2. TP Monitors

CSE 593 Transaction Processing

Philip A. Bernstein

Copyright © 1999 Philip A. Bernstein

Outline

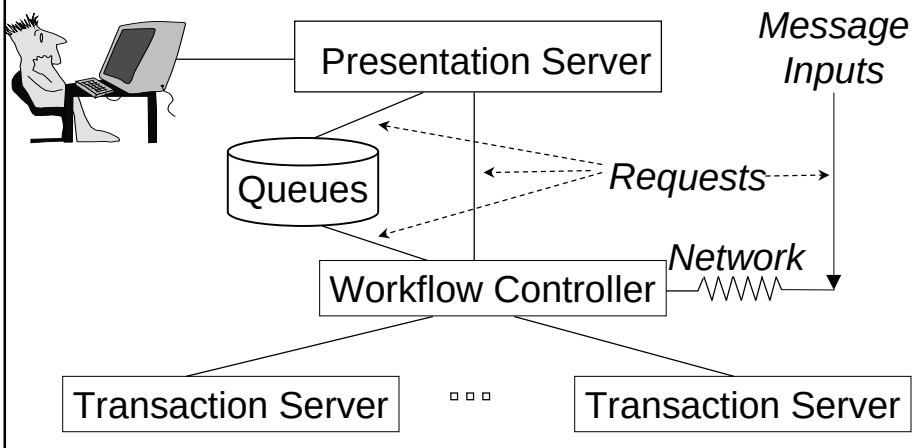
1. Introduction
2. Presentation Servers
3. Workflow Controllers
4. Transaction Servers
5. Database Servers vs. TP Monitors
6. Transactions over the Internet

2.1 Introduction

- A request is a message that describes a unit of work for the system to execute
- A TP monitor coordinates the flow of requests between message sources (displays, etc.) and application programs that perform transactions.
- Basic control flow:
  - Translate the display input (form/menu selection, etc.) into a standard-format request.
  - Find the transaction type in the request header
  - Start the transaction
  - Invoke the transaction type's application program
  - Send the transaction's output to the display

3-Tier TP Monitor Architecture

- TP monitor should make the previous control flow scale up
- Boxes below can be distributed around a network

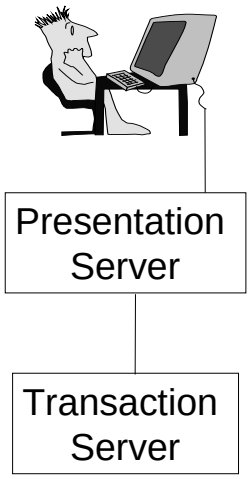


TP Monitor Functions

- Glue and veneer for TP applications
  - glue fills in gaps in system functionality
  - covers the interface with a seamless veneer
- Provides run-time functions for applications (presentation servers, workflow control, and transaction servers). Often wired tightly to underlying OS platform.
- Provides system mgmt for the running application. E.g., interprets OS and DBMS metering in TP terms
  - fault monitoring and repair (fail over), server mgmt
  - security management
  - performance monitoring and tuning (load balancing)
- Provides some application development tools

2-Tier vs. 3-Tier

- Presentation server is usually a 4GL on a PC: Visual Basic, Powerbuilder, Delphi, … (a.k.a. RAD - rapid app development)
- 2-tier alternative ("TP Lite") - connect PCs directly to transaction servers
- 2-tier is feasible, but
  - doesn't scale as well as 3-tier (more later …)
  - doesn't match OO application designs, which are inherently 3-tier



3-Tier Applications

- Application structure mimics system structure
- Fits business rule / business object architecture
- Presentation server - forms/menus, data validation
  - UI for invoking a transaction
- Workflow controller - map requests to calls on the right program(s)
  - Business rules to transform request into right calls on basic objects (automatic loan payment, money transfer)
- Transaction server - do the work
  - Business objects (customer, account, loan, teller)

2.2 Presentation Servers

- Presentation independence - application is independent of the display device used
- Presentation server has two layers:
  - Gathering input and
  - Constructing requests

Gathering Input

- Gathering input, usually in a 4GL
  - Select transaction type (menu item or control/widget)
  - Fill in a form and validate input against cached tables
- Forms managers for character-cell devices
  - Character-at-a-time, block mode (screen-at-a-time)
  - WYSIWYG form editor, compile form, execute it (with record defn binding to programming language)
- Many specialized device types
  - ATMs, bar code readers, credit card authorization, gas pumps, cash registers, robots, ticket printers, etc.
  - Often they conform to terminal standards - IBM 3270 or asynch terminal; dialup modem or standard protocol, etc.

Gathering Input

- Web browser is now a standard type of input device
  - Each Web protocol is a new device type
  - HTML forms
  - XML - application-defined structures
  - ...

Constructing Requests

- TP monitor defines a request format, which includes
  - User id
  - Device id
  - Device type - what kinds of messages can it understand?
  - Request type - name of transaction type requested
  - Request-specific parameters
- Can implement using ordinary RPC
  - E.g. Microsoft Transaction Server (MTS)
  - May transparently pass monitor-specific context too
- Often, it's explicit message passing.

Constructing Requests (cont'd)

- Helpful if send-request returns a request id, to persist on client and/or simulate multi-threading
- Need a cancel function, to (try to) kill a request

Authentication

- Authentication - determining the identity of a user and/or display device
  - Client system may do authentication, but server usually does too (doesn't trust clients)
  - Encrypt the wire to avoid wiretapping and spoofing
- Geographical entitlement - check that a particular *device* is allowed access (e.g. security trading room)
- Need system mgmt functions to create accounts, initialize passwords, bracket hours of access (simplify using role abstractions)
- Major activity in TP application development

Other Presentation Functions

- Communications
  - map device ids to meaningful names, for easier programming and system mgmt
  - similarly for other system variables
- Logging
  - record all messages sent and received by a presentation server
  - useful for auditing and error analysis

2.3 Workflow Controllers

- Routing - Maps the request type into the network id of the server to process it, and sends the reply back to the client system
- Bracket the transaction with Start and Commit
- Handle exceptions

Routing

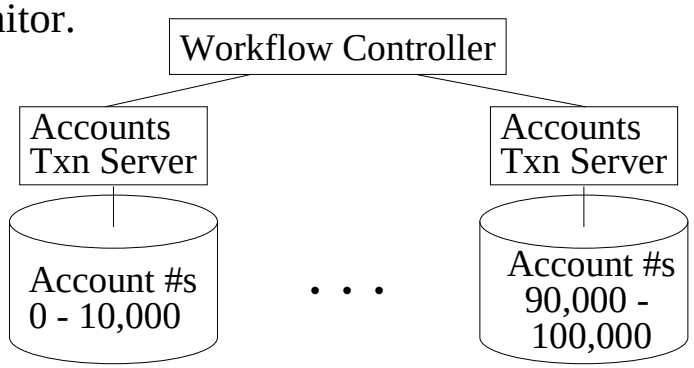
- Need a directory service to map symbolic request type names (usually path names) into server ids.
  - E.g. in DCOM, it's a registry entry
  - In MTS, package is the unit of installation
  - MTS creates a self-extracting .exe per package that updates the registry entry (among other things)
- The mapping should be dynamic, to support
  - reconfiguration - add new request type, move server
  - fault tolerance - redirect requests to backup server

Routing (cont'd)

- Use network-wide directory service, if available
  - reconfigure using predefined backups, since propagating new configuration network-wide takes too long
- Most TP monitors do routing themselves (too), for portability and legacy reasons.
  - Usually a central copy of map is periodically sent to cached copies.

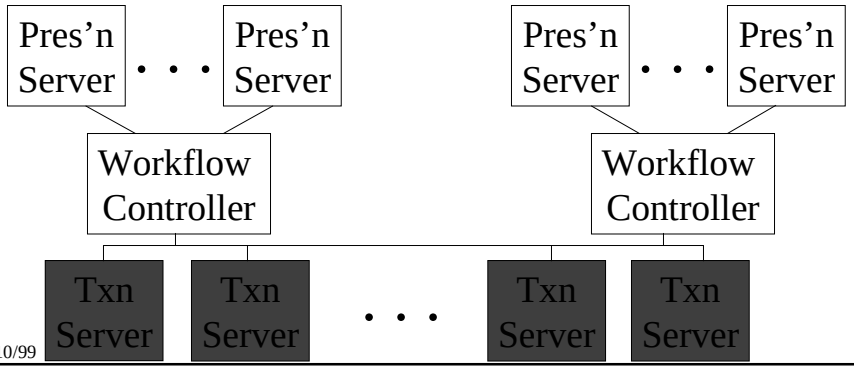
Parameter-based Routing

- Use a set of identical servers, each handling a different range of parameters.
- Use request's parameter to select the server.
- Can be done in the application, or automated by the TP monitor.



Session Management

- Session - shared state between communicating parties
- Sessions reduce amount of per-request context passing (comm addr's, authenticated user/device)
- But N clients and M servers implies too many sessions
- Need a middle tier to reduce number of sessions



Session Management (cont'd)

- Example
  - 10^5 presentation servers (devices) and 100 servers implies 10^7 sessions (10^5 per server)
  - Now, add 100 workflow controllers, which partition the set of presentation devices. Implies 10^3 sessions per workflow controller and 10^2 sessions per server, or $10^5 + 10^4 = 11{,}000$ sessions
- Now guess why http is stateless … no sessions.

Workflow Logic

- The middle tier program is a plain old application
- Most TP monitors support most standard languages.
  - It's still important to distinguish workflow controller layer, for scalability, isolated business rules, and portability to other TP monitors
  - MTS supports all COM languages (VB, C++, Java, COBOL, …)
- But some TP monitors require that it's written in a special language
  - Digital's ACMSxp - Structured Txn Defn Language
  - Tandem Pathway - Screen COBOL

Transaction Bracketing

- Workflow controller brackets the transaction with Start, Commit, Abort.
- Chained - All programs execute in a transaction. A program can commit/abort a transaction, after which another transaction immediately starts
  - E.g., CICS syncpoint = Commit&Start
  - Prevents programmer from accidentally issuing resource manager operations outside a transaction
- Unchained - Explicit Start operation, so some statements can execute outside a transaction
  - No advantages, unless txns have overhead even if they don't access resources.

Transparent Transaction Bracketing

- Transaction-hood may be a property of the workflow controller application.
- In MTS, a class is declared:
  - requires new - callee always starts a new transaction
  - required - if caller is in a transaction, then run in caller's transaction, else start a new transaction
  - supported - if caller is in a transaction, then run in caller's transaction, else run outside of any transaction
  - not supported - don't run in a transaction

MTS Transaction Bracketing (cont'd)

- Caller can create a transaction context, which supports Commit and Abort (chained model).
- SetComplete - object is finished with transaction's work (not just this call). Commit if method was called outside of a transaction.
- SetAbort - abort the transaction
- We'll revisit this in more detail in Chapter 3.

Nested Transaction Calls

- What does Start do, when executed within a txn?
  1. it starts an independent transaction, or
  2. it does nothing, or
  3. it increments a nested transaction count (which is decremented by each commit and abort), or
  4. it starts a sub-transaction
- If (1), then be careful not to start a transaction from within a transaction
  - E.g. only start transaction in workflow control, not in a transaction server
  - (3) avoids this problem

Integrity of Requests

- To permit request retries, it's useful if **`get-request`** runs inside the request's transaction:

```
Start;
 get-request;
    . . .
Commit;
```

- If the transaction aborts, the request becomes available for the next **`get-request`**.
- In the RPC or "push model," make the "catch-the-call" operation explicit, so it can be undone. Possibly hidden in the dispatch mechanism. Often requires a queue manager.

Exception Handling

- Workflow control brackets the transaction, so it must say what to do if the transaction aborts
- An exception handler must know what state information is available
  - cause of the abort, e.g. a status variable
  - possibly program exception separate from abort reason
  - for system failures, application must save state in stable storage -- none of the aborted txn's state will be available
- Chained model - exception handler starts a new txn
- MTS - component returns a failure hresult

Savepoints

- Savepoint - a point in a program where an application saves all its recoverable state
- Can restore a savepoint within the transaction that issued the savepoint. (It's a partial rollback.)
- Usually supported by SQL DBMSs, since it helps them support atomic SQL statements.

```
Start;
get-request;
Savepoint("B"); . . .;
if (error) {Restore("B"); …; Commit;}
. . .;
Commit;
```

Savepoints (cont'd)

- Savepoints are not recoverable. If the system fails or the transaction aborts, the txn is completely undone.
- Persistent savepoints have appeared in the research literature, but aren't supported commercially. There are formidable technical difficulties.

2.4 Transaction Servers

- Transaction server - application program that does the "real work" of processing the request
- For application portability, it's desirable to write them in a standard programming language (C, C++, COBOL) and standard data manipulation language (SQL)
- These are ordinary application programs. Only two issues are worth discussing
  - mapping them onto OS processes
  - transaction server communication

Process Architecture Issues

- TP monitor architecture is greatly affected by
  - which components share an address space
  - how many control threads per address space
- TP grew up in the days of batch processing, and reached maturity in the days of timesharing.
- TP users learned early that a process-per-user fails:
  - too much context switching
  - too much fixed memory overhead per process
  - process per user per machine, when distributed
  - some OS functions scan the list of processes
  - load control is hard

Multithreading

- Have multiple threads of control in an address space
- Often, the TP monitor implements multithreading
  - TP monitor switches between threads when app calls a TP monitor function that blocks
  - so the OS thinks there's just one thread
  - all app calls that can block must go to the TP monitor, or the whole process will block
  - possible conflict between TP monitor and OS scheduling
  - simplify the implementation by using an interpretive language (e.g. ACMSxp STDL, Tandem SCOBOL)

Multithreading (contd')

- Or use OS multithreading
  - no problem with blocking operations
  - can run a process's threads on different processors (SMP)
  - possibly more expensive to switch threads (system calls)
- Whether at the user or OS level, multithreading has
  - fewer processes
  - reduced context switching
- Disadvantages
  - little protection between threads
  - server failure affects many transactions

Server Pools

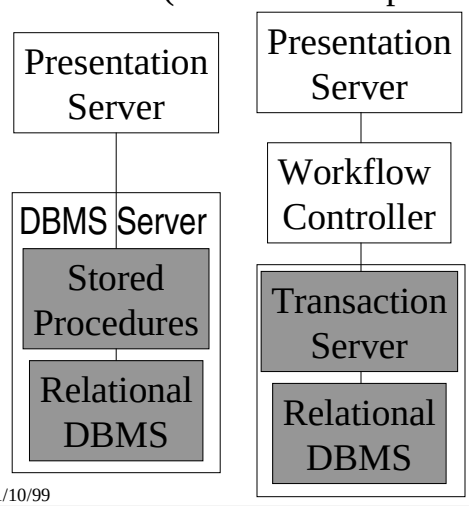
- Use a set of processes (server pool or server class) to emulate multithreading
  - better protection and fault isolation than multithreading
  - avoids problems of user-level multithreading - blocking operations and conflicts between scheduling levels
- How to dispatch calls?
  - Randomly select a server
  - Server class shares a dispatch queue. Clients enqueue, servers dequeue.
  - Use a dispatch process (adds a context switch per call)
- Number of servers is proportional to number of active transactions, so not too many processes

Mapping Servers to Processes

- Presentation servers - separate processes
- Workflow controllers
  - usually multithreaded, serving many presentation servers
  - if single-threaded, then 1 per presentation server (2-tier)
- Transaction servers
  - multithreaded servers, or
  - server pools
- What does all this cost?
  - 1500 - 25,000 instructions per process call, vs. 50 instructions per local procedure call …
  - but it scales, with flexible configuration and control

2.5 DB Servers vs. TP Monitors

- How many tiers are best?
- DB servers with stored procedures look a lot like a TP monitor (RPC and two-phase commit)



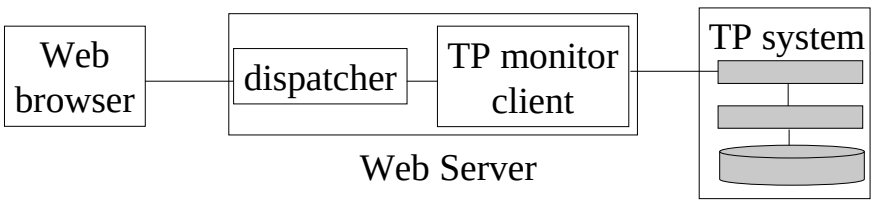
- Main difference is the 3rd tier (workflow controller), for scalability
- In 3-tier, can still be good to run transaction server as a stored procedure
- Remaining differences are features, which change with each product release

DB Servers vs. TP Monitors (cont'd)

- Database Servers
  - nonstandard stored procedure language, usually less expressive with somewhat weaker development tools
  - limited interoperability of cross-server calls
  - limited interoperability of distributed transactions
  - it's another language to learn
- TP monitors
  - more system complexity
  - better scalability (most TPC-C benchmarks (all high-throughput ones) use a TP monitor)

2.6 Transactions over the Internet

- Web browser = presentation manager
- Message from web browser to server is a transaction request
- http = client/server protocol
- Web server = workflow controller + transaction server



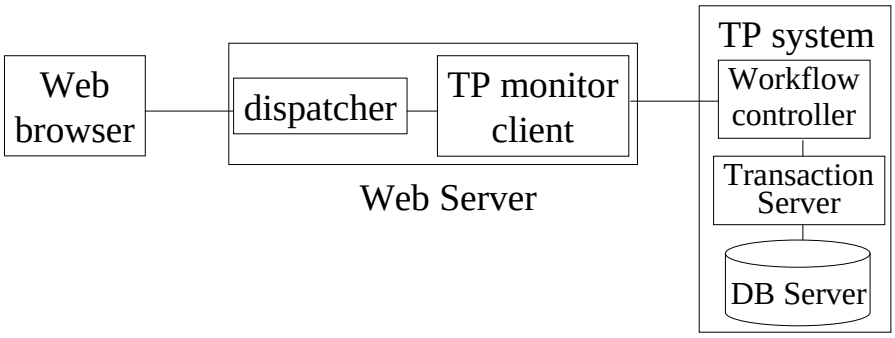
- It's déjà vu, all over again.

Let's Count Processes

- Initially, dispatcher used CGI (fork a process)
  - Each request creates a process
- Now ISAPI (Microsoft) and NSAPI (Netscape) are popular
  - Web server calls an in-proc.dll instead of creating a process
  - But if the callee must run as a transaction, what process does it run in?

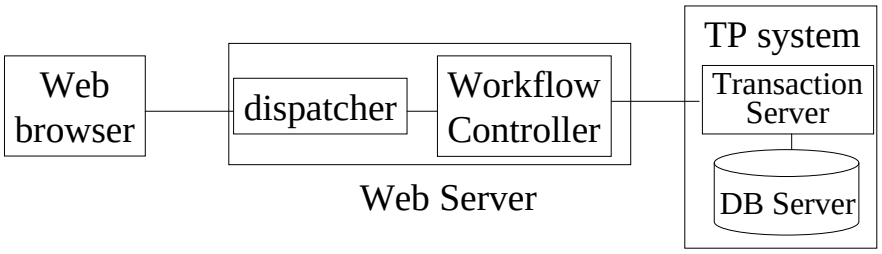
TP Monitor Implications

- In many TP monitors, Web server is an ordinary client
- So it can't be a workflow controller (can't bracket a transaction) and you're doomed to an extra context switch.
  - There goes 15K instructions of your 100K budget



TP Monitor Implications (cont'd)

- Solution - Make the web server a workflow controller. I.e., it includes the TP monitor runtime.



- Most TPC benchmarks use a web browser
- TP monitors and DB servers are now integrating with Web servers to minimize # of processes.