Design and Implementation of Extended Parallel Sequoia 2000 Benchmark

Wang, Botao
Department of Intelligent Systems, Graduate School of Information Science and Electrical Engineering, Kyushu University : Visiting Researcher

Jin, Taiyong
Department of Intelligent Systems, Kyushu University : Graduate Student

Tamura, Kenichi
Department of Intelligent Systems, Kyushu University : Graduate Student

Kimura, Kenichiro
Department of Intelligent Systems, Kyushu University : Graduate Student

他

https://doi.org/10.15017/1500429

出版情報：九州大学大学院システム情報科学紀要. 5 (1), pp.1-6, 2000-03-24. 九州大学大学院システム情報科学研究院
バージョン：
権利関係：

九州大学大学院
システム情報科学研究科報告
第5巻 第1号 平成12年3月

Research Reports on Information Science and
Electrical Engineering of Kyushu University
Vol.5, No.1, March 2000

Design and Implementation of Extended Parallel Sequoia 2000 Benchmark

Botao WANG*, Taiyong JIN**, Kenichi TAMURA**, Kenichiro KIMURA**,
Kunihiko KANEKO***and Akifumi MAKINOUCHI***

(Received December 10, 1999)

Abstract: Performance is a major issue in the acceptance of database, especially for the database holding massive data. It's natural to apply parallel technology to handle these large data sets. ShusseUo is an Object Database Management Systems(ODBMSs) and it provides persistent global object management on persistent Distributed Shared Virtual Memory (DSVM) distributed on Network Of Workstations (NOWs). This paper introduces the performance evaluation of ShusseUo using extended Sequoia 2000 benchmark. The goal is to test the scalability and speedup of ShusseUo while dealing with mass size of extended spatial benchmark data in parallel. Experiments show that good scalability and speedup can be gotten by ShusseUo.

Keywords: Distributed shared virtual memory, Parallel spatial query, Benchmark

1. Introduction

There is the same trend that applications such as scientific database, data warehouse and digital libraries, will generate and use massive data. It is natural to apply parallel technology to improve performance in such kinds of applications. ShusseUo is an ODBMS built on NOW which is a platform for parallel processing. Benchmark is the crucial tool on performance evaluation of database system. In this report, we will introduce our design and implementation of extended parallel Sequoia 2000 benchmark [6] on ShusseUo. The scalability and speedup of ShusseUo are tested based on the benchmark.

2. Background

2.1 ShusseUo

NOW is a set of workstations which are connected via high speed network. Originally NOW was proposed for the parallel computing applications. However, database system can take advantage of the merits of NOW [1]. Distributed Shared Virtual Memory (DSVM) of ShusseUo is a virtual shared memory space which consists of the memory space distributed on NOW. ShusseUo is an ODBMS. It is built using DSVM and currently consists of two layers: WAKASHI and INADA. WARASA is under development now. The **Fig.1** shows the layers of ShusseUo.

2.1.1 WAKASHI

WAKASHI[8] is a distributed persistent object storage system. It runs on the distributed UNIX platform. WAKASHI is peer-to-peer model. The main idea of WAKASHI is mapping: persistence is provided by memory-to-disk mapping and global data sharing is supported by DSVM mapping. In detail, memory-to-disk mapping is realized by UNIX **mmap()** system call and DSVM mapping is realized in WAKASHI server by means of OS's paging mechanism. After mapping, all the clients can deal with data in database locally by local disk caching. The heap is the basic storage unit. **Fig.2** shows the architecture of WAKASHI. WAKASHI consists of two parts: 1) servers that run as daemon processes, 2) a client library that is linked by user programs (client programs). The server performs data access control and transaction management. Page-level locking and cache coherence protocol are used. The client library provides communication interfaces between a client and the server on the a site.

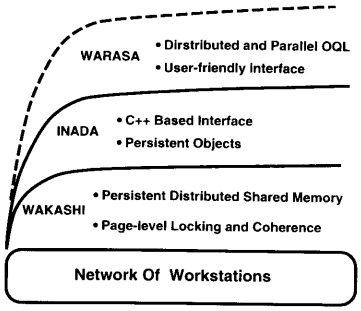

Fig.1 Layers of ShusseUo (WARASA is currently under development)

* Department of Intelligent Systems, Visiting Researcher
** Department of Intelligent Systems, Graduate Student
*** Department of Intelligent Systems

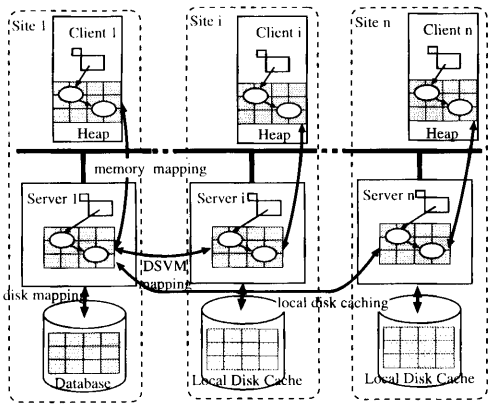

Fig.2 Architecture of WAKASHI

2.1.2 INADA

INADA offers an application platform for building object databases with a database language for object management. The language provides the facilities to manage persistent objects with C++ based interface. An INADA program runs as a WAKASHI's client and it uses the client library provided by WAKASHI. INADA provides users with a library that allows the users to directly manipulate global persistent objects (i.e., persistent objects shared by different clients on different sites). In detail, it provides facilities of manipulating collections and writing distributed parallel programs [7]. ODMG[2] C++ binding interface is integrated into INADA.

2.2 Sequoia 2000 Benchmark

The Sequoia 2000 Project [6] explores technologies to earth science problems. The Sequoia 2000 benchmark [6] uses real data sets and defines 11 queries. It is designed to represent the needs of engineering and scientific computing. The raster, point, polygon and graph types of data are defined and used in the benchmark queries.

3. Design for Parallel Processing

Basically, the main source of parallelism is partitioned parallelism[9)4)]. There are two main requirements for achieving effective partitioned parallelism. First, good data declustering techniques are required to evenly distribute the data across the nodes in the parallel system. It is related to data partition. Second, the operations must be designed such that the operations running at a particular node accesses only the data stored locally. It is related to the design of parallel query.

3.1 Data Partition

There are 3 basic partitioning techniques, round-robin partition, range partition and hash partition. As analyzed in 9)4), considering its best possible data load balance, the round-robin partitioning is selected in our implementation of benchmark. For N sites, round-robin algorithm distributes the kth object to site((k-1)mode N)+1. The storage unit heap is used as one partition unit. While building database, all kinds of data will be partitioned according to round-robin algorithm to different heaps and these heaps will be distributed to different sites for parallel queries.

3.2 Parallel Query

The parallel query here is based on round-robin partition. The architecture of parallel query execution is shown in **Fig.3**. There are N+1 sites. One site is named coordinate site, mainquery processes run there. This process coordinates the running of parallel queries. At the other N sites which are named subquery sites, the query commands sent from coordinate site are executed by subquery process there.

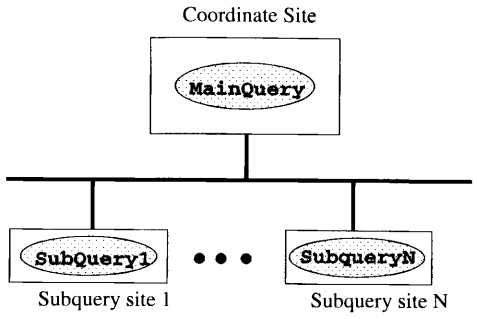

Fig.3 Architecture for parallel query

The parallel query executes in the four phases shown in **Fig.4**:

- Phase1
  The mainquery process creates a number of threads[†1] in the coordinate site. Each of the threads corresponds to one of the subquery

†1 Threads are "light weight processes" (LWPs). The idea is that a process has five fundamental parts: code ("text"), data (VM), stack, file I/O, and signal tables. "Heavy-weight processes" (HWPs) have a significant amount of overhead when switching: all the tables have to be flushed from the processor for each task switch. Threads reduce overhead by sharing fundamental parts. By sharing these parts, switching happens much more frequently and efficiently.

sites. The mainquery process sends the query commands to the threads, then waits for the results from the threads.

- Phase2
  On the coordinate site, the threads created at phase1 send the query commands to the subquery sites where the subquery processes are started beforehand, and then the threads wait for the results from the subquery sites. The query commands and their inputs are same for all the subquery sites.

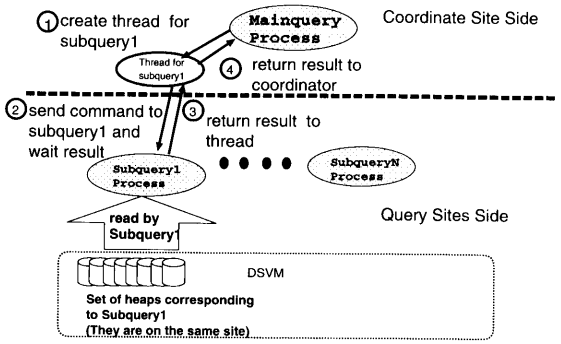

Fig.4 Procedure of query

- Phase3
  One each subquery site, the subquery process receives the command from the thread and executes the command locally. After the execution of query, the results are returned to its corresponding thread created by mainquery process on coordinate site.
- Phase4
  The threads receive the results from their corresponding subquery sites and transfer them to mainquery process. After collecting results from all threads, the mainquery process merges the results and returns the results to caller.

3.3 Parallel Queries Based on Spatial Types

Most queries over spatial types involve proximity rather than exact matching. The result objects are generally gotten by two steps: filtering and refinement. Filtering is the step based on Minimum Bounding Rectangle(MBR) of objects. The refinement is done based on the exact shape of objects. For the queries like spatial join and spatial recursive, the procedure shown in **Fig.4** will be looped more than one time.

Because the parallel queries are based on round robin partition, for each loop, the input should be broadcasted to all the subquery sites. Compared with numeral attribute data, the spatial attribute data(polygon and graph) become very large. For example, the polygon have average 50 sides[6] which means average 50 points data(the size of one point is 4Bytes) are used to represent its exact geometry shape, because the exact geometric shape is necessary for refinement. The size of query command sent by mainquery process become much larger. Considering the time of synchronization operations of mainquery and the time of data transferring, it's kind of overhead compared to traditional parallel computation.

4. Related Work

As far as we know, 5) is related to this topic. In 5), a parallel geospatial DBMS is built on PARADISE.

PARADISE[5)3)] is an Object-Relational Database Management System aiming at handling geographical applications and provides an extended-relational model for the applications. The built_in data types for spatial data management are provided. Its implementation is based on the Client-Server model.

ShusseUo is an ODMG-based ODBMS, which is not limited to special applications. The database design is based on ODMG object database model and data structures. Its implementation is based on peer-to-peer model.

In 5), the data are spatially partitioned across all nodes on the network statically by range. The advantage of range partition is that high performance can be expected for spatial join which do join operation based on spatial relationships(i.e. intersection, nearest). The spatial join algorithm used there is derived from parallel hash join[9] where spatial object can be grouped by their position(coordinates). But there are two problems which are critical for the parallel range portioning.

- One is that, when range partitioning non-point data (i.e., polygons or graph) is partitioned, the objects which span the partitions must be replicated in order to ensure that queries on the range attribute produce the correct result. For example, consider the two objects which are intersected each other in the real world. If they were put in different partitions where they span without replication, they can not be checked out by spatial join.
- Another problem is skew. If the number of partitions is small, skew becomes a major problem. As the number of partitions increases, the number of spatial objects spanning multiple parti-

tions will increase.

In our implementation, round robin partitioning is used and the query command is broadcasted, so the above two problems don't exist. The overhead in our implementation is that the spatial data representing their exact geometry shape used as input data must be broadcasted.

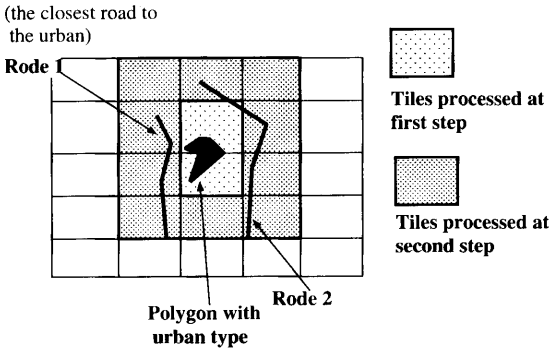

Fig.5 Example of spatial aggregation

Further, for spatial aggregation type queries, the parallel algorithm based on range partition can not guarantee to get result at one step. One example of such a query is 'find the closest road for each polygon with type of urban'. It's because the closest road may be in the adjoining partition as shown in **Fig.5**. At the same time, the copy of exact geometry shape must be sent to the sites where the adjoining partitions are located. Conversely, the algorithm based on round-robin can get result by one step.

5. Experimental Results

5.1 Data Scaleup of Sequoia 2000 Benchmark

The goal of this report is to test the scalability and speedup of ShusseUo based on benchmark queries. This requires the data set can provide data at different level of size. The source data of the benchmark are scaled up in our test. The "resolution scaleup[5]" is chosen in our experiment. It means that the region under consideration is kept constant while it is viewed at a higher resolution. It's introduced in 5). The primary idea is that when a user moves to a data set with a higher resolution, the existing spatial features become more detailed, and at the same time a number of smaller "satellite" features that hover around the existing feature become visible.

Table 1 and **Table 2** show the number and the size of extended(scaleuped) data set when scalability is tested on the case the number of sites is 3, 6, 12. The data set used for speedup test are the data set when the number of sites is 3.

Table 1 Number of data set after scaleup

Unit: Million

Sites	*3*	*6*	*12*
Point	0.25	0.50	1.00
Polygon	0.32	0.64	1.27
Graph	0.81	1.61	3.22

Table 2 Size of data set after scaleup

Unit: MB

Sites	*3*	*6*	*12*
Point	8.54	17.2	34.4
Polygon	255.4	509.8	1019.6
Graph	265.6	528.9	1057.8

5.2 Environment

For subquery sites, a cluster of 12 Ultra5 Sun workstations is used and each has 128 MB memory, 270 Mhz processor and 20GB disk for data caching. For coordinate site, an Ultra10 Sun workstation is used, which has 512MB memory and 440Mhz processor and 100GBs disk where the database is built. All these workstations are connected to 100M-bit Ethernet switch.

The OS is Solaris 7, all the query codes are implemented in C++ and the compiler is SUN Workshop C/C++ version 5.0.

5.3 Results

Mainly, we used queries related to range query, spatial join and spatial recursive to test scalability and speedup of ShusseUo †2 . The results are shown in **Table 3** and **Table 4**. For each query, it is executed 20 times continuously. The input of the first 10 times are created randomly and the input the second 10 times are the same as those of first 10 times correspondingly. The average response time of first 10 running is called warm result, and the average response time of second 10 running is called hot result †3.

†2 The queries related to raster is not used in our test.

†3 Cold, warm and hot is the term of database benchmark 001 and 007. The status of hot means all the data needed is in memory; the status of cold means that no data needed is in memory; the status of warm means part of data needed

Table 3 The Scalability result

Unit: seconds

Sites	3	6	12
Query	warm/hot	warm/hot	warm/hot
6	4.7/0.38	5.8/0.40	9.0/0.49
7	4.8/0.39	7.3/0.58	12.9/0.50
8	6.1/0.51	10.9/0.66	17.3/0.59
11	14.0/2.4	17.9/3.8	25.4/3.6

Table 4 The speedup result

Unit: seconds

Sites	3	6	12
Query	warm/hot	warm/hot	warm/hot
6	4.72/0.38	2.87/0.25	2.61/0.16
7	4.85/0.39	3.37/0.25	2.88/0.16
8	6.06/0.512	3.25/0.35	3.13/0.14
11	14.0/2.4	9.08/1.40	6.87/0.83

5.4 Analysis of Results

5.4.1 Range Query

The scalability and speedup of range query are shown in **Fig.6** and **Fig.7** It is result of query6.

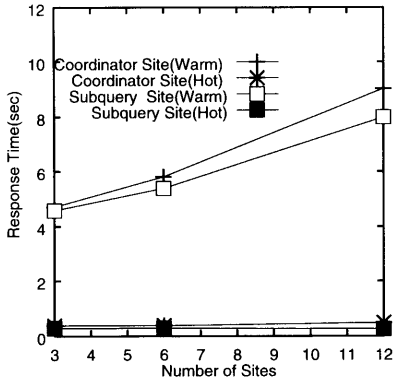

Fig.6 Scalability of range query

The scalability at warm status becomes worse when number of sites increases. The reason is that the database is put in the coordinate site. The data in database will be mapped(distributed) to the subquery sites after the query begins. The more the number of site is, the more the time will cost. But at hot status, the data used by query have been mapped to subquery sites, there are no such kind of data distribution, so the hot is ideal.

The speedup of both warm and hot results be-

is in memory. Here the warm result includes the cold result, because 1) the average time is used,2)the input is created randomly and the response time is depended on input.

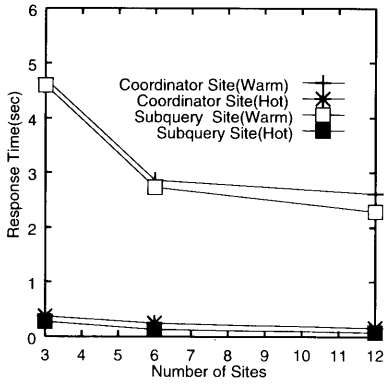

Fig.7 Speedup of range query

come slower when number of sites becomes bigger. The reason is that the cost for synchronization of subqueries and broadcasting spatial input data become larger.

The difference of time measured on coordinate site and subquery site becomes larger for the above reason too. But compared to the total response time, such kind of cost is smaller.

The result of query7 is range query too. From **Table 3** and **Table 4**, we can find that it has the same changing trend with that of query6.

5.4.2 Spatial Join

The scalability and speedup of spatial join are shown in **Fig.8** and **Fig.9** They are results of

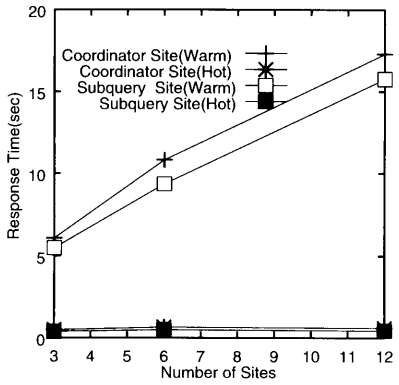

Fig.8 Scalability of spatial join

query8. The trend on scalability and speedup are same with that of range query.

5.4.3 Spatial Recursive

The scalability and speedup of spatial query are shown in **Fig.10** and **Fig.11** They are results of query11. The changing trends of scalability and speedup are same with that of range query. In this query, the times measured on coordinate site and subquery site are almost same, the reason is that multiple loops(analyzed in 3.3) are done in the query. The response time is measured when the

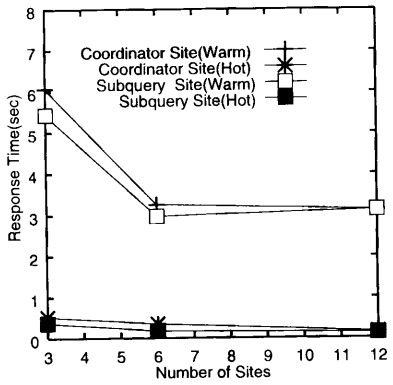

Fig.9 Speedup of spatial recursive

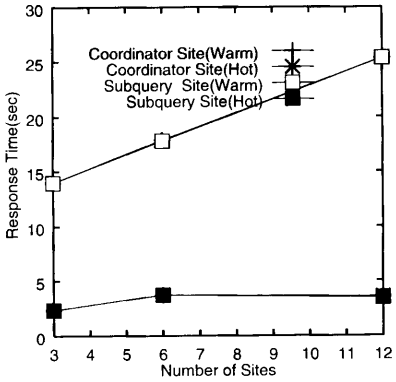

Fig.10 Scalability of spatial recursive

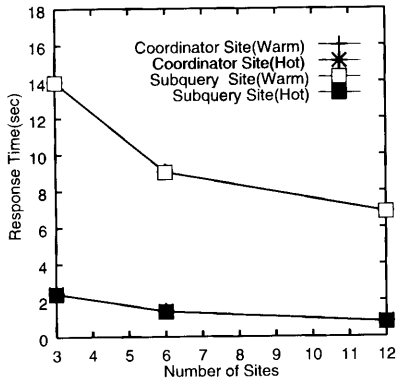

Fig.11 Speedup of spatial recursive

query is started to the time when final results are gotten on both coordinate site and subquery sites.

6. Conclusion

In this paper, we introduced the design and implementation of extended parallel sequoia 2000 benchmark on an ODBMS ShusseUo. The data partition and related parallel query algorithm were discussed and compared with that of related system. We measured the scalability and speedup of ShusseUo. From results, we can concludes that

- In the hot state, the system is scalable. In the warm state, the scalability become worse with the increment of number of sites.
- The system has a good speedup on both warm and cold state.
- The overhead derived from broadcasting spatial input data and synchronization of subqueries become larger when the number of site increases. Compared to the total cost, such kinds of overhead have a little influence on system performance.

References

1) A.C. Arpaci-Dusseau, R.H. Arpaci-Dusseau, D.E. Culler, J.M. Hellerstein, and D.A.Patterson, "High-performance storing on network of workstation", Proc. of the 1997 ACM SIGMOD Conference, 1997.
2) R.G.G. Cattell and Douglas K.Barry, The Object Database Standard: ODMG 2.0. Morgan Kaufmann Publishers, Inc. 1997 ISBN 1-55860-463-4
3) David J. DeWitt, Navin Kabra, et.al., "Client-Server Paradise", Proc. of VLDB 94
4) David DeWitt, Combining Object Relational & Parallel: Like Trying to Mix Oil and Water. VLDB 96.
5) Jignesh Patel, JieBing Yu, Navin Kabra, et al., Building s Scalable Geo-Spatial DBMS Technology, Implementation, and Evaluation. SIGMOD 1997
6) M. Stonebraker, J.Frew, K.Gardels and J.Meredith. The SEQUOIA 2000 storage benchmark. SIGMOD 1993
7) Kan Yamamoto, Multimedia Data Storage for the Object-Oriented Persistent Programming Language I-NADA, 1997, February. Master thesis, the Department of Intelligent Systems, Kyushu University, Japan
8) 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.
9) Clement T. Yu, Weiyi. Meng. Principles of Database Query Processing for Advanced Applications. Morgan Kaufmann, San Francisco, 1998.