A P2P Meta-Index for Spatio-Temporal Moving Object Databases

Cecilia Hernandez[1], M. Andrea Rodriguez[13] and Mauricio Marin[23]

[1] Dept. of Computer Science, Universidad de Concepción, Chile, {cecihernandez,andrea}@udec.cl
[2] Yahoo! Research, Chile, mmarin@yahoo-inc.com
[3] Center for Web Research, Universidad de Chile, Chile

Abstract. In this paper we propose a distributed meta-index using a peer-to-peer protocol to allow spatio-temporal queries of moving objects on a large set of distributed database servers. We present distribution and fault tolerance strategies of a meta-index that combines partial trace of object movements with aggregated data about the number of objects in the database servers. The degrees of distribution and fault tolerance were compared by using discrete-event simulators with demanding spatio-temporal workloads. The results show that the distributed meta-index using Chord provides good performance, scalability and fault tolerance.

1 Introduction and Related Work

For query processing, an important aspect of distributed moving object applications is the distribution of the index structure. Advances in this sense are studies that distribute spatial index structures, such as the Quadtree and R-tree structures, in peer-to-peer (p2p) networks [4, 7]. In similar way, studies in distributed spatio-temporal databases have typically considered a distributed global index structure, which organizes spatio-temporal information in a structure defined in terms of space and temporal partitions [2, 5]. These studies have addressed window queries (i.e., time-instant and time-interval queries) and not queries about locations of particular objects or aggregated queries.

In this paper, we describe an alternative approach to distributed spatio-temporal database servers. First, and unlike previous works, we design a structure to support different types of queries including coordinate-based queries (i.e., time-slice and time interval queries), aggregated queries (top-K servers with the largest number of objects and geographic extent), and combined queries of the form "Where was an object o at a time instant t or time interval $[t_1, t_2]$?" We argue that it is impractical to think of solving these types of queries with a global index when the system is composed of a large and dynamic number of servers. Second, instead of having a global and distributed spatio-temporal structure, we propose to handle independent local indexes in database servers and a global and distributed meta-index structure. This could be seen as a two-level architecture;

however, we are not forcing any coordination between the meta-index and the local indexes.

A preliminary study for this meta-index [3] analyzes the viability of creating and updating a centralized structure in a highly dynamic environment of moving objects. This paper continues with our previous work and describes the distributed meta-index structure using Chord [6]. In this paper we present experimental evaluation that shows the good performance, scalability and fault tolerance properties of the proposed system.

The structure of the paper is as follows. Section 2 describes the meta-index, whose distribution and fault tolerance strategies are presented in section 3. Section 4 presents experimental evaluation supporting our claims on performance, scalability and fault tolerance. Finally, we outline our conclusions in Section 5.

2 Meta-Index Description

A basic component of the system is the meta-index structure that guides the search process to local servers or gives approximated answers to different queries. In particular, we studied the following types of queries: (1) time-slice or time-interval queries that return objects or trajectories within a query window and time instant or interval, (2) aggregated queries concerning the top-K servers with largest number of objects or trajectories and largest extent area, (3) nearest neighbors to a specific object and time instant, and (4) queries about the location of a particular object at a specific time instant.

The meta-index stores partial data about the time-varying location of objects. These data include: time-varying number of objects per server (statistical or aggregated information), time-varying geographic extent including the location of objects in a server, and coarse traces of object visits across servers. Coarse traces of objects in the meta-index are not "real" sparse trajectories, but lists of servers that objects have visited sorted by the time of the data collection.

Among all queries of interest, this paper concentrates only on queries about the location of specific objects at a particular time instant, since they represent a challenging and not previously addressed query in the context of distributed spatio-temporal databases. This type of queries uses the object traces and number of objects per server at different time stamps in the meta-index. Using coarse traces for answering this type of queries is novel since we do not use the classical space partition to organize or distribute spatio-temporal information.

The search algorithm, based on object traces, finds the first location of the object at a time instant t' such that $|t' - t|$ is minimum, with t the query time. If the trace of an object is not found in the meta-index, the algorithm starts search on servers with largest number of objects at query time. When a first location of the object is found, the algorithm follows the object path in the corresponding servers until finding the location of the desired object at query time.

To maintain the meta-index, we define a crawling strategy that collects data from database servers asynchronously [3]. In every data collection from a server, a crawler transfers to the meta-index aggregated data and the ids of objects which have been in the server but not transfered in previous visits. Consequently, some objects might no longer be present in the server at the time of the data collection.

3 Meta-Index Distribution

We base the distribution of the meta-index on a p2p network using Chord protocol [6]. Chord defines a common address space for nodes and data keys and provides a lookup algorithm. This algorithm enables distribution by mapping data keys among a changing set of nodes with $O(\log n)$ routing cost.

In order to use bandwidth efficiently, we propose two strategies for distributing the meta-index data. First, we introduce the concept of *MSB (Most Significant Bits)* to map moving object trace data to Chord keys. Crawlers use the MSB to group object trace data in *Composite Objects*. The number of bits defined by the MSB constitutes a threshold for the update message maximum size performed by crawlers' robots. Second, since the amount of aggregated data per update message is very small we piggy it back to composite objects. This scheme allows crawlers to send fewer update messages into the p2p network avoiding overloading the network. Figure 1 shows an example of how a robot groups object traces into two composite objects. In this case the MSB is 28 which means that the maximum number of moving object ids in a composite object is 16 (2^4). Here, composite objects are identified by $compositeObj_1$ and $compositeObj_2$ with their corresponding moving object ids and aggregated data (SD1).

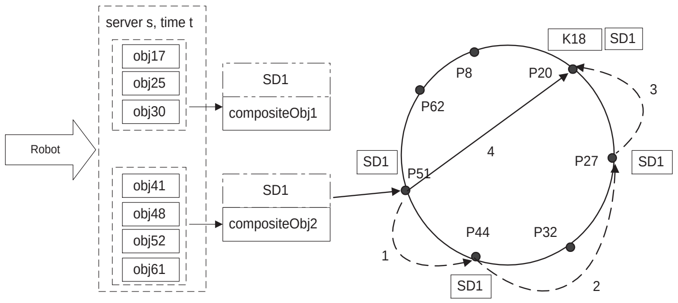


Fig. 1. Meta-index distribution

The meta-index distribution strategy requires the administration of two types of messages. *S-messages* (messages 1-3 in Figure 1) and *T-messages* (message 4 in Figure 1). *S-messages* carry control and aggregated data but they do not carry composite object content. Aggregated data are stored in all peers visited until finding the destination peers. *T-messages* transfer composite object content from the entry peer to the peer that will store the composite object.

The distribution of the meta-index supports fault tolerance in two ways: by the partial replication of aggregated data in all peers, and by distributing the object traces uniformly at random in p peers. The replication of aggregated data is useful when no data about the trace of the desired object is found in the distributed meta-index. In such case, the number of objects stored in any of the peers is used to rank the servers where to most likely find the desired objects. Second, data collected by a crawler are grouped into *sub-composite* object ids based on MSB and a random number between 1 and p. Then, object traces are associated randomly with one of the p possible sub-composite objects and stored in the corresponding peer. When a client looks for a specific object at a given time, it starts the search at a random peer, this peer looks up the p sub-composite objects in parallel and sends back the best answer to the client, where best here means the estimated location of the object at the closest time to the query. If one of the peers holding the requested object is down, another peer among the p ones may have it. Even though the reply might not be the best, it would serve as starting search point for the client.

Operations on our meta-index system architecture are decentralized. First, crawlers collect data from database servers and update the meta-index in the p2p network in parallel. Crawlers build composite objects based on MSB and p, and send them to the peer-to-peer network through *Update*() messages. Second, peers receive and delegate *Update*() operations (Algorithm 1) to other peers in order to find the destination peer. We allow peers to receive queries from any node on the network through the *RecvClientQuery*() procedure. This procedure takes in a moving object and query time and finds at most p peers that resolve the query. The procedure $DoQuery()$ looks up recursively the composite object id in the peer-to-peer network and then the query resolving involves applying the search algorithm described in 2. Peer operations are defined in Algorithm 1.

4 Experimental Evaluation

Our simulation environment uses event-driven simulators. One simulates the database servers and crawling generating the data in the form of the meta-index. Another simulates the Chord protocol to allocate and lookup meta-index data. We also used the network simulator NS-2 ($http://www.isi.edu/nsnam/ns$) in tandem with GT-ITM ($http://www.cc.gatech.edu/projects/gtitm$) to simulate the network topology and environment.

We use workload data generated by a public spatio-temporal dataset generator; the Network-based Generator of Moving Objects (NGMO) [1]. The data set contains 50,000 initial moving objects, existing around 150,000 along the simulation time. In similar ways as seen in [7], we run the experiments with 4, 8, 16, and 32 crawlers' robots and 16, 32, 64 and 128 peers. We chose peers and clients randomly from stub nodes defined in a transit-stub network of 588 nodes created with NS-2/GT-ITM.

Search performance of centralized versus distributed meta-index. We first evaluate the performance of the search process for a centralized versus

Algorithm 1 Algorithms at peers

```
 1: procedure Update (peer entrypeer, composite object id coid, aggregated data sd)
 2: if aggregated data sd is not in thispeer then
 3:     Add sd into this thispeer
 4: end if
 5: if coid must be stored in thispeer then
 6:     Get composite object co with $co.id = coid$ from entrypeer
 7:     Add composite object co with $co.id = coid$ into thispeer
 8: else
 9:     call Update message to nextpeer (based on Chord protocol)
10: end if
11: end procedure
```

```
1: procedure RecvClientRequest (moving object id moid, timestamp queryTime)
2: Get all composite object ids coidList using moid, $MSB$ and $P$
3: for each coid in coidList in parallel do
4:     call DoQuery(coid) on a random peer
5: end for
6: Resolve best reply to client
7: end procedure
```

```
1: procedure DoQuery (composite object id coid)
2: if coid is in thispeer then
3:     call Algorithm1 defined in section 2
4:     return
5: else
6:     call DoQuery(coid) on nextpeer (based on Chord protocol)
7: end if
8: end procedure
```

distributed meta-index by using the number of visits to local servers as a performance metric to answer a query of type "find object o at time stamp t," (Figure 2 (a)) . We used three types of benchmarks, each with 200 queries: (b1) random objects (average trajectory length of 32 servers), (b2) objects with largest trajectories (average trajectory length of 175 servers), and (b3) objects with shortest trajectories (average trajectory length of 2 servers). Here, the meta-index contains traces for 71% of the total number of moving objects (150,000) that exist during simulation time; that is, approximately 30% of queries for the whole data set would need to use statistical data from the meta-index.

Figure 2 (b) shows the performance of the distributed meta-index for 200 random queries using different percentages of searches that require the statistical data. Here, using statistical data in a search is important because this data is partially recovered from peers, which can affect the performance with respect to the number of visits to local servers during searches. The quality of the statistical data recovery would improve when using p greater than 1, since query solving would combine statistical data from p peers. We present the ratio between a centralized and distributed meta-index search using 8 robots and statistical data recovery from only one peer. In this figure, values close to 1.0 mean that the

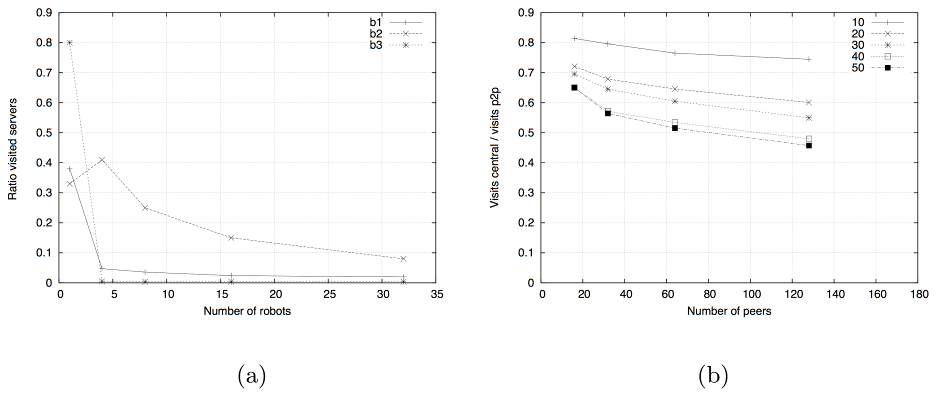


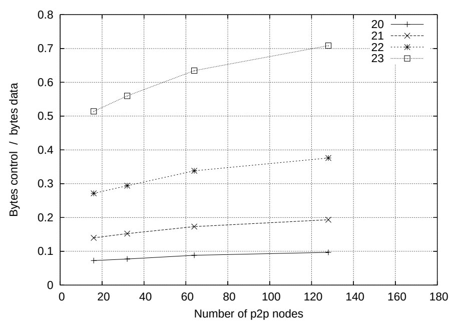
Fig. 2. Performance of the meta-index: (a) The ratio number of servers visited using the meta-index to the number of servers visited using the server random selection strategy (The x-axis indicates the different numbers of robots updating the meta-index), (b) Search performance with a distributed meta-index with respect to a centralized meta-index (Different curves indicate percentages of use of aggregated data in the search)

distributed and centralized meta-index have similar performance with respect to the number of visits to local servers.

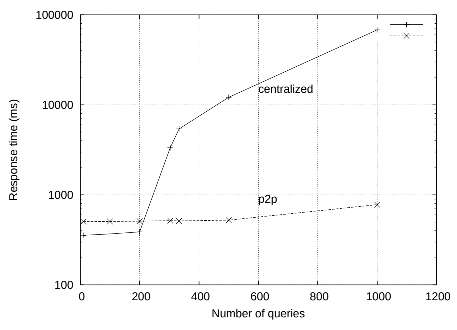
Communication Overhead. In order to evaluate the communication overhead associated with the distribution of the meta-index in the p2p network, we run an experiment that shows the number of bytes transmitted in update operations (i.e., bytes in S-messages and T-message). This number of bytes is normalized by the total number of bytes of the meta index, such that we can obtain an overhead with respect to the original data content we want to distribute. Figure 3(a) shows the normalized number of bytes with respect to different numbers of peers and different setting of MSB.

Elasticity. We run an experiment to compare the scalability of the distributed meta-index with the centralized system. We measured the response time for a protocol that use MSB=20, P=1, 128 peers and different rates of query concurrency, from 10 to 1000 queries per second given to the system. Figure 3 (b) indicates that over 200 queries per second, the performance of the centralized system starts to be affected significantly due to the workload. In this case, the response time in the centralized scheme does not scale because of network congestion. The P2P system has a great amount of available bandwidth allowing it to scale gracefully with increasing query rate.

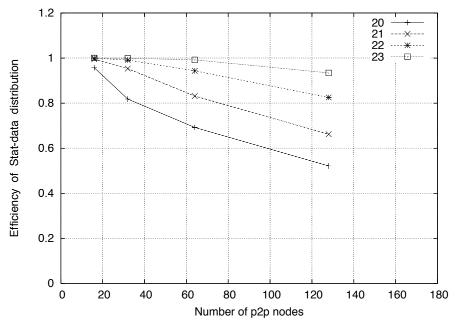
Fault Tolerance. We analyzed the fault tolerance in terms of the strategies described in 3. Figure 3(c) shows the efficiency of the distribution of the aggregated data among peers. We define efficiency in a strict hand-to-hand manner by averaging over all servers and peers on the ratio average number of objects to the maximum number of objects observed in any peer for the same server.



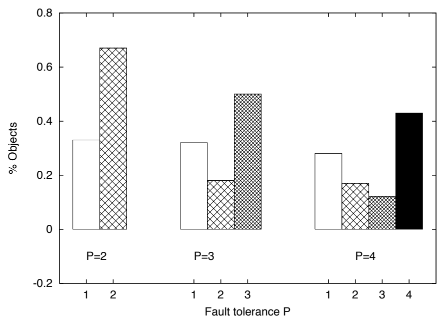
(a)



(b)



(c)



(d)

Fig. 3. Communication overhead, elasticity, and fault tolerance of the meta-index: (a) relative number of bytes transmitted in update operations, (b) response time for different query rates given to the system for centralized and p2p scheme (MSB=20, p=1 and 128 peers), (c) efficiency of distribution of statistical data in peers, and (d) quality of fault tolerance distribution for different values of p

This is a demanding distribution test for both the degree of completeness and precision of the samples kept in each real p2p node. There is a trade-off with MSB, since fewer bits lead to a small number of composite objects circulating among the peers.

We also measured the distribution of an object trace in the participating p peers. We show results about the percentage of objects that maintain traces between 1 and p peers. In Figure 3(d), the first bar for each p represents the percentage of objects whose traces are only in one peer (objects with shortest traces), the second bar represents the percentage of objects whose traces are in two peers, and so on. The results indicate that our strategy is able to provide fault tolerance for object traces for 70% of the objects in the system.

5 Conclusions

Overall our strategy allows the two main processes, namely crawling and searching to make efficient use of the p2p network. Crawlers can store their payload in any peer and user queries can also start up in any peer. Thus the scheme can accommodate many crawlers, which improve the probability of registering at least one trace per moving object. Also, user queries throughput can be increased by evenly distributing the start of searches across all of the p2p nodes.

Our experiments results show that combining trace-based data with statistics about the number of objects per server in the meta-index provides performance guarantees in comparison with using random access to database servers to resolve the query addressed in this paper. Moreover, the centralized trace meta-index data does not experiment any loss in the distribution scheme. However, the distribution of the statistical data suffers partial loss in comparison with the centralized scheme, but, as seen in Figure 2, it improves in at least 50% the overall performance.

Acknowledgements. Cecilia Hernandez is partially funded by DIUC Grant - 206.091.044-1.0, University of Concepcion, Chile. Andrea Rodriguez and Mauricio Marin are funded by Nucleus Millennium Center for Web Research, Grant P04-067-F, Mideplan, Chile.

References

1. Thomas Brinkhoff. A framework for generating network-based moving objects. *GeoInformatica*, 6(2):153–180, 2002.
2. H. Lee, J. Hwang, J. Lee, S. Park, C. Lee, and Y. Nah. Long-term location data management for distributed moving object databases. In *Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, pages 451–458. IEEE Press, 2006.
3. Mauricio Marín, Andrea Rodríguez, Tonio Fincke, and Carlos Román. Searching moving objects in a spatio-temporal distributed database servers system. In Robert Meersman and Zahir Tari, editors, *OTM Conferences (2)*, volume 4276 of *Lecture Notes in Computer Science*, pages 1388–1401. Springer, 2006.
4. Anirban Mondal, Yi Lifu, and Masaru Kitsuregawa. P2pr-tree: An r-tree-based spatial index for peer-to-peer environments. In Wolfgang Lindner, Marco Mesiti, Can Türker, Yannis Tzitzikas, and Athena Vakali, editors, *EDBT Workshops*, volume 3268 of *Lecture Notes in Computer Science*, pages 516–525. Springer, 2004.
5. Y. Nah, J. Lee, W.J. Lee, H. Le, M.H. Kim, and K.J. Han. Distributed scalable location data management system based on the GALIS architecture. In *Tenth IEEE International Workshop on Object-Oriented Real Time Dependable Systems*, pages 397–404. IEEE Press, 2005.
6. Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, 2003.
7. Egemen Tanin, Aaron Harwood, and Hanan Samet. Using a distributed quadtree index in peer-to-peer networks. *VLDB J.*, 16(2):165–178, 2007.