Implementation and Performance Evaluation of WAKASHI

2H−2

Guangyi Bai, Akifumi Makinouchi
Faculty of Engineering, Kyushu University

1 Introduction

We are now working for a project named "Shusse-Uo (出世魚)" [Makinouchi 91] [Makinouchi 92] which is to develop a system for the object-oriented and database (or persistent) programming. The system is enhanced step by step by adding new functionality. The WAKASHI is the most lean system, which provides C programmers with the most primitive facilities to deal with distributed shared persistent data.

WAKASHI is constructed on Mach Operating System using the virtual-memory-based approach. This approach is to support persistent data in virtual memory by binding files into virtual address space.

Through previous work on persistent heap [Bai 91] and virtual-memory-based database [Bai 92], we have implemented and evaluated a shared persistent heap on one site and we came to the conclusion that virtual-memory-based approach is adaptable to advanced data applications in new computing environments such as distributed and parallel computing systems.

In this paper, we describe an implementation of the distributed shared persistent heap of WAKASHI using the Mach Operating System, and present a performance evaluation of the persistent heap.

2 Implementation of WAKASHI using the Mach OS

Mach Operating System provides EMMI (External Memory Management Interface) that allows users to define and manage the content of memory object that may be mapped into virtual address. Exporting this interface to the user programs simplifies the construction of complex virtual memory applications and allows them to control sharing, consistency, and secondary storage of their data without being embedded in the operating system kernel.

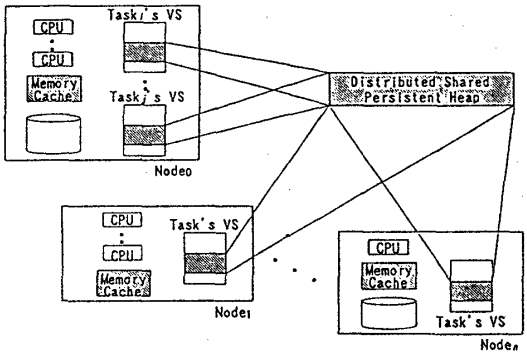

Figure 1: Distributed Shared Persistent Heap

WAKASHIの実現及び実行評価
白 光一、 牧之内 顕文
九州大学 工学部

We extended the Mach EMMI to implement a centralized WAKASHI Server (i.e., an external pager) to support the distributed shared persistent heap object that is a abstraction of distributed shared secondary storage.

As Figure 1 shows, the WAKASHI Server provides an distributed shared persistent heap object to clients on different machines. Any number of clients may map the distributed shared persistent heap object into their own virtual address space to create a distributed shared persistent heap area.

Since, in the virtual-memory-based approach, a persistent heap area is virtual address space, we may extend the distributed shared virtual memory techniques to implement the distributed shared persistent heap. In a typical implementation of distributed shared virtual memory, a memory mapping routine in each processor maps the local memory onto the shared virtual address space. Memory pages are paged not only between a local physical memory and the local paging area on secondary storage, but also between physical memories of different processors. Our implementation of the distributed shared persistent heap is based on the distributed shared virtual memory. The key idea is to replace the local paging area on secondary storage by a user-specified file.

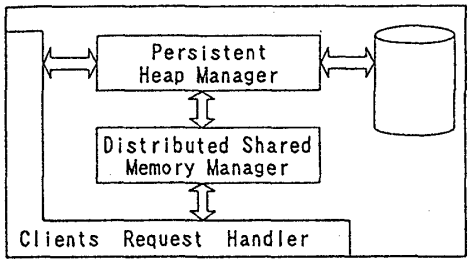

Figure 2: WAKASHI Server Structure

As Figure 2 shows, the WAKASHI Server is composed of:

The Client Request Handler ; This is the WAKASHI Server's interface (i.e. the extended EMMI) to clients on the same machine or different machines. The clients use this interface to ask either Persistent Heap Manager or Distributed Shared Memory Manager to create or destroy one or more than one distributed shared persistent heap object, and control WAKASHI Server to avoid unnecessary page-in and page-out. The interface also handles all page accesses by the clients.

The Persistent Heap Manager ; This supports persistence of data on the same machine. It implements paging between the physical memory and the files on the same machine.

The Distributed Shared Memory Manager ; This implements paging between remote physical memories to guarantees data coherence on the same machine or different machines. The Distributed Shared Memory Manager uses a fairly simple page scheduling policy to ensure that progress is made when more the one request is outstanding for a given persistent page. The underlying processor scheduling and network delays limit

the rate at which pages can be reclaimed and these limitations effectively eliminate thrashing. The Distributed Shared Memory Manager also handles multiple page sizes and different data representations in order to support sharing in heterogeneous distributed systems. To support multiple page sizes, the Distributed Shared Memory Manager may provide data or make lock requests in large units than the kernel's page. To support different data formats, it allows its clients to associate data types with its persistent heap objects.

3 Performance Evaluation

To see if the WAKASHI Server can serve as the storage manager with reasonable performance for both the remote RDBMSs and the remote OODBMSs, we developed simple simulation programs in C that simulats the Wisconsin Benchmark and the Engineering Database Benchmark. The programs run under Mach 2.5 on a OMRON LUNA-88K workstation. The machine has 3 M88100CPU (25Mhz) processors and 32 Mbytes physical memory. The measurement was performed in a-single-user environment. To simplify the simulation programs, a persistent pointer of data contains offset address of the data location in the file. Therefor, transformation from a persistent pointer to a volatile pointer is simpler than the one that would be needed to implement real DBMSs. However, we think that CPU time increase which would be expected when using table look-up might be negligible. (We plan another simulation for this, though.) As a by-product of this simulation, we found that programming dealing with persistent data, called "persistent programming", is much easier than expected.

Wisconsin Benchmark : We created a remote relation file and an index file for the Wisconsin Benchmark test. The remote relation file contains a relational table whose tuple length is 208 bytes. In the measurement, tuples increase from 10000 up to 200000. The index file is for indexing the table. The index is a binary tree whose nodes points to tuples of the table. Using the binary tree instead of the B-tree simplifies the programming.

We measured the following two operations:
(1) Selection with 10% selectivity factor without index.
(2) Selection with 10% selectivity factor with index.
Figure 3 shows the execution time of the first operation. As the Figure 3 shows, in the "Cold" cases, every page in the file are paged-in once and only once during the operation. In the "Warm" cases, for a database enough small to be cached on the physical memory, the performance is as good as the one of the Main-Memory Database. When it becomes large, some pages are paged-out. Hence, after that point, the performance is similar to the Cold cases.
The second operation requires small work space, so the performance is much better than the first operation.

Engineering Database Benchmark A remote database of "parts" with unique part-id is created. The part-id is actually represented by byte offset in the file. A part connects itself to three other parts. The size of a part is 28 bytes. We measured the following two operations:
(1) Lookup: Looking up objects.
(2) Traversal: Following connections between objects
The program maps the remote "parts" database file into its own virtual address space. The in-memory parts in the persistent heap area have the same format as in-disk parts stored in the file. A part is pointed by part-id. The part-id is the byte offset of the part in the file. When a part is accessed by the program, it must be accessed using its virtual address. Transformation from the part-id to its virtual address in the persistent heap area is done by adding its byte offset to starting address of the persistent heap area.

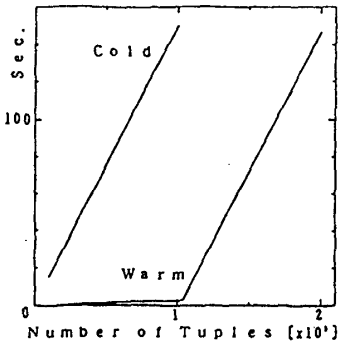

Figure 3: Selection without index

4 Conclusions and Future Works

WAKASHI gives the C programmers the uniform address space to handle distributed persistent data as well as volatile data. The centralized WAKASHI Server is constructed on the Mach Operating System using virtual-memory-based approach. The WAKASHI Server is designed for use in distributed database system designer or other data intensive applications that perform complex and high-performance manipulation of distributed persistent data.

Such a centralized solution has several drawbacks. The server may become a bottleneck and proximity of a client to the server may affect its performance. In addition, the server host performs an unfair amount of computation, possibly degrading other tasks on that host, and so on. Therefor, we are now designing a distributed WAKASHI Server that can allocate any number of servers on any number machines is more usable.

References

[Makinouchi 91] Akifumi Makinouchi and Masayoshi Aritsugi, "The Object-Oriented Persistent Programming Languages for Multimedia Databases", Technical Report CSCE-91-C04, March 1991.

[Makinouchi 92] Akifumi Makinouchi, "Shusse-Uo Project", Proc. of 44th IPSJ conf, 2H-1, 1992.

[Bai 91] Guangyi Bai, Masayoshi Aritsugi, and Akifumi Makinouchi, " Implementation of the Persistent Heap", Proc. of 43th IPSJ conf, 1L-7, 1991 [in Japanese].

[Bai 92] Guangyi Bai, and Akifumi Makinouchi, "Implementation and Evaluation of a New Approach to Storage Management for Persistent Data - Towards Virtual-Memory Databases - ", Second Far-East Workshop on Future Database, Kyoto Japan, 1992.