Design and Implementation of Transaction Coordinator at Network Of Workstations Environment

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

Kaneko, Kunihiko
Department of Intelligent Systems, Graduate School of Information Science and Electrical Engineering, Kyushu University

Makinouchi, Akifumi
Department of Intelligent Systems, Graduate School of Information Science and Electrical Engineering, Kyushu University

https://doi.org/10.15017/1515681

出版情報：九州大学大学院システム情報科学紀要. 5 (2), pp.173-178, 2000-09-26. 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
システム情報科学紀要
第5巻 第2号 平成12年9月

Research Reports on Information Science and
Electrical Engineering of Kyushu University
Vol.5, No.2, September 2000

Design and Implementation of Transaction Coordinator at Network Of Workstations Environment

Taiyong JIN*, Kunihiko KANEKO** and Akifumi MAKINOUCHI**

(Received June 16, 2000)

Abstract: To support the highly concurrent processing of transactions with low price at parallel and distributed database systems, Network Of Workstations(NOW) is utilized. All workstations cooperate to perform the jobs submitted by database applications. Each job consists of several transactions and these transactions are executed on NOW. Each transaction sent to NOW is allocated to a certain workstation by the coordinator running at a workstation. In this paper, we present a Distributed-Transaction Coordinator (DTC). In DTC, the cost to finish transactions is collected automatically and each transaction is assigned to an appropriate site based on the collected information.

Keywords: Distributed shared virtual memory, Network of workstations, Parallel transaction process, Memory coherence, Object database system

1. Introduction

1.1 Network Of Workstations (NOW) & Distributed-Transaction Coordinator

NOW [12] is a group of workstations connected via a network, and it is expected to act as a parallel machine with lower price, via utilizing the commodities CPU, disk, memory, and network. NOW is utilized in many kinds of application including distributed database server.

A distributed database server receives many transactions issued by the application jobs submitted by users. Each transaction is executed at one of workstations. Among these workstations of NOW, a workstation is assigned to coordinate the transactions. The program running on the workstation for coordinating is called as Transaction Coordinator (TC).

For example in **Fig.1**, the workstation *WS4* is the TC workstation of the NOW. Transaction *T0, T1, T2, T3.....* are sent to *WS4*. And *T0* is forwarded to *WS3*; *T1, T3* are sent to *WSn*; T2 is sent to *WS2*.

In this paper, we introduce a transaction coordinator which is implemented in *WAKASHI* [2),13)]. WAKASHI is the lowest layer of an object database system named *SHUSSEUO*, which is developed by Kyushu University, Japan.

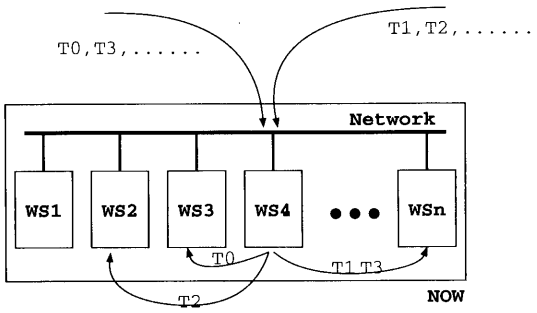

Fig.1 Network of Workstations

1.2 Distributed Shared Virtual Memory (DSVM) & WAKASHI

To ease sharing the resources distributed on the workstations in NOW and to ease the distributed parallel programming, two main approaches were proposed. One is the *Message Passing*(MP) [9], and the other one is *Distributed Shared Virtual Memory*(DSVM) [3),4),5)]. The DSVM seems to provide more friendly user interface [7].

WAKASHI is based on the DSVM technique and it provides transparent persistent storage facility for the object database applications. In WAKASHI, a DSVM space shared by distributed processes is called a heap.

In WAKASHI, a database can be composed of several independent heaps, as shown in **Fig.2**. To create a persistent DSVM space, we use two kinds of mapping mechanism. One is *disk mapping* supported by UNIX OS[11]. The other is *DSVM mapping*

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

between the virtual memory spaces of two different sites. By the disk mapping, the data becomes persistent. By the DSVM mapping, the virtual memory space is shared by the processes at different sites.

The functionality of each site to manage a heap in NOW is different. These sites are distinguished into two kinds, *Primary Site* and *Mirror Site*. There is only one *Primary Site* of a heap and several *Mirror Sites*.

- **Primary site**
  If a heap in a site is created via *disk mapping*, this site is called the *primary site* of this heap. And the WAKASHI server in this site is called *primary server*. Primary server performs the following 3 tasks.
  (1) Primary server is responsible to keep the heap's data persistent. When a transaction commits at other site, all of its updated pages of this heap are sent back to the primary site and the primary site writes the updated pages back to the disk.
  (2) Primary server also manages the global accesses on this heap. If a site wants a lock on a page and the lock is not retained locally, the lock-request is forwarded to the primary site. In this scene, the primary server is a global lock monitor of the heap.
  (3)As all of the updated pages are sent back to the primary site, it always holds the latest copies. So while another site wants to fetch a latest copy, the primary site can satisfy the request quite well.
- **Mirror site**
  If a heap in a site is created via DSVM mapping, this site is called *mirror site* of this heap. And the WAKASHI server in this site is called *mirror server*. A mirror server manages the local accesses. While there is an access-request on a page which does not reside in the site, the request for the page is forwarded to the primary site by the mirror server.

In the rest of the paper, at section 2 we describe the cost that a WAKASHI transaction has to pay. In section 3 the transaction coordinator named as Distributed-Transaction Coordinator(DTC) is presented. The related work is introduced at section 4. Finally, the conclusion is given at section 5.

2. The cost paid by WAKASHI transactions

In WAKASHI each transaction is coordinated to a site. To which site is decided by considering the cost of the transaction. The cost of a transaction which accesses a heap is mainly determined by two values, the number of pages read and the number of pages written.

In this section, we firstly introduce how to collect such values. Secondly, we give some cost factors. With these factors and the collected values, the total cost of a transaction is calculated.

2.1 The way to collect the cost values

In WAKASHI, we use the implicit page lock mechanism to synchronize the transactions. Before a transaction begins, all the pages of the heaps are protected. When the transaction wants to access a protected page, a pagefault occurs and is trapped by the pagefault handler of the process in which the transaction runs. The information of the pagefault tells the type of the operation, read or update. According to the type, page lock request is sent to the page server. At the same time, the number of read(written) pages of the heap is increased by 1. When the transaction is to commit, such information of each heap is gathered.

2.2 The cost factors

In WAKASHI, there are mainly 3 kinds of cost factor.

- Disk I/O Factor (f_{io}):
  Each heap exists in the virtual memory space at either the primary site or a mirror site. So there are swapping-in or swapping-out between main memory and the disk of the site. In WAKASHI, there are two kinds of operation incurring disk I/O. (1) When a page is accessed and the page doesn't reside in the memory, the page has to be read from disk into memory. (2) When a page is written, there is a disk I/O needed for writing the updated page from memory to disk. We use the f_{io} to represent the cost caused by disk I/O.
- Data Transfer Factor(f_{dt}):
  When a page is read at a mirror site where the page doesn't reside, the page is fed from the primary site. In WAKASHI, the page's transfer induces the network communication cost. We call such cost as *data transfer cost* and use f_{dt} to represent it.
- Remote Control Message Factor (f_{rct}):
  In order to synchronize concurrent transactions, some control messages are necessary. Among these message, some messages have to be exchanged between the different sites. We

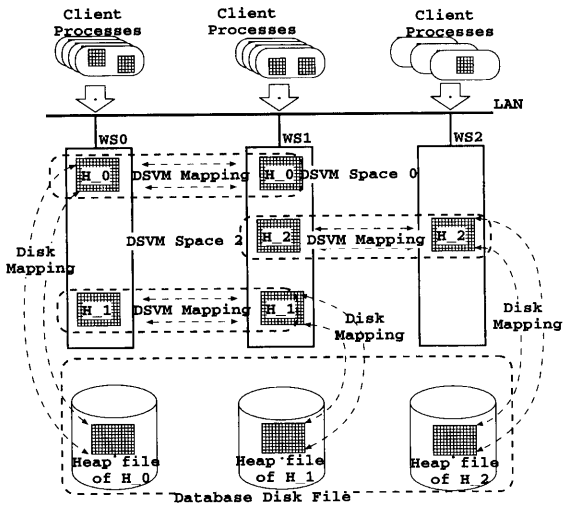

Fig.2 Database Structure in WAKASHI

call the cost for exchanging the message as *remote control message cost* and use f_{rct} to present it.

As measured in current WAKASHI, these three cost factors are found to satisfy the following formulas roughly.

$$f_{rct} = 15 \times f_{io} \quad (1)$$

$$f_{dt} = 60 \times f_{rct} \quad (2)$$

Using these three basic cost factors, we can formulate the following four more complex cost factors.

1. The cost factor f_{pr} of a *page_read_access* at the primary site:

$$f_{pr} = f_{io} \quad (3)$$

2. The cost factor f_{pw} of a page write access at the primary site:

$$f_{pw} = 2 \times f_{io} \quad (4)$$

In WAKASHI, before a page is written, the page must be read from disk to memory. Writing the updated page back to the disk includes another disk I/O.

3. The cost factor f_{mr} of a *page_read_access* at a mirror site:

$$f_{mr} = 2 \times f_{io} + f_{dt} + 2 \times f_{rct} \quad (5)$$

As shown in **Fig.3**, when a page is read at a mirror site, firstly, a *remote_page_read* request is sent to the primary site. If the request is granted, the

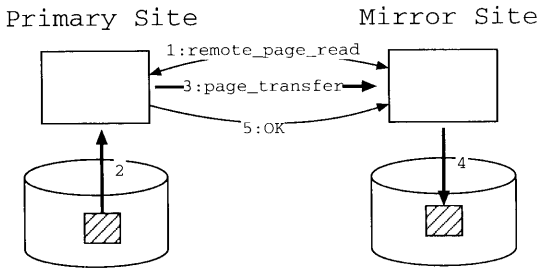

Fig.3 Read a page at mirror site

page is loaded from the disk and transfered to the mirror site. Finally, the primary site sends an OK message to the mirror site. When the mirror site receives the page, the page is written to the disk. As the result, two disk I/Os, one data transfer and two remote messages are necessary.

4. The cost factor f_{mw} of a *page_write_access* at a mirror site:

$$f_{mw} = 4 \times f_{io} + 2 \times f_{dt} + 4 \times f_{rct}; \quad (6)$$

As shown in **Fig.4**, before a page is written at a mirror site, the page is read at first. So the cost for a page read access at a mirror site is necessary. Additionally a *remote_page_write*, and an OK message are necessary to grant the write lock from the primary site. When the transaction is to commit, the updated page has to be transfered back to the primary site. When the primary site receives the page, writing the updated page back into the disk implies

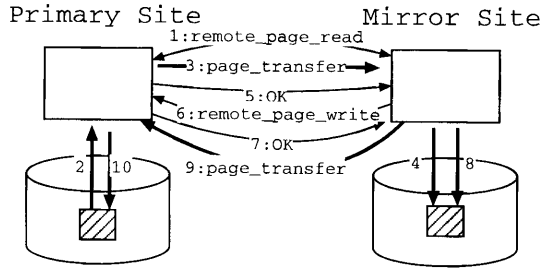

Fig.4 Write a page at a mirror site

a disk I/O. As the result, four disk I/Os, two data transfers and four remote messages are necessary.

When transaction t is scheduled to be executed at site s, the total cost for execution of t is summed up as follows.

$$\left(\sum_{i=0}^{n, i \neq S} (f_{mr} \times N_{ri} + f_{mw} \times N_{wi})\right) + (f_{pr} \times N_{rs} + f_{pw} \times N_{ws}) \quad (7)$$

Here N_{rx} is the number of the pages read at heap x and N_{wx} is the number of the pages written at the same heap.

3. Distributed-Transaction Coordinator (DTC)

In DTC, a transaction is scheduled to be executed at the workstation where the cost of executing it is less than the cost of executing it at other workstations.

In DTC, there are 4 modules, *Transaction Pool*(TP), *Cost Information Manager*(CIM), *Database Distribution Manager*(DDM) and *Transaction Scheduler* (TS). At each site of NOW, there are several *Execute Element*(EE)s and DTC communicates with these EEs via the communication interface.

The structure of the DTC is shown in **Fig.5**.

In the following sections, before the introduction of the structures of TP, DDM, CIM, and the algorithm of TC, EE is described at first.

3.1 Execute Element (EE)

Each EE has two kinds of functionality.

- Execute the transaction
  Initially EE is blocked. When a transaction's type and parameters are sent from DTC by socket, the EE is activated to execute the transaction.
- Collect the cost information and feed it back to DTC
  When EE finishes the transaction, the number of page read locks and page write locks of each heap are collected and sent back to DTC via a RPC message.

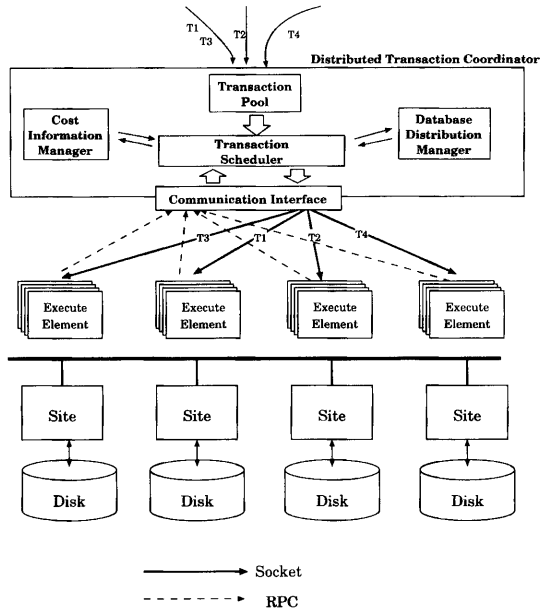

Fig.5 Structure of DTC

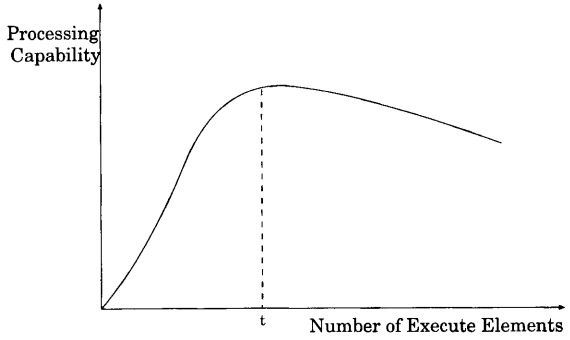

Fig.6 Relationship between the processing capability and the number of the Execute Elements

Although the number of EEs at each site can be tuned dynamically by DTC, there is a limit of the number of EEs, because the resource of a site, such as CPU or memory, is limited. As shown in **Fig.6**, the site's processing capability rises when the number of EEs increases until t. While the number of EEs becomes over than t, the processing capability can not be improved.

3.2 Transaction Pool (TP)

When DTC receives a transaction, the information of the transaction, such as type and parameters are stored in the Transaction Pool. When a transaction is coordinated, the information of the transaction is removed from TP. The structure of TP is a transaction list as shown in **Fig.7**.

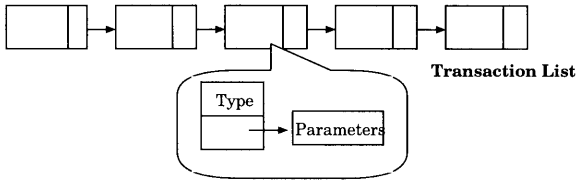

Fig.7 Structure of Transaction Pool

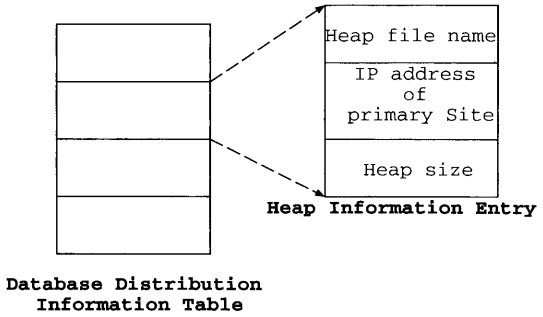

Fig.8 Structure of DDM

The transactions are retrieved one by one from the beginning of the list. And the transactions stored at the list are sorted in order of their cost values. In other words, the transactions with lower cost are allocated prior to the ones with higher cost. Consequently the transactions with lower cost are coordinated in advance. We found that it is a better way to keep the workload balance of the NOW.

3.3 Database Distribution Manager (DDM)

The information of the database distribution, such as the location of each heap and its size, is stored in DDM. The structure of DDM is shown in **Fig.8**.

A main component of DDM is a table named as *Database Distribution Information Table*. An entry of this table stores the distribution information concerning a heap.

3.4 Cost Information Manager (CIM)

In any database application, the number of the transaction types is fixed and decided during designing it. For each type of transaction, a vector $< ov_0, ov_1, ...ov_n >$ named as *Cost Vector(CV)* is used to describe the cost. Here n is the number of the heaps composing the application database. Each ov_i is also composed of a vector $< Nr, Nw >$. Here Nr (Nw) is the number of the pages read(written) in heap i.

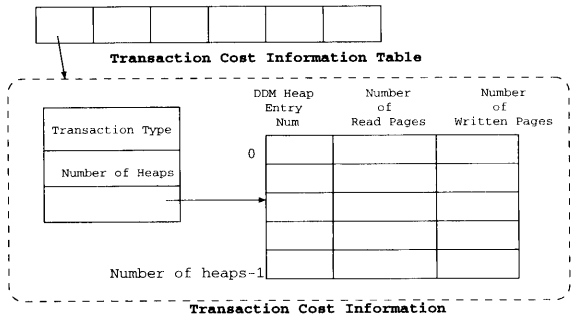

Fig.9 Structure of CIM

In DTC, the CVs of every type of transaction are stored in CIM. Its structure is shown in **Fig.9**.

Initially, the table is empty. Once a transaction commits, the cost information is sent to CIM and is stored in the corresponding entry.

3.5 Transaction Scheduler Algorithm

- When a beginning request of a transaction t whose type is T arrives and there isn't any idle EE,
  (1) Get the cost information of type T.
  (2) Insert the T and parameters to the TP.
- When a begin request of a transaction t whose type is T arrives and there are some idle EEs,
  (1) Get the cost information of type T.
  (2) For each site s of the idle EEs, calculate the t's total cost if t would be executed at site s.
  (3) Schedule transaction t to be executed at site k at which the total cost of t is the smallest.
- When transaction t commits,
  (1)EE Sends the cost value of transaction t to DTC
  (2)When DTC receives such information from an EE at site s,
  (2.1) if the corresponding entry in CIM is empty,
  (2.1.1) register an entry to store the information.
  (2.2) if the entry is not empty, average the vector stored in CIM and newly-received one.

(2.3) if the TP is not empty, get a transaction whose cost of executing at site s is minimum and schedule the transaction to site s.

4. Related Work

In order to keep the workload balanced in NOW or multicomputer systems, some process migration models[1),6)] are proposed. In these models, both migration policy and location policy are based on the cost of the running processes. The cost mainly consists of CPU cost. And the life time of a process is used to describe the CPU cost. Disk I/O and network communication cost are not considered.

In our model, the life time of a transaction can not be used to represent the transaction cost, since some transactions may be blocked because of page lock conflicts. During the blocking period, the transaction doesn't spend CPU. Moreover, common database applications are I/O intensive. So we consider the disk I/O and network communication cost as the cost of a transaction.

5. Conclusion

In this paper, we introduce the DTC which is implemented at WAKASHI, a distributed database platform based on Distributed Shared Virtual Memory environment. In DTC, the disk I/O and communication cost are considered. It differs from other process migration based models. We give several cost factors to describe the execution cost of a transaction accessing a distributed database in WAKASHI. The cost of each transaction is collected and stored automatically. Each transaction is scheduled to be executed at the site where its cost is minimum. The structure and algorithm of DTC are introduced.

6. Acknowledgements

This research is partially supported by the research project on Advanced Databases of Scientific Research on Priority Area in Japan(Grant-No. 08244105).

References

1) M.Harchol-Balter, and A.B.Downey, "Exploiting Process Lifetime Distributions for Dynamic Load Balancing", ACM Transaction on Computer Systems, Vol.15, No. 3, August 1997, pp. 253-285.
2) G.Bai, and A.Makinouchi, "WAKASHI/D: A Distributed Paged-Object Server for Storage Management of New Generation Databases", Proc. of the Int'l Symposium on ADTI, Nara, 1994, pp.137-144.
3) B.N.Bershad and M.J.Zekauskas, "Midway: Shared memory parallel programming causal distributed shared memory", Proc. of the 11th International Conference on Distributed Computing Systems, October 1991, pp.152-164.
4) B.Nitzberg and L.Virginia, "Distributed Shared Memory: A Survey of Issues and Algorithms", ACM Computer. Surv. August 1991, pp.52-61.
5) J.B.Carter,J.K.Bennett and W. Zwaenepoel. "Implementation and Performance of Munin", Proc. of the Thirteenth Symposium on Operating Systems Principles, October 1991, pp.104-123.
6) F.Douglis and J.Ousterhout, "Transparent process migration: Design alternatives and the sprite implementation" Software, Pract. Exper. Vol.21, No.8, August 1991, pp.757-785.
7) K.Li and P.Hudak. Memory coherence in shared virtual memory systems. ACM Transactions on Computer Systems, Vol.7, No.4, November 1989, pp.321-359.
8) K. Li and R.Schaefer, "A Hypercube shared virtual memory system for parallel computing", Proc. of the 1988 International Conference on Parallel Processing, 1989. pp.1022-1047
9) Message Passing Interface Forum, "MPI: A message passing interface standard", http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html.
10) U.Ramachandran and M.J.A.Khalidi, "An Implementation of Distributed Shared Memory", Software-Practice and Experience, Vol.21, No.5, 1991, pp.443-464.
11) Sun Microsystems, Inc. SunOS 5.2 Reference Manual, July 1993.
12) T.E.Anderson, D.E.Culler, D.A.Patterson and the NOW team, "A Case for NOW(Networks of Workstations)", IEEE Micro. Feb.1995, pp.54-64.
13) G.Yu, K.Kaneko, G.Bai and A.Makinouchi, "Transaction management for a distributed object storage system WAKASHI – design, implementation and performance", Proc. of 12th ICDE, New Orleans, February 1996, pp.496-509.