VICTORIA UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF COMPUTER AND MATHEMATICAL SCIENCES

Terabyte Database Simulation Model

Kevin H. Liu and Clement H. C. Leung

(73 COMP 27)

April 1996

(AMS : 68P15)

TECHNICAL REPORT

VICTORIA UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF COMPUTER AND MATHEMATICAL SCIENCES
P O BOX 14428
MCMC
MELBOURNE, VICTORIA 8001
AUSTRALIA

TELEPHONE (03) 9688 4492
FACSIMILE (03) 9688 4050

Footscray Campus

TERABYTE DATABASE SIMULATION MODEL

Kevin H. Liu and Clement H. C. Leung

Department of Computer and Mathematical Sciences
Victoria University of Technology, Ballarat Road, Footscray,
PO Box 14428 MMC, Melbourne, Victoria 3000, AUSTRALIA
Email: {kevin, clement}@matilda.vut.edu.au
Fax: +613-688 4050

ABSTRACT

With the query complication and many hardware and software options in parallel databases, both analytical and simulation model need to be considered to accurately describe the system behaviour. Terabyte Database Simulation Model is developed at Victoria University of Technology by the Terabyte Database Group, to carry out further research on terabyte database, identify potential research problems, and gain research experiences. This paper will provide a manual for the model. Facilities of the simulation model are discussed and the analytical model behind the relational operations is presented. Several research topics are included in Application Guide such as skew of parallel database, site allocation of distributed database, and parallel query processing in Object-Oriented database.

1. TDB Simulation Model Overview

Parallelism has been employed in database system to obtain higher performance and availability. [2,4] However, it also introduces more possible ways to execute one query on multiple connected processors than on conventional single processor. Therefore, parallelism complicates not only query processing but also performance evaluation. Although analytic model can be used, to some extent, accurately to describe the system, most of the cases, the system is too complex to be modelled by analytic model alone and the stochastic feature and the bottleneck of the system are less likely to be reflected in an analytic model. [9,10,11] As a result, simulation is a better way in terms of the cost, efficiency and preciseness.

1.1 Facilities

The Terabyte Database Simulation Model (TDBSM) is designed to provide a user friendly environment which facilitates the users carrying out research in parallel database system particularly in query optimisation, data mining, and parallel architecture design. The model presents an TDBSM platform (an integrated development environment) which provides

⇒ online help,
⇒ a built-in calendar,
⇒ a built-in calculator,
⇒ a built-in ASCII table (facilitates programmers),
⇒ multiple, movable, and resizable windows,
⇒ fully mouse support and dialogue boxes,
⇒ put down menus and online status options,
⇒ menu colour syntax highlighting to accelerate the search,
⇒ online clock and heap view,
⇒ and temporarily DOS exit.

1.2 Configuration

TDBSM employs a hybrid architecture where nodes are loosely coupled and each node may have a number of processors with varied powers, and different interconnections among the processors within the node. Currently, the Terabyte Database Group (TDBG) is focusing on query processing and data mining, and most of the proposed algorithms are hardware independent. As a result, more experiments will be carried out before decisions are made on topology among the nodes. However, to reflect the characteristics of the current generation of commercially available multiprocessors, the group will pay more attention to the fat-tree network structure interconnections adopted by the Thinking Machine CM-5, the two-dimension mesh interconnections used by the Intel Paragon, and the hypercube topology employed by nCUBE multiprocessors and Intel iPSC/860.

The simulation parameters are classified into systems, database sites, and workload categories. The default values of all parameters are provided through dialog boxes which can be modified by the user. Due to the limitations of the TDialog Class of Turbo Vision, checking is not enforced on any parameters. Therefore, users must be cautious on settings to achieve rational results.

1.3 Operations and Applications

In addition, the model provides basic unary relational algebraic operations such as selection, projection (with and without duplicate removal), and aggregation (minimum or maximum), and binary operations such as nested-loops join, and hash based join. Moreover, three applications are discussed in detail, Skew in relational parallel systems, Path expression operation in object-oriented parallel systems and Load Balancing in distributed systems. The objective is to illustrate how users can integrate their existing models or proposing algorithms into the model with ease. An example of data representation (Skew in relational parallel database systems) is included in Appendix B.

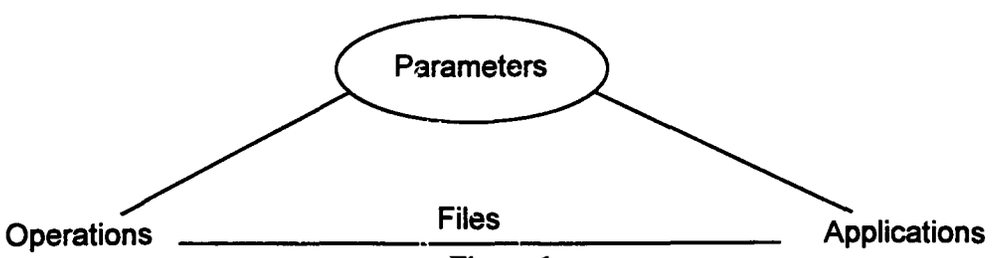

Figure 1

Figure 1 shows that applications and operations are co-related through files since interprocess communication is expensive. All parameters in the model are global variables so they are visible in any operations and application functions.

This manual is a complete description of the TDBSM. The simulator is entirely written in C++ and it is compiled using the Borland C++ 3.1 under DOS 5.0. The model is based on event driven simulation which consists of four main types of events, keyboard events, mouse events, message events, and nothing event, respectively.

The remainder of this paper is organised as follows. Section 2 discusses hardware and software requirements, and Section 3 contains menus and options references. Library primitives of TDBSM are included in Section 4. In Section 5 and 6, we present application guides and, briefly, show users how to make contributions. Finally, Section 7 provides our concluding remarks.

2. Hardware and Software Requirements

2.1 Hardware Requirements

In order to use all the facilities of the TDBSM, your PC must have a certain minimum hardware configuration. Your PC must be of IBM PC compatible family of computers, including the AT and PS/2, along with all true IBM-compatible 286, 386, or 486 computers using DOS 3.3 or higher. Besides, you need a hard disk, and at least 640k RAM.

Mouse system must be installed properly along with your PC. A math co-processor (80x87) is preferred in your PC to improve performance dealing with floating point math operations although it is not compulsory.

To represent the data in graphics, you need a monitor graphics display card. The model can detect automatically and support several graphics systems. The graphics cards currently supported are CGA, MCGA, EGA, EGA64, EGAMONO, IBM8514, HercMono, ATT400, VGA, and PC3270.

2.2 Software Requirements

Directly under the root directory of your hard disk create a subdirectory. Then, copy all files to this directory using command x*copy a:*.*/s*. List files to check there is one BGI directory and all **.bgi* files are in the BGI directory. The following files should be included,

tdbsimu1.exe	executable file of the model,
tdbsimu1.hlp	help file messages,
readme.doc	supplementary information about the system and some last minute updates,
**.obj*	object modules of the system model,
**.bgi*	Borland graphic device drivers used by the system.

All source files names' listing can be found in Appendix A.

2.3 Getting Started

Type *dir* to check all object files are in the same directory with the executable file. Next, give the command *tdbsimu1.exe* to start the model. If there is an error message that the graphic drives are not found, go to menu "Options-SetBGIPath" and input the right path through the dialog box.

3. Menus and Options Reference

Due to the space limitations of status line, not all keyboard shortcuts are displayed on the screen. However, the following standard accelerators in Table 1 are always available no matter where you are in the simulation model.

Key Combinations	Usage
Alt-x	quit the TDBSM to Dos
Alt-F3	close the active window
F10	activate the main menu
F5	zoom in or out the active window
Ctrl-F5	move and resize the active window
F6	activate the next window
Shift-F6	activate the previous window

Table 1

Menu options are discussed below in detail to provide a reference for users.

3.1 System Menu

The system menu is located at the far left of the menu bar. It provides a submenu with five options. "About" will give a Copyright modal dialog box; "Puzzle" is small game won by arranging the characters in order (A -- O) with the few moves; "Ascii" presents a table which can show the decimal and hexadecimal number of any ascii characters; "Calendar" and "Calculator" are two facilities supplied to users.

Parameters	Explanation
NoNode	the number of nodes in the system
NoRela	the number relations in the database
IntCst	fixed network time for message initiation per block
TerCst	fixed network time for message termination per block
BlkSiz	system block size (number of bytes per block)
NetBan	network bandwidth (transmission time per block)
nLarge	flag to represent the data in graphics

Table 2

3.2 File Menu

As any other software packages, file menu lets you look at the contents of the files, change file directories, exit to DOS, and quit the simulation model. "Open" will display the file in a full-sized window with scroll bar available for the sake of

convenience. Multiple windows can be opened simultaneously, but only one window is active at one time. Unfortunately, you can not edit the file within the window. However, you may temporally exit to DOS and use any editors (e.g. DOS editor) to make modifications. Then, type EXIT to return simulation model.

3.3 Settings Menu

Simulation parameters setting can be done in the menu by dialog box. "System" submenu lets you set up systems parameters shown in Table 2.

"DB_Site" submenu lets you set up database sites parameters shown in Table 3.

Parameters	Explanation
NoPro	the number of processors per node
MemSiz	the memory size of local processors
SekTim	disk seek time per block
LatTim	disk latency time per block
XerTim	data transfer time from disk to memory in local sites per block
RevTim	the time to retrieve one block from buffer
PCTim	time to compare two tuples
PETim	time to evaluate one predicate involving a single field
PPTim	time to insert one tuple to the hash table
PHTim	time to probe one tuple with the hash table
WriTim	the time to write one block data to output buffer.

Table 3

"Workload" submenu lets you set up query workload parameters shown in Table 4.

Parameters	Explanation
LenPre	the length of predicates in the query
NoColn	the number of columns to be projected
TupSiz	the number of bytes per tuple
R_Size	|R| -- the total number of records in the relation R
S_Size	|S| -- the total number of records in the relation S
selfac	Selectivity factor i.e. the ratio of the number of selected records to the number of records of the specified relation.
UniRat	the ratio of tuple that are unique when dealing with projection without duplicates

Table 4

Click O.K. if you want to save your modifications on the parameters. Otherwise, close the dialog box or click Cancel to quit.

3.4 Operations Menu

TDBSM provides unary relation algebraic operations such as selection, projection, and aggregation and binary relational operations such as join. The real database is not presented because the need of a real running system and the necessary design of the measurement. In addition, real database may bring about inaccurate results representing the general performance of a system. By comparison, analytical and

simulation model will not only efficiently give solutions based on mathematical theories but also precisely describe the stochastic behaviour of database operations.

The following cost functions will serve as the basis of the operations over multiple processors in TDBSM. The assumptions are as below and additional notations are listed in Table 5.

Assumptions: Loosely coupled multiprocessors are fully connected, skew does not take into account, and cost is primarily expressed in terms of the elapsed time taken to carry out an operation. In addition, data are well distributed on the disks initially, i.e. the reading time for each block is the same. Finally, data are stored among the memory associated with each processors without replication, i.e. each processor need to transmit $\left\lceil \frac{R\_Size \times TupSiz}{N \times BlkSiz} \right\rceil$ blocks of data to each processor at the beginning of processing.

Parameters	Explanation
Ts	the elapsed time for a select operation
T_{p1}	the elapsed time for a project operation (with duplicates)
T_{p2}	the elapsed time for a project operation (without duplicates)
T_{A1}	the elapsed time for an aggregation operation (min or max)
T_{jns}	the elapsed time for a join operation with nested-loops method
T_{jhh}	the elapsed time for a join operation with hybrid hash method
ReaTim	reading time from disk to local memory of each block of data
N	the total number of processors used for parallel processing
r_k	the input relation R in number of blocks
s_k	the input relation S in number of blocks

Table 5

In addition, we derive

ReaTim=SekTim+LatTim+XerTim (for each block of data),

$$N = \sum_{i=1}^{i=NoNode} No\,\mathrm{Pr}\,o_i,$$

$$r_k = \left\lceil \frac{R\_Size \times TupSiz}{BlkSiz} \right\rceil,$$

and $s_{k=}\left\lceil \frac{S\_Size \times TupSiz}{BlkSiz} \right\rceil$.

Therefore, we have

Selection

$$T_s = \text{ReaTim} \times r_k + \left\lceil \frac{R\_Size \times TupSiz}{N \times BlkSiz} \right\rceil \times NetBan + \left\lceil \frac{R\_Size \times TupSiz}{N \times BlkSiz} \right\rceil \times \text{RetTim}$$
$$+ \frac{R\_Size \times \text{LenPre} \times \text{PETim}}{\text{N}} + \left\lceil \frac{selfac \times R\_Size \times TupSiz}{N \times BlkSiz} \right\rceil \times WriTim \quad ,$$

Projection

Projection provides a submenu of with and without duplicates removal. Projecting without duplicate removal is to project the vertical columns instead of select the horizontal rows in selection. Projecting with duplicates removal can be treated as a combination of projecting with duplicates and selecting the unique tuples. Therefore, the former is more expensive than the later since a nested loops search is needed for the selection of removing duplicates.

Projection without duplicate removal

$$T_{p1} = \text{ReaTim} \times r_k + \left\lceil \frac{R\_Size \times TupSiz}{N \times BlkSiz} \right\rceil \times NetBan + \left\lceil \frac{R\_Size \times TupSiz}{N \times BlkSiz} \right\rceil \times \text{RetTim}$$
$$+ \frac{R\_Size \times \text{NoColn} \times \text{PETim}}{N} + \left\lceil \frac{R\_Size \times TupSiz}{N \times BlkSiz} \right\rceil \times \text{WriTim} \quad ,$$

Projection with duplicate removal (projection + selection)

$$T_{p2} = T_{p1} + \frac{R\_Size \times (R\_Size + 1) \times PETim}{2} +$$
$$\left\lceil \frac{UniRat \times R\_Size \times TupSiz}{N \times BlkSiz} \right\rceil \times WriTim \quad ,$$

Aggregation (minimum or maximum)

Unlike other unary operations, aggregation over multiple processors has not been investigated in depth especially for many complex functions, such as dividing the whole population into a number of groups and collecting statistics in each group (Grouped by / Count). In this section, the simple operation (minimum or maximum) is considered and we assume one level parallelism, i.e. at the beginning, relation is partitioned into the number of parallel processors and, at the end of the processing, processors will transmit the result (one tuple) to one and only one processor which will carry on further comparisons.

$$T_{A1} = \text{ReaTim} \times r_k + \left\lceil \frac{R\_Size \times TupSiz}{N \times BlkSiz} \right\rceil \times NetBan + \left\lceil \frac{R\_Size \times TupSiz}{N \times BlkSiz} \right\rceil \times \text{RetTim}$$
$$+ \frac{(R\_Size - 1) \times \text{PCTim}}{\text{N}} + \left\lceil \frac{1 \times TupSiz}{BlkSiz} \right\rceil \times NetBan + \left\lceil \frac{N \times TupSiz}{BlkSiz} \right\rceil \times \text{RetTim}$$
$$+ (N-1) \times PCTim + \left\lceil \frac{1 \times TupSiz}{BlkSiz} \right\rceil \times WriTim$$

Join

Join is one of the most expensive and frequently used relational operations. It has been an active research area for decades. Initially, we will implement two algorithms, parallel nested loops join and parallel hash based join method. Although the hash based join leaves a good impression from the analytic model, it does suffer from choosing hash functions and avoiding hash collision. Parallel nested loops join is employing range partitioning and nest loops join is carried out in each individual processor, whereas parallel hash based join is using hash partitioning and hash join is processed at each processor.

Join Equation (assuming both operand relations are unsorted on the join attribute, R_Size is larger than S_Size, Mem_Siz is larger than S_Size, and *parallel nested loops method* with simple range partitioning is employed)

$$T_{jn} = \mathrm{ReaTim} \times r_k + \left\lceil \frac{R\_Size \times TupSiz}{N \times BlkSiz} \right\rceil \times NetBan + \mathrm{ReaTim} \times s_k +$$
$$s_k \times NetBan + \frac{(r_k + s_k) \times \mathrm{RetTim}}{N} + \frac{R\_Size \times S\_Size \times PCTim}{N},$$
$$+ \left\lceil \frac{selfac \times R\_Size \times S\_Size \times TupSiz}{N \times BlkSiz} \right\rceil \times WriTim$$

Join Equation (assuming R_Size is larger than S_Size, and *parallel hash based method* with hash partitioning is employed with uniformity partitions)

$$T_{jhh} = \mathrm{ReaTim} \times r_k + \left\lceil \frac{R\_Size \times TupSiz}{N \times BlkSiz} \right\rceil \times NetBan + \mathrm{ReaTim} \times s_k +$$
$$\left\lceil \frac{S\_Size \times TupSiz}{N \times BlkSiz} \right\rceil \times NetBan + \frac{(r_k + s_k) \times \mathrm{RetTim}}{N} + \frac{S\_Size \times PHTim}{N} +,$$
$$\frac{R\_Size \times PPTim}{N} + \left\lceil \frac{selfac \times R\_Size \times S\_Size \times TupSiz}{N \times BlkSiz} \right\rceil \times WriTim$$

Note: the partitions of the smaller relation is used to build the hash table

3.5 Applications Menu

The existing algorithms among the TDBG are integrated into the model to form this menu. Thus, the algorithms are classified into Architecture design, Data mining, and Query processing and data placement according to the subgroups interest. The main topics of query processing and data placement are transaction updating, skew in relational parallel database, path expression operation of object-oriented parallel database, and load balancing in distributed database. Moreover, the skewness are modelled using Random allocation, Interval simulation, and Double hashing according to the characteristics of input relation(s).

In fact, the application menu is aimed as a library and it will grow as the time goes on.

3.6 Windows Menu

The menu is dealing with window management commands. Multiple windows can be opened at the same time with one and only one activated, and each window has scroll bars, a zoom button, and a closing button. "Resize/move" lets you change the position and size of the window. ENTER is to confirm your choice, SHIFT-Arrow keys are to resize, and Clicking and Dragging the title of the window are to quickly reposition. "Zoom" lets you select the size of the active window from maximum and default sizes. "Next" will make the next window active. "Close" lets you close active window. "Tile" will tile all your opening windows. "Cascade" lets you stack all open windows.

3.7 Options Menu

The menu lets you make environment settings and all the submenu items have default values. Certainly you can change the settings through dialog box. The BGI path tells the system where is the Borland graphics device drivers and there will be device errors without them detected properly. Before using graphics representation, make sure that path is right through "SetBGIPath". "Mouse" lets you choose the speed of system responding to mouse click and you can activate the right mouse button by checking the Reverse Mouse Button. "Colours" can let you choose your colour preference for every component of TDBSM platform. When you select the item or group, set foreground and background colours, and do not forget to click OK to save changes. In addition, you may save and, later, retrieve the current desktop by using "Save Desktop" and "Retrieve Desktop".

4. Library Primitives

TDBSM is build on C++ Object-Oriented programming (OOP) language. The distinguishing features of OOP go to polymorphism, encapsulation, and inheritance. Therefore, the biggest advantage of OOP is to reuse code as much as possible. If you are interested in developing algorithms and their visual representation, you can have more efficient code by knowing the following library primitives.

4.1 Constructor and Destructor

The main function in the simulator is very simple,

```
int main (int argc, char **argv)
{
        TDBSimu tdbProgram(argc, argv);
        tdbProgram.run( );
        return 0;
},
```

since the constructor will take care of all the initialisation and destructor will destroy the object or class. Thus, you only need to call the run( ) which is a member function of TDBSimu class (a derived class of TApplication defined in the TDBSimu.h header file).

There are many types of constructors and copy constructor is used in the simulator,

```
TDBSimu::TDBSimu( int argc, char **argv):
```

```
        TProgInit( &TDBSimu::initStatusLine, &TDBSimu::initMenuBar,
        &TDBSimu::initDeskTop)
{
...(Body)...
}.
```

Although a few lines of code, you can have a fairly good and consistent user interface which has one top line for menu bar, one bottom line for status line, and a shaded background against the rest of the application appears. In addition, all the data initialisation may be done here by writing relevant code in the function body, i.e. constructor lets you provide a default setting for all parameters and variables used in the model.

In contrast, destructor is even easier than constructor because it only has one type and does not return any values. TDBSM employs

```
TDBSimu::~TDBSimu( )
{
        graphAppDone( );
        delete bgiPath;
}
```

to close the graph mode if it is active, free the driver memory, and delete the Borland graphics device drivers path before terminating the program.

Keep in mind that a class may have multiple constructors and may only have one destructor, and properly using them can have efficient codes for your algorithm design.

4.2 TView

Any objects seen on the screen are views and TView is the most fundamental and central class of any applications build on top of Borland Turbo Vision. In other words, all classes in TDBSM are derived from TView class. Going a bit further, it means when you develops your proposing algorithms or submodels, you will never modify the source code of the existing classes directly, instead you extend classes by deriving new classes from them. See reference [1] for detail.

4.3 TMenu, TSubmenu, and TMenuItem

It goes without saying that as the time goes on, the applications and relational operations in the TDBSM will also grow. However, the extending menu is so easy that take you less effort (a few modifications in one member function -- initMenuBar), and you only need recompile the source code.

The menu line at the top of the screen is the menu bar, the titles on the menu bar are submenus, and each submenu consists of several menu items. The definition of menu is done through initMenuBar member function which is only one statement,

```
return new TMenuBar (r, *new TSubMenu(...)+
                                *new TMenuItem (...)+
                                ...
                                *new TMenuItem (...)+
```

```
                *new TSubMenu(...)+
                    *new TMenuItem (...)+
                    ...
                    *new TMenuItem (...)+
                *new TSubMenu(...)+
                    ...
                    ...);
```

where r is the location of the menu.

4.4 TDialog

So far, there are views (objects seen on the screen) and mute objects (proposing algorithms and standard relational operations), but still lack the engine to drive the whole simulator. In TDBSM, it is the events communicating through dialog boxes and handled by the member function of TDBSimu -- handleEvent(TEvent &event).

Dialog box object in TDBSM includes labels and controls. The former can be activated but can not accept any input values, while the later may receive input lines or check and it has several styles (Button, RadioButtons, CheckBoxes, ListBox, and InputLine).

To execute the dialog, you need to call the setData( ) member function of TDialog class to pass the data to the dialog as did in TDBSM,

```
sdialog->setData(&Data);
ushort result=TProgram::deskTop->execView(sdialog);
if (result==cmOK)
        sdialog->getData(&Data);
```

and call the desktop evecView( ) function which will return a value telling you the terminating status (OK or Cancel). If user presses OK, system calls getData( ) function to save changes.

As any objects in C++, they are created dynamically and have to destroyed to free all resources.

```
TObject::destroy(sdialog);
```

4.5 Default values setting

Finally, there is a better way to provide default values for parameters setting through dialog boxes considering struct and its constructor such as following.

```
struct TextData {                    // declaration of the dialog box variables
        char   sNoNode[numLen];
        char   sNoRela[numLen];
        char   sIntCst[numLen];
        char   sTrmCst[numLen];
        char   sNetBan[numLen];
        char   sBlkSiz[numLen];
        ushort nLarge;
        TextData( );                 // constructor
};
```

```
TextData::TextData( )              // initialise the dialog box variables
{
        strcpy(sNoNode,"100");
        strcpy(sNoRela,"200");
        strcpy(sIntCst,"300");
        strcpy(sTrmCst,"400");
        strcpy(sNetBan,"500");
        strcpy(sBlkSiz,"600");
        nLarge=0;
}
```

Do not confuse Struct with Class. Both of them can have constructor, but the former is used only for storing data which by default are public while the later not only stores data but also the actions on the data (functions) which by default are private.

5. Applications Guide

5.1 Limits to Database Speedup for Data Partitioned Parallelisation in Relational Operations

A major obstacle to performance improvement in parallel database processing is the presence of skew, which in extreme cases, can contribute to performance degradation to a level below that of uniprocessing. In [8], a new taxonomy of skew for parallel database systems is presented, with attention focused on the relational model operating in a shared-nothing distributed memory environment in SPMD mode. Three types of skews are identified: data skew, load skew, and operations skew.

The Load Skewness is introduced when data is partitioned over multiprocessors. A homogeneous distributed memory configuration and unary relational operations have been chosen to study the skew effect which is brought about by intra-operation parallelism. The conventional performance estimates are far from accuracy and the closed-form expressions are proposed in [5]. The skewness complicates in binary relational operations and the skew estimation of join is discussed in [6].

The input variables of the skew application include the number of parallel processors, the cadinality of the relation(s), and the number of experiments per run. When the tuples generated, their destinations depend on random numbers. The output from the application is the load of the maximum loaded processor(s).

5.2 Site Allocation for Parallel Query Execution in Locally Distributed Databases

The application is dealing with exploiting parallelism for distributed query processing. Unlike parallel database systems, the communication cost plays an significant role in total processing cost of distributed database systems because of data fragmentation and data replication. See reference [3] for detail.

The input of the site allocation model are number of sites, number of base relations, with or without replication, overlapping factors, number of joins per query, cardinality of relations, length of a tuple, join methods. Moreover, you need unit time for data probing, hashing, joining, and initiation. The output from the model includes average number of phases and site utilisation.

5.3 Parallel Query Processing in Object-Oriented Database Systems

The object-oriented model (OOM) provides a more natural way to represent the large datasets and their complex relationships. As there has not been a standard developed for OOM, reference [7] gives an overview of the object-oriented data model and query, some parallelisation techniques and their implementations are discussed, and query optimisation issues of OOM are presented. The simulation is carried out on path expression operation.

The input of the path expression object-oriented operation application includes the number of processors, the number of classes, the number of objects of the root class, and the average degree of skewness where value one means no skew. Moreover, you need predicate selectivity for the output and the length of predicate to describe the coming work load. In addition, the cost unit for variable distribution, fixed distribution, reading, evaluating, writing, and scheduling must be set before you can obtain rational results.

The output of the application goes to elapsed time and linear speedup factor. Taking account of the possibility of re-distribution, elapsed time is given in terms of with or without re-distribution, with full data replication, and with uniprocessor.

6. How To Contribute

Any members of TDBG or Computer and Mathematical Science Department who have written C routines or C++ classes coping with standard database operations or proposing algorithms are encouraged to integrate their methods or models into TDBG's simulation model.

The merging steps are as follows,

- add a menu option in *initMenuBar( )* function in the *tdbsimu3.cpp* file
  e.g. PEJoin
- create the corresponding function in *tdbsimu2.cpp* file with the body of *main( )* function in your source code
  e.g. TDBSimu::pe_join( )
- add the function prototype (declaration) in the *tdbsimu.h* file
  e.g. pe_join( )
- put all other codes except *main( )* function to a new file
  e.g. cost51.cpp
- create a header file with function prototypes which used in the *main( )* function, and include this header file in *tdbsimu2.cpp* and the new function file
  e.g. cost51.h

- *make* to recompile and link.

7. Concluding Remarks

To study the behaviour of parallel computer database systems and gain deep understanding of the system, there are several approaches to achieve the objective. First, use benchmarks which are general database sets or high level programming language written programs that can represent a given class of applications. Second, use prototype which is a first form from which copies, reproductions, or further developments can be produced. Third, employ simulation to provide conditions which are encountered only in real operations. Fourth, study a real system by conducting experiments on an existing computer system to identify the current contentions and the potential future problems.

Among them, the benchmark is the simplest but limited to existing systems, often ,with hardware and software requirements, and the proposed systems must be available in its entirety (configuration). Moreover, the benchmark is less likely to use in a changing environment where system might get complicated in every minute. It goes without saying that not every research group can afford a real running system. And also, the measurement of real systems need design carefully. In addition, how to represent and identify the general performance of real database systems is a key question since most of the database systems behaviour are stochastic. Therefore, prototyping and simulation are the most cost-effective ways to study and research a database system particularly for small or medium sized research groups.

We believe the two methods are not mutually exclusive but sequentially related, i.e. prototyping is treated as the first step of simulation. By comparison, the former takes less effort, can run experiments faster, and is easier to modify to achieve desired objectives whereas the later can show more details of the system such as the bottleneck.

In other words, we suggest to have a prototype first to identify the potential research problems and help participants gain experience dealing with one particular system. Next, the corresponding simulation may be built up depending on the resource constraints.

Appendix A: Source File Names Listing

Header Files

ascii.h
bgi.h
bgii.h
calculator.h
calendar.h
cost51.h
dialog.h
fileopen.h
hc_view.h
graphapp.h
mouse.h
gpuzzle.h
sk_simu.h
sksim1.h
tdbhelp.h
tdbsimu.h
tvcmds.h

Standard Include Header Files

borlandc\include
borlandc\tvision\include

Program Files

ascii.cpp	// ascii table
calculator.cpp	// calculator
calendar.cpp	// calendar
cost51.cpp	// path expression operation
fileopen.cpp	// open file
hc_view.cpp	// clock and heap
mouse.dlg	// mouse selection
gpuzzle.cpp	// game
sk_simu.cpp	// skewness
tdbsimu1.cpp	// main program one
tdbsimu2.cpp	// main program two
tdbsimu3.cpp	// main program three
utls.cpp	// graphics utilities

Libraries

borlandc\lib	// C++ library
borlandc\tvision\lib	// Library for application framework Turbo Vision

Appendix B: Example of Data Representation -- Skew in Parallel Databases

The following 11 figures (Figure B1-B11) output directly from one run of the running program.

- Each 3D-bar in the figure represents one processor
- The length of the 3D-bar represents the number of tuples in that processor
- Two vertical lines with coordinates on them provide scale of workload
  e.g. 60 means 60 tuples

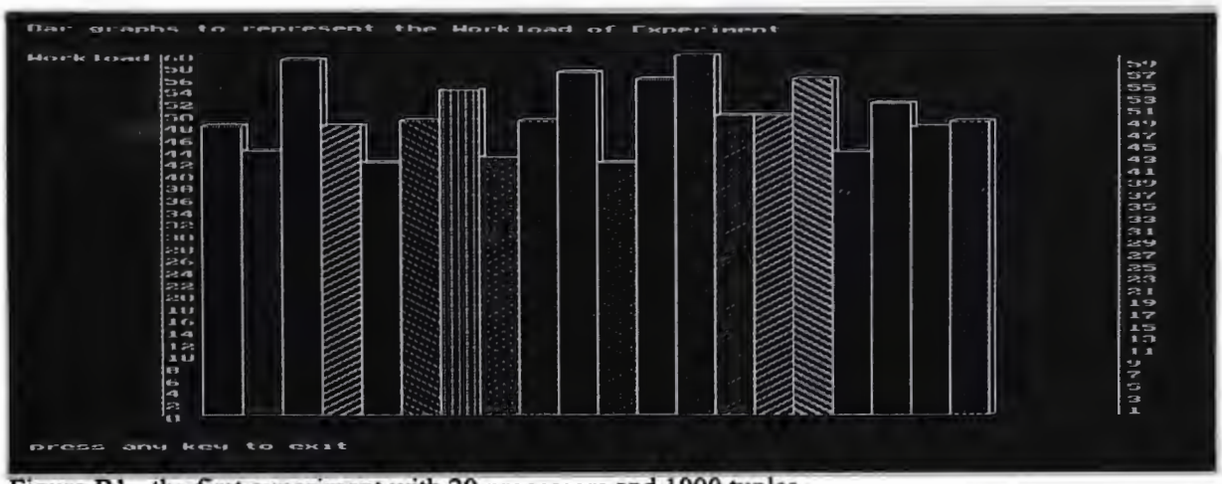

Figure B1: the first experiment with 20 processors and 1000 tuples

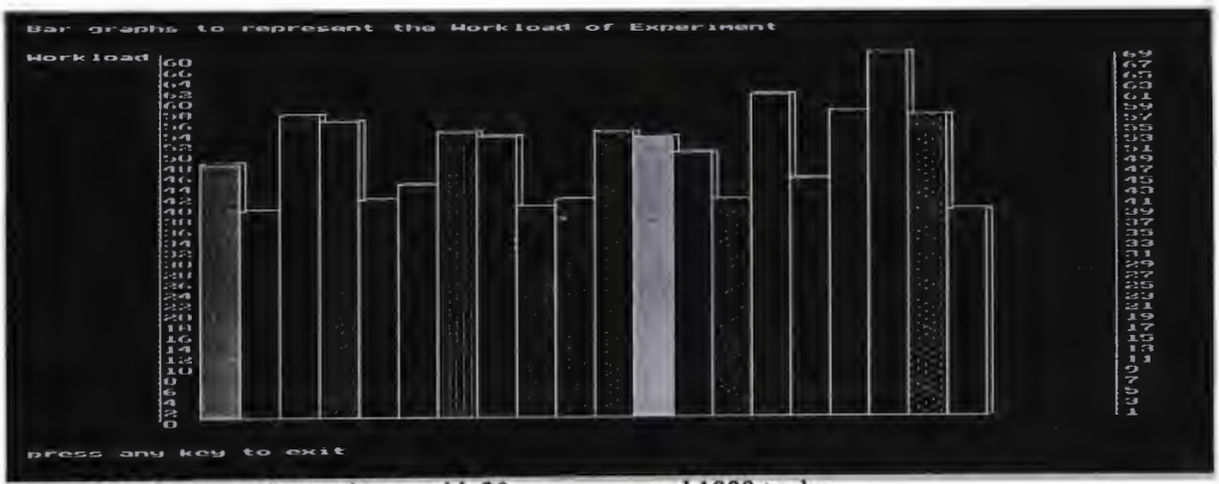

Figure B2: the second experiment with 20 processors and 1000 tuples

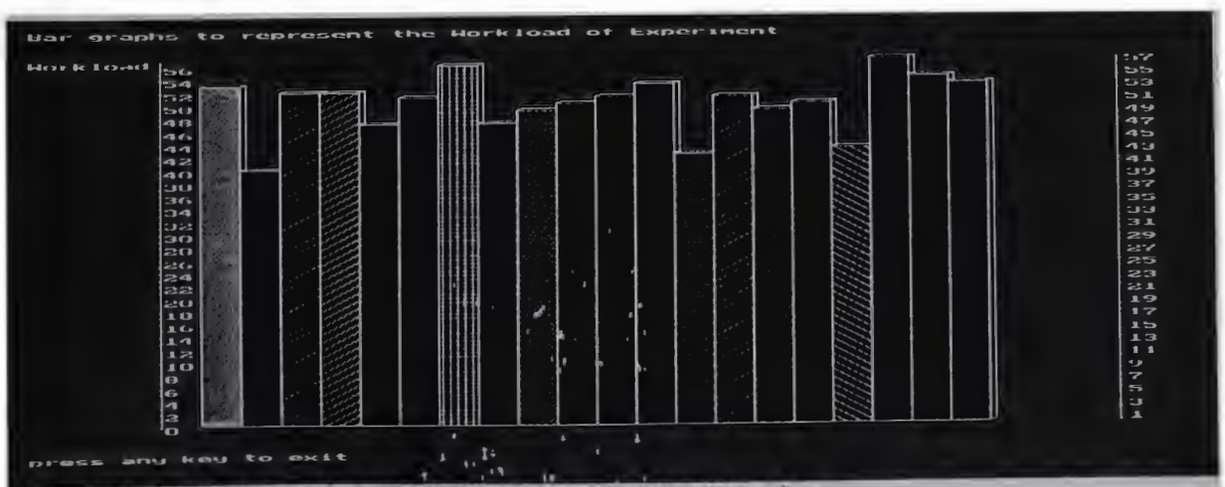

Figure B3: the third experiment with 20 processors and 1000 tuples

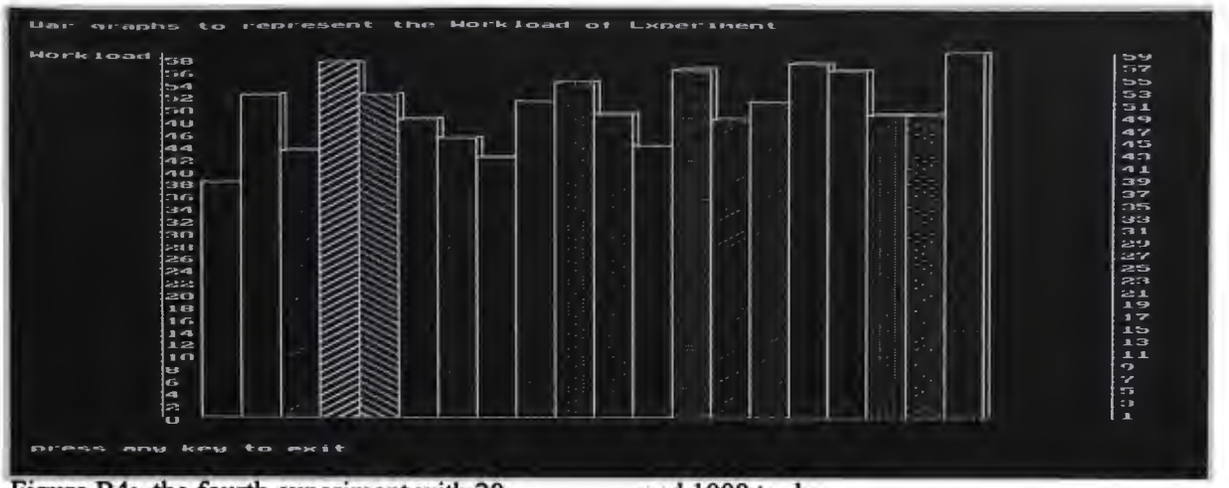

Figure B4: the fourth experiment with 20 processors and 1000 tuples

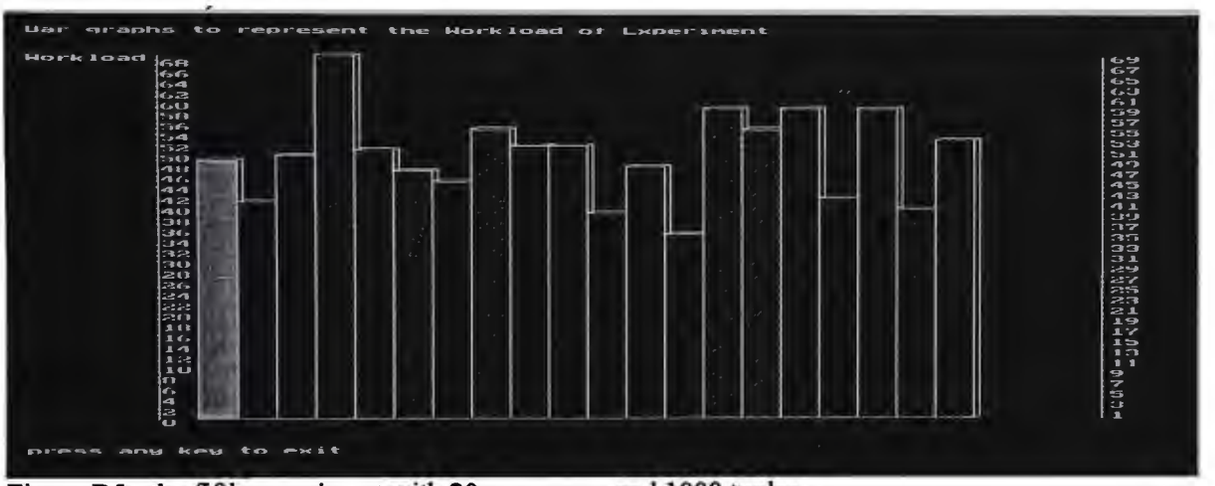

Figure B5: the fifth experiment with 20 processors and 1000 tuples

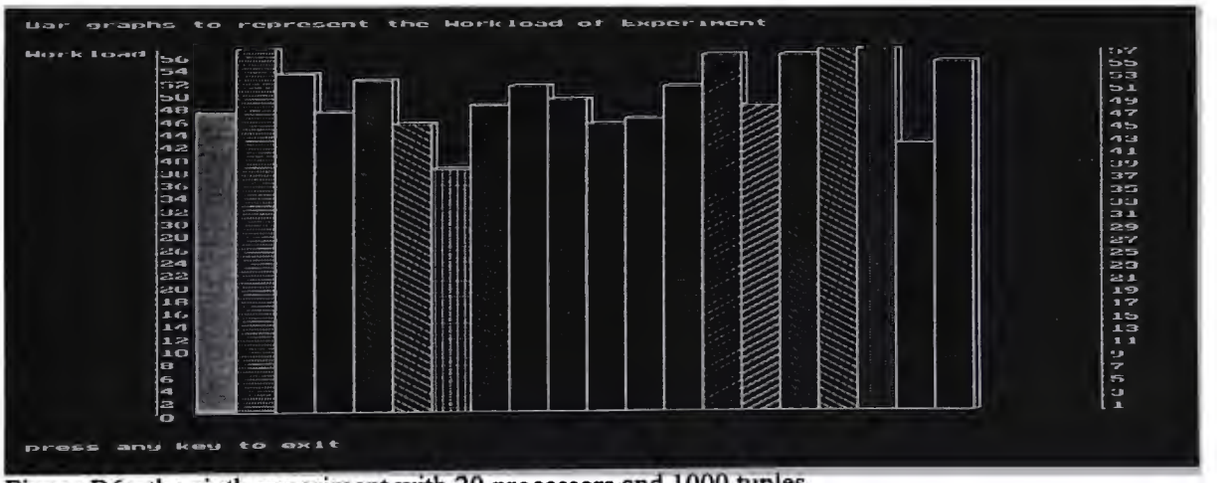

Figure B6: the sixth experiment with 20 processors and 1000 tuples

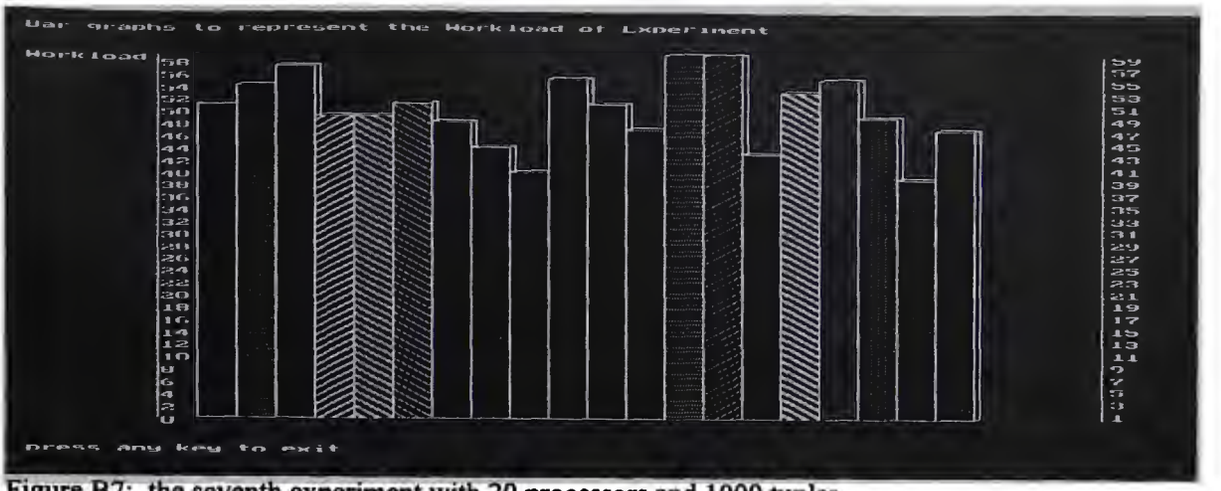

Figure B7: the seventh experiment with 20 processors and 1000 tuples

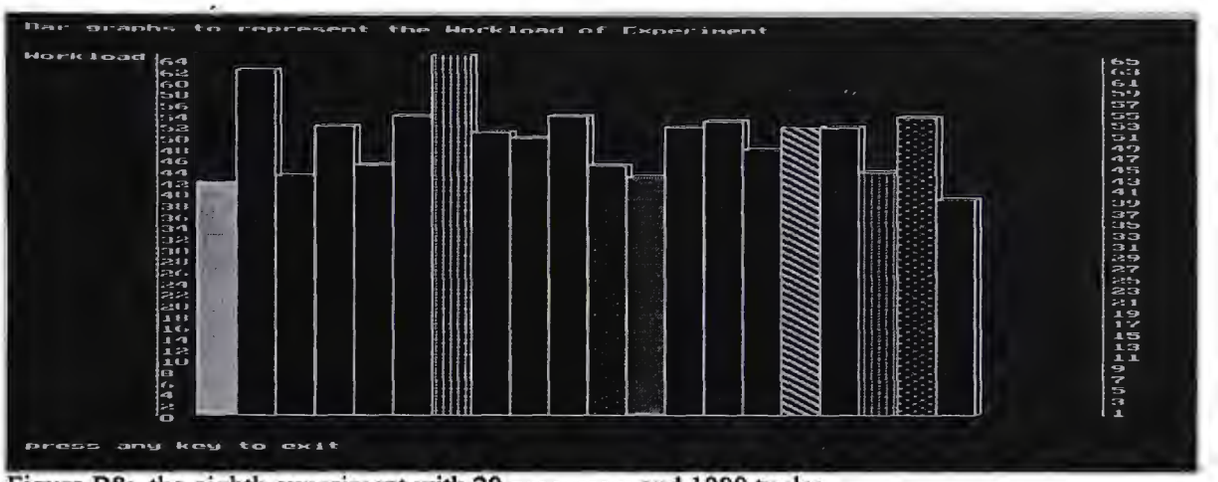

Figure B8: the eighth experiment with 20 processors and 1000 tuples

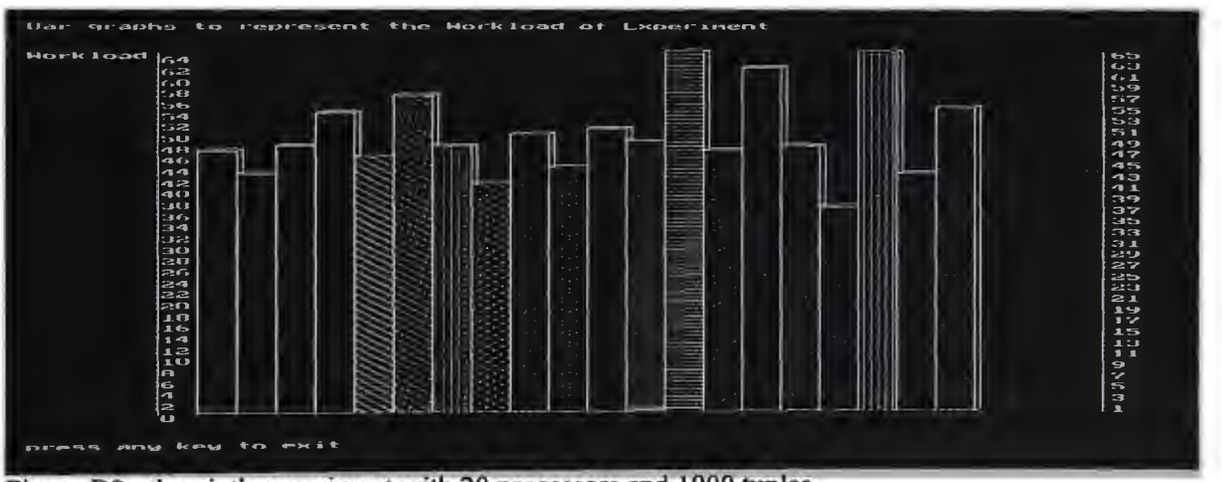

Figure B9: the ninth experiment with 20 processors and 1000 tuples

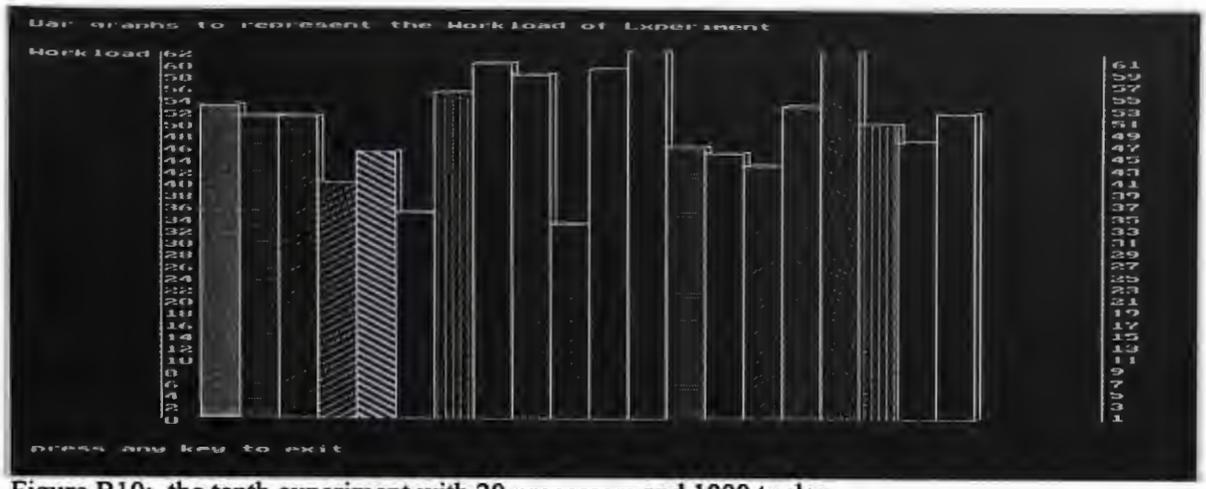

Figure B10: the tenth experiment with 20 processors and 1000 tuples

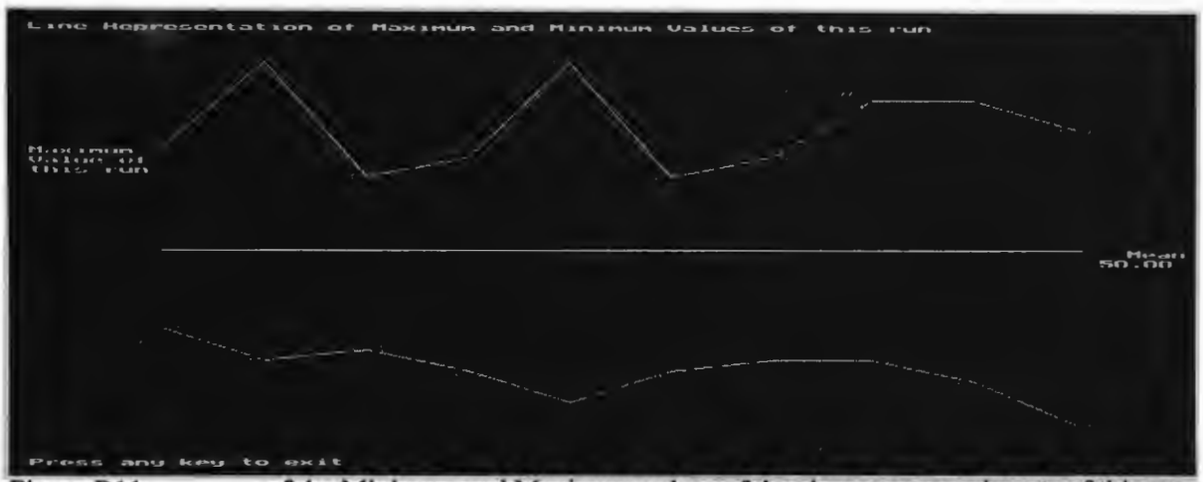

Figure B11: average of the Minimum and Maximum values of the above ten experiments of this run

- There are two points in the figure (minimum value and maximum value) for each experiment
- The relative positions of the points in the figure are determined by their maximum and minimum values and standard deviation from the mean load
- In the figure, White colour stands for Mean workload
- In the figure, Yellow colour stands for Maximum workload
- In the figure, Green colour stands for Minimum workload

As we mention before, more experiments can be carried out in one run to reduce the variation.

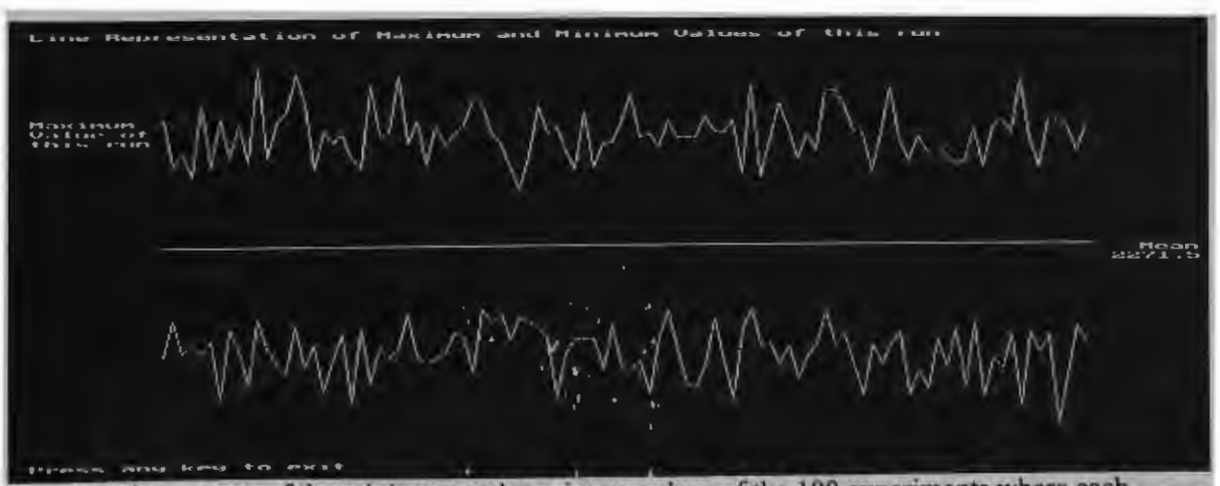

Figure B12: average of the minimum and maximum values of the 100 experiments where each experiment consists of 25 processors and 56,789 tuples

References

[1] Borland International, "Turbo vision for C++ 3.1: user's guide", 1992

[2] DeWitt D., et al, "The Gamma database machine project", *IEEE Trans. on Knowledge and Data Engineering*, Vol.2, No.1, March 1990

[3] Jiang Y. and C. H. C. Leung, "Site allocation for parallel query execution in locally distributed databases", *Proceedings of the 7th IASTED International Conference on Parallel and Distributed Computing and Systems*, Washington D. C., October 1995

[4] Leung C. H. C. and H. T. Ghogomu, "A high-performance parallel database architecture", *Proceedings 7th ACM International Conference on Supercomputing*, Tokyo, 1993

[5] Leung C. H. C. and K. H. Liu, "Limits to database speedup for data partitioned parallelisation in unary relational operations", submitted for publication, 1994

[6] Leung C. H. C. and K. H. Liu, "Skewness analysis of parallel join execution", submitted for publication, 1994

[7] Leung C. H. C. and D. Taniar, "Parallel query processing in object-oriented database systems", submitted for publication, 1994

[8] Liu K. H., Leung C. H. C., and Y. Jiang, "Analysis and taxonomy of skew in parallel database systems", *High Performance Computing Symposium'95*, Montreal, Canada, July 1995

[9] Mcgeoch C., "Analysing algorithms by simulation: variance reduction techniques and simulation speedups", *ACM Computing Surveys*, Vol.24, No.2, June 1992

[10] Metha M. and D. J. DeWitt, "Dynamic memory allocation for multiple-query workloads", *Proceedings of the 19th Very Large Databases Conference*, Dublin, Ireland, 1993

[11] Patel J. M., Carey M. J., and M. K. Vernon, "Accurate modelling of the hybrid hash join algorithm", *Proceedings ACM Sigmetrics Conference on Measurement & Modeling of Computer Systems*, Tennessee, U.S.A., may 1994, PP56-67