1. Introduction

CSEP 545 Transaction Processing
Philip A. Bernstein
Copyright ©2005 Philip A. Bernstein

Outline

1. The Basics
2. ACID Properties
3. Atomicity and Two-Phase Commit
4. Performance
5. 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.
- Place a bid at an on-line auction.
- Submit a corporate purchase order.

The "ilities" 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 - fits 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 systems market. One of the largest applications of computers.

TP System Infrastructure

- User's viewpoint:
  - Enter a request from a browser or other display device
  - The system performs an application-specific task, which includes database accesses.
  - Receives 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

(requests)

Workflow Control

routes requests and
supervises their execution

Transaction Program

Back-End

(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
  - thousands of disk drives
  - thousands of transactions per second, peak

Availability

- Fraction of time the 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

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

Availability Measurements

<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>99.96%</td>
</tr>
<tr>
<td>1 hour/year</td>
<td>99.9886%</td>
</tr>
<tr>
<td>1 hour/20 years</td>
<td>99.99942%</td>
</tr>
</tbody>
</table>

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 program 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.)
- Enterprise Java Beans, IBM WebSphere, Microsoft .NET (COM+), BEA WebLogic, Oracle Application Server

App Server Architecture, pre-Web

Boxes below are distributed on an internet

Presentation Server

Message Inputs

Workflow Controller

Network

Transaction Server

Transaction Server
**Automated Teller Machine (ATM) Application Example**

- **Workflow Controller**
- **CIRRUS Accounts**
- **Credit Card Accounts**
- **Loan Accounts**

**Application Server Architecture**

- **Web Browser**
  - **Workflow Controller**
  - **Queues**
  - **Transaction Server**

**Internet Retailer**

- **Web Server**
  - **Workflow Controller**
  - **Music**
  - **Electronics**
  - **Computers**

**Web Services**

- Interface and protocol standards to do application server functions over the internet.

**Enterprise Application Integration (EAI)**

- A software product to route requests between independent application systems. Often include:
  - A queuing system
  - A message mapping system
  - Application adaptors (SAP, PeopleSoft, etc.)

EAI and Application Servers address a similar problem, with different emphasis.

**ATM Example with an EAI System**

- **Workflow Controller**
  - **Queues**
  - **EAI Routing**

- **CIRRUS Accounts**
- **Checking Accounts**
- **Credit Card Accounts**
- **Loan Accounts**
**Workflow Systems**
- A software product that executes multi-transaction long-running scripts (e.g., process an order)
- Product components:
  - A workflow script language
  - Workflow script interpreter and scheduler
  - Workflow tracking
  - Message translation
  - Application and queue system adaptors
- Transaction-centric vs. document-centric
- Structured processes vs. case management
- IBM WebSphere MQ Workflow, Microsoft BizTalk, SAP, Vitria, Oracle Workflow, FileNET, Documentum, ...

**System Software Vendor’s View**
- TP is partly a component 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 the Database System and Application Server

**Outline**
1. The Basics
2. ACID Properties
3. Atomicity and Two-Phase Commit
4. Performance
5. 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**
- A ll-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 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...
    Commit
    Dispense Money
```

System crashes
Deferred operation never gets executed
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
  - E.g. File an invoice, cancel contract
  - Contract law talks a lot about appropriate compensations
- A well-designed TP application should have a compensation for every transaction type

Consistency

Every transaction should maintain DB consistency
- Referential Integrity - E.g., each order references an existing customer number and existing part number
- 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.

Some Notation

- $r_i[x] = \text{Read}(x)$ by transaction $T_i$
- $w_i[x] = \text{Write}(x)$ by transaction $T_i$
- $c_i = \text{Commit}$ by transaction $T_i$
- $a_i = \text{Abort}$ by transaction $T_i$

A history is a sequence of such operations, in the order that the database system processed them.

Consistency Preservation Example

- $T_1$: Start;
  - $A = \text{Read}(x)$;
  - $A = A - 1$;
  - $w(y, A)$;
  - $\text{Commit}$

- $T_2$: Start;
  - $B = \text{Read}(y)$;
  - $C = \text{Read}(y)$;
  - If $B > C + 1$ then $B = B - 1$;
  - $w(y, B)$;
  - $\text{Commit}$

Consistency predicate $s \equiv \{ x > y \}$
- Serial executions preserve consistency.
  - Interleaved executions may not.

- $H = r_1[x] r_1[x] r_2[y] w_2[x] w_1[y]$
  - E.g., try it with $x=4$ and $y=2$ initially

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

T1: Start;
A = Read(x);
A = A + 1;
Write(x, A);
Commit;

T2: Start;
B = Read(x);
B = B + 1;
Write(y, B);
Commit;

H = r1[x] r2[x] w1[x] c1 w2[y] c2

H is equivalent to executing T2 followed by T1

Note, H is not equivalent to T1 followed by T2

A loo, note that T1 started before T2 and finished before T2, yet the effect is that T2 ran first.

Serializability Examples (cont’d)

Client must control the relative order of transactions, using handshakes
wait for T1 to commit before submitting T2.

Some more serializable executions:

- r1[x] r2[y] w2[y] w1[x] “ T1 T2 ” T2 T1
- r1[y] r2[y] w2[y] w1[x] “ T1 T2 ” T1 T2
- r1[x] r2[y] w2[y] w1[y] “ T2 T1 ” T1 T2

Serializability says the execution is equivalent to some serial order, not necessarily to all serial orders

Non-Serializable Examples

- r1[x] r2[x] w2[x] w1[x] (race condition)
  - e.g. T1 and T2 are each adding 100 to x
- r1[x] r2[y] w2[x] w1[y]
  - e.g. each transaction is trying to make x = y, but the interleaved effect is a swap
- r1[x] r2[y] w1[x] r2[y] c1 w2[y] c2
  - e.g. T1 is moving $100 from x to y,
  - T2 sees only half of the result of T1
  - 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, it adds a record “commit(T) ” 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. Performance
5. 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". Each participant RM 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.

System Architecture

Outline

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

1.4 Performance Requirements

- Measured in max transaction per second (tps) or permutes (tpm), and dollars per tps or tpm.
- Dollars measured by list purchase price plus 5 year vendor maintenance ("cost of ownership")
- Workload typically 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

The TPC-C Order-Entry Benchmark

- TPC-C uses heavier weight transactions

<table>
<thead>
<tr>
<th>Table</th>
<th>Rows/Row</th>
<th>Bytes/Row</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warehouses</td>
<td>1</td>
<td>89</td>
</tr>
<tr>
<td>District</td>
<td>10</td>
<td>95</td>
</tr>
<tr>
<td>Customers</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-A/B — Bank Tellers

- Check 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
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
- tpmC = 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
  - Some high-end sales require custom benchmarks.

Typical TPC-C Numbers

- All numbers are highly sensitive to date submitted.
- $1.50 - $9 /tpmC for results released in 2004.
- Low-end numbers are slim: EM SQL Server & Win.
- High-end is only a scale and IBM, Linux, BEA Tuxedo
- System cost: $27K (HP) - $17M (IBM)
- Examples of high-throughput (64-processor systems)
  - IBM: 3.2M tpmC, $16.7M, $5.19/tpmC
  - HP: 1.2M tpmC, $28M, $5.50/tpmC
- Examples of low-cost (MS SQL Server, Win, COM+)
  - HP ProLiant, 18K tpmC, $32K, $1.70/tpmC, 4/14/04
  - Dell: 26K tpmC, $40K, $1.50/tpmC, 12/04

TPC-W - Web Retailer

- Introduced 12/99. Effectively retired in 2003 because it allowed “benchmark special” solutions
- Features: dynamic web page generation, multiple browser sessions, secure UI & payments (via secure socket layer)
- Profiles - shop (WIPS), browse (WIPSb), order (WIPSo)
- Tables - {Customer, Order, Order-Line, Item, Author, CreditCardTxns, Address, Country}
- Transactions - HomeWeb, ShoppingCart, Admin-Request, AdminConfirm, CustomerRegister, Buy-Request, BuyConfirm, OrderInquiry, OrderDisplay, Search, SearchResult, NewProducts, ...
- Web Interactions per sec (WIPS) @ ScaleFactor
  - ScaleFactor=1K - 10M items (in the catalog).

Comming Soon

- TPC App
  - A replacement for TPC-W. Completely different but web-focused. Unclear if it will be approved.
- TPC-E
  - Like TPC-C, it’s database-centric, but a different application
  - More realistic disk configuration (a smaller % of total price)
  - Possibly will have a processor scalability metric

Outline

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

- TP is System Engineering
- Compare TP 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 processor's display.
- Real Time - Submit requests that have a deadline.
- Client/Server - PC calls a server over a new network to access files or run applications.
- Decision Support - Submit queries to a shared database, and process the results with desktop tools.
- TP - Submit a request to run a transaction.

TP vs. Batch Processing (BP)

- A BP application is usually uniprocessor so serializability is trivial. TP is multiprocessor.
- 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 the work that has weak response time requirements.
- So, TP systems must also do BP well.

TP vs. Time Sharing (TS)

- TS is a utility with highly unpredictable load. Different programs run each day, exercising features in new combinations.
- By contrast, 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's usually a physical process.
- RT deals with more specialized devices.
- RT doesn't need or use a transaction abstraction.
- In RT, response time goals are usually more important than correctness. In TP, correctness is paramount.

TP and Client/Server (C/S)

- Is common only used for TP, where client prepares requests and server runs transactions.
- In a sense, TP systems are the first C/S systems, whereas the client was a terminal.

TP and Decision Support Systems (DSSs)

- DSSs run long queries, usually with low data integrity requirements compared to TP.
- A.K.A. data warehouse. DSS is the more generic term.
- TP systems provide raw data for DSSs.
Outline

1. The Basics
2. ACID Properties
3. Atomicity and Two-Phase Commit
4. Performance
5. 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.