updates the locations of objects in the region. To speed up the processing of trajectory data, a distributed index may be built, for example, [8] builds a distributed trajectory index similar to a skip list so as to accelerate the routing of a query to the server which stores the queried segments.

Pub/Sub middleware provide a mechanism to filter and disseminate the events according to user's requirements. Existing systems include SIENA [11] for wide-area network applications, PADRES [12] for workflow management and business process, OncePubSub [13] for RFID applications, APUS [14] for VANET applications. But in a ride sharing service, the response to ride sharing requests should be on very short notice or even en-route. Such requirement does not take into accounts in the existing Pub/Sub systems.

Migrating existing systems to cloud infrastructure seems a sound solution to cater for load fluctuation. Many efforts have been done, including migrating n-tier applications [15], service-oriented systems [16] and Pub/Sub systems [17]. In general, while being migrated to the cloud infrastructure, a computation-intensive system can obtain the scalability by replicating its components. But a data-intensive system may spend a lot of time to transfer the data in memory and bear heavy burdens of network traffic while utilizing the elastic resources in the cloud. The situation will be more complex while involving the persistent data, since persistent data are required to replicate the updates. At the worst, if the applications put the constraints on the storage locations of data and data routing, it may lead to re-construct the system for the cloud. Similarly, a ridesharing system over the cloud also relates to server consolidation and migration. In regards to data transferring between servers, our way is to take advantage of the cloud storage so as to decrease the amount of data transferring and speed up the procedure of load balance.

V. CONCLUSIONS

Ridesharing can be viewed as a kind of Pub/Sub application which suffers from load fluctuation but is required to respond quickly. We discuss its implementation method over the cloud in the paper. Next step is to implement and evaluate the ride sharing service comprehensively.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China under Grant 60970027.

REFERENCES

[1] N. Jing, Y.-W. Huang, E.A. Rundensteiner, Hierarchical Encoded Path Views for Path Query Processing: An Optimal Model and Its Performance Evaluation, IEEE Trans. Knowledge and Data Eng., vol. 10, no. 3, pp. 409-432, 1998.

[2] Y.-W. Huang, N. Jing, E.A. Rundensteiner, Effective Graph Clustering for Path Queries in Digital Map Databases, Proc. Fifth Int'l Conf. Information and Knowledge Management, 1996.

[3] N.karmarkar and R. M. Karp. The Differencing Method of Set Partitioning. Technical Report UDB/CSD 82/113, UC, Berkeley, 1982.

[4] Keivan Ghoseiri, Ali Haghani, Masoud Hamedi, Real-Time Rideshare Matching Problem, January 2011

[5] Agatz, N., Erera, A., Savelsbergh, M., Wang, X., Optimization for Dynamic Ride-Sharing: A Review, European Journal of Operational Research, 2012, doi: http://dx.doi.org/10.1016/j.ejor.2012.05.028

[6] Guttman, A. R-trees: A dynamic index structure for spatial searching. In Proceedings of the 1984 ACM SIGMOD International Conferencee on Management of Data, Jun 1984, pp. 47-57.

[7] Yannis Manolopoulos, Alexandros Nanopoulos, Apostolos N. Papadopoulos, Yannis Theodoridis, R-Trees: Theory and Applications, Springer-Verlag London Limited 2006

[8] Ralph Lange, Frank Dürr, Kurt Rothermel, Scalable Processing of Trajectory-Based Queries in Space-Partitioned Moving Objects Databases, ACM GIS08, November 5-7, 2008, Irvine, CA, USA.

[9] X. Xiong, H. G. Elmongui, X. Chai, W. G. Aref. PLACE*: A Distributed Spatio-temporal Data Stream Management System for Moving Objects. In Proc. of 8th MDM, May 2007.

[10] J. Zhang, G. Zhang, L. Liu. Geogrid: A scalable location service network. In Proc. of 27th ICDCS, June 2007.

[11] Carzaniga A, Rosenblum DS, Wolf AL. Design and evaluation of a wide-area event notification service. ACM Transaction on Computer Systems, 2001, 19(3)

[12] Li GL, Jacobsen A. Composite Subscriptions in Content-based Publish/Subscribe Systems, the 6th ACM/IFIP/USENIX International Middleware Conference 2005.

[13] Beihong Jin, Xinchao Zhao, Zhenyue Long, Fengliang Qi, Shuang Yu, Effective and Efficient Event Dissemination for RFID Applications, the Computer Journal, Volume 52, Issue 8, Nov. 2009.

[14] Fusang Zhang, Beihong Jin, Wei Zhuo, Zhaoyang Wang, Lifeng Zhang, A Content-Based Publish/Subscribe System for Efficient Event Notification over Vehicular Ad hoc Networks, The 9th IEEE International Conference on Ubiquitous Intelligence and Computing, Fukuoka, Japan, 2012

[15] Deepal Jayasinghe, Simon Malkowski, Qingyang Wang, Jack Li, Pengcheng Xiong, Calton Pu, Variations in Performance and Scalability when Migrating n-Tier Applications to Different Clouds, IEEE 4th International Conference on Cloud Computing, 2011

[16] Muhammad Aufeef Chauhan, Muhammad Ali Babar, Migrating Service-Oriented System to Cloud Computing An Experience Report, IEEE 4th International Conference on Cloud Computing, 2011

[17] Biao Zhang, Beihong Jin, Haibiao Chen, Ziyuan Qin, Empirical Evaluation of Content-based Pub/Sub Systems over Cloud Infrastructure, The 8th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, 2010, Hong Kong, China.