Keeping Track of “Flying Elephants”: Challenges in Large-Scale Management of Complex Mobile Objects

T. Drosdol[1], T. Schwarz[1], M. Bauer[1], M. Großmann[1], N. Hönle[1], D. Nicklas[1]

Universität Stuttgart, Institute of Parallel and Distributed Systems
Universitätsstraße 38, 70569 Stuttgart, Germany
{drosdol,schwarts,bauer,grossmann,hoenle,nicklas}@informatik.uni-stuttgart.de

Abstract: The management of mobile objects like cars, persons, or workpieces in a factory is an important task in many context-aware environments. So far, most solutions can either cope with many small objects (few properties) or with a limited number of complex objects in a centralized way. In this paper, we face the challenge of managing a large number of bulky mobile objects (flying elephants). We state requirements, propose basic components, and discuss alternative architectures based on the main influencing factors like mobility and update rate.

1 Introduction

Keeping track of mobile objects is an important task when managing mobile environments: wireless carriers need to know where to route an incoming call to, road authorities wish to know the location of traffic jams, or parents want to find their lost children in an amusement park. Hence, more and more mobile objects are tracked, and selecting only the desired ones requires utilizing a variety of additional information besides the objects' positions: *“Who else at this banquet is a Mozart-loving violinist?”* Such information could comprise user profiles, inventories, or a 3D model of the object to be used as a virtual avatar. In this paper we present the solution space for dealing with complex mobile objects (flying elephants) in a large-scale (possibly world-wide) environment.

Current research in location management for mobile networks focuses on highly scalable, distributed systems for tracking users in order to route calls [MD04]. However, the data maintained per object is small – just object ID (OID), position (pos), and some billing data – and is accessed solely using the OIDs, but not using spatial queries. On the contrary, moving object databases [Wo98] and spatio-temporal databases [Gü00] store complex objects and allow for spatial and non-spatial access to them. However, current systems are centralized and can only cope with a relatively small number of objects. In Nexus, our Location Service tracks mobile objects and offers spatial queries [LR02]. It already is highly scalable, too. However, in order to work efficiently, it assumes the amount of data per mobile object to be small, so that handovers remain reasonably cheap.

As illustrated above, there is a clear demand for providing spatial access to many complex mobile objects. Since none of the current approaches is satisfactory, we investigate how to construct a large-scale system meeting this demand. In the following, we state the system's requirements, derive its basic components, and discuss architectural alternatives.

[1] funded by the German Research Foundation (DFG) within the Center of Excellence (SFB) 627.

2 Requirements

A large-scale system for managing complex mobile objects has to meet several functional and non-functional requirements. Supporting *position updates* and different kinds of *queries* are the major functional requirements. The frequency of *position updates* directly influences the accuracy of an object's stored position. Consequently, as position is one of the primary access paths, frequently changing positions need to be reflected in the spatial index and may necessitate handovers of objects to different servers.

Location-based services and context-aware applications typically access data using three types of queries: *OID queries*, *spatial queries*, and *arbitrary filter* queries. Note that these queries only select information but do not perform joins of any kind, as the applications we looked at did not require them. Yet, we intend to consider joins in future work.

OID queries find the objects corresponding to given OIDs. They are used to follow references or relations between objects. In the cell-phone environment only this type of query is employed. It is imperative that objects can be efficiently retrieved using just their OID. *Spatial queries* subsume all kinds of queries that filter objects based on their position: range, window, and nearest-neighbor queries. Efficiently accessing objects by position is especially important for location-based applications, which issue queries like *"Which other players are closer than 100 meters to the user?"* As location-based applications extensively use OID queries as well, we conclude that OID and position are the two primary access paths that require special attention. *Arbitrary filter queries* select objects using arbitrary combinations of conditions on spatial and non-spatial properties of the objects. Queries issued by context-aware applications thereby typically involve some kind of spatial condition: *"Which is the closest non-smoking cab that is currently available?"* Hence, they profit from an efficient spatial access path as well.

The major non-functional requirements are *efficiency, scalability, and openness*. *Efficiency* demands high throughput of position updates and queries as well as short answering times of individual queries. Our experiences with the Nexus Location Service have shown that a single server is able to meet this requirement for at most 30,000 objects (in typical usage scenarios). Thus, a large-scale system inevitably comprises many servers that need to cooperate somehow. *Scalability* requires that the system can easily accommodate an increasing number of users by adding new servers. *Openness* demands that the system has well-defined interfaces so that external mobile object management systems of different providers can be integrated into the global system.

3 Basic Components

In this section we present the conceptual architecture of a large-scale system for managing a huge number of complex mobile objects. We have identified several basic components that are necessary to meet the presented requirements, as shown in Fig. 1.

To manage a very high number of complex mobile objects we distribute them to many independent *data servers*. Each one manages only a limited number of objects at a time. All of them must be capable of resolving arbitrary filter queries. Yet, their implementations may vary from traditional database servers to lean main memory solutions.

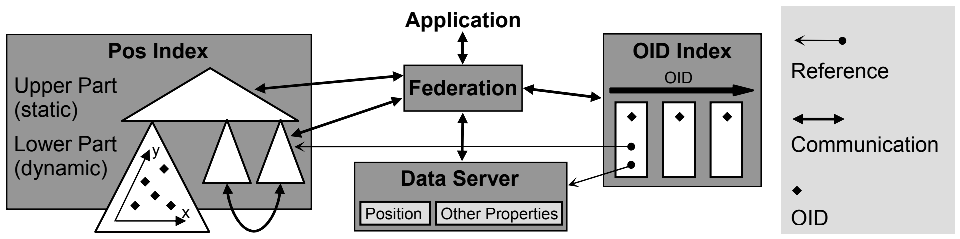

Fig. 1: Basic components of a large-scale system for managing mobile objects

In order to achieve scalability it is essential to minimize the number of servers accessed for a given query. Therefore, we introduce special index components to enable efficient retrieval of objects based on their OIDs and positions: the *OID-index* and the *pos-index*.

The *OID-index* supports an efficient processing of OID queries by keeping track of the physical storage location of each object. This actually incorporates more than one data server if the object's data is either partitioned or replicated. Most of all, the OID-index allows a seamless migration of mobile objects between different servers.

The *pos-index* is responsible for retrieving mobile objects based on their positions. For every spatial query it supplies the resulting OIDs. Because it has to cope with frequent position updates of many objects, the pos-index has a very dynamic nature and can maintain only the least possible amount of data: the OIDs and their current positions.

In a similar way, additional indexes on any property of the mobile objects can optimize arbitrary filter queries containing conditions on these properties. For example, a type-index supports retrieving only the cabs among all mobile objects.

Finally, the *federation* component serves as a single access point to the system and hides the complexity of the distributed components just introduced. It coordinates the communication between these components, forwards queries and updates to them, and integrates their results. Moreover, it is a good place to cache frequently accessed data.

Assume we are looking for the Mozart-loving violinists mentioned earlier. Within the presented architecture, this query will be processed as follows: First, the pos-index provides the OIDs of all objects within the extent of that banquet. The OID-index then supplies the relevant data servers. Finally, these servers are queried for objects having any of the provided OIDs and being a Mozart-loving violinist.

In order to achieve scalability of the overall architecture, it is essential that all components themselves are in turn scalable. This is accomplished by distributing them over a variable number of physical servers. The federation component can be duplicated without restraint because it offers a stateless service and does not store any data. The OID-space may be partitioned to distribute the management of the OID-index among several servers.

Likewise, the pos-index is distributed to several servers. Each server of the *lower pos-index* is responsible for a small partition of geographic space – its *service area*. It manages only the positions of mobile objects residing within this fixed service area. For maximum flexibility this division of space is neither regular nor perfect: service areas may overlap and have any shape. When a mobile object leaves a service area a handover to a different server has to be performed. For that reason, the OID-index also maintains a pointer to the current server of the lower pos-index for each object. Furthermore, this collection of index servers is complemented by an *upper pos-index*: it uses the service areas to determine the servers of the lower pos-index that possibly contain objects satisfying a given spatial query. As service areas are mostly static, this part of the pos-index can be replicated easily.

4 Discussion

The conceptual architecture proposed so far still contains a lot of room for further optimizations. Especially communication costs may be reduced significantly by placing more than one component on the same physical server, by shifting tasks between components, and by considering the network topology when placing the components. The impact of these adaptations is influenced by a number of factors.

First, the frequency of position updates depends on the speed and movement patterns of the mobile objects in combination with the required accuracy. Secondly, the range of an object's movement in relation to the size of service areas (which in turn depends on the number of objects to manage and their update frequency) determines its retention period on a single server of the pos-index (and thus the frequency of handovers). Thirdly, the size of data maintained per object determines the cost of handovers. Fourthly, the ratio of updates to queries, and the frequency and types of queries have to be considered, all of which depend mainly on the application domain. So do the selectivity of certain properties and the usefulness of additional indexes, which both affect the optimal processing sequence of arbitrary filter queries. Finally, queries and position updates are typically local, i.e. the issuer is within the queried area or at the updated position.

Considering these influencing factors, several architectural variants are conceivable. We introduce the most promising ones here, knowing well that there are many others.

In the first variant (left side of Fig. 2), we merge a server of the lower pos-index with a data server into a so called *context server*. Thus, it stores all parts of an object's context data and is able to process arbitrary filter queries efficiently. However, the cost of handovers increases significantly because all the objects' data has to be moved. In Nexus [Ni01], we currently use context servers to store stationary objects like buildings and roads. Storing mobile objects on context servers as well enables the federation to process queries for stationary and mobile objects in the same way.

Alternatively, we can combine a data server with a server of the OID-index into a single component called *home server* (right side of Fig. 2). This obliterates all lookups for the

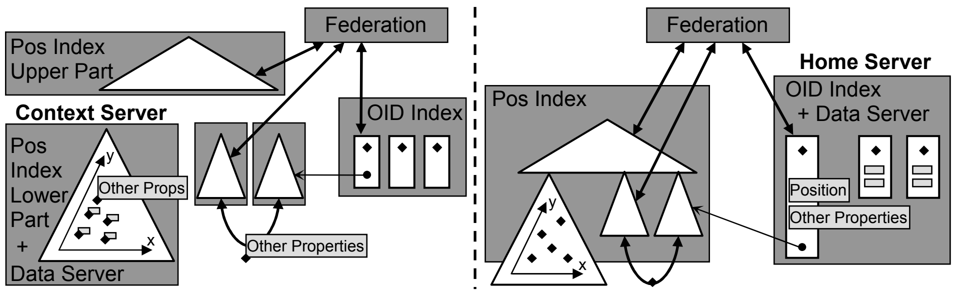

Fig. 2: Alternative system architectures

data server corresponding to a given OID, as the OID-index already knows all data of the searched object. However, arbitrary filter queries require extra effort. The federation has to get the OIDs of candidate objects from the pos-index before letting the home servers check the remaining conditions on these candidates.

For optimization, the objects' positions may be stored only within the lower pos-index servers. Then, position updates need to be reflected only at one component, but OID queries have to fetch the objects' positions from the lower pos-index. Arbitrary filter queries may be further sped up if the lower pos-index knows the corresponding data server for each object. Position updates and OID queries may also skip lookups at the OID-index, if the application remembers the lower pos-index or data server it used the last time. Additionally, applications may send position updates directly to the server of the lower pos-index, which may then take care of handovers itself.

5 Challenges

As we have discussed in this paper, taking care of "flying elephants" is a challenging task. Tracking many mobile objects requires a scalable system that is distributed over many servers. With regard to the distribution schemes, two conflicting goals exist: On the one hand, efficiency demands optimal data placement within the system. The optimum would be achieved if the data accessed by each individual query was always stored on a single server (close to the issuer of the query). Due to the objects' mobility, this data would therefore have to be moved between different servers accordingly. For resolving arbitrary filter queries the relevant data can thereby comprise all properties of the objects. On the other hand, minimizing the introduced communication overhead calls for transferring only the least possible amount of data. Ideally, no data would ever change its storage location. Clearly, both goals cannot be met simultaneously for a large number of complex mobile objects. Therefore, a suitable trade-off between these conflicting aspects is essential. It depends on the presented factors of influence.

In this paper, we proposed special, partitioned index components to support spatial and OID queries alike. We also discussed the most important approaches to structure the system's architecture, but still several more are well conceivable. Next, we intend to implement the most promising variants and analyze their behavior.

References

[Gü00] Güting et al.: A Foundation for Representing and Querying Moving Objects. In: ACM Trans. Database Syst., Vol. 25, No. 1, March 2000.

[LR02] Leonhardi, Rothermel: Architecture of a Large-scale Location Service. In: Proc. 22nd Int. Conf. on Distributed Computing Systems (ICDCS 2002), Vienna, Austria, July 2002.

[MD04] Mao, Douligeris: A Distributed Database Architecture for Global Roaming in Next-Generation Mobile Networks. In: IEEE/ACM Trans. Netw., Vol. 12, No. 1, Febr. 2004.

[Ni01] Nicklas et al.: A Model-Based, Open Architecture for Mobile, Spatially Aware Applications. In: Proc. 7th Int. Symp. on Advances in Spatial and Temporal Databases (SSTD 2001), Redondo Beach, USA, July 2001.

[Wo98] Wolfson et al.: Moving Objects Databases: Issues and Solutions. In: Proc. 10th Int. Conf. on Statistical and Scientific Database Management (SSDBM 1998), Capri, Italy, July 1998.