INFORMIX-OnLine's Dynamic Scalable Architecture

By Gregg A. Christman

This article describes how INFORMIX-OnLine is evolving to meet the technological demands of symmetric multiprocessing and massively parallel processing architectures.

Just as computer systems are evolving from uniprocessor to Symmetric Multiprocessing (SMP) and eventually to Massively Parallel Processing (MPP) architectures, INFORMIX-OnLine must also evolve to meet these technological demands. This article discusses the multiple phases of Dynamic Scalable Architecture (DSA) evolution as well as an alternative architecture that was considered, but later rejected in favor of DSA.

Figure 1 illustrates the three phases of OnLine's evolution as it relates to the changing computing environments. The OnLine phases include the following releases:

- INFORMIX-OnLine/DSA
- DSA/Parallel Database Queries (PDQ)
- DSA/Extended Massively Parallel (XMP) Processing

Phase 1: INFORMIX-OnLine/DSA

The Dynamic Scalable Architecture, released in 1993, provides the foundation for the generations of INFORMIX-OnLine that follow. The new architecture provides a flexible threading architecture for both Online Transaction Processing (OLTP) and Decision-Support System (DSS) environments. In OLTP environments, a small number of server processes efficiently handle multiple user sessions. In DSS environments and batch jobs, a single-user session can efficiently spawn multiple threads that run in parallel, utilizing more of the hardware's resources.

The initial release of DSA took advantage of the inherent parallelism within the SMP architecture by using parallel database backups and restores and building indexes in parallel. True database parallelism came when DSA evolved to the next phase: DSA/PDQ.

Phase 2: INFORMIX-OnLine/DSA with PDQ

DSA with PDQ, released in 1994, splits a single database operation into a set of parallel operations. Parallelization can dramatically reduce the processing times for both OLTP and DSS operations since multiple processors perform work for a single transaction.

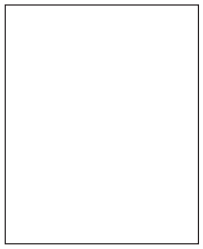
DSA/PDQ runs on tightly coupled SMP designs. This release is targeted at several computer environments including OLTP, DSS, Online Complex Processing (OLCP), and batch processing.

Phase 3: INFORMIX-OnLine/DSA with XMP

DSA with Extended Massively Parallel (XMP) Processing is scheduled for release in 1995. DSA/XMP will run on loosely coupled and MPP architectures. DSA/XMP can take advantage of loosely coupled distributed cluster designs such as IBM's High Availability Clustered Multiprocessing/6000 (HACMP/6000).

Architecture of DSA/XMP

DSA/XMP will enhance performance on loosely coupled architectures by taking advantage of multinode partitioning. It will process SQL functions in parallel across nodes or processors. Multinode partitioning enables the database server to place pieces of the same logical table on different



Gregg A. Christman

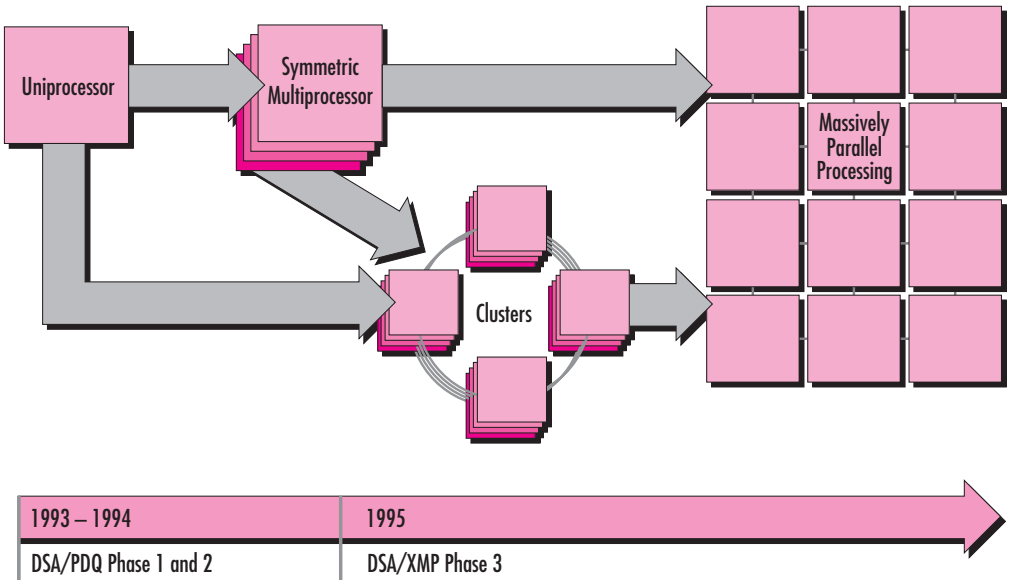


Figure 1. Evolution of computing and DSA

nodes. The partitioning is defined with the `CREATE TABLE` command and allows data to be placed on the various nodes based on a range of values (other partitioning methods are also supported). For example, a customer table being placed on a cluster with five nodes can have all customers with zip codes beginning with 1 placed on node 1, those with zip codes beginning with 2 on node 2, and so on.

Informix can parallelize operations such as scans and joins across nodes, which maximizes the use of the hardware for database processing. SQL requests can span many database servers and many nodes or processors. This scalability is achieved through DSA's dynamically configurable pool of database server processes called Virtual Processors (VPs).

Virtual Processors

Rather than initiating UNIX processes to provide client access to the database, DSA spawns lighter-weight mechanisms called *threads* to execute user requests. Threads can execute in parallel as processors are available. The database server processes manage active threads and can effectively switch among them. The database server processes—the VPs—are similar to physical processes in this sense.

DSA contains a pool of VPs. Multiple threads can run concurrently on different VPs. Since all data is in shared memory, any VP can execute any thread. This allows threads to migrate across processors for load balancing.

For efficiency and tuning versatility, VPs are grouped into *classes* (Figure 2 shows examples). Each class is optimized for a particular function. Threads are transparently scheduled across the VPs in the relevant class. The VP pool can be adjusted online to accommodate periods of unusual activity or load mixes.

Figure 3 illustrates an example of DSA utilizing multiple CPU VPs to parallelize an SQL scan and join request. The steps shown in the figure are as follows:

1. The network VP receives the scan request.

Class	Description
`cpu`	Performs SQL data manipulation for clients
`tlitcp` and `soctcp`	Processes incoming client requests
`aio`	Performs asynchronous I/O

Figure 2. Examples of VP classes

Parallelized Scan and Join

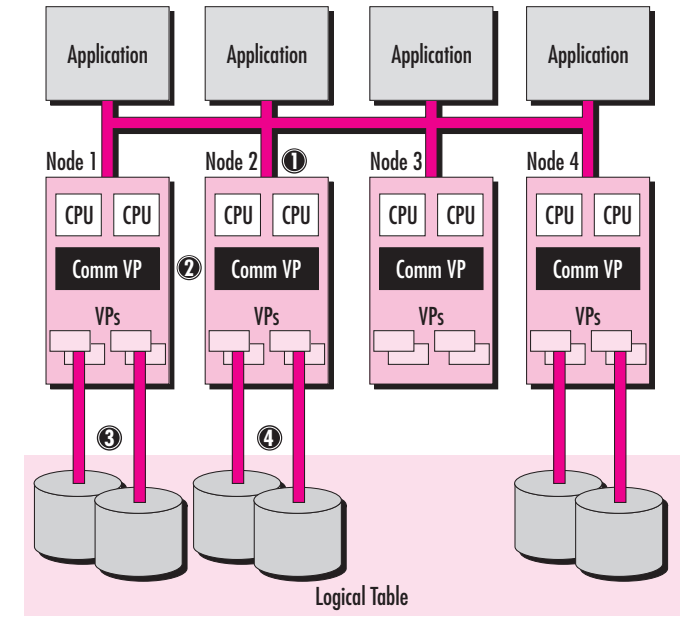


Figure 3. An example of multiple VPs used to parallelize an SQL scan and join request

2. The network VP passes the request to the database servers involved in scanning the data.
3. The CPU VPs on the appropriate database servers issue non-blocking stored-procedure or SQL requests in parallel. The requested data is retrieved from the pieces of the database table that reside on the node.
4. If required, the resulting data is joined in parallel and returned to the destination database server. No intermediate temporary files are required for a join operation.

On SMP nodes within the loosely coupled architecture, multiple CPUs on the same machine can be used in the parallel query.

Database Partitioning Methods

Informix supports four different database partitioning methods. Each node can run its piece of the XMP server and own portions (partitions) of the table.

DSA/XMP allows a table to be partitioned across any combination of nodes within the system using any of these four methods.

Range partitioning: This is ideal for requests that are based on a range of values and are commonly accessed by one index key. DSA/XMP range partitioning can be based on either the index keys or the actual data. Range partitioning enables queries to be selectively sent to a subset of the nodes in the system, reducing the overall load on the system and improving throughput.

Figure 4 shows range partitioning for a customer table. Partitioning is determined by the first character of the customer's last name. Customers with last names beginning with A to F have records stored on part of the customer table on node one. The remainder of the customer records are stored on the second or third node.

Round-robin partitioning: Round-robin partitioning provides perfectly balanced partitioning. The price of perfect balancing is the cost inherent in sending queries to all nodes for applying against all partitions. As illustrated in Figure 5, the records are sent to a specific node based on the order in which the record was inserted.

Round-robin partitioning increases the overall system load. It can also decrease throughput since all the storage nodes will be accessed. For this reason, it is unlikely that round-robin partitioning would be used in heavy OLTP environments. The concept of round-robin partitioning is similar to disk striping in the filesystem.

Hash partitioning: Hash partitioning approximates the perfect population balancing of round-robin partitioning while enabling queries to access a single partition selectively. Since the DSA hash algorithm can quickly determine the node on which the required data resides, hash partitioning avoids the overhead associated with round-robin partitioning.

Hash partitioning is useful when the characteristics of the data values in the partitioning attri-

Range Partitioning

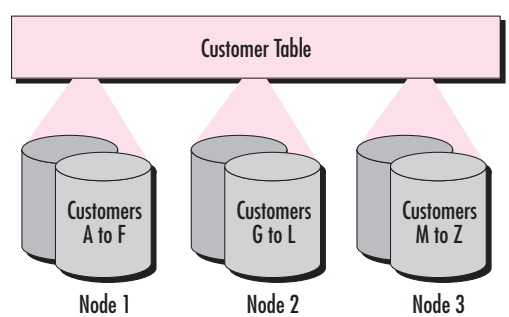


Figure 4. Example of range partitioning based on the first character of the customer's last name

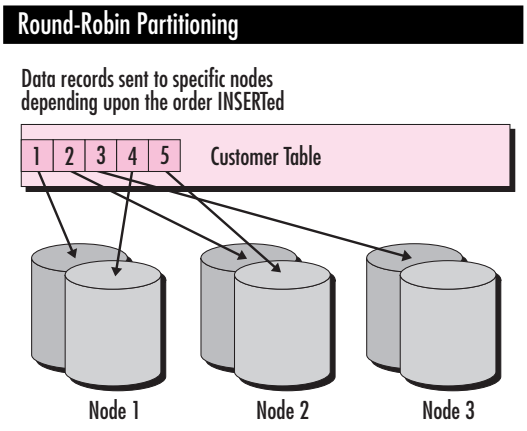


Figure 5. Round-robin partitioning provides perfectly balanced partitioning

bute are either not well known or dynamic by nature.

Expression partitioning: Expression partitioning, illustrated in Figure 6, enables queries to be selectively sent to a subset of the nodes in the system, similar to range partitioning. The partitioning is based on a predefined SQL expression (including the use of `or` expressions). The expression is evaluated and, based upon the results of the expression, the data is sent to the appropriate node.

DLM: An Alternative to DSA/XMP

Informix considered using a Distributed Lock Manager (DLM), but later rejected it. A DLM allows a common database on shared disks to be accessed by multiple nodes in a cluster system.

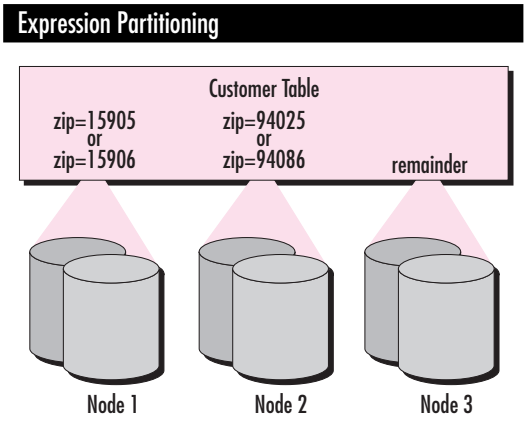


Figure 6. Partitioning based on a predefined SQL expression

Although the DLM is a centralized resource, which can be a problem, it is not the DLM concept that is inherently a problem. The problem was that with existing DLM-based database management solutions, the DLM is used to enable data shipping—data being sent from the shared database on disk to the local buffer cache of the node requesting the data. As shown in Figure 7, the node must then write the data back to the disk when it is requested by another node. In the worse case, data is shipped from one cache to disk, then from disk to another cache.

Lock latency is an issue with DLM since the system will typically require two milliseconds for buffer latch message passing. As more nodes are added to the system, data shipping via the I/O subsystem becomes a bottleneck and performance degrades.

Within a small cluster, scalability of a DLM-based solution might be adequate if the data being requested is partitioned appropriately across the nodes to reduce lock contention and I/Os. Scalability would be achieved through application-level partitioning of access; that is, the application is structured so that access to a given subset of data is restricted to one node, particularly for updates. Using the TPC-A schema (BRANCH, TELLER, ACCOUNT) as an example, applications would be structured so that all

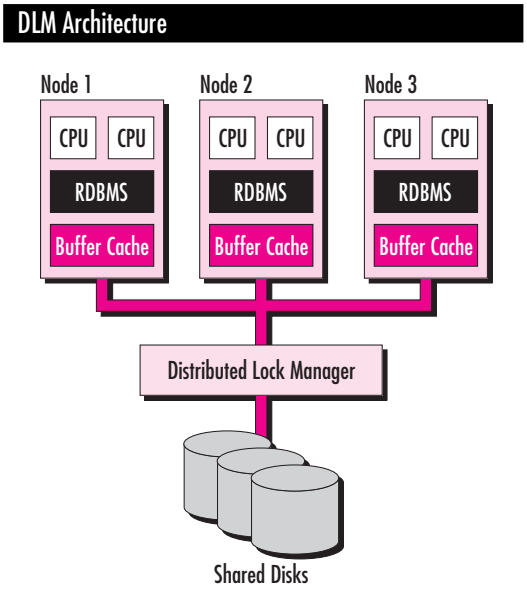


Figure 7. Distributed Lock Manager architecture

requests for a given BRANCH go to the same node.

The application-level partitioning would be required to achieve effective database performance. This is because each node is restricted to accessing a subset of data, and DLM requests and data shipping are minimized. But this forces an unnecessary burden of complexity upon the users, because in practice, it can be difficult to partition applications. For many applications, a physical database design in which data access can be restricted to one node is not possible. Also, applications should not be required to know how data is partitioned (for data independence). Even if the application developer had this knowledge, it can be difficult to decide which node to access for best performance.

DSA was designed to use the breadth of open systems hardware.

Scalability of a DLM solution that uses data shipping becomes an issue because as the number of nodes increases, the data-shipping problem can easily compound. The additional throughput gained by adding another node to the database cluster decreases or becomes negative. If application partitioning is not being done, data shipping is required. As nodes are added, internode DLM traffic increases and more data must be shipped between nodes—using up interconnect bandwidth. Once the interconnect is saturated, throughput cannot be increased by adding more nodes.

Transactions also wait longer for data. The problem negatively affects data that is frequently accessed. In some situations, there could be so much waiting and interference between nodes that throughput can decrease by adding an additional node.

DSA/XMP For Database Performance

The distributed database partitioning approach provides superior scalability over the DLM approach. Interference (contention among processes) is kept low by minimizing resource sharing. The volume of data shipped across the network is minimized since only questions (SQL functions, parallel scan operations, parallel join operations) and answers (resultant data) are moved through the network.

DLM-based databases typically require many messages to be generated (to coordinate the buffer caches and locking) to retrieve the page. DSA/XMP filters out unneeded data before the results are returned. Since there is no need to manage distributed locks or segmented buffer caches, only the resultant data is moved across the network.

Raw memory accesses and raw disk accesses are performed locally in a processor. Only the filtered (reduced) data is passed to the client program. Minimizing traffic on the interconnection network allows more nodes to be added to the system without performance degradation, making it a more scalable design.

DSA/XMP distributes function to where data resides, rather than shipping the data to where the work needs to be done. Distributing the function provides for parallelism and reduces data shipping. This algorithm will run well with modest amounts of memory, achieving near linear speed-up. It will also take advantage of additional memory when available.

DSA was designed to use the breadth of open systems hardware, available from uniprocessor to SMP to loosely coupled clusters and MPP architectures. This is unlike alternate technologies, such as DLMs, which create inherent scalability bottlenecks that could prevent RDBMS users from fully utilizing the power of their computing environments.

***Gregg A. Christman**, Informix Software Inc., 4100 Bohannon Drive, Menlo Park, CA 94025. Mr. Christman is the manager of the Marketing Analysis Group at Informix. He provides research about the database server industry and how to keep Informix's database server products at the leading edge of that industry. Previously, Mr. Christman had managed the INFORMIX-OnLine database server product line. He has been with Informix for over seven years, starting in the Technical Support Group and later serving as the senior product specialist for INFORMIX-TURBO. He holds a degree in Engineering Psychology from the University of Pittsburgh.*