

1. Introduction

CSEP 545 Transaction Processing
Philip A. Bernstein

Copyright ©2003 Philip A. Bernstein

3/27/03

1

Outline

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

3/27/03

2

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
- Connect to video-on-demand and pay for it

3/27/03

3

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

3/27/03

4

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.

3/27/03

5

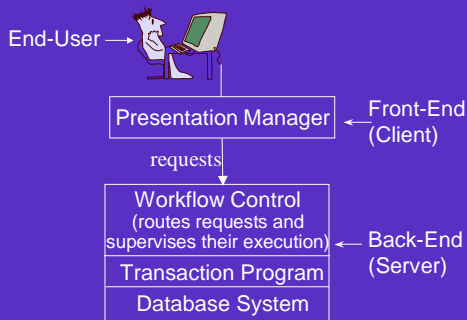
TP System Infrastructure

- 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

3/27/03

6

TP System Infrastructure ... Defines System and Application Structure



3/27/01

7

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

3/27/01

8

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

| Downtime | Availability |
|----------------|--------------|
| 1 hour/day | 95.8% |
| 1 hour/week | 99.41% |
| 1 hour/month | 99.86% |
| 1 hour/year | 99.9886% |
| 1 hour/20years | 99.99942% |

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

3/27/01

9

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

3/27/01

10

Application Servers (cont'd)

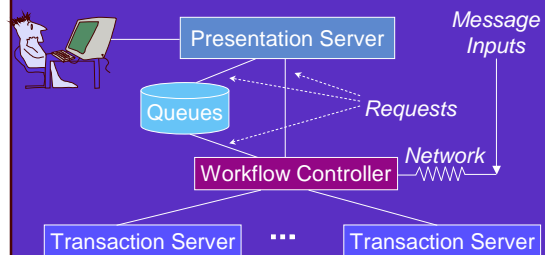
- Components include
 - an application programming interface (API) (e.g., Enterprise Java Beans)
 - tools for program development
 - tools for system management (app deployment, fault & performance monitoring, user mgmt, etc.)
- Enterprise Java Beans, IBM Websphere, Microsoft .NET (COM+), BEA Weblogic, Oracle Application Server

3/27/01

11

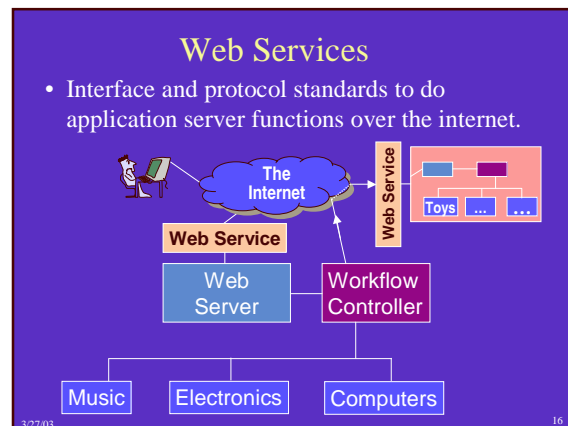
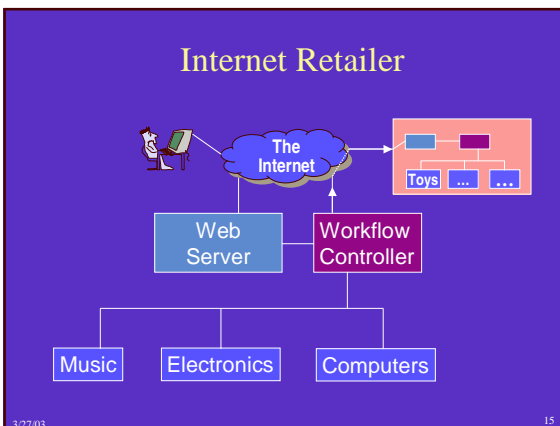
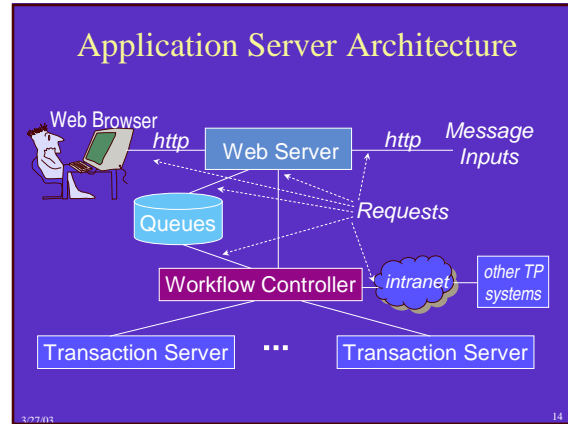
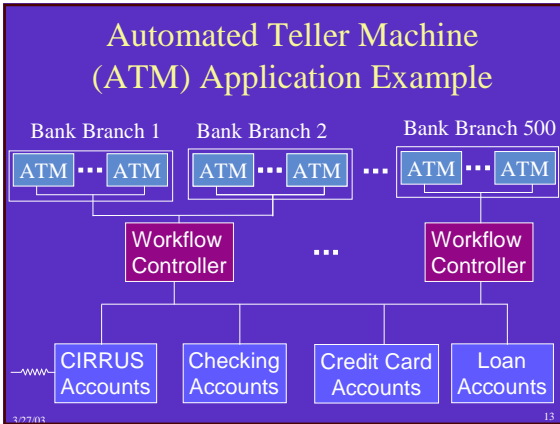
App Server Architecture, pre-Web

- Boxes below are distributed on an intranet



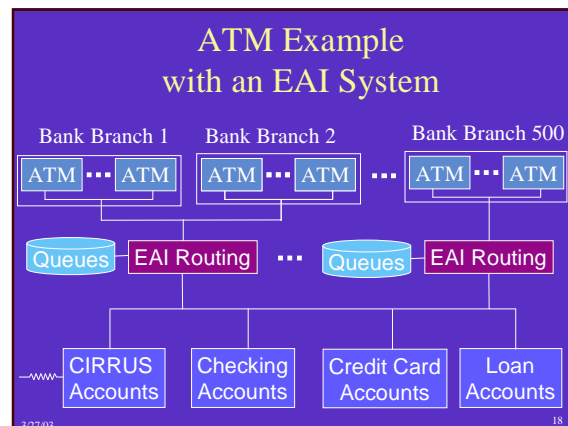
3/27/01

12



Enterprise Application Integration

- 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
- IBM Websphere MQ, TIBCO, Vitria, SeeBeyond



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 vs. document centric
- Structured processes vs. case management
- IBM Websphere MQ Workflow, Microsoft BizTalk, Vitria, Oracle Workflow, FileNET, Documentum

3/27/03

19

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

3/27/03

20

Outline

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

3/27/03

21

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

3/27/03

22

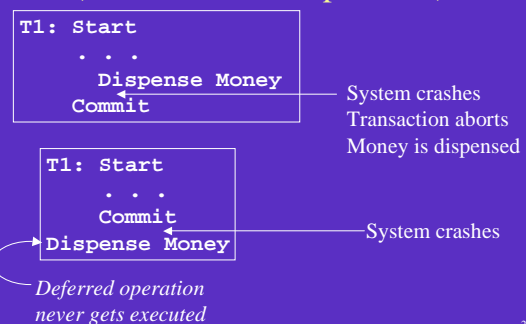
Atomicity

- 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

3/27/03

23

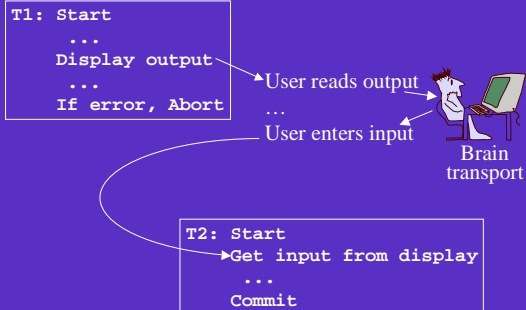
Example - ATM Dispenses Money (a non-undoable operation)



3/27/03

24

Reading Uncommitted Output Isn't Undoable



3/27/03

25

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 talks a lot about appropriate compensations
- ♣ A well-designed TP application should have a compensation for every transaction type

3/27/03

26

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.

3/27/03

27

Some Notation

- $r_i[x]$ = Read(x) by transaction T_i
- $w_i[x]$ = Write(x) by transaction T_i
- c_i = Commit by transaction T_i
- a_i = Abort by transaction T_i
- A *history* is a sequence of such operations, in the order that the database system processed them.

3/27/03

28

Consistency Preservation Example

```

T1: Start;      T2: Start;
  A = Read(x);  B = Read(x);
  A = A - 1;    C = Read(y);
  Write(y, A);  If (B > C+1) then B = B - 1;
  Commit;      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

3/27/03

29

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.

3/27/03

30

A Serializability Example

| | |
|-------------------------|-------------------------|
| T ₁ : Start; | T ₂ : Start; |
| A = Read(x); | B = Read(x); |
| A = A + 1; | B = B + 1; |
| Write(x, A); | Write(y, B); |
| Commit; | Commit; |

- $H = r_1[x] r_2[x] w_1[x] c_1 w_2[y] c_2$
- H is equivalent to executing T₂ followed by T₁
- Note, H is *not* equivalent to T₁ followed by T₂
- Also, note that T₁ started before T₂ and finished before T₂, yet the effect is that T₂ ran first.

3/27/03

31

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:
 $r_1[x] r_2[y] w_2[y] w_1[x] \equiv T_1 T_2 \equiv T_2 T_1$
 $r_1[y] r_2[y] w_2[y] w_1[x] \equiv T_1 T_2 \not\equiv T_2 T_1$
 $r_1[x] r_2[y] w_2[y] w_1[y] \equiv T_2 T_1 \not\equiv T_1 T_2$
- Serializability says the execution is equivalent to *some* serial order, not necessarily to *all* serial orders

3/27/03

32

Non-Serializable Examples

- $r_1[x] r_2[x] w_2[x] w_1[x]$ (*race condition*)
 – e.g. T₁ and T₂ 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. T₁ is moving \$100 from x to y.
 – T₂ sees only half of the result of T₁
- Compare to the OS view of synchronization

3/27/03

33

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(T₁)” to the log
 – when the commit record is on disk, the transaction is committed.
 – system waits for disk ack before acking to user

3/27/03

34

Outline

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

3/27/03

35

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.)

3/27/03

36

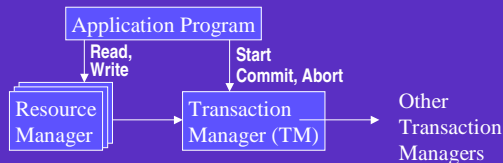
Two-Phase Commit

- 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.

3/27/03

37

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.

3/27/03

38

Outline

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

3/27/03

39

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 & W

3/27/03

40

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

3/27/03

41

The TPC-C Order-Entry Benchmark

| Table | Rows/Whse | Bytes/row |
|-----------|-----------|-----------|
| Warehouse | 1 | 89 |
| District | 10 | 95 |
| Customer | 30K | 655 |
| History | 30K | 46 |
| Order | 30K | 24 |
| New-Order | 9K | 8 |
| OrderLine | 300K | 54 |
| Stock | 100K | 306 |
| Item | 100K | 82 |

- TPC-C uses heavier weight transactions

3/27/03

42

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.

3/27/03

43

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.
 - IBM has claimed DB2 is optimized for a different workload, so they only started publishing TPC numbers a few years ago.

3/27/03

44

Typical TPC-C Numbers

- \$3 - \$50 / tpmC. Most are under \$20 / tpmC.
 - Top 24 price/performance results on MS SQL Server & Windows.
 - One of the top 56 is Oracle, Linux, BEA Tuxedo
- System cost \$36K (Dell) - \$12M (Fujitsu)
- Examples of high throughput
 - HP ProLiant cluster, 709K tpmC, \$10.6M, \$15/tpmC (MS SQL, MS COM+)
 - IBM 428K tpmC, \$7.6M, \$18/tpmC (Oracle, Websphere)
- Examples of low cost (all use MS SQL Server, COM+)
 - HP ProLiant cluster, 411K tpmC, \$5.3M, \$13/tpmC
 - Dell, 16.7K tpmC, \$47K, \$3/tpmC
- Results are very sensitive to date published.

3/27/03

45

TPC-W – Web Retailer

- Introduced 12/99.
- 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, AdminRequest, AdminConfirm, CustomerRegister, BuyRequest, BuyConfirm, OrderInquiry, OrderDisplay, Search, SearchResult, NewProducts, BestSellers, ProductDetail,

3/27/03

46

TPC-W (cont'd)

- Scale factor: 1K – 10M items (in the catalog).
- Web Interactions per sec (WIPS) @ ScaleFactor
 - IBM: 21K WIPS@10K items; \$33 / WIPS; \$690K total
 - Dell: 8K WIPS@10K items; \$25 / WIPS; \$190K total

3/27/03

47

Outline

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

3/27/03

48

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 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

3/27/03

49

TP vs. Batch Processing (BP)

- A BP application is usually unprogrammed 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.

3/27/03

50

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.

3/27/03

51

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.

3/27/03

52

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

3/27/03

53

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.

3/27/03

54

Outline

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

3/27/01

55

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.

3/27/01

56