Odysseus: a High-Performance ORDBMS Tightly-Coupled with Spatial Database Features

Kyu-Young Whang, Jae-Gil Lee, Min-Soo Kim, Min-Jae Lee, and Ki-Hoon Lee
Department of Computer Science and Advanced Information Technology Research Center (AITrc)
Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea
{kywhang, jglee, mskim, mjlee, khlee}@mozart.kaist.ac.kr

Abstract

We have earlier proposed the tight-coupling architecture for adding new data types into the DBMS engine. In this paper, we introduce the Odysseus ORDBMS and present its tightly-coupled spatial database features. We demonstrate a geographical information system (GIS) implemented using Odysseus.

1 Introduction

Conventional ORDBMS vendors provide extension mechanisms for adding new data types and operations to their own DBMSs. Examples are Cartridge for Oracle and Extender for IBM DB2. In these products, new data types are added by using user-defined types, their operations by using user-defined functions, and their indexes by using extensible indexing [5]. Here, user-defined types, functions, and extensible indexing are implemented through the high-level (typically, SQL-level) interface provided by the DBMS [1, 5]. We call this mechanism *loose-coupling*.

In the loose-coupling architecture, the high-level interface causes the following problems. First, inter-process communication overhead is incurred because operations on new data types are performed outside the core DBMS engine. Second, concurrency control and recovery in fine granularity are hard to perform because low-level functions of the DBMS engine cannot be fully utilized for new data types through the high-level interface [5].

We have earlier proposed the tight-coupling architecture [8, 9] to solve these problems. In the *tight-coupling* architecture, new data types and operations are implemented directly into the core of the DBMS engine (i.e., the storage system). Hence, the problems above do not occur in the tight-coupling architecture. This tight-coupling architecture is being used to incorporate IR and spatial database features into the Odysseus ORDBMS [9][1] that has been under development at KAIST/AITrc for 16 years. Whang et al. [9] have introduced the tightly-coupled IR features of Odysseus. [2]

In this paper, we present the tightly-coupled spatial database features of Odysseus. Then, we demonstrate excellence of the tightly-coupled spatial database features through a geographical information system (GIS) implemented using Odysseus.

[1] The Odysseus ORDBMS consists of approximately 450,000 lines of C and C++ precision codes.

[2] This work received the **Best Demonstration Award** from IEEE ICDE 2005.

2 Tight-Coupling Architecture

The storage system of the Odysseus DBMS
Odysseus has a storage system *Odysseus/COSMOS*, which is a sub-system that stores and manages objects in the database. Odysseus/COSMOS uses the Multi-Level Grid File (MLGF) [6, 7] for spatial indexing and supports fine or coarse granularity concurrency control and recovery on spatial data. Odysseus/COSMOS contains the *extensible type layer* for tight-coupling.

The Extensible Type Layer
We propose to employ the notion of the extensible type layer to facilitate tight-coupling. We define the *extensible type layer* as the layer that provides new data types, their operators, and their indexes at the level of the storage system. Spatial database features are currently implemented in the extensible type layer.

The distinct characteristic of tight-coupling is that the extensible type layer is located *inside the storage system*. Due to this characteristic, new data types are supported directly from the storage system in the same way as built-in types are.

The Pseudo (Built-in) Type
We call a data type defined in the extensible type layer as the *pseudo built-in type* (simply, the *pseudo type*). The reason for including the term "pseudo" is as follows: although it is not a *real* built-in type, the pseudo built-in type has the performance benefits equivalent to that of a built-in type. In general, built-in types mean those specified in the SQL3 standard, e.g., *int* and *varchar*.

We define that a data type has performance benefits *equivalent* to those of a built-in type if using the data type does not require accessing the database catalog. Using pseudo types does not require accessing the database catalog because the information of pseudo types is hard-coded in the extensible type layer.

3 Tightly-Coupled Spatial Database Features

In Odysseus, users can specify the database schema using the spatial types and the MLGF index just in the same way as using nonspatial types and their indexes. Figure 1 shows the physical structure of a data record abiding by a schema involving the Point type, an MLGF index, the integer type, and a B^+-tree index. The Point type is specified just in the same way as the integer type is specified. Likewise, an MLGF index is specified in the same way as a B^+-tree index is.

1-4244-0803-2/07/$20.00 ©2007 IEEE

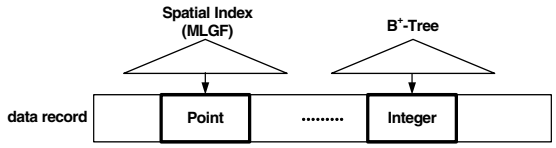

Figure 1. The structure of a data record involving the Point type and an MLGF index.

3.1 Spatial Types and Operators

The spatial types and operators supported conform to the OpenGIS [3] standard. The spatial types are implemented as *pseudo types* in the extensible type layer. The spatial operators supported are classified into three categories. First, *relational operators* return true or false depending on whether a specified topological spatial relationship exists between two spatial objects; they include `Equals`, `Disjoint`, `Intersects`, `Crosses`, `Overlaps`, `Touches`, `Within`, and `Contains`. Second, *geometric operators* return a geometric measure of a spatial object or between two spatial objects; they include `Area`, `Length`, and `Distance`. Third, miscellaneous operators include `Intersection`, `Union`, `Difference`, and `SymDifference`.

3.2 Spatial Index (MLGF)

The MLGF is a balanced tree consisting of a multilevel directory and data pages [6, 7]. A directory entry consists of a region vector and a pointer to a data page or a lower-level directory page. A region vector in an n-dimensional file consists of n hash values that uniquely identify the region. The MLGF uses an order-preserving hashing function to map attribute values to four-byte signed integers. The i-th hash value of the region vector is the common prefix of the hash values for the i-th attribute of all the records that belong to the region.

3.3 Spatial Query Processing

Region Query

A *region query* finds all objects satisfying a given spatial relationship (e.g., Intersects and Within) with a query region. Odysseus processes region queries by using the MLGF.

Transform-Based Spatial Join

Spatial join finds all pairs of objects satisfying a given spatial relationship (e.g., Intersects and Within) between two sets of spatial objects [4]. Odysseus uses the transform-based spatial join (TBSJ) algorithm proposed by Song et al. [4]. The TBSJ algorithm transforms spatial objects with extents into points without extents using corner transformation [2, 4], and then, performs spatial join. *Corner transformation* transforms a spatial object in the n-dimensional original space (o-space) into a point in the $2n$-dimensional transform space (t-space).

Nearest Neighbor Search

Nearest neighbor search finds the spatial object nearest to a given query point. The nearest neighbor search algorithm implemented in Odysseus is simple, consisting of two steps. In the first step, the algorithm visits the data page containing a query point, and then, selects a *candidate object* whose z-order value is closest to that of the query point. In the second step, the algorithm executes a region query over the region where spatial objects nearer to the query point than the candidate object may exist. This region is a circle whose center is the query point, and whose radius is the distance between the query point and the candidate object. If spatial objects are found in this region, the nearest object among them is selected as the nearest neighbor. Otherwise, the candidate object is guaranteed to be the nearest neighbor.

4 Demonstration

We demonstrate a geographical information system (GIS) implemented using Odysseus. We store approximately 850,000 spatial objects in the database; among them, 250,000 objects represent buildings in Seoul, and 600,000 objects the center lines of roads in Seoul. Our demo system runs on a PC with 2.5 GHz CPU and 1 GB of main memory.

Our demo system supports the following four kinds of queries: (1) finding the k buildings nearest to a point (*k-nearest neighbor search*), (2) finding the nearest neighbor of every point on a path (*continuous nearest neighbor search*), (3)finding the districts overlapping with a subway line (*spatial join*), and (4) finding the shortest path between two points (*shortest path search*). Our demo system displays results very fast (in a fraction of a second) for all queries. These results demonstrate excellence of the tightly-coupled spatial database features of Odysseus.

Acknowledgement

This work was supported by the Ministry of Science and Technology (MOST) / Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc) and by BK 21 Project, the school of information technology, KAIST in 2006.

References

[1] Banerjee, S., Krishnamurthy, V., and Murthy, R., All Your Data: The Oracle Extensibility Architecture, White Paper, Oracle Corp., Redwood Shores, California, Feb. 1999.

[2] Lee, M., Whang, K., Han, W., and Song, I., "Transform-Space View: Performing Spatial Join in the Transform Space Using Original-Space Indexes," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 18, No. 2, pp. 1–16, Feb. 2006.

[3] Open GIS Consortium, Inc., OpenGIS Simple Features Specification For SQL, Rev. 1.1, OpenGIS Project Document 99-049, May 1999.

[4] Song, J., Whang, K., Lee, Y., and Kim, S., "Spatial Join Processing Using Corner Transformation," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 11, No. 4, pp. 688–698, Aug. 1999.

[5] Srinivasan, J., Murthy, R., Sundara, S., Agarwal, N., and DeFazio, S., "Extensible Indexing: A Framework for Integrating Domain-Specific Indexing Schemes into Oracle8i," In *Proc. 16th Int'l Conf. on Data Engineering*, San Diego, California, pp. 91–100, Feb./Mar. 2000.

[6] Whang, K. and Krishnamurthy, R., Multilevel Grid Files, IBM Research Report RC11516, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, Nov. 1985.

[7] Whang, K., Kim, S., and Wiederhold, G., "Dynamic Maintenance of Data Distribution for Selectivity Estimation," *The VLDB Journal*, Vol. 3, No. 1, pp. 29–51, Jan. 1994.

[8] Whang, K., Park, B., Han, W., and Lee, Y., An Inverted Index Storage Structure Using Subindexes and Large Objects for Tight Coupling of Information Retrieval with Database Management Systems, U.S. Patent No. 6,349,308, Feb. 19, 2002, Appl. No. 09/250,487, Feb. 15, 1999.

[9] Whang, K., Lee, M., Lee, J., Kim, M., and Han, W., "Odysseus: a High-Performance ORDBMS Tightly-Coupled with IR Features," In *Proc. 21st Int'l Conf. on Data Engineering*, Tokyo, Japan, pp. 1104–1105, Apr. 2005.