TP Monitors Solve Server ScaleUp Problems.

Jeri Edwards (Tandem), Jim Gray (UCB)

Introduction:

TP monitors and TP monitor techniques solve the upsizing problem in client/server computing -- going from 50 clients to 500 or more clients. Once past this basic performance problem, TP monitors focus on tools to design, configure, manage, and operate client/server systems. They place particular emphasis on the management and control of server.

The general flow of client/server systems is that each client makes simple requests to the server. The server processes the request and responds. Typically there a many clients and only one or a few servers. This simple request-response flow has many refinements and variants discussed below. Before getting into such refinements, let's first look at the fundamental scaleup problems and the solutions that TP systems have evolved over the last 30 years.

Scaleable Clients and Server Connections: The Evolution of TP

Dedicating a **server-process-per-client** is the obvious way to build a client/server application. Having a process for each window gives concurrent execution and also gives protection among applications so that if one malfunctions, the others are not affected.

The server-process-per-client has two severe scalability problems.

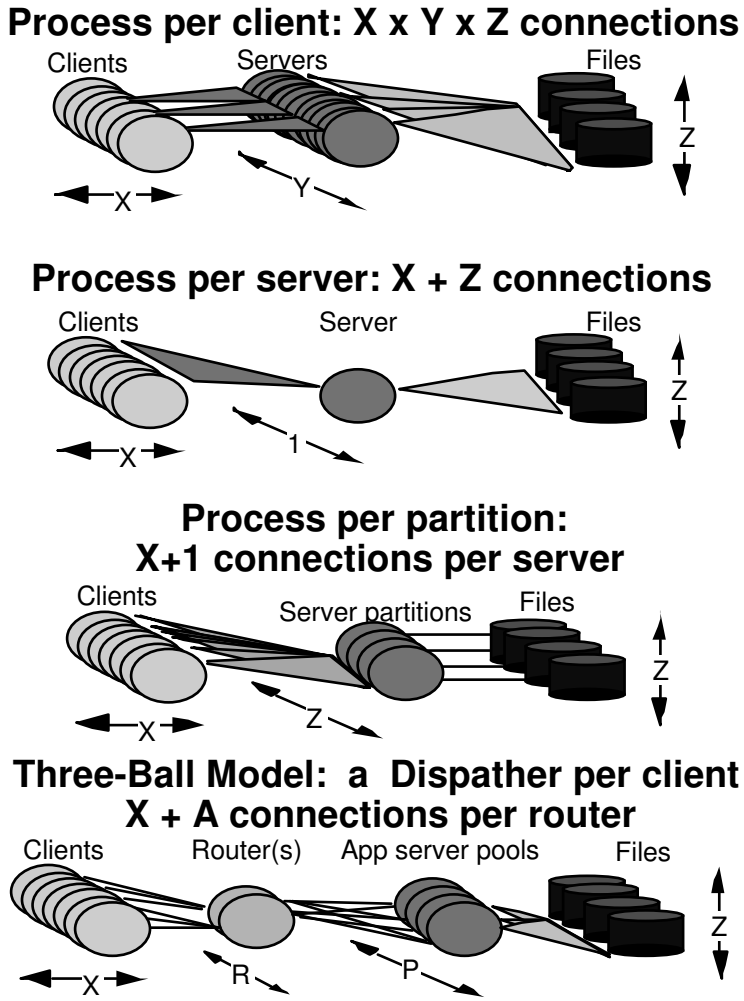
The percentage problem: With ten clients, each client can have 10% of the server -- but with 100 clients, each client can only count on 1% of the server. The shared server and its data becomes a precious resource as one goes past about 30 clients. TP monitors optimize the server to support many clients.

The polynomial explosion problem: Each client typically wants to open several applications. Each application wants to open several files. So X clients opening Y applications each with Z open files results in an X*Y*Z connections and opens. The system has to manage X*Y processes. As the application grows, X, Y and Z grow. Quickly, one is managing thousands of processes and tens of thousands of connections. This breaks most operating systems.

The obvious solution to these problems was to go from a server-process-per-client to a **server-process-per-server**. Many companies implemented an efficient operating system within the operating system -- folding the entire application within one OS process. This TP operating system implemented a private thread library, and built a private file system based on the host operating system raw file interface. This approach is typified by IBM's original CICS (Customer Information Control System) could support hundreds of clients on a 100 KB, half-mip server running atop DOS/360. The idea was reincarnated in the 1980s by Novel NetWare and by Sybase SQL Server.

In 1984 Sybase multi-threaded a single UNIX process, added stored procedures, and so got a 3x speedup and a 10x scaleup advantage over the process-per-client servers from Oracle, Ingress, and Informix. Today all the database vendors have copied the Sybase design and offer multi-threaded servers. The 1982 introduction of Novell's NetWare file server quickly evolved to a database and application server. By using inexpensive threads and by emphasizing the performance of the simple requests, NetWare was able to support many clients with a relatively modest server. NetWare Loadable Modules (originally called Value Added Programs) performed server applications and used NetWare services. General-purpose operating systems from IBM and Microsoft and others could not compete with NetWare's ability to scale. OS/2's and NT's threading mechanisms are substantially more expensive than Netware's. If history repeats itself, this will give NetWare a big advantage in scaling to hundreds of clients.

The process-per-server design addresses the percentage problem by providing very efficient services. NetWare is proud that it can service a client's disk request in less than a thousand instructions. This is 10x to 100x better than general purpose operating systems. The process-per-server design solves the polynomial explosion problem by having only one server process. There are X client connections to the server and the Y applications at the server collectively open Y*Z files. These opens are managed within the server's file system. These are manageable numbers.



Routers: A More Scaleable Design.

CICS, NetWare, and Sybase have been extraordinarily successful -- but there are problems with the process-per-server design. It does not scale to shared memory multiprocessors (SMP) because the single operating system process uses a single processor. The other processors just sit idle. Also, if the single process faults or waits in any way, the whole server stalls. Even worse, the process-per-server design does not scale to clusters of servers.

Beyond these scalability problems, the process-per-server model has a manageability problem. The design creates a monolithic process that collapses all applications into one address space. A bug in any application may crash the whole server. Changing any application may impact all others.

These scaling and management problems obviously suggest the idea of a **process-per-application-server**. The idea is to functionally specialize a process or processor to service a particular application function. One scales the system by adding servers for

each application. If an application saturates a single server, the application data is partitioned and a server is dedicated to each partition.

The **process-per-application-partition** technique is widely used to scaleup CICS, NetWare, Sybase and Oracle applications. The difficulty is that it re-introduces the polynomial explosion problem. The clients must connect to each server partition, log on to it and maintain a connection with it. The client code needs to route requests to the appropriate partition.

It is not be easy to partition most applications. A particular request my touch many partitions. There are often central files or resources used by all partitions or applications (the customer list, the price list, the bindery, ...). Partitioning such resources is not possible, rather they must be replicated or managed by a shared server. Nonetheless, process-per-application-partition is the most widely used scalability technique today.

All the solutions described so far involve two kinds of processes: clients or servers. These designs are called generically called t**wo-ball** models. All the two-ball models count on the client to find the servers, and count on the client to route requests to the appropriate server. Each server authenticates the client and manages the connection to the client.

The **three-ball** model of introduces a **router** function. The client connects to a router and the router brokers client requests to servers. The client is authenticated once and sends all it's requests via a single connection to it's router. This design scales by adding more routers as the number of clients grows.

Routers typically create and manage **application server pools**. A server pool may be distributed across the several nodes of a cluster. The routers balance the load across the pool. Each application can have a separate server pool. The router can run different pools at different priorities to optimize response time for simple requests. Should a server fail, the router redirects the request to another member of the pool. This provides load balancing and transparent server failover for clients.

IBM's IMS built in 1970 was the first three-ball system. It had a single router process. With time, the ideas were generalized by Tandem (Pathway, 1979), Digital (ACMS 1981 and RTR 1987), AT&T (Tuxedo 1985 and Topend 1991), and Transarc (Encina 1993) to provide many additional features. The rest of this article dwells on those other features.

The driving force for all these designs has been the need to scaleup to hundreds or thousands of clients per server -- for small systems the process-per-client model works fine. For applications involving more than 30 clients, conventional operating systems approaches suffer the percentage problem or the polynomial explosion problem. TP products and concepts addressed this scaleup problem. Once past the scalability problem TP monitors have gone on to address other key issues like programmability, manageability, security, integrity, and availability.

The process-per-client model had the virtue of implementation simplicity and gave the benefit of giving each client its own server process. But the design did not scaleup because of the percentage problem and the polynomial explosion problem. The two-ball model collapsed all the applications together thereby solving these two problems but creating some scalability problems.

The three ball model multiplexes the many clients down to a few server processes. This solves the polynomial explosion problem. The percentage problem remains an issue. The tree-ball model uses the operating system to provide processes. The benefit is that the router and applications can use SMPs and clusters. If the router is programmed carefully, and if the operating system dispatches well, the three-ball model can compete with the uni-processor two-ball systems. In addition, it the three ball router offers the ability to scale to clusters and so has a much better scalability story.

The three ball model allows the application designer to use either a process per server cpu, a process per application, or a process per client. The process-per-client is the most interesting case. By dynamically connecting the client to a server on an as-needed basis, the 3-ball router increases the duty cycle on each server. This solves the polynomial explosion problem while still allowing the application to have simple interactions with the client.

Initially classic CICS (on the mainframes) was implemented as a 2-ball model. But when CICS was re-implemented on UNIX to be portable, it was implemented as a 3-ball system built above Transarc's Encina toolkit. Now it is fair to say that all the popular TP-monitors (IMS, CICS, ACMS, Pathway, Topend, and Tuxedo) are three-ball systems.

TP Monitors and ACID Transactions

So far we have not mentioned the word transaction -- rather we used the ambiguous term TP. Originally TP-monitor meant teleprocessing monitor -- a program that multiplexed many terminals (clients) to a single central server. With time, the term grew to include much more than the multiplexing and routing function. As this happened, TP came to mean transaction processing.

Transaction processing is a *power term* in our vocabulary, evoking all sorts of products and concepts. To many, the core concept is a transaction as an ACID collection of actions (hence the word trans-action). ACID is an acronym for Atomic (all actions or no actions happen), Consistent (the actions transform the state to a new correct state), Isolated (the actions appear to execute as though there were no concurrent actions by others), and Durable (once the transaction commits, its effects are preserved despite any failures). There is an OSI/TP standard that defines how to make transactions atomic. There is an X/Open Distributed Transaction Processing (DTP) standard that defines a system structure and applications programming interface that allows servers to participate in atomic transactions.

The transaction tracking system in NetWare, the resource manager interfaces in TP systems, and the transaction mechanisms the many SQL products all contribute to building ACID applications.

TP monitors go well beyond a database system's narrow view of ACID applications. A TP monitor treats each subsystem (database manger, queue manager, message transport) as an ACID resource manager. The TP monitor coordinates transactions among them, assuring that the job is done **exactly-once** -- not zero times, and not two times. For example, a TP system will assure that the database gets updated and the output message is delivered (exactly once) and an entry is made in the work queue. Either all these things will happen or none of them will happen.

For most TP monitors, ACID is an afterthought or sidelight. What they really do is configure and manage client-server interactions. They help applications designers build and test their code. They help system administrators install, configure, and tune the system. They help the operator with repetitive tasks and they manage server pools. They connect clients to servers and provide efficient system services to applications.

Generalized Client/Server: Queued, Conversational, and Workflow

We have seen that most of TP-monitors have migrated from a two-ball to a three ball model in which the client performs data capture, local data processing and eventually sends a request to a router. The router broker's the client request to one or another server process. The server in turn executes the request and responds. Typically the server is managing a file system, database, or bulletin board shared among many clients.

This simple application design has evolved in three major directions: (1) queued requests, (2) conversational transactions, and (3) workflow.

Queued transaction processing is convenient for applications where some clients are produce data while others process or consume it. Electronic mail, job dispatching, electronic data interchange (EDI), print spooling, and batch report generation are typical examples of queued transaction processing. TP monitors routinely include a subsystem that manages transactional queues. The router inserts a client's request into a queue for later processing by other applications. The TP monitor may manage a pool of application servers to process the queue. Conversely, the TP monitor may attach a queue to each client, and inform the client when messages appear in his queue. Messaging applications are examples of queued transactions.

Simple transactions are one-message-in one-message-out client/server interactions, much like a simple remote-procedure call. **Conversational transactions** require the client and server to exchange several messages as a single ACID unit. These relationships are sometimes not a simple request-response, but rather small requests answered by a sequence of responses (e.g., a large database selection) or a large request (e.g., sending a file to a server). The router acts as an intermediary between the client and server for conversational transactions. Conversational transactions often invoke multiple servers and maintain client context between interactions. Menu and forms-processing systems

are so common that TP systems have scripting tools to quickly define menus and forms and the flows among them. The current menu state is part of the client context. Application designers can attach server invocations and procedural logic to each menu or form of the hierarchy. In these cases, the TP-monitor may manage the client context and controls the conversation with a workflow language.

Workflow is the natural extension of conversational transactions. In its simplest form, a workflow is a sequence of ACID transactions following a workflow script. For example, the script for a person-to-person electronic mail message has the script compose-deliver-receive. More commonly, scripts are quite complex. Workflow systems capture and manage individual flows. A client may advance a particular workflow by performing a next step in the script. The system designer defines workflow scripts as part of the application design. Administrative tools report and administer the current work-in-process.

How Do You Write Applications in a TP System?

TP systems vary enormously, but the general programming style is to define a set of services that the server will provide. Each service has message interface. The implementation task is then reduced to defining client programs to generate these messages and server programs to service the messages.

If the system provides a two-ball model, then your server program will run inside the TP monitor and follow the TP monitor rules (e.g., a NetWare NLM, a Sybase Transact SQL program, an Oracle PL/SQL or a CICS application program). If the system provides a three-ball model, then you write your service program in a conventional programming language (C, C++,. COBOL, or whatever) which runs in a standard process using the TP system library to get and send messages.

The service programs are invoked with a message, either from a queue or directly from a client. The service program executes the application logic, and responds to the client with a response message.

TP-systems generally provide some tools to automatically construct forms-oriented clients and to automatically construct simple application servers which can be customized.

System Configuration, Management, and Operation

TP monitors allow us to build huge client server systems. Before three-ball TP-monitors appeared on UNIX systems, they had several difficulties going much beyond 300 clients. Even that was almost unmanageable. Today, most UNIX vendors are reporting TPC-A and TPC-C benchmarks demonstrating thousands of clients attached to a single server and tens of thousands attached to a cluster of servers.

Now that you can build such systems, how do you manage them? TP monitors have evolved a broad suite of tools to configure and manage servers and large arrays of clients.

To start the TP monitor thinks of each database and application as a **resource manger**. Resource mangers are register with the TP monitor. Thereafter, the TP monitor will start (and restart) the resource manager), inform it of system checkpoints, and system shutdowns, and will provide an overall operations interface to coordinate orderly system startup and shutdown. Database systems, transactional queue managers, and remote TP systems all appear to be resource managers.

Each application is viewed as a collection of service programs. The service programs may all be packaged into a server, or they may each be packaged in separate servers. Typically short-running or high-priority services are packaged together, and batch or low-priority work is packaged in separate server processes. The TP monitor provides tools to define the packaging of services into servers.

Once the service packaging is decided, each service is give security attributes. Security is typically done in terms of roles: each client has a role. Security is determined by role, workstation, and time. For example, it may be that clerks are only allowed to make payments when they are using an in-house workstation during business hours. The TP monitor provides a convenient way to define users and roles. It also provides a way to specify the security attributes of each service. It authenticates clients and checks their authority on each request. The TP-monitor rejects requests that violate the security policy.

TP-monitors manage the size an priority of server pools so that each service meets it's response time goals. Each service has a desired response time. Some services are long-running reports or mini-batch transactions, while others are simple requests. The administrator can control how many processes or threads are available to each service. These server pools can be spread over multiple nodes of a cluster. This number can be dynamic, growing as load increases. When the number of requests for service exceeds the maximum size of the server pool, requests are queued until the next server becomes available.

Systems involving thousands of clients and hundreds of services have many moving parts. Change is constant. TP-monitors manage online change, while the system is operating. They allow a new service to be installed by creating a server pool for it. More interesting, they allow a new version of an existing service to be installed by installing the new version of the program. New servers will be created using this new program. The system can gradually kill off the old servers as they complete their current tasks and after a minute or so the new application is fully installed. Of course this only works if the new service has the same request-reply interface as the old one.

TP-systems mask failures in many ways. At the most basic level, they use the ACID transaction mechanism to define the scope of failure. If a server fails, that transaction is backed-out and restarted. If a node fails, server pools at that node are migrated to others until the node is restarted. When the node restarts, resource mangers at that node are restarted and recovered using the TP-systems transaction log.

Modern database systems can maintain multiple replicas of a database. Each time one replica is updated, the updates are cross-posted to the other replicas. TP monitors can complement database replication in two ways. First they can submit transactions to multiple sites so that update transactions are applied to each replica. This avoids the need to cross-posting database updates. It is difficult to get transaction replication to work because each node must generally execute the transactions in the same order to get the same end result. This ordering is difficult to control.

More typically, TP systems use database replicas in a fallback scheme -- leaving the replication to the underlying database system. If a primary database site fails, the router sends the transactions to servers the fallback replica of the database. This hides server failures from clients. Because the router uses ACID transactions to cover both messages and database updates, each transaction will be processed exactly once. Without this router function, clients must explicitly switch servers when a primary database server fails. They probably will not get exactly-once semantics in making this switch over.

TP-monitors include sub-systems to automate the repetitive management tasks of tracking

Emerging Transaction Processing Trends

The main trend in transaction processing, as with all of computing, is to make things easier and more automatic. Other articles in this issue discuss the automatic construction of clients, servers, and automating the split between client and server.

The concepts of remote procedure call, and object request brokering are familiar to TP users. These are the bread-and-butter concepts of a TP-system. Current ORBs give new meaning to the term "bad performance." They are pioneering the concept of seconds per transaction. Once they realize the importance of performance, they are very likely to embrace the three-ball model and the techniques used by TP-system.

Meanwhile the venerable TP monitors (CICS, IMS, ACMS, Pathway, Tuxedo, Encina, and Topend) will continue to evolve as high-performance message routers and server pool mangers.