データベース・システム 63－3
（１９８８．１．１８）

データベース･オペレーティングシステム･シミュレータの作成

仲 興国　陳 道灼　上林 弥彦
九州大学･工学部

計算機の応用分野におけるデータベースの比重が高くなるにつれ、データベースに適したオペレーティングシステムが重要となってきている。従来のオペレーティングシステムはデータベースシステムにとって不適なところが多くある。並行処理制御(複数の処理単位の実行に直列可能性を保持しつつ、効率よく実行する制御)を考慮に入れていないことが1つの問題点である。並行処理制御の研究および並行処理制御に適したオペレーティングシステムの研究のために、我々はデータベース･オペレーティングシステム･シミュレータを設計開発した。本稿では、その実現および利用について述べる。

A Database Operating System Simulator

Xingguo ZHONG, Daozhuo CHEN and Yahiko KAMBAYASHI

Dept. of Computer Science and Communication Engineering,
Kyushu University, Fukuoka, 812 Japan

Due to the increasing importance of databases in computer application areas, it becomes very important to develop operating systems suitable for databases. There are some conventional techniques for operating systems unsuitable for database systems. One of the serious problems of conventional operating systems is that its support for concurrency control (which tries to improve efficiency while keeping the correctness called serializability) is not enough for databases. In order to evaluate techniques for concurrency control and techniques in the operating system level that are considered to be suitable for concurrency control, we have developed a database operating system simulator. In this paper, we describes the implementation of the simulator and its applications.

1 Introduction

From our research on database concurrency control, there is a requirement to develop a simulator to give some quantitative evaluations of the techniques we have developed. The techniques we need to evaluate consist of two classes: techniques of constructing efficient concurrency control mechanisms and techniques of providing suitable environment under the operating system level for concurrency control. For this purpose, we have developed a database operating system simulator.

In this paper, we will discuss the simulator which can simulate the combined effects of concurrency control supported by operating system and databases. Because of the difficulty of comparing various concurrency control mechanisms analytically, there are papers on concurrency control simulators [1, 3, 6, 8]. It is known that some conventional techniques used in operating systems are not suitable for database [9]. Thus we need to develop operating system which can support for database functions efficiently by proper concurrency control mechanisms. Features of our simulator are as follows.

(1) We designed the simulator from operating system level. Thus simulation on techniques in the operating system level is possible.

(2) Besides READ and WRITE operations in transactions, actions of transactions can be arbitrarily defined.

(3) We define transaction as a partial order of actions [2, 4, 5], which can model parallel transactions [7] and is especially desired in distributed database systems.

(4) Due to the modular organization, the simulator is assumed to be extendable. Simulation on buffer management suitable for database systems, for example, is also planned.

From the viewpoint of the internal organization, the simulator has been designed as a virtual database operating system. It provides process management, buffer management, concurrency control, I/O management and interrupt control.

The implementation of the simulator is realized on a Ustation E20 using the C language (about 3,000 lines of source code). In Section 2 the transaction and data access models of this simulator will be shown. In Section 3, the organization of the simulator is described. In Section 4, the interface to define a concurrency control mechanism is presented. Section 5 shows one of the experiments performed using this simulator.

2 Transaction and Data Access Models

The parallel transaction model was introduced in [5]. In [4] realization of efficient partial rollback under such model was discussed. Formal definition is also given in [2]. Fig. 1 is an example of a transaction defined as a partial order of actions. Each node of the partial order shows an action and each edge expresses a dependent relationship between two actions. A transaction

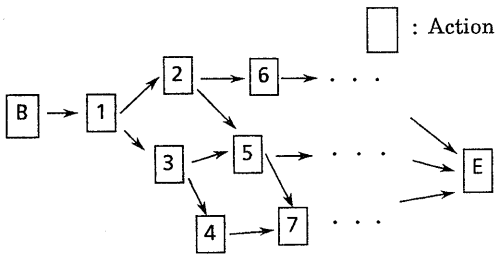

Fig. 1 Transaction Model

starts with a BEGIN (B) action and terminates with an END (E) action. The contents of other actions are not decided. They can be defined by users of the simulator. According to the definition of the actions, an interpreter is needed for the simulator to simulate the execution of these actions. All actions have predecessors (successors) except the BEGIN (END) action. In simulating the execution of transactions, an action of a transaction is executable, if all of its predecessors have been executed. The contents of actions used for simulations on concurrency control are as follows.

ACTION:: = BEGIN | REQUEST | ACCESS | COMP | COMMIT | END

In the simulator, a database is assumed to be a

set of data items. A data item may be simply considered as a logical block of I/O system, which is also handled as a unit of concurrency control. A transaction is defined to be a partial order of the above actions with REQUEST actions on data items being serial. REQUEST is a read or write request to a certain data item. A read request is always followed by an ACCESS action, which copies the data item from the common main memory space of the system to the private space of the transaction. When the data item is not in the main memory at the time of access, the ACCESS action issues an I/O access to fetch the data item from the database to the common space. COMP is interpreted as a computation using certain CPU time.

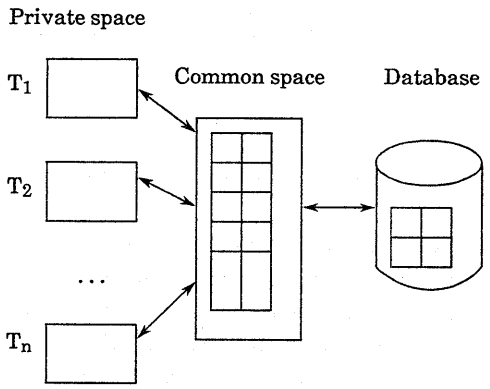

Fig. 2 Data Access Model

COMMIT appears one time in each transaction. By this action, the transaction finishes its work and is committed. All write operations of a transaction are performed in the COMMIT action. Two-phase commit protocol is assumed to be used in terminating a transaction. Thus, writing each data item need two I/O accesses between main memory and secondary storage.

Fig. 2 shows the data access model of the simulator. When a transaction is issued to the system, a private space is prepared by the system for buffering all the data items it will read and write. All the read operations are performed by copying the data items to the private space. Therefore, a transaction will read one data item at most once. Write operations of a transaction does not reflect their value of data items to the database until the transaction is committed. Thus no data item will be written to the database more than once by one transaction. A transaction generating program is implemented satisfying these constraints. Setting up the common space for buffering data items avoids redundant data accesses.

3 Organization of the Simulator

The simulator is constructed as a virtual database operating system. It provides an environment to simulate the behavior of the concurrent execution of transactions under certain system assumption. Since the transactions described in the previous section are only symbols, not real programs, the simulator is required to interpret each action of a transaction and reflect the corresponding effects to a virtual system.

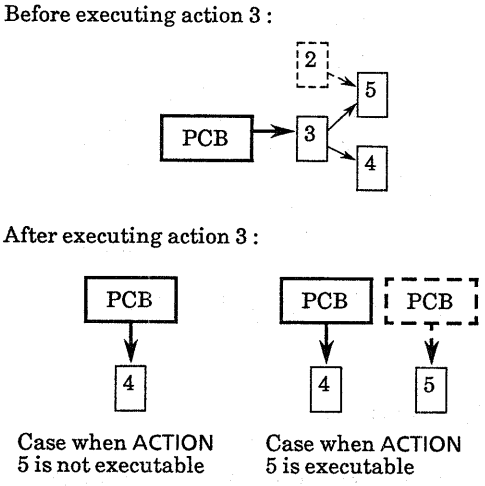

Fig. 3 Control Execution of Actions

(1) Controlling execution of a transaction

The execution of transactions is controlled at two levels, transaction level and process level. For each transaction, when it is issued to the system, a Transaction Control Block (TCB) is created to control it. The TCB records the current state and

other information on this transaction. When the transaction starts its execution from the BEGIN action, a Process Control Block (PCB) is created to control the execution of the action. When the execution of one action finishes, the control of the PCB is moved to an executable successive action. If there is no executable successive action, the PCB will be destroyed. If, however, there is more than one executable successor, new PCBs are created for other executable successors.

Fig. 3 shows an example of such control. When the execution of ACTION 3 finishes, the control of the PCB is moved to ACTION 4. If ACTION 5 is executable at the time, a new PCB is created for ACTION 5.

Fig. 4 shows execution control of a transaction. Note that when a transaction is created, a unique PCB is created for controlling the BEGIN action. When the transaction terminates, only one PCB exists to control the END action.

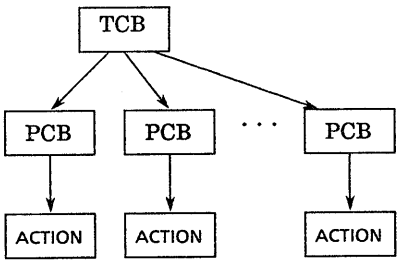

Fig. 4 Execution Control of a Transaction

(2) State transition

Fig. 5 shows the state transition of transactions, which demonstrates the lifecycle of an execution of a transaction. When a transaction is issued to the system, it is put into active state. The state of the transaction moves to blocked state when it waits for a data item held by another transaction. When the data item is released by the other transaction, the transaction returns to active state. A transaction in blocked or active state may be aborted by the system. When a transaction is aborted, it moves to abort state and then is forced to sleep for a while. The state of a transaction will be converted from active state to commit and then terminated, when the last REQUEST action of the transaction is granted and all the COMP actions finish.

The TCB of a transaction does not control the execution of its actions directly. It controls PCBs belonging to it. The state of transaction depends on states of its PCBs. A TCB is blocked if one of its PCBs controls a REQUEST action which conflicts with other transactions.

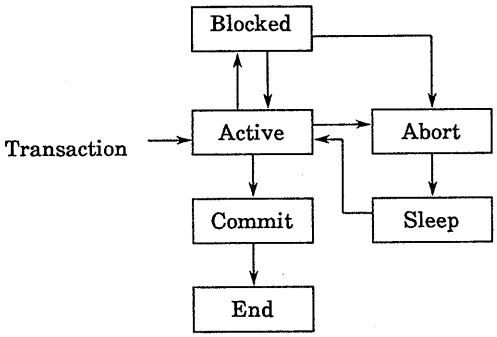

Fig. 5 State Transition of Transactions

The state transitions of processes are shown in Fig. 6 which demonstrates the possible lifecycle of a process. When a process is created, it controls the execution of a certain action. The ready, suspending and running states of processes are the same as in general operating systems. The blocked state of processes is separated from the suspended state to express that the process is controlling a REQUEST action that conflicts with other transactions. When a process enter the blocked state, the transaction it belongs to will also enter the blocked state. A process will be destroyed when it finishes in controlling an action and no successive action is executable. A suspended state of a process shows that the process controls an ACCESS action that causes an I/O access. The process will enter the ready state when the I/O access is completed.

(3) Program construction

The simulator consists of program modules, each of which provides a certain function. Unlike real operating systems, these modules are not controlled by system processes. The program is

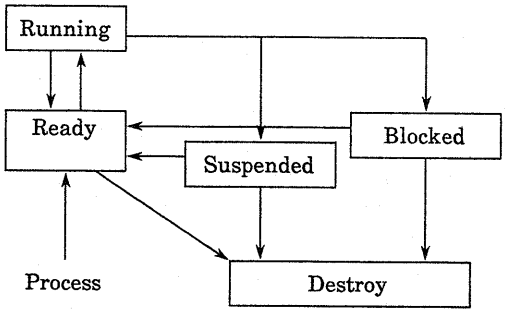

Fig. 6 State Transition of Processes

designed to support the connections of the following modules.

* Interrupt Controller
* Execution Interpreter
* Console Controller
* Issuing Transaction Manager
* Terminating Transaction Manager
* Concurrency Control Mechanism
* Buffer Manager
* Aborting Transaction Manager
* I/O and CPU Manager
* System Clock Manager
* Common Functions

The interrupt controller is realized by a conditional branch program. When a module needs to start another module, it only needs to set the corresponding flag for that module to let it satisfy the execution condition. When the execution of this module finishes, the control of the system will go to that module, if that module has the highest priority at the time.

Fig. 7 shows the connection of the system modules. An action of a transaction controlled by a PCB is executed by the running interpreter. If the action is a REQUEST for a certain data item, the concurrency control mechanism will be called to handle the request. If the request is accepted, the PCB will enter the ready state. If the request is forced to wait, the PCB and the TCB it belongs to will enter the blocked state. When the concurrency control mechanism decided to abort a transaction, the aborting manager will be called. More than one transaction may be aborted depending on concurrency control mechanisms.

When the action is an ACCESS, the buffer manager will be called. According to the state of the system buffer, the buffer manager will determine whether or not an I/O access is required. It issues an I/O access to the I/O and CPU manager when necessary.

When the action is a COMP, the I/O and CPU manager will be called. Transaction commitment is managed by the termination manager. In order to maintain the system load under a limit, the multiple transaction level (number of transactions being executed concurrently) is determined as a system parameter. When a transaction

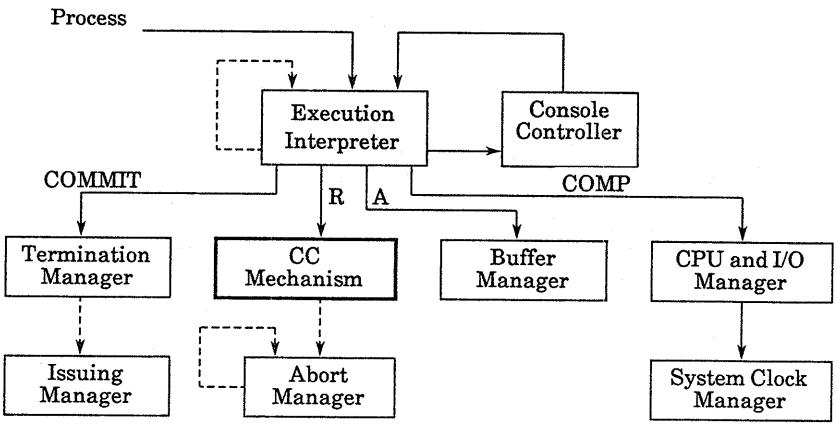

Fig. 7 Modular Construction of Simulator

terminates, a new transaction is required to be generated and issued to the system.

(4) Building virtual CPU and I/O devices.

Processes in the ready state are collected in a ready process queue. The interrupt controller takes the processes one after another to execute them according to a certain strategy. Running a process one time executes one or several actions of a transaction. As all the actions of each transaction are only symbolic, the time concept is virtual. A COMP action consumes CPU time. An ACCESS action consumes I/O time when it needs an I/O access between main memory and the database. In order to describe the time consuming, a virtual COST block is used to express the cost (CPU time or I/O time)that will consume.

Fig. 8 shows the concurrent execution of COSTs with the system having one CPU and two I/O devices. For the CPU and each I/O device, a COST queue is set up. Each COST belongs to a process of a transaction. The vertical axis expresses the virtual time supposed. The sequence in which the processes will enter the ready state should be the same as the sequence of termination time of their corresponding COSTs. A process will be executed in the next time earlier if it enters the ready process queue earlier under FIFS (Frist-In-First-Service) strategy for process management.

In the Fig. 8, the CPU and I/O COSTs, C_1, C_2, C_3, I_{11}, I_{12}, I_{21}, I_{22}, I_{23} will be terminated in the sequence of events shown at the left. Three CPU COSTs are shown in the figure. In fact, when executing a process causes the CPU COST queue to be not empty, the CPU and I/O manager will be soon called to handle COSTs until the CPU COST queue becomes empty.

An advantage of setting the above virtual time is that system overhead for concurrency control and others can be set up arbitrarily by giving certain COSTs for system management.

(5) System parameters.

In order to perform the simulation under desired system assumption, the system is defined by system parameters. All the parameters are defined as external variables in the simulator. An initializing function is used to assign certain values to these parameters. Therefore changing the system assumption does not need to perform re-compilling and linking of any source program. As an example, some system parameters are shown below.

DBSIZE	:	Database size
MTL	:	Multiple Transaction Level
NUM-IO	:	Number of I/O devices
NUM-CPU	:	Number of CPU units
TIMEPIECE	:	Largest time for a process to occupy a CPU unit

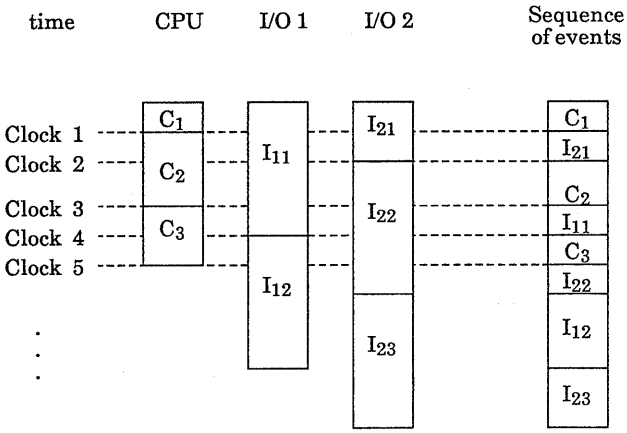

Fig. 8 Concurrent Execution of Vertual COSTs

COM-SPACE : Size of the common space
IO-COST : Time consumed by one data access between main memory and secondary storage

4 Interface to Concurrency Control Mechanisms

The simulator provides a standard interface for constructing a concurrency control mechanism by six functions. When users perform experiment on concurrency control, different concurrency control mechanisms are usually constructed under the same environment. Since the contents of each of these functions depend on the concurrency control mechanism to be constructed, only a rough explanation is given as follows.

(1) Initializing a simulation

When starting a simulation, preprocessing works required by the users are performed by this function.

(2) Terminating a simulation

When the simulation finishes, the results of the simulation need to be extracted. Such work can be done by this function.

(3) Preprocessing for issuing a transaction

When a new transaction is issued to the system or an aborted transaction is awaken to restart, the required preprocessing is performed in this function. For example, in timestamp ordering, a timestamp is required to be assigned to the transaction. In two-phase locking, a node corresponding to the transaction may be added to the wait-for graph.

(4) Preprocessing for committing a transaction

Preprocessing desired for committing a transaction is performed in this function. The major work of this function is to perform the reflection of data items written by the transaction to database. Performing unlocks for data items it locks and awaking the blocked transactions waiting for its termination, are also needed in certain concurrency control mechanisms.

(5) Preprocessing for aborting a transaction

Preprocessing desired for aborting a transaction is performed in this function. This function is very similar to the above function (4).

(6) Handling an operation request

This function is the main procedure of concurrency control mechanisms. It receives a request of a transaction and checks if this request can be granted or not. If not, it checks if it is necessary to abort any transactions.

The details of each function depend on the concurrency control mechanisms to be constructed. Data under certain data structures corresponding to meta-information in a real concurrency control mechanism are also needed to be defined for constructing the concurrency control mechanisms.

5. Applications of the simulator

Using the database operating system simulator, some experiments on evaluating techniques we have developed have been performed. In this section, we describe one of these experiments performed for simulating a technique under the operating system level called execution selection strategy.

The execution selection strategy under the operating system level has been proposed to be suitable for concurrency control [10]. The basic idea of this technique is to give the ongoing transactions increasing priorities determined by the length of their execution. This can also be considered to be the method providing as much serial executions of transactions as possible under the concurrent execution environment. By the execution selection strategy, the possibility of causing transaction abort may be reduced.

The execution selection strategy is realized by process management and I/O management. In the process management, instead of the FIFS strategy, the process in ready state with the largest executed part at the time of interest is always selected to be executed. The same strategy is also used in I/O management. Instead of performing I/O access (between main memory and secondary storage) in the sequence of IFIS, the I/O COST belonging to the process with the largest executed part is selected to be performed. There are several definitions in measuring the executed part. The experiment performed defines the executed part by the number of data items held by each transaction.

That is, the process with holding thelargest number of data items will always be selected to be executed.

The experiments were performed for both two-phase locking and multi-version timestamp ordering. The major purpose of these experiments is to know to what degree will the execution selection strategy effect the transaction abort rate in database systems. The results of the experiments have dramatically proved the effectiveness of this technique.

Fig. 9 shows the elapsed times and transaction abort rates for different multiple transaction levels by executing 5,000 transactions for each of the multiple transaction levels. For space limitation, we omit the results of multi-version timestamp ordering and further discussion on the results of the figure.

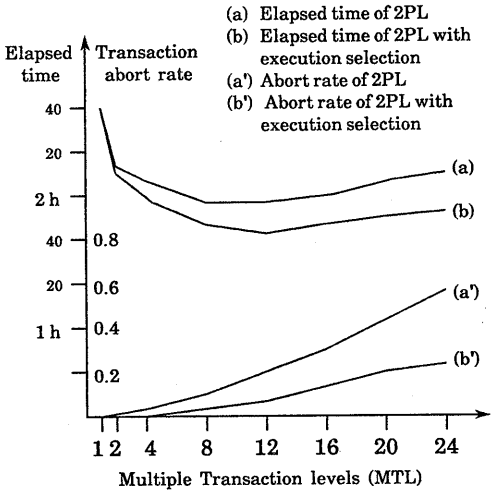

Fig. 9 Comparison of Two-phase Locking with its Version Utilizing Execution Selction Strategy (DBSIZE=3000)

6. Summary

In this paper, implementation of a database operating system simulator has been described. The motivation of developing this simulator is to evaluate techniques of concurrency control and techniques in the operating system level that are considered to be suitable for concurrency control. Besides techniques under operating system level, experiments for techniques in realizing efficient concurrency control mechanisms have also been performed.

References

[1] R.Agrawal and M.Carey, Models for Studying Concurrency Control Performance Alternatives and Implications, SIGMOD 1985, pp. 108-121.

[2] J.Brzozowski and S.Muro, On Serializability, International Journal of Computer and Information Science, Vol.14, No.6, 1985.

[3] M.Carey and M.Stonebraker, The Performance of Concurrency Control Algorithms for Database Management Systems, VLDB 1984, pp.107-118.

[4] S.Kondoh, Y.Kambayashi and S.Yajima, Concurrency Control Procedures Utilizing Dependencies of Basic Operations, RIMS Record 494, Kyoto University, Research Institute of Mathematical Sciences, pp.90-101, June 1983 (In Japanese).

[5] R.Krishnamurthy and U.Dayal, Theory of Serializability for a Parallel Model of Transaction, ACM Proc. of Principle of Database Systems, pp.293-305, 1982.

[6] W. Lin and J. Nolte, Basic Timestamp, Multi-Version Timestamp and Two-Phase Locking, VLDB 1983, pp. 109-115.

[7] K.Saisho and Y.Kambayashi, Multi-Wait Two-Phase Locking Mechanism and its Hardware Implementation, Proc. on 5th Inter. Workshop on Database Machine, Japan, 1987. pp 198-211.

[8] S.Nishio et al., Performance Evaluation on Several Cautious Schedulers for Database Concurrency Control, Proc. on 5th Inter. Workshop on Database Machine, Japan, 1987. pp 212-225..

[9] M.Stonebraker, Operating System Support for Database Management, Comm. of ACM, Vol. 24, No. 7, 1981. pp. 412-418.

[10] X.Zhong and Y.Kambayashi, Execution Selection Strategies Suitable for Database Concurrency Control, Proceedings of 36th Annual Convention, IPS Japan, Mar. 1988. (in Japanese)