1 Goals

To study the internals of database systems as an introduction to research and as a basis for rational performance tuning.

The study of internals will concern topics at the intersection of database system, operating system, and distributed computing research and development. Specific to databases is the support of the notion of transaction: a multi-step atomic unit of work that must appear to execute in isolation and in an all-or-nothing manner. The theory and practice of transaction processing is the problem of making this happen efficiently and reliably.

Tuning is the activity of making your database system run faster. The capable tuner must understand the internals and externals of a database system well enough to understand what could be affecting the performance of a database application. We will see that interactions between different levels of the system, e.g., index design and concurrency control, are extremely important, so will require a new optic on database management design as well
as introduce new research issues. Our discussion of tuning will range from
the hardware to conceptual design, touching on operating systems, transac-
tional subcomponents, index selection, query reformulation, normalization
decisions, and the comparative advantage of object-oriented database sys-
tems. This portion of the course will be heavily sprinkled with case studies
from database tuning in biotech, telecommunications, and finance.

Because of my recent research interests, this year will include frequent
discussions of

- “decision support,” the activity of exploring aggregated data to find
trends; and
- “array databases,” the extension of relational systems to support or-
dered data such as time series in finance

2 Mechanics

YOU MUST BE ENROLLED IN THIS CLASS TO SIT IN ON THE LEC-
TURES.

2.1 Texts and Notes

The first text will be used for the first half of the course and the second text
in the second half. The notes will be used throughout the course.

- *Concurrency Control and Recovery in Database Systems* by Bernstein,
  Now available for free at: http://research.microsoft.com/pubs/ccontrol/
- *Database Tuning: Principles Experiments and Troubleshooting Tech-
niques* Dennis Shasha and Philippe Bonnet, Morgan Kaufmann Publish-

There are also two optional books which are very nicely written:

- *Principles of Distributed Database Systems* M. Tamer Ozsu and Patrick
  Valduriez, Prentice-Hall, 1999. A nice introduction to the general
  problem of distributed database systems.

2.2 Prerequisites

Fundamental Algorithms I plus Data Base Systems I or equivalent (first 6 chapters of Ullman). If you don’t have the database prerequisites, then you may take the course, but you must be responsible for understanding material covered in Database I: a reading knowledge of SQL and basic familiarity with indexes and third normal form.

2.3 Course Requirements

(2 or 3) problem sets (30%), project (70%).

LATE HOMEWORKS OR PROJECTS WILL NOT BE ACCEPTED without a note from your physician or from your employer. (We will discuss the solutions on the day you hand in the assignment. That’s why I don’t want any late homeworks. As for projects, this is a question of fairness.)

On the other hand, collaboration on the problem sets IS allowed. You may work together with one other partner and sign both of your names to a single submitted homework. Both of you will receive the grade that the homework merits.

3 Syllabus — times are estimated

1. Overview of transaction processing, distributed systems, and tuning (1 week)

2. Principles of concurrency control for centralized, distributed, and replicated databases. (3 weeks)

3. Principles of logging, recovery, and commit protocols. (3 weeks)

4. Database Tuning (7 weeks)
Tuning principles.
Hardware, operating system, and transaction subsystem
Transaction Chopping
Index tuning
Tuning relational systems
Tuning object-oriented systems
Benchmarking
Case Studies from Wall Street and Elsewhere

5. Special topics: data warehousing, data mining, main memory databases.

4 Project

Your project is due the second to last class. The three possible project topics (you choose one) are:

1. Distributed replicated concurrency control and recovery. You may do this in a team of two.

2. An experimental or theoretical study of tuning issues.

3. An individual term paper. Note, however, that it must be good.

4.1 Possibility 1 — Replicated Concurrency Control and Recovery (RepCRec for short)

Implement a distributed concurrency control algorithm and commit algorithm with replication. Variables x1, ..., x20 (that is, only 20 variables in whole database — the numbers between 1 and 20 will be referred to as indexes below). Sites are 1 to 10. A copy is indicated by a dot. Thus, x6.2 is the copy of variable x6 at site 2. The odd indexed ones are at one site each (i.e., index number mod 10 plus 1). Even indexed ones are at all sites. Each variable xi is initialized to the value 10i.

Implement the available copies approach to replication using two phase locking (using read and write locks) at each site and validation at commit time. Read-only transactions should use multiversion read consistency. You should detect deadlocks by using timeouts (it’s easier) of 10 steps where a step is marked by a tick command. You may assume that the processors
work in lock-step That is, you may assume that all operations between ticks occur concurrently.

Input instructions come from a file in or the standard input, output goes to a file out. (That means your algorithms may not look ahead in the input.) Input instructions occurring in one step begin at a new line and end with a carriage return. Thus, there will be several activities in each step, though only one per transaction. (Obviously, some of these activities may be blocked due to conflicting locks.) Input is of the form:

begin(T1) says that T1 begins
beginRO(T3) says that T3 is read-only

R(T1, x4) says transaction 1 wishes to read x4 (provided it can get the locks or provided it doesn’t need the locks (for read-only transactions)). It should read any up copy and return the current value.

W(T1, x6,v) says transaction 1 wishes to write all copies of x6 (provided it can get the locks) with the value v.

dump() gives the committed values of all copies of all variables at all sites, sorted per site.

dump(i) gives the committed values of all copies of all variables at site i.

dump(xj) gives the committed values of all copies of variable xj at all sites.

dump(T1) sees whether T1 can commit.

Your program goes through a two-phase commit with pre-commit, etc. It returns whether a transaction commits, is blocked, or aborts and explains why (two sentences max).

fail(6) says site 6 fails. (This is not issued by a transaction, but is just an event that the tester will execute.)

recover(7) says site 7 recovers. (Again, a tester-caused event) We discuss this further below.

tick(n) says the clock advances by n time steps.

Example (partial script with six steps in which transactions T1 and T2 commit, and one of T3 and T4 may commit)
begin(T1); begin(T2); begin(T3) tick
W(T1, x1,5); W(T3, x2,32); tick(1)
W(T2, x1,17); tick(1) — will have to wait, we won’t issue more from T2 while it waits

end(T1); begin(T4); tick(1) — T2’s write succeeds now

W(T4, x4,35); W(T3, x5,21); end(T2); tick(1) — T2 can commit

W(T4,x2,21); W(T3,x4,23); tick(1) — both of these are delayed, one or both are aborted

Your program should consist of two parts: a single transaction manager that translates read and write requests on variables to read and write requests on copies using the available copy algorithm described in the notes. The transaction manager never fails. (Having a single global transaction manager that never fails is a simplification of reality, but it is not too hard to get rid of that assumption.)

If the TM requests a read for transaction T and cannot get it due to failure, the TM should try another site (all in the same step). If no relevant site is available, then T must wait. T may also have to wait for conflicting locks. Thus the TM may accumulate an input command for T and will try it on the next tick. If T waits long enough, then it may abort. While T is blocked (whether waiting for a lock to be released or a failure to be cleared), no new operations for T will appear, so the buffer size for messages from each transaction can be of size 1.

A data and lock manager at each site performs concurrency control. You should implement a simple message buffer at each site. In one step each working DM reads its message buffer from the TM in that step, performs some processing and perhaps responds to the TM. The TM won’t send more than one message to a DM in one step though that message may contain several operations each from a different transaction.

Failures are indicated only by the fail statement. The site should forget any previous messages sent to it (because these are held in volatile storage) as well as lock information. If a site fails and recovers, the DM would normally perform local recovery first (perhaps by asking the TM about transactions that the DM holds pre-committed but not yet committed), but this is unnecessary since, in the simulation model, commits are atomic with respect to failures. This makes all non-replicated variables available for reads and writes. Regarding replicated variables, the site makes them available for writing, but not reading. In fact, reads will not be allowed until a committed write takes place (see notes on recovery when using the available copies
algorithm).

During the execution, your program should say which transactions commit and which abort and for what reason. For debugging purposes you should implement the command querystate which will give the state of each DM and the TM as well as the data distribution and data values. Finally, each read that occurs should show the value read.

4.1.1 Recommendations from a previous grader regarding programming project

1) Running the projects: I should be able to sit down and access the projects either at my terminal or at one of the pc's in the education building.

   a) Don't have the project on an account that has a fancy login script that requires a particular terminal.

   b) Either the executable file should be in the account, or source files and explicit instructions on compiling and linking.

   c) Don't expect me to understand intricacies of the NYU computers. I need explicit instructions on how to access, setup and run the project (i.e., if I need to set the terminal type, tell me exactly how).

   d) Don't assume that I have accounts on any machines.

   e) Don't assume that I have a P.C. and a modem so that I can access your computer.

   f) I don't want to load your project into my account.

   g) Don't develop the project on a "sensitive" account. At any rate, I am an honest kind of person and I will leave a message saying that I finished grading if you wish.

   h) If you develop the project on a P.C., make sure that it runs on one of the P.C.s in the education building. Tell me which one to run it on. Please note that they have no computers that accept high-density (1.2 mb) 5 1/4 inch floppy disks (this last point may be out of date, but you get the idea).

   i) Please tell me about any peculiarities about your project that might make me assume that it has a fatal bug (i.e., under what conditions will it decide that otherwise valid input is incorrect).

2) Interpreting the results: While you understand exactly how your pro-
gram works, I need to be convinced that the program does what it should. Every relevant aspect of the program’s behavior should be reported. "relevant aspect" should be given a liberal interpretation.

a) uninterpretable aspects of the program’s behavior will be considered to be incorrect.

b) it should not be my tough luck if the project is difficult to grade because of scanty status reports, even if I can cleverly deduce the program’s behavior.

After working so long to get the project finished, it is silly to not do the extra small effort to make the project easy to grade.

4.2 Possibility 2 — Benchmarking/Tuning Project

Take a section of the tuning book and see whether its recommendations make quantitative sense on a real system that is available to you. Use substantial relations, e.g. 1 million rows and up. Specify the database management system, operating system, hardware platform including disks, memory size, and processor. Back up your conclusions with graphs drawn from real data. The example benchmarks should come from some TPC benchmark found in the transaction processing council’s web page. I recommend TPCH. You can download one database system from www.kx.com. Use at least two systems, compare their performances before tuning and then tune them up as much as possible and give me the performance afterwards.

4.3 Possibility 3 — Paper

Each student wanting to write a paper will read several research papers, write a survey-style paper (7-10 single-spaced pages) describing the results, create four non-trivial problems based on that material and give their solutions. (The problems should be at the level of the best problems in the problem sets I give you.) Feel free to do original work, but it must be grounded in the papers you read.

The basic idea will be to follow the trail of one paper in some area, e.g. fault tolerance in the real world, the computerization of medical records, spatial access methods, performance results in concurrency control on real computers, backup and recovery for parallel transactions, extensions to 2
phase locking, semantic serializability theory, replication management, real-
time databases. In general, the topic should have some interesting (to be
determined by me) distributed database or tuning aspect to it. It should
not be a fuzzy topic, because then your problems and hence your grade will
be poor.

5 Project Schedule — depends on project you choose

5.1 Schedule for Programming Project

Last class in September: Letter of intent that you are going to do program-
ning project. Partner chosen if any.

   Last class in October: status reports.

   Project is due on December 2. Between December 2 and December 8, your project will be graded. You will make an appointment that week with
   the grader. We will figure out a randomized way to do this.

5.2 Schedule for Benchmarking/Tuning Project

Last class in September: project outline (should fit on one page). Tuning
problem you intend to address. System you plan to use and experimental
question you plan to ask. This must be approved before you go on.

   Last Class in October: status report. How are you doing? Any show-
stoppers.

   First Class In December: final report to me and begin to set up appoint-
ment for testing with grader.

5.3 Schedule for Papers

Last class in September, outline of strategy (less than one page). Problem
whose literature you intend to explore. Papers chosen. Some motivation.

   Last Class in October, status report. Papers read, anything you have
written up. If you are finished by this time, you will be able to give your
talk early on.

   First Class in December, final report to me.