1. Introduction

CSE 593 Transaction Processing
Philip A. Bernstein

Outline
1. The Basics
2. ACID Properties
3. Atomicity and Two-Phase Commit
4. Availability
5. Performance
6. Styles of System

1.1 The Basics - What’s a Transaction?

- The *execution* of a program that performs an administrative function by accessing a *shared database*, usually on behalf of an *on-line* user.

**Examples**
- Reserve an airline seat. Buy an airline ticket
- Withdraw money from an ATM.
- Verify a credit card sale.
- Order an item from an Internet retailer
- Download a video clip and pay for it
- Play a bid at an on-line auction

The “ities” are What Makes Transaction Processing (TP) Hard

- Reliability - system should rarely fail
- Availability - system must be up all the time
- Response time - within 1-2 seconds
- Throughput - thousands of transactions/second
- Scalability - start small, ramp up to Internet-scale
- Security – for confidentiality and high finance
- Configurability - for above requirements + low cost
- Atomicity - no partial results
- Durability - a transaction is a legal contract
- Distribution - of users and data

What Makes TP Important?

- It’s at the core of electronic commerce
- Most medium-to-large businesses use TP for their production systems. The business can’t operate without it.
- It’s a *huge* slice of the computer system market — over $50B/year. Probably the single largest application of computers.

TP System Infrastructure

- User’s viewpoint
  - Enter a request from a browser or other display device
  - The system performs some application-specific work, which includes database accesses
  - Receive a reply (usually, but not always)
- The TP system ensures that each transaction
  - is an independent unit of work
  - executes exactly once, and
  - produces permanent results.
- TP system makes it easy to program transactions
- TP system has tools to make it easy to manage
TP System Infrastructure …
Defines System and Application Structure

End-User → Presentation Manager → Front-End (Client)

Workflow Control (routes requests) → Back-End (Server)

Transaction Program
Database System

System Characteristics
- Typically < 100 transaction types per application
- Transaction size has high variance. Typically,
  - 0-30 disk accesses
  - 10K - 1M instructions executed
  - 2-20 messages
- A large-scale example: airline reservations
  - 150,000 active display devices
  - plus indirect access via Internet travel agents
  - thousands of disk drives
  - 3000 transactions per second, peak

Application Servers
- A software product to create, execute and manage TP applications
- Formerly called TP monitors. Some people say App Server = TP monitor + web functionality.
- Programmer writes an app. to process a single request. App Server scales it up to a large, distributed system
  - E.g. application developer writes programs to debit a checking account and verify a credit card purchase.
  - App Server helps system engineer deploy it to 10s/100s of servers and 10Ks of displays
  - App Server helps system engineer deploy it on the Internet, accessible from web browsers

Application Servers (cont’d)
- Components include
  - an application programming interface (API)
    (e.g., Enterprise Java Beans)
  - tools for program development
  - tools for system management (app deployment, fault & performance monitoring, user mgmt, etc.)

App Server Architecture, pre-Web
- Boxes below are distributed on an intranet

Presentation Server → Message Inputs

Requests → Workflow Controller

Network

Transaction Server → Queues

Automated Teller Machine (ATM) Application Example

Bank Branch 1
Bank Branch 2
Bank Branch 500

ATM → Workflow Controller

CIRRUS Accounts
Checking Accounts
Credit Card Accounts
Loan Accounts
### System Software Vendor’s View

- TP is partly a component product problem
  - Hardware
  - Operating system
  - Database system
  - Application Server
- TP is partly a system engineering problem
  - Getting all those components to work together to produce a system with all those "ilities".
- This course focuses primarily on Database System and Application Server

### Outline

1. The Basics
2. ACID Properties
3. Atomicity and Two-Phase Commit
4. Availability
5. Performance
6. Styles of System

### 1.2 The ACID Properties

- Transactions have 4 main properties
  - Atomicity - all or nothing
  - Consistency - preserve database integrity
  - Isolation - execute as if they were run alone
  - Durability - results aren’t lost by a failure

### Atomicity

- All-or-nothing, no partial results.
  - E.g. in a money transfer, debit one account, credit the other. Either debit and credit both run, or neither runs.
  - Successful completion is called *Commit*.
  - Transaction failure is called *Abort*.
- Commit and abort are irrevocable actions.
- An Abort *undoes* operations that already executed
  - For database operations, restore the data’s previous value from before the transaction
  - But some real world operations are not undoable. Examples - transfer money, print ticket, fire missile
Example - ATM Dispenses Money (a non-undoable operation)

```
T1: Start
... Dispense Money
Commit
```

System crashes
Transaction aborts
Money is dispensed

```
T1: Start
... Dispense Money
Commit
```

Deferred operation never gets executed

```
T2: Start
... Get input from display
Commit
```

User reads output
... User enters input
Brain transport

Reading Uncommitted Output Isn’t Undoable

```
T1: Start
... Display output
... If error, Abort
```

Consistency

- Every transaction should maintain DB consistency
  - Referential integrity - E.g. each order references an existing customer number and existing part numbers
  - The books balance (debits = credits, assets = liabilities)

> Consistency preservation is a property of a transaction, not of the TP system
  (unlike the A, I, and D of ACID)

- If each transaction maintains consistency, then serial executions of transactions do too.

Compensating Transactions

- A transaction that reverses the effect of another transaction (that committed). For example,
  - “Adjustment” in a financial system
  - Annul a marriage
- Not all transactions have complete compensations
  - E.g. Certain money transfers (cf. “The Firm”)
  - E.g. Fire missile, cancel contract
  - Contract law has a lot to say about appropriate compensations

△ A well-designed TP application should have a compensation for every transaction type

Consistency Preservation Example

```
T1: Start;
A = Read(x);
A = A - 1;
Write(y, A);
Commit;
```

```
T2: Start;
B = Read(x);
C = Read(y);
If (B > C+1) then B = B - 1;
Write(x, B);
Commit;
```

- Consistency predicate is x > y.
- Serial executions preserve consistency. Interleaved executions may not.
- H = r₁[x] r₂[x] r₂[y] w₂[x] w₁[y]
  - e.g. try it with x=4 and y=2 initially

Some Notation

- ri[x] = Read(x) by transaction Ti
- wi[x] = Write(x) by transaction Ti
- ci = Commit by transaction Ti
- ai = Abort by transaction Ti
- A history is a sequence of such operations, in the order that the database system processed them.
Isolation

- Intuitively, the effect of a set of transactions should be the same as if they ran independently.
- Formally, an interleaved execution of transactions is *serializable* if its effect is equivalent to a serial one.
- Implies a user view where the system runs each user’s transaction stand-alone.
- Of course, transactions in fact run with lots of concurrency, to use device parallelism.

A Serializability Example

<table>
<thead>
<tr>
<th>T1: Start;</th>
<th>T2: Start;</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = Read(x);</td>
<td>B = Read(x);</td>
</tr>
<tr>
<td>A = A + 1;</td>
<td>B = B + 1;</td>
</tr>
<tr>
<td>Write(x, A);</td>
<td>Write(y, B);</td>
</tr>
<tr>
<td>Commit;</td>
<td>Commit;</td>
</tr>
</tbody>
</table>

- \( H = r_1[x] r_2[x] w_1[x] c_1 w_2[y] c_2 \)
- \( H \) is equivalent to executing \( T_2 \) followed by \( T_1 \)
- Note, \( H \) is *not* equivalent to \( T_1 \) followed by \( T_2 \)
- Also, note that \( T_1 \) started before \( T_2 \) and finished before \( T_2 \), yet the effect is that \( T_2 \) ran first.

Serializability Examples (cont’d)

- Client must control the relative order of transactions, using handshakes (wait for \( T_1 \) to commit before submitting \( T_2 \)).
- Some more serializable executions:
  - \( r_1[x] r_2[y] w_2[y] w_1[x] \equiv T_1 T_2 \)
  - \( r_1[y] r_2[y] w_2[y] w_1[x] \equiv T_1 T_2 \)
  - \( r_1[x] r_2[y] w_2[y] w_1[y] \equiv T_2 T_1 \equiv T_1 T_2 \)
- Serializability says the execution is equivalent to some serial order, not necessarily to all serial orders.

Non-Serializable Examples

- \( r_1[x] r_2[x] w_2[x] w_1[x] \) *(race condition)*
  - e.g. \( T_1 \) and \( T_2 \) are each adding 100 to \( x \)
- \( r_1[x] r_2[y] w_2[x] w_1[y] \)
  - e.g. each transaction is trying to make \( x = y \), but the interleaved effect is a swap
- \( r_1[x] r_1[y] w_1[x] r_2[x] r_2[y] c_2 w_1[y] c_1 \) *(inconsistent retrieval)*
  - e.g. \( T_1 \) is moving $100 from \( x \) to \( y \).
  - \( T_2 \) sees only half of the result of \( T_1 \)
- Compare to the OS view of synchronization

Durability

- When a transaction commits, its results will survive failures (e.g. of the application, OS, DB system … even of the disk).
- Makes it possible for a transaction to be a legal contract.
- Implementation is usually via a log
  - DB system writes all transaction updates to its log
  - to commit, it adds a record “commit(\( T_i \))” to the log
  - when the commit record is on disk, the transaction is committed.
  - system waits for disk ack before acking to user

Outline

- ✓ 1. The Basics
- ✓ 2. ACID Properties
- ✓ 3. Atomicity and Two-Phase Commit
- ✓ 4. Availability
- 5. Performance
- 6. Styles of System
1.3 Atomicity and Two-Phase Commit

- Distributed systems make atomicity harder
- Suppose a transaction updates data managed by two DB systems.
- One DB system could commit the transaction, but a failure could prevent the other system from committing.
- The solution is the two-phase commit protocol.
- Abstract “DB system” by resource manager (could be a SQL DBMS, message mgr, queue mgr, OO DBMS, etc.)

Two-Phase Commit

- Main idea - all resource managers (RMs) save a durable copy of the transaction’s updates before any of them commit.
- If one RM fails after another commits, the failed RM can still commit after it recovers.
- The protocol to commit transaction T
  - Phase 1 - T’s coordinator asks all participant RMs to “prepare the transaction”. Participant RMs replies “prepared” after T’s updates are durable.
  - Phase 2 - After receiving “prepared” from all participant RMs, the coordinator tells all participant RMs to commit.

1.4 Availability

- Fraction of time system is able to do useful work
- Some systems are very sensitive to downtime
  - airline reservation, stock exchange, telephone switching
  - downtime is front page news

<table>
<thead>
<tr>
<th>Downtime</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 hour/day</td>
<td>95.8%</td>
</tr>
<tr>
<td>1 hour/week</td>
<td>99.41%</td>
</tr>
<tr>
<td>1 hour/month</td>
<td>99.86%</td>
</tr>
<tr>
<td>1 hour/year</td>
<td>99.9886%</td>
</tr>
<tr>
<td>1 hour/20years</td>
<td>99.99942%</td>
</tr>
</tbody>
</table>

- Contributing factors
  - failures due to environment, system mgmt, h/w, s/w
  - recovery time

1.5 Performance Requirements

- Measured in max transaction per second (tps) or per minute (tpm), and dollars per tps or tpm.
- Dollars measured by list purchase price plus 5 year vendor maintenance (“cost of ownership”)
- Workload has this profile:
  - 10% application server plus application
  - 30% communications system (not counting presentation)
  - 50% DB system
- TP Performance Council (TPC) sets standards
- TPC A & B (’89-’95), now TPC C & W
TPC-A/B — Bank Tellers
- Obsolete (a retired standard), but interesting
- Input is 100 byte message requesting deposit/withdrawal
- Database tables = [Accounts, Tellers, Branches, History]

Start
Read message from terminal (100 bytes)
Read+write account record (random access)
Write history record (sequential access)
Read+write teller record (random access)
Read+write branch record (random access)
Write message to terminal (200 bytes)
Commit

- End of history and branch records are bottlenecks

The TPC-C Order-Entry Benchmark

<table>
<thead>
<tr>
<th>Table</th>
<th>Rows/Whse</th>
<th>Bytes/row</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warehouse</td>
<td>1</td>
<td>89</td>
</tr>
<tr>
<td>District</td>
<td>10</td>
<td>95</td>
</tr>
<tr>
<td>Customer</td>
<td>30K</td>
<td>655</td>
</tr>
<tr>
<td>History</td>
<td>30K</td>
<td>46</td>
</tr>
<tr>
<td>Order</td>
<td>30K</td>
<td>24</td>
</tr>
<tr>
<td>New-Order</td>
<td>9K</td>
<td>8</td>
</tr>
<tr>
<td>OrderLine</td>
<td>300K</td>
<td>54</td>
</tr>
<tr>
<td>Stock</td>
<td>100K</td>
<td>306</td>
</tr>
<tr>
<td>Item</td>
<td>100K</td>
<td>82</td>
</tr>
</tbody>
</table>

- TPC-C uses heavier weight transactions

TPC-C Transactions
- New-Order
  - Get records describing a warehouse, customer, & district
  - Update the district
  - Increment next available order number
  - Insert record into Order and New-Order tables
  - For 5-15 items, get item record, get/update Stock record
  - Insert Order-Line Record
- Payment, Order-Status, Delivery, Stock-Level have similar complexity, with different frequencies
- $\text{tpmC} = \text{number of New-Order transaction per min.}$

Comments on TPC-C
- Enables apples-to-apples comparison of TP systems
- Does not predict how your application will run, or how much hardware you will need, or which system will work best on your workload
- Not all vendors optimize for TPC-C. E.g., IBM has claimed DB2 is optimized for a different workload, so they have only recently published TPC numbers

Typical TPC-C Numbers
- $\$10 - $50 / \text{tpmC}$. Uniform spread across the range.
  - Top 49 price/performance results on MS SQL Server & Win 2000.
  - Fujitsu at $21. Sybase at $27. IBM DB2 at $32. Oracle at $36
- System cost $153K (Intergraph) - $14.2M (IBM)
- Examples of high throughput
  - Compaq 550K $\text{tpmC}$, $10.4M$, $210/\text{tpmC}$ (MS SQL, MS COM+)
  - IBM 441K $\text{tpmC}$, $14.2M$, $32/\text{tpmC}$ (IBM DB2, MS COM+)
- Examples of low cost (all use MS SQL Server, COM+)
  - Compaq, 20.2K $\text{tpmC}$, $201K$, $10/\text{tpmC}$
  - Dell, 30.2K $\text{tpmC}$, $335K$, $11/\text{tpmC}$
  - HP, 33.1K $\text{tpmC}$, $393K$, $12/\text{tpmC}$
- Results are very sensitive to date published.

TPC/W – Web Retailer
- Introduced 12/99. One published measurement so far.
- Features - DB accesses to generate dynamic web pages, secure UI, secure payments (via secure socket layer (SSL))
- Scale factor: 1K – 10M items (in the catalog).
- Web Interactions per sec (WIPS) @ ScaleFactor
  - IBM: 1262 WIPS@ 10,000; $277 / WIPS; $350K total
- Profiles - shop (WIPS), browse (WIPSb), order (WIPSo)
- Tables – {Customer, Order, Order-Line, Item, Author, CreditCardTxns, Address, Country}
- Transactions – HomeWeb, ShoppingCart, AdminRequest, AdminConfirm, CustomerRegister, BuyRequest, BuyConfirm, OrderInquiry, OrderDisplay, Search, SearchResult, NewProducts, BestSellers, ProductDetail,
Outline

✓ 1. The Basics
✓ 2. ACID Properties
✓ 3. Atomicity and Two-Phase Commit
✓ 4. Availability
✓ 5. Performance
  6. Styles of System

1.6 TP is System Engineering
• Compare it to other kinds of system engineering …
• Batch processing - Submit a job and receive file output.
• Time sharing - Invoke programs in a process, which may interact with the process’s display
• Real time - Submit requests that have a deadline
• Client/server - PC calls a server over a network to access files or run applications
• Decision support - Submit queries to a shared database, and process the result with desktop tools
• TP - Submit a request to run a transaction

TP vs. Batch Processing (BP)
• A BP application is usually uniprogrammed so serializability is trivial. TP is multiprogrammed.
• BP performance is measured by throughput. TP is also measured by response time.
• BP can optimize by sorting transactions by the file key. TP must handle random transaction arrivals.
• BP produces new output file. To recover, re-run the app.
• BP has fixed and predictable load, unlike TP.
  • But, where there is TP, there is almost always BP too.
    – TP gathers the input. BP post-processes work that has weak response time requirements
    – So, TP systems must also do BP well.

TP vs. Timesharing (TS)
• TS is a utility with highly unpredictable load. Different programs run each day, exercising features in new combinations.
• By comparison, TP is highly regular.
• TS has less stringent availability and atomicity requirements. Downtime isn’t as expensive.

TP vs. Real Time (RT)
• RT has more stringent response time requirements. It may control a physical process.
• RT deals with more specialized devices.
• RT doesn’t need or use a transaction abstraction
  – usually loose about atomicity and serializability
• In RT, response time goals are usually more important than completeness or correctness. In TP, correctness is paramount.

TP and Client/Server (C/S)
• Is commonly used for TP, where client prepares requests and server runs transactions
• In a sense, TP systems were the first C/S systems, where the client was a terminal
TP and Decision Support Systems (DSSs)

- DSSs run long queries, usually with lower data integrity requirements than TP.
- A.k.a. data warehouse (DSS is the more generic term.)
- TP systems provide the raw data for DSSs.

Outline

- 1. The Basics
- 2. ACID Properties
- 3. Atomicity and Two-Phase Commit
- 4. Availability
- 5. Performance
- 6. Styles of System

What’s Next?

- This chapter covered TP system structure and properties of transactions and TP systems
- The rest of the course drills deeply into each of these areas, one by one.