

*UNIVERSITY OF*

*NORTHERN BRITISH COLUMBIA*

Mani Samani

eslamis@unbc.ca

Abstract

This simulator simulates a distributed transaction processing database system. It currently can simulate a distributed system of 8 servers, each which store data, process transactions, and communicate on the network between them.

Distributed real-time database symulator

User Manual

Contents

[Distributed Database Simulator Manual 2](#_Toc484281609)

[Setting up the Database 3](#_Toc484281610)

[Using the simulator 4](#_Toc484281611)

[Parameters 4](#_Toc484281612)

[Results 6](#_Toc484281613)

[Implementation Notes 7](#_Toc484281614)

[Consumers and Suppliers instead of interface 7](#_Toc484281615)

[Components 8](#_Toc484281616)

[Server 8](#_Toc484281617)

[Disk 8](#_Toc484281618)

[Disk Job 8](#_Toc484281619)

[Processor 8](#_Toc484281620)

[Processor Job 8](#_Toc484281621)

[Lock Manager 9](#_Toc484281622)

[Network 9](#_Toc484281623)

[Transaction Manager 9](#_Toc484281624)

[Transaction 9](#_Toc484281625)

[Event Queue 9](#_Toc484281626)

[Event 10](#_Toc484281627)

[Protocols 11](#_Toc484281628)

[Two Phase Commit Protocol 11](#_Toc484281629)

[Transaction Life Cycle 12](#_Toc484281630)

[Lock Management 13](#_Toc484281631)

# Distributed Database Simulator Manual

The simulator has a few parameters that can be varied. These include: the number of pages, the arrival rate of transactions, the deadlock detection protocol, the deadlock resolution protocol, the priority protocol, the deadlock detection interval, maximum active transactions available in the system, and transactions’ update ratio.

It utilizes an event queue to simulate the progression of time. The various components of the simulator post events at future times into the event queue. The event queue is traversed in order of when events are scheduled to run.

# Setting up the Database

Download MySql database from <http://dev.mysql.com/downloads/mysql/>

This is the database server instance.

Download MySql Workbench from <http://dev.mysql.com/downloads/workbench/>

This is the database management software.

For both of these download the 64bit MSI installer (unless you have a 32bit machine)

Once both products are installed, open the workbench, connect to the local instance. Open a new query window (directly below the file menu) and copy and paste the code in the CreateDatabase.sql file provided in the project. This will create the database and the table that is used in the simulator.

# Using the simulator

To run an experiment, ensure the params.txt file has been set up correctly, and run the main method in the class Main. The params.txt file has to be set up in the following format:

LOGGING:false   
SEED:121,14331,1  
Topology:HyperCube

NumPages:280

ArrivalRate:300,400,500,600,700  
DDP:AgentDeadlockDetectionProtocol  
DRP:AgentDeadlockResolutionProtocol,PriorityDeadlockResolution  
PP:EarliestDeadlineFirst  
DetectionInterval:100  
MaxActiveTransactions:30  
AgentsHistoryLength:3  
UpdateRate:0.5

Settings separated by commas will create separate simulations. As you can see, SEED has 3 values, ArrivalRate has 5 values and DRP has 2 values. This will produce a total of 30 simulations, which will be run in parallel.

## Parameters

* **LOGGING:** Enables or disables the user interface. Turning this off greatly improves performance.
  + The possible values for this are: *True* or *False*.
* **SEED:** The seed the random number generator uses. This can be used to run the same parameters multiple times to get an average performance for that parameter combination.
  + The possible values for this are between: *0* and *264*
* **Topology:** This is the network arrangement of the servers.
  + Only *HyperCube* is available for this setting.
* **NumPages:** This is the total number of different pages in the simulation. Setting this lower should increase deadlocks, increasing this should decrease deadlocks as there will be less contention for each page.
  + The possible values for this are *4* – *232*, limited to the amount of ram on the machine.
* **ArrivalRate:** This setting changes the rate at which transactions enter the system.
  + Possible values are between: *0* and *232*.
* **DDP:** The deadlock detection protocol is the protocol used to detect deadlocks.
  + The possible values for this are: AgentDeadlockDetectionProtocol, MAEDD, TimeoutDeadlockDetection, and ChandyMisraHaasDDP.
* **DRP:** The deadlock resolution protocol is the protocol used to determine which transaction in a deadlock is to be aborted.
  + The possible values for this are: AgentDeadlockResolutionProtocol, PriorityDeadlockResolution, and FirstDeadlockResolution.
* **PP:** The priority protocol is used to prioritize transactions.
  + The possible values for this are: EarliestDeadlineFirst, FirstComeFirstServe, LeastSlackFirst, and RandomPriority.
* **DetectionInterval:** The detection interval is the time between deadlock detections.
  + The possible values for this are between: *0* – *232*. Practically though, they should be between *100* – *5000*.
* **MaxActiveTransactions:** The maximum active transactions is the total number of transactions actively performing in the system.
  + The possible values for this are between: *0* – *232*. Practically though, they should be between *10* – *100*.
* **AgentsHistoryLength:** This setting changes agents’ history keeping length in the protocols using agents.
  + The possible values for this are between: *0* – *232*.
* **UpdateRate:** This is the ratio of number of read pages to number of write pages in a transaction workload.
  + The possible values for this are between: *0* – *1*.

# Results

The results from the experiment are inserted into a database. The schema required is included in the project files. Run the CreateDatabase.sql file on a MySQL database to create the schema.

Ensure the database is hosted locally and there is a user created which matches the username and password found in the DBConnection class.

Each experiment (a set of parameters defined like the example in the previous section) will output an experiment number. All of the simulations ran from a single parameter file will have the same simulation number. You can use this number to compare the results from that experiment.

There is a separate project ResultsViewer which is used to graph the results. There are several classes in this project which graph different results on different axes. Ensure you use the correct experimentNumber when running the main method in any of these classes.

# Implementation Notes

## Consumers and Suppliers instead of interface

This project uses the Consumer and Supplier classes for interfaces between components. These classes are really for representing method references. Instead of creating interface classes for components to interface with each other, Consumers and Suppliers are used to references each other’s methods.

For example, the GUI needs to display what’s happening in the simulation to the user, but because of good programming practice we do not let the GUI classes have direct reference to the simulation and we don’t let the simulation have direct references to the GUI classes. Instead Consumers are used for the simulation to push information to the GUI.

In the Main class you will see

Consumer<Integer> updateTime = null;

…

updateTime = gui::updateTime; //This refers to the updateTime(int time) method of the GUI class

…

SimSetupParams params = new SimSetupParams(SEED, numPages, maxActiveTrans, 8, arrivalRate, DDP, DRP, log, stats, getSleepTime, updateTime);

This updateTime Consumer is really a method reference and is used for the simulation (specifically the Event Queue) to pass the current simulation time to the GUI.

These Consumers and Suppliers are used in various other places in the simulator as well; such as, for the deadlock detection protocol to pass the deadlock to the deadlock resolution protocol.

# Components

## Server

The server class holds the transaction manager, lock manager, disk, processor, and network interface.

Its main task is to be a central hub for communication between the components mentioned above and to relay messages to and from the network interface.

Each server holds only one of the five components mentioned above.

There are always 8 servers involved in every simulation. They are always involved in a hypercube topology.

The various servers in the simulation hold different pages. There are two copies of every page in the system. Servers 0 and 4 hold the first quarter, servers 1 and 5 hold the second quarter, etc.

## Disk

The disk holds a number of pages. The number of pages can be set in the parameters file.

The disk handles read and write jobs sent to it from the transaction manager.

The disk can process one job at a time.

When it begins processing a job, it posts an event to the event queue for when the job is completed.

### Disk Job

Disk jobs are posted by the transaction manager to load pages from, or save pages to, the disk.

## Processor

The processor processes a single job at a time. When it begins processing a job, it posts an event to some point in the future. When that event triggers, the transaction manager is informed the job is done, and the processor checks its queue for the next processor job.

### Processor Job

The processor job holds a page number, and a transaction ID. The transaction manager posts processing jobs to the processor on the behalf of transactions so they can process their required pages.

## Lock Manager

The lock manager is responsible for ensuring isolation between concurrent transactions.

It does this by ensuring there is at most one exclusive lock on a local page at a time. There can be multiple shared locks on pages at any given time.

The lock manager ensures isolation.

## Network

The network deals with routing messages between servers.

The network consists of a number of network connections (NetworkConnection.java). Each network connection has a certain bandwidth, there can only be a certain number of messages on each connection at a time. Each connection has an incoming message queue. When a message is to be sent on the connection, it is added to the queue, and when there is room on the wire, the message is sent and an event is posted into the event queue for when it arrives.

Each server has a connection to a network interface, the interface consists of a number of network connections to its neighboring servers.

## Transaction Manager

The transaction manager deals with progressing the transactions through their stages of: acquiring locks on the required pages, reading those pages from the disk, processing the pages, then writing the results back to the disk.

### Transaction

Transactions hold a number of read pages and a number of write pages. They are produced at each server at consistent but random intervals. They may need to access pages which are not at that server, when this occurs they create cohort transactions which are sent to the required servers.

## Event Queue

This class holds a sorted list of events. Events are taken from the front and each ‘ran’. New events are placed in the queue in the order which they are to be ran.

### Event

Each event contains a number of operations to perform on specific components on behalf of a transaction, and a time when these operations occur.

# Protocols

There are various protocols that are used in the simulator.

## Two Phase Commit Protocol

This protocol syncs the cohorts up with their master transaction. It ensures that the master does not commit until all cohorts are ready to commit. When this occurs the master commits and informs all cohorts to commit, then the transaction begins performing its write jobs.

# Transaction Life Cycle

Transaction created with a random number of pages between 1 and 8. Some of these pages are read pages, and some are write pages.

Transaction Created

First, the transaction must acquire locks on all of its pages. For read pages, they acquire shared locks, for write pages they acquire exclusive ones.

Read Phase

Acquire Locks

The transaction then begins reading each of its pages independent of one another.

Complete

Write Phase

Once each page is read from the disk, it is processed by the processor. Process pages happens independent of one another.

Commit

Processing Phase

Once all of the transactions pages have been processed, the transaction is able to commit.

After committing, the transactions writes each of its pages to disk. This write occurs on all servers where this page resides.

Once all the pages have been written to disk, the transaction completes and releases all of its locks.

What is left out from this is the creation of cohort transactions. If pages, that the transaction must read, are not on the server where the transaction arrived, a cohort transaction is sent to a server with those pages. Before a transaction is able to commit, all its cohorts must be ready to commit as-well.

# Lock Management

Proper locking is necessary to ensure the atomicity constraint on database transactions. The lock manager in this simulator uses progressive locking; the locks a transaction requires are independent from one another, they are acquired as soon as they are available, and the page begins processing after the lock is acquired. This allows the transaction to begin working on some of its pages, before it has locks on all of its pages. This creates the possibility of deadlocks to occur, as multiple transactions can begin acquiring locks on separate pages and then end up waiting for the locks the others hold.

Example:

The arrows indicate a transaction waiting on another transaction.

T4

Pages: 3, 0

Locked: 3

Waiting for: 0

T3

Pages: 2, 3

Locked: 2

Waiting for: 3

T2

Pages: 1, 2

Locked: 1

Waiting for: 2

T1

Pages: 0, 1

Locked: 0

Waiting for: 1