1. Introduction

CSEP 545 Transaction Processing
Philip A. Bernstein
Sameh Elnikety

Copyright ©2012 Philip A. Bernstein
Outline

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

Front End Program

Request Controller
(routes requests and supervises their execution)

Transaction Server

Database System

Client

Back-End (Server)
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
  – Hundreds of thousands of active display devices
  – Indirect access via Internet
  – Tens of thousands of transactions per second, peak
Availability

• Fraction of time system is able to do useful work

• Some systems are very sensitive to downtime
  – Airline reservation, stock exchange, on-line retail, …
  – 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
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.)

• Enterprise Java Beans, IBM Websphere, Microsoft .NET (COM+), Oracle Weblogic and Application Server
App Server Architecture, Pre-Web

- Boxes below are distributed on an intranet
Automated Teller Machine (ATM) Application Example

Bank Branch 1
- ATM
- ATM

Request Controller

Interbank Transfer

Checking Accounts

Credit Card Accounts

Bank Branch 2
- ATM
- ATM

Request Controller

Bank Branch 500
- ATM
- ATM

Loan Accounts
Application Server Architecture

Web Browser

http

Web Server

http

Message Inputs

Requests

intranet

other TP systems

Queues

Request Controller

Transaction Server

Transaction Server

...
Service Oriented Architecture (SOA)

- Web services - interface and protocol standards to do app server functions over the internet.
Enterprise Application Integration (EAI)

• A software product to route requests between independent application systems. It often includes
  – A queuing system
  – A message mapping system
  – Application adaptors (SAP, Oracle PeopleSoft, etc.)

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

• Examples
  – IBM Websphere MQ, TIBCO, Vitria, Sun SeeBeyond
ATM Example with an EAI System

Bank Branch 1

ATM

ATM

Queues

EAI Routing

Bank Branch 2

ATM

ATM

Bank Branch 500

ATM

ATM

Queues

EAI Routing

Interbank Transfer

Checking Accounts

Credit Card Accounts

Loan Accounts
Workflow, or Business Process Mgmt

• 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

• Examples: IBM Websphere MQ Workflow, Microsoft BizTalk, SAP, Vitria, Oracle Workflow, IBM FileNET, EMC Documentum, TIBCO
Data Integration Systems
(Enterprise Information Integration)

- Heterogeneous query systems (mediators). It’s database system software, but …
- It’s similar to EAI with more focus on data transformations than on message mgmt.
Transactional Middleware

• In summary, there are many variations that package different combinations of middleware features
  – Application Server
  – Enterprise Application Integration
  – Business process management (aka Workflow)
  – Enterprise Server Bus
• New ones all the time, that defy categorization
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 the Database System and Application Server
Outline

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

Deferred operation never gets executed

System crashes
Transaction aborts
Money is dispensed

System crashes
Reading Uncommitted Output Isn’t Undoable

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

User reads output ...
User enters input

Brain transport

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

1/4/2012
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., Fire missile, 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 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
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 = Read(x);
   A = A - 1;
   Write(y, A);
   Commit;

$T_2$: Start;
   B = Read(x);
   C = Read(y);
   If (B - 1 > C) then B = B - 1;
   Write(x, B);
   Commit;

- Consistency predicate is $x > y$
- Serial executions preserve consistency.
  Interleaved executions may not.
- $H = r_1[x] \ r_2[x] \ r_2[y] \ w_2[x] \ w_1[y]$
  - 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.
Serializability Example 1

\( T_1: \) Start;  
\( A = \text{Read}(x); \)  
\( A = A + 1; \)  
\( \text{Write}(x, A); \)  
\( \text{Commit}; \)

\( T_2: \) Start;  
\( B = \text{Read}(y); \)  
\( B = B + 1; \)  
\( \text{Write}(y, B); \)  
\( \text{Commit}; \)

- \( H = r_1[x] \ r_2[y] \ w_1[x] \ c_1 \ w_2[y] \ c_2 \)
- \( H \) is equivalent to executing
  - \( T_1 \) followed by \( T_2 \)
  - \( T_2 \) followed by \( T_1 \)
Serializability Example 2

$T_1$: Start;
A = Read(x);
A = A + 1;
Write(x, A);
Commit;

$T_2$: Start;
B = Read(x);
B = B + 1;
Write(y, B);
Commit;

• $H = r_1[x] \; r_2[x] \; w_1[x] \; c_1 \; w_2[y] \; c_2$

• $H$ 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

• Client must control the relative order of transactions, using handshakes
  (wait for $T_1$ to commit before submitting $T_2$)

• Some more serializable executions
  
  $r_1[x] \ r_2[y] \ w_2[y] \ w_1[x] \equiv T_1 \ T_2 \equiv T_2 \ T_1$

  $r_1[y] \ r_2[y] \ w_2[y] \ w_1[x] \equiv T_1 \ T_2 \neq T_2 \ T_1$

  $r_1[x] \ r_2[y] \ w_2[y] \ w_1[y] \equiv T_2 \ T_1 \neq T_1 \ T_2$

• 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. Performance
  5. Scalability
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
Two-Phase Commit System Architecture

1. Start transaction returns a unique *transaction identifier*
2. Resource accesses include the transaction identifier
   For each transaction, RM registers with TM
3. When application asks TM to commit, the TM runs two-phase commit
Outline

✓ 1. The Basics
✓ 2. ACID Properties
✓ 3. Atomicity and Two-Phase Commit
  4. Performance
  5. Scalability
1.4 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 typically has this profile
  - 10% application server plus application
  - 30% communications system (not counting presentation)
  - 50% DB system
- TP Performance Council (TPC) sets standards
  - http://www.tpc.org
- TPC A & B (‘89-’95), now TPC C & E
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

1/4/2012
## TPC-C Order-Entry for Warehouse

<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

• \( tpmC = \text{number of New-Order transaction per min} \)
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
  – Some high-end system sales require custom benchmarks
Current TPC-C Numbers

- All numbers are sensitive to date submitted
- Systems
  - cost $60K (Dell/HP) - $12M (Oracle/IBM)
  - mostly Oracle/DB2/MS SQL on Unix variants/Windows
  - $0.40 - $5 / tpmC
- Example of high throughput
  - Oracle, 30M tpmC, $30.0M, $1/tpmC, Oracle/Solaris
- Example of low cost
  - HP ProLiant, 290K tpmC, $113K, $0.39/tpmC, Oracle/Linux
TPC-E

• Approved in 2007
• Models a stock trading app for brokerage firm
• Should replace TPC-C, it’s database-centric
• More complex but less disk IO per transaction
TPC-E

- 33 tables in four sets
  - Market data (11 tables)
  - Customer data (9 tables)
  - Broker data (9 tables)
  - Reference data (4 tables)

- Scale
  - 500 customers per tpsE
TPC-E Transactions

• Activities
  – Stock-trade, customer-inquiry, feeds from markets, market-analysis

• tpsE = number of Trade-Result transaction per sec

• Trade-Result
  – Completes a stock market trade
  – Receive from market exchange confirmation & price
  – Update customer‘s holdings
  – Update broker commission
  – Record historical information
# TPC-E Transactions

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broker-Volume</td>
<td>RO</td>
<td>DSS-type medium query</td>
</tr>
<tr>
<td>Customer-Position</td>
<td>RO</td>
<td>“What am I worth?”</td>
</tr>
<tr>
<td>Market-Feed</td>
<td>RW</td>
<td>Processing of Stock Ticker</td>
</tr>
<tr>
<td>Market-Watch</td>
<td>RO</td>
<td>“What’s the market doing?”</td>
</tr>
<tr>
<td>Security-Detail</td>
<td>RO</td>
<td>Details about a security</td>
</tr>
<tr>
<td>Trade-Lookup</td>
<td>RO</td>
<td>Look up historical trade info</td>
</tr>
<tr>
<td>Trade-Order</td>
<td>RW</td>
<td>Enter a stock trade</td>
</tr>
<tr>
<td><strong>Trade-Result</strong></td>
<td><strong>RW</strong></td>
<td><strong>Completion of a stock trade</strong></td>
</tr>
<tr>
<td>Trade-Status</td>
<td>RO</td>
<td>Check status of trade order</td>
</tr>
<tr>
<td>Trade-Update</td>
<td>RW</td>
<td>Correct historical trade info</td>
</tr>
</tbody>
</table>
Current TPC-E Numbers

• Systems
  – Cost $60K - $2.3M
  – Almost all are MS SQL on Windows
  – $130 - $250 / tpsE

• Example of high throughput
  – IBM, 4.5k tpsE, $645k, $140/tpsE, MS SQL/Windows

• Example of low cost
  – IBM, 2.9K tpsE, $371K, $130/tpsE, MS SQL/Windows
Outline

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

• Techniques for better performance
  – Textbook, Chapter 2, Section 6

• Scale-up
  – Caching
  – Resource Pooling

• Scale-out
  – Partitioning
  – Replication
Caching

• Key idea
  – Use more memory
  – Keep a copy of data from its permanent home
  – Accessing a cached copy is fast

• Key issues
  – Which data to keep
    • Popular read-only data
  – Cache replacement
  – What if original data is updated
    • Invalidations
    • Timeouts
Caching

• Applied at multiple levels
  – Database and application server

• Updates
  – Write through
    • Better cache coherence
  – Write back
    • Batching and write absorption

• Example products
  – Memcached, MS Velocity
Resource Pooling

• Key idea
  – If a logical resource is expensive to create and cheap to access, then manage a pool of the resource

• Examples
  – Session pool
  – Thread pool
Partitioning

• To add system capacity, add server machines
• Sometimes, you can just relocate some server processes to different machines
• But if an individual server process overloads one machine, then you need to partition the process
  – Example – One server process manages flights, cars, and hotel rooms. Later, you partition them in separate processes.
  – We need mapping from resource name to server name
Partitioning: Routing

• Sometimes, it’s not enough to partition by resource type, because a resource is too popular
  – Example: flights

• Partition popular resource based on value ranges
  – Example – flight number 1-1000 on Server A, flight number 1000-2000 on Server B, etc.
  – Request controller has to direct its calls based on parameter value (e.g. flight number)
  – This is called parameter-based routing
    • E.g., range, hashing, dynamic
Replication

• Replication - using multiple copies of a server or resource for better availability and performance.
  – Replica and Copy are synonyms

• If you’re not careful, replication can lead to
  – worse performance - updates must be applied to all replicas and synchronized
  – worse availability - some algorithms require multiple replicas to be operational for any of them to be used
Replicated Server

• Can replicate servers on a common resource
  – Data sharing - DB servers communicate with shared disk

• Helps availability for process (not resource) failure

• Requires a replica cache coherence mechanism, so this helps performance only if
  – Little conflict between transactions at different servers or
  – Loose coherence guarantees (e.g. read committed)
Replicated Resource

- To get more improvement in availability, replicate the resources (too)
- Also increases potential throughput
- This is what’s usually meant by replication
Outline

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

• We covered
  – Chapter 1
  – Chapter 2, Section 6

• Assignment 1

• Teams for the project