データベースシステム 113－12
（１９９７．７．１４）

出世魚の６４ビット化とその上でのセコイヤ２０００ベンチマークデータベース

王 波濤，牧之内 顕文，金子 邦彦
九州大学大学院システム情報科学研究科

本発表では，６４ビット計算機の特徴と，その特徴を生かしたINADA/ODMGの６４ビット化について報告する．INADAはメモリマップドファイル機能に基づくデータベースシステムである．従って，３２ビット計算機版では一度に扱うのデタベス量に限界がある．６４ビット化によりこの制限はなくなる．又，セコイヤ２０００ベンチマークの実装についても報告する．

64 Bit Shusse-Uo and Sequoia 2000 Benchmark Database on It

Botao Wang, Akifumi Makinouchi, Kunihiko Kaneko
Gradudate School of Information Science and Electrical Engineering, Kyushu University

In this paper, we discuss properties of 64 bit environment and the redesign of our 32 bit INADA/ODMG for 64 bit environment. INADA is a database system based on memory mapped files. INADA 32 bit version has the limitation of maximum data size. In 64 bit INADA, a very large database in disks can be mapped onto a large size virtual memory, practically without considering the size limitation. At the same time, we give our design of implementation on Sequoia 2000 benchmark on the extended INADA.

64 Bit Shusse-Uo and Sequoia 2000 Benchmark Database on It

Botao Wang, Akifumi Makinouchi, Kunihiko Kaneko
Department of Intelligent Systems
Graduate School of Information Science and Electrical Engineering
Kyushu University, Fukuoka, Japan 812-81
botaow@db.is.kyushu-u.ac.jp, {akifumi, kaneko}@is.kyushu-u.ac.jp

Abstract

In this paper, we discuss properties of 64 bit environment and redesign of our 32 bit INADA/ODMG for 64 bit environment. The new implementation is introduced based on the 64 bit properties. In the new design, a very large database in disk can be mapped onto a large size virtual memory without considering the size limitation. High query performance and storage capability are expected in the new INADA/ODMG. While designing the implementation of Sequoia 2000 benchmark, we think OQL is flexible to handle mass and complicated data.

1 Introduction

Since the computer applications become more complicated, and the users become more sophisticated based on 32 bit environment, the move to 64 bit environment is inevitable. Many computer companies such as, DEC, SGI, IBM and Sun, announced their timetables for 64 bit operating system. The gains from 64 bit computing environment can be summed up in three aspects: performance, precision, and capacity[1]. The most important thing for database management is the ability to address Very Large Memory(VLM64) and very large file system[2]. These aspects should be reconsidered and would provide a new solution for computing in the applications which manipulate multimedia, scientific, and data warehouse databases. They are usually huge in size and complicated in operations.

Now, DEC Digital UNIX is the world's leading 64 bit UNIX operating system. Informix, Oracle, and Sybase are combining the power of their VLM64 database solutions with the power of Digital's 64 bit AlphaServer system[3]. For the 64 bit architecture, **LP64** model is chosen as the solution by Open System community considering the portability, and interoperability with 32 bit environment[4].

In this paper, we present our design of extending the object management system INADA/ODMG([5][6]) based on benefits of 64 bit system - VLM64 and very large file systems. What we want to do is to extend database space with improvement of system performance.

The remainder of the paper is organized as follows. Section 2 describes the background. The extension is presented in Section 3. Section 4 introduces the design of implementing Sequoia 2000 benchmark[9] using our INADA/ODMG. Section 5 contains related systems. Finally section 6 concludes the paper and describes the future work.

2 Background for Extension

2.1 Issues

The Sequoia 2000 Project([7][8][9]) explores the application of emerging database, network, storage, and visualization technologies to earth science problems, resulting in a "database-centric" metaphor for scientific computation. The Sequoia 2000 Storage Benchmark[9] uses real data sets and defines 11 queries. It is designed to represent the needs of engineering and scientific computing. In this paper, we consider two issues related to this benchmark:

- One is that the data size is huge. In Sequoia 2000 benchmark, the regional benchmark is about 1GB bytes, the national benchmark data is about 18 GBytes and the globe benchmark is multiple terabytes.
- Another is how to implement the special operators such as POSTGRES defined build_in operators "||", " < | >" and" * &*" for geographic spatial operations which are used in the queries[9]. The data processing functions like **clip** and **lowerRes** are defined too. Such special operations can not be avoided in engineering and scientific computing. The semantics of operations are different in different applications, and new operations will be required from application to application.

As mentioned in [9], above issues are not limited to geographic information system. Multimedia databases and data warehouse databases face the same issues. The implementation of Sequoia 2000 benchmark is meaningful for all such kind of applications which deal with mass and complicated data.

2.2 INADA/ODMG

INADA offers an application platform for building databases with a database language for

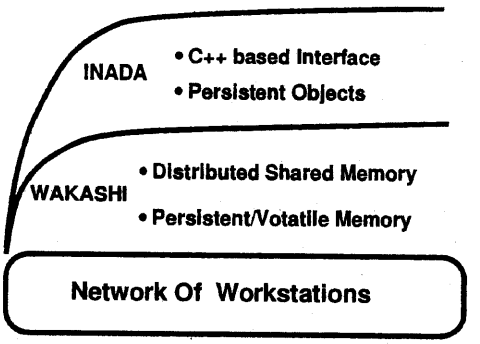

Figure 1: Architecture of Shusse-Uo

object management. It is a part of our ongoing project "**Shusse-Uo**". The object database programming language provides the facilities to manage persistent objects with C++ based interface. "**WAKASHI**"[10] is used as distributed object storage manager which is independent of data models. ODMG93 C++ binding interface[11] is integrated into INADA and it is called INADA/ODMG. The figure1 shows the architecture of **Shusse-Uo** .

For the management of persistent heap where persistent objects are allocated, INADA uses UNIX **mmap()** function to map files to the virtual memory. The current 32 bit INADA shows some incapabilities to deal with files of huge size, because of 2 GBytes memory size limitation. The largest virtual memory available to the users is 2 GBytes. This means that very large database such as Sequoia 2000 are very difficult, if not impossible with distribution of database, to be handled by the current 32 bit INADA.

The target of our extension is to make good use of the gains from the 64 bit architecture, to build 64 bit INADA/ODMG to overcome the above mentioned issues based on our current 32 bit INADA/ODMG, and to realize the Sequoia 2000 benchmark on it so as to show the performance of INADA in running such a large and complicated database application.

3 Extending INADA Address Space

3.1 Adding Data Types

In **LP64** (known as 4/8/8) model, the size of basic data type **long** and **pointer** is 64 bit. In current **ILP32** (known as 4/4/4) model which is the predominating data model available with UNIX systems that execute on 32 bit computer architecture, the size of basic data type **long** and **pointer** is 32 bit. In ODMG93 [11], only **Long** and **Unsigned Long** are defined. There is no **Integer** type. The variables with **Long** and **Unsigned Long** become 64 bit long while being compiled with **LP64** compiler. It's problematic while dealing with 64 applications because integer type and long-integer type have different bit size, 32 and 64 in **LP64**, and only long-integer type is defined currently. For some applications, 64 bit integer is too long.

Table 1: Bit Size Comparison

	Pointer	Long Unsigned Long	Int Unsigned Int	Short
INADA/ 64	64	64	32	16
ODMG93 for 64	none	64	none	16
INADA/ 32	32	32	none	16
ODMG 93	none	32	none	16
LP64	64	64	32	16
ILP32	32	32	32	16

We think the new data type, *integer type*, should be added into ODMG93 [11], which is defined using keywords **Int** and **Unsigned Int** for 32 bit size variables. The width of long-integer type. **Long** and **Unsigned Long** should be redefined from 32 bit to 64 bit. Without this extension, the applications based on ODMG93 are limited, because long-integer type can not be 32 bit and 64 bit at same time. The table 1 shows the different size definitions about the data types in INADA/64, ODMG93 for 64 (current version with LP64 compiler), INADA/32, ODMG93, LP64 and ILP32.

3.2 Using 64 Bit Address Space

Address space becomes vast on the 64 bit processor. Besides directly addressing much large memory, the file sizes become much more huge than current 2 GBytes limitation. So the GBytes of data, such as multimedia data or data warehouse data, need not to be stored in different files any more and all the data can be accessed via a single file if necessary. The file can be mapped on the virtual memory using the **mmap()** function. With this, the record I/O which would be required in handling the traditional files is expected to be improved too, which can help the performance of DB. It means it becomes possible that data operations can perform in the memory(i.e. virtual memory) with the whole data set such as data or indexes in the memory address space. The larger the main memory is, the smaller the number of data swapping is. So the number of I/O operations will be reduced benefited from the 64 bit very large memory. And 64 bit computers are

usually provided with much more physical memory than 32 bit computers.

The INADA/ODMG is built on **WAKASHI** which supports persistent virtual and distributed shared memory on Network Of Workstations (NOW). The basic storage unit is heap. It is the unit used in **mmap()** function. One heap is one clustering unit too. Logically, the heap offers storage space for objects practically without size limitation. In the 32 bit implementation, one heap file can not exceed 2 GBytes. So the DBMS system, if the size of DB is larger than that size, has to manage multi heaps distributed on multi sites.

In the extended design based on 64 bit architecture, the size of one heap can be much larger than 2 GBytes, which allows DBMSs to get a file mapped on the very large virtual memory. Based on this property, the management method is designed as follows. Simply, one database file is defined as one heap. A very large database may have different types of data. At least, it usually has a dataset and indexes which accelerate access to the data. One heap is dedicated to data of a similar characteristic. A very large database may be composed of many heaps. In 64 bit environment, it becomes possible to map all these database files on the virtual memory while opening DB.

The mass data are saved with the same image as its memory binary image. They do not need to be put on different location in different format. The user can manipulate all the data in the same view as that in the main memory. In the implementation of Sequoia 2000 benchmark, one index file is stored in one heap. For example. for **Point** data, there are one data heap file, one Btree index heap file and one Rtree index heap file. For **Polygon** data, there are one data heap file, one Btree index heap file and one Rtree heap file.

3.3 Extension of OID

Besides the above changes the OID structure is redesigned too. OID represents the identity of an object. In INADA/ODMG, OID is composed of two parts, Heap Identifier(h_id) and Object Reference Table Entry No. (ort_id). H_id is used to identify the heap file where the object that OID referees to exists and ort_id is used to identify the object inside the heap. The size of h_id is changed from 8 bit to 16 bit, and the size of ort_id is changed from 24 bit to 48 bit. Figure 2 show the structure of OID and its changes. The maximum heap number is changed from 2^8(256) to 2^{16}(65,536). The maximum object entry number is changed from 2^{24}(16,777,216) to 2^{48}(281,474,976,710,656). Apparently the storage capability is improved greatly by this change in 64 bit INADA/ODMG.

4 Benchmark Design

4.1 Schema

The implementation of Sequoia 2000 benchmark is designed in INADA/ODMG in order to

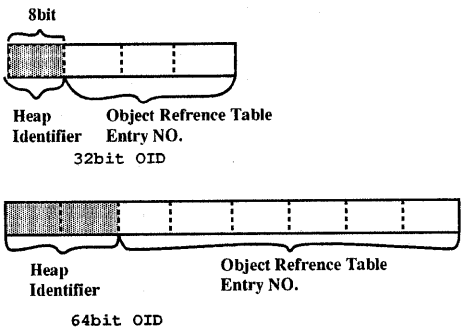

Figure 2: INADA/ODMG 32 bit OID and 64 bit OID

test the performance. Based on the object oriented design method, the schema classes **Point, Raster, Polygon, Graph** and additional classes **BTree, RTree** are defined. The instance objects of a type are stored in a set. There are four collection types, **Set, Bag, List** and **Array**. In ODMG93, the **extent** of a type is the set of all instances of the type within a particular database. For example the **extent of Polygon** is defined in INADA/ODMG C++ binding **d_Set** as following:

```
d_Set<d_Ref<Polygon>> POLYGONS;
```

According to ODMG93, OQL is a superclass of SQL. It allows method invocations with or without parameters wherever the result type of the method match the expected type in the query. With the methods the complicated queries can be performed easily and directly in OQL without system extension, especially in the case of the application like GIS or MMDB which usually requires special operations for data processing as well as the queries.

For Sequoia 2000 benchmark, the data are complicated. The raster image data are large in size. The data size of polygon and graph are variable. At the same time, there are special operations on spatial relationships and results. For example, spatial operation between **Raster** and **Polygon** " polygon intersects rectangle" is defined in POSTGRES as operator "||" [9]. On the retrieved objects, there are operations such as **clip, lowerRes**, which clips raster image by shape and lowers the resolution of the raster image, respectively. OQL shows great flexibility to deal with these kinds of complicated queries. In the followings, we list **Raster** and **Polygon** schema classes defined by INADA/ODMG C++ binding:

The definition of **Raster** is shown as followings:

```
class RasterBase:d_Object{
// RasterBase definition
public:
```

```
// Attributes
  Boxdata          data;
// Boxdata is a literal structure
// describing box data.
};

class Raster:RasterBase{
// RASTER definition
enum { DataSize = 2048 };
//8MB for regional data
public:
// Attributes
d_Int       time;
d_Int       band;
// wavelength to get data.
d_Short     image[DataSize][DataSize];
// Raster image data

// Constructor and Destructor
Raster();
Raster(Boxdata Location, d_Int time, d_Int
 band, d_Short  image[DataSize][DataSize]);
~Raster();

// Operation for queries
Raster clip(Boxdata box);
// clip operation for Raster.
Raster  lowerRes(d_Int size);
// do resolution reduction on Raster.
d_Double raster-avg();
//calculates the average value
//of all pixels
};
```

In the above definitions, class **RasterBase** defines the primitive geometric attributes. Class **Raster** inherits these attributes and adds its own attributes for raster image, such as, *time, band* and *image*. The image data is defined as 2D short integer array with the size of 8 MBytes [9]. The methods for raster image processing used in the queries are defined too. They are **clip, lowerRes** and **raster-avg.**

The definition of **Polygon** is defined as follows:

```
struct Polygondata{
d_Varray<Pointdata> data;
// Polygon data, variable length array
// Point data is a literal structure
// describing point data.
};

PolygonBase: d_Object{
// PolygonBase definition
public:
// Attributes
  Polygondata  data;

// Method for Rtree Creating and
// Searching
Boxdata  getBoundingbox();
// gets its boundingbox
d_UInt size();
// gets polygon size.
};

class Polygon:PolygonBase{
// POLYGON definition
public:
// Attributes
d_Int      landuse;

// Constructor and Destructor
Polygon();
Polygon(d_Int landuse, Polygondata location);
~Polygon();


// Method for queries
d_Boolean interactBox(Boxdata rectangle);
//represent operator ''||'' in query 6,8
d_Boolean insideCircle(Circledata circle);
//represent operator ''<|>'' in query 7
d_Boolean includePoint(Pointdata point);
//represent  operator ''||'' in query 10
};
```

In the above polygon definitions, **PolygonBase** defines the primitive geometric attributes too. Class **Polygon** inherits these attributes and adds its own attributes for query operations too. Polygon data is defined as variable length array by **d_Varray.** In class **PolygonBase**, the methods for creating and searching Rtree are defined. Class **Polygon** is a subclass of **PolygonBase** and a new attribute *landuse* is added. Further, the spatial predicates , such as **interactBox, insiderCircle** and **includePoint** are defined. They represent Postgres build_in operators, "||" and " < | >". For the detail of each query, please referee to [9].

The relationship of the schema classes is shown in figure3. Based on the the ODMG object model, the system is easy to extend to other applications. The reason we define two level classes is that we want to distinguish the geometric attributes which are not dependent on applications from the other attributes which are dependent on applications. For example, while creating Rtree indexes, only geometric attributes are used. The raster image data and polygon landuse data are the data used in GIS queries. It is not necessary to include these data while creating Rtree indexes.

4.2 Queries

Based on the above definitions of **Raster** and **Polygon**, the benchmark query 3 written in POSTGRES

```
retrieve (raster-avg{
  clip(RASTER.data), RECTANGLE})
where RASTER.time = TIME
```

which selects AVHRR data for a given time and geographic rectangle, and then calculates an arithmetic function of the five wavelength band values for each cell in the rectangle to be studied can be written in OQL as follows:

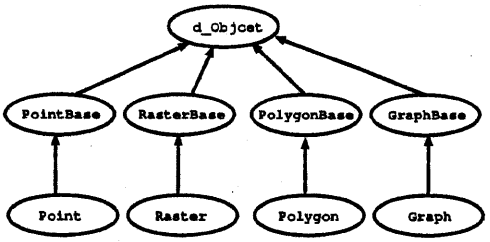

Figure 3: Classes Relationship

```
select (r.clip(RECTANGLE)).raster-avg
from RASTERS r
where r.time = TIME
```

Since we have not yet developed INADA/ODMG OQL on our platform, the query is directly implemented using ODL C++ binding and OML C++ binding. The procedure is shown as the follows:
1)Find all rasters which meet the given condition using *time* **Btree** index.
2)Call the **Raster.clip** method for step 1) result.
3)Call the **Raster.raster-avg** method for step 2) result.
The benchmark query 6 written in POSTGRES

```
retrieve into FOO-2(POLYGON.all)
where POLYGON.location||RECTANGLE
```

finds all the polygons that intersect a specific rectangle and store them in database. Operator "||" means 'polygon intersect rectangle'. The query can be written in OQL as follows:

```
select polygon
from polygon in POLYGONS
where polygon.interactBox(RECTANGLE)
```

The procedure using OML C++ binding is shown as follows:
1)Find all polygons whose bounding box interacts with RECTANGLE using polygon **Rtree** index.
2)Check step 1) result by **Polygon.InteractBox** method, and get the polygons which interact with RECTANGLE.

Compared the queries based on relational model or extended relational model, the queries based on OQL describes the operation in a more natural and clear way, because the methods defined in the classes can be called in the queries.

5 Related Systems

The systems related to Sequoia 2000 storage benchmark have two sorts. One is geographic information oriented systems, such as **ARC-INFO**, **GRASS** and **IPW**, which are introduced in [9]. Another one is the extension of existing systems, **Paradise** ([12][13][14]) and **Monet**([15][16]) are some examples. Because the formers are geographic information application systems, which differ from INADA, an object database for general data extensive applications. The following comparisons focus on the later two systems because they are database systems on which sequoia 2000 are built.

Paradise is a database system aimed at handling GIS applications and provides an extended-relational data model for modeling GIS application. The build_in data types for GIS management are provided. The spatial data types provide a set of spatial operators that can be accessed from an extended version of SQL. All the types are implemented based on Abstract Data Types (ADT).

Monet is a type- and algebra-extensible data system based on Binary Association Tables(BAT). It is table-oriented. For user-defined types, ADT facility is provided too. Besides allowing addition of new datatypes and operators like **Paradise**, it further allows for addition of new relation operators. Its implementation strategy is "extensible-toolkit" approach. The support for new data types and accelerators is implemented through ADT interfaces and the C compliant object-code are linked by **Monet** utility for new relation operators.

Paradise uses extended build-in datatypes. Its capabilities of data processing is limited to its build_in datatypes. **Monet** has good capability for extension on new datatypes and relation operations. To extend the operators, **Monet** system utility is used and the operators are added using dynamic linkage of C object-file or library. At implementation level, the operator is a kind of module, it is not internal part of the data.

In our implementation the database design is based on ODMG object database model. The data and operations on it, are defined together. This is very flexible to handle above complicated data.

For very large arrays, **Paradise** chunks arrays into subarrays, and stores them in a file different from other data. The data compression is used for performance enhancement. The **Monet** performs all operations in main memory based on unix **mmap()**, on which INADA/ODMG is based too. **Monet** puts fixed-sized type and variable-size type data in different storage location. In INADA/ODMG, fixed-sized type and variable-size type data are put together. This is allows clustering the data.

6 Conclusion and Future Work

In this paper, we discussed properties of 64 bit environment and redesign of our 32 bit INADA/ODMG for 64 bit environment. The extension provides very large space for database system, and high performance operations can be expected based on our design. The design has same interface as that of ODMG C++ binding. The size of build in data type in ODMG93 are upgraded and extended.

We discussed also an implementation of Sequoia 2000 benchmark on INADA/ODMG extension for 64 bit address space. The memory address limitations derived from 32 bit cpu architecture are eliminated by its extension and can run the benchmark. The database is built using ODMG object database methodology.

Now we are implementing the 64 bit INADA/ODMG and Sequoia 2000 benchmark and hope that the performance result will be reported soon.

References

[1] Solaris Products White Papers: 64-bit Computing in Solaris, 1996, December. //http: www. sun. com/ sunsoft/ solaris/ whitepaper/ solaris64.html

[2] Digital Announces New Release of Industry's Leading 64-bit Unix Operating System, 1996, March. http://www. digital.com/ infor/ PR00HG

[3] Digital Opens Database Technology Center Supporting Transition to 64-Bit, Very Large Memory (VLM64) Computing. 1995, December. http://www. europe. digital. com/ infor/ PR00F7

[4] 64-bit
Computing Today, http:// decwww. epfl. ch/ connections/ cdrom /html/dec_unix/ 64bit/ index.html

[5] Kan Yamamoto, Multimedia Data Storage for the Object-Oriented Persistent Programming Language INADA, 1997, February. Master thesis, the Department of Intelligent Systems, Kyushu University, Japan

[6] Yamamoto, Kan, et.al, "Integrating ODMG interface into the object database SHUSSEUO", Information Processing Society of Japan, the 53th national convention, 1996, Japan

[7] Allison Woodruff, Sequoia 2000 slide show, 1992. http://s2k-ftp CS.Berkeley. EDU:8000/ sequoia/demo/

[8] Micahael Stonebraker, et.al., "THE SEQUOIA 2000 ARCHITECTURE AND IMPLEMENTATIONS STRATEGY", Sequoia 2000 Technical Report,93/23. 1993, University of California, Berkeley, CA 94720

[9] Stonebreaker, M. et.al., "The SEQUOIA 2000 Benchmark", Proc. of 93 ACM SIGMOD Conference on Management of Data, 1993, Washington

[10] G.Yu, K.Kaneko, G.Bai, and A. Makinouchi, "Transaction Management for a Distributed Object Storage System WAKASHI-Design, Implementation and Performance", Proc. of ICDE 96, New Orleans, Louisiana, USA.

[11] R.G.G. Cattell, et.al., "The Object Database Standard:ODMG-93 Rel 1.2", Morgan Kaufmann Publishers, San Francisco, California Inc. 1996

[12] Computer Sciences Department University of Wisconsin, Madison, Paradise Version 0.1 Reference Manual. 1994, January. file:// ftp. cs. wisc. edu/ paradise/ papers,

[13] David J. DeWitt, Navin Kabra, et.al., "Client-Server Paradise", Proc. of VLDB 94

[14] Jignesh Patel, JieBing Yu, et.al. , "Building a Scalable Geo-Spatial DBMS:Technology, Implementation, and Evaluation", Proc. of SIGMOD 97

[15] Peter A. Boncz, Martin L.Kersten. "Monet: An Impressionist Sketch of an advanced Database System", Proc. of BIWIT 95

[16] Perter A. Boncz, Wilko Quak, Martin L. Kersten, "Monet And Its Geographic Extensions", Proc. of EDBT 96