Parallel Approaches in Relational Database Systems

Cormac Burke, Cray Research Business Systems Division

Introduction

With the relatively recent development of new parallel implementations in Open Database Systems and the coming to market of new parallel System Technologies there is a great deal of confusion as to how these technologies can address the data processing requirements of today's corporate IS departments. This paper will attempt to present an overview of the primary applications of today's IS departments, an overview of different database/systems architectures, and a discussion of how these applications may fit into such architectures. For the purposes of this discussion we'll put the primary business applications into two categories; Online Transaction Processing (OLTP) and Decision Support Systems (DSS). The OLTP side can be characterized by large numbers of users (1000s), relatively small/simple transactions, high update activity, and a requirement for minimum downtime. OLTP data is typically considered 'Operational', and contains the data generated by the transactional activity of a business (e.g. financial data from sales, banking transactions, etc.). DSS applications can be characterized by a smaller user community (e.g. < 100 users), more complex transactions which tend to be read only and often will require table scans. Decision Support applications often utilize operational data to determine trends, which will then influence business decisions (e.g. inventory levels for particular location, marketing strategies, sales forecasts, etc.).

The ways in which different database systems and computer architectures address the requirements of these two applications vary widely. To approach the problem I'll first describe the current database approaches, move onto different computer architectures, and them discuss how the different database products map onto the computer architectures.

DBMS Architectural Overview

In the past 18 months, both Oracle and Informix have come to market with new parallel systems to enhance their performance and functionality, with major emphasis on parallelization of Decision Support workloads. Additionally, Sybase has released their Navigation Server product, another implementation of a parallel database system. While all of the new systems are going after the same basic function (parallelization of queries), they do so with very different approaches.

With Oracle version 7.1, Oracle released their Oracle Parallel Query option (PQO). This approach at parallelization basically extends the standard, process based Oracle model of parallelization by creating 'Query Servers', which are OS level processes that are used to simultaneously read/sort/join data. The parallelization is done from an operating system level (with Oracle's Query Coordinator deciding how to divide work among slaves), and the optimizer is actually unaware of the parallelization. Some number of Query Servers reads data, then passes data on to other servers (this is considered a producer/consumer relationship) We've seen near linear scaling on certain DSS queries with Oracle Parallel Query on the Cray 6400. Some functions, such as Index Creation, are still relatively immature in terms of ability to run in parallel, and present Oracle Corporation an area to enhance their product. Oracle Parallel Query Option can also run on Oracle's Parallel Server product, which is a shared disk system, described below.

Informix brought a new architecture to market with their version 7.1. Referred to as Informix-Online Dynamic Server, this version of their RDBMS is multithreaded internally, which gives it the capability to run on parallel processors by virtue of a small number of Unix processes or 'Virtual Processors'. Informix utilizes Operating System threads to parallelize functions internally. In addition to multi-threading in the RDBMS, Informix implemented new functionality to enhance parallel query performance. Two key components of this are 'fragmentation' and hash joins. Fragmentation basically allows data to be 'striped' or evenly distributed from within the RDBMS, and provides some control as to what algorithm determines how data is spread (e.g. round robin or value based distribution). Scaling appears to be near linear with some queries in Informix 7.1 on the CS6400. Informix's next release, version 8 (also called XPS), will run on MPPs as a shared nothing. It is described in more detail below.

The Sybase Navigation Server is basically an extension or amalgamation of Sybases' Virtual Server Architecture. This DBMS is parallelized by running multiple instances of the server, with an internal threading mechanism to handle scheduling of work. Queries are not parallelized, but can be spread to different database/servers by going through the Navigation Server, which will route queries. Given that Sybase is not currently available on the CS6400, scalability data is not readily available. However, it is generally acknowledged that Sybase does not scale much beyond 8 processors, and given that the Navigation Server is not generally available it seems unlikely that this approach to parallelism will work well.

Systems Architecture Overview

There are 3 primary architectural trends in the Open Systems market today; Shared Everything, Shared Disk, and Shared Nothing. One can draw correlation to known technologies as follows: Shared Everything and Symmetrical Multiprocessors, Shared Disk and Clusters (such as the VAX Cluster), and Shared Nothing and MPPs. Different database systems are available on some but not all different architectures. To take into account the various different potential combinations I'll touch briefly on the systems architecture first, then go into specific database implementations on various architectures.

Shared Everything Architectures are probably most familiar, common, and commercially acceptable architectures. They basically consist of a shared set of processors, with a single Operating System image, and shared memory and disk, all on a common bus. These systems have been in existence for a number of years, and are produced by most open systems manufacturers, including Sun, HP, IBM, and Cray. Typically the performance and scalability of these systems has been limited by the Operating System environment, application scalability, and bus bandwidth. New releases of Operating environments such as Solaris 2.4 have addressed the OS limits, latest generations of RDBMS products have begun to address the application scaling limitations, and systems like the CS6400 with 1.76GB of bus bandwidth have begun to address the hardware limitations.

Shared Disk/Clusters evolved from the VAX Cluster products of the late'80s. They basically consist of multiple distinct nodes sharing a disk farm. There is some dedicated interconnect for inter-node communication (often Ethernet, fast Ethernet, or ATM), but the key issue is that the nodes share disks and can both concurrently access the disks. This approach has been popular for Digital, and with the development of Oracle Parallel Server has become a popular approach to parallelism with Open Systems vendors such as Sequent, Pyramid and Sun. This approach has also been used in some cases with MPP systems such as Ncube and IBM’s SP2 (while this system is really a shared nothing system, IBM uses a virtual file system layer to allow non-shared disks to be seen by the application as shared disks).

Shared Nothing/MPP systems are relatively new entries into the commercial market. The key components of MPPs are separate, distinct nodes, each running it's own instance of an operating system, with a dedicated disk/memory subsystem per node. Typically the nodes are connected via some kind of high speed, packet switched interconnect to attempt to minimize latency in the internode communication that is required with these systems. IBM's SP2 is probably the most widely known MPP. ICL Goldrush fits into this category as does the Meiko and the Pyramid Meshine.

Database Implementations

All of the major database products are currently supported on shared everything/SMP systems, with varying levels of scalability. They take different approaches (e.g. Informix with shared 'Virtual Processors' versus Oracle with dedicated server or 'shadow' processes), but all run in a similar fashion on SMP machines. The implementations differ much more distinctly as you move into clusters and shared nothing environments, with significant impacts on business applications.

In Shared Disk/Cluster systems, Oracle is currently the only available product. Oracle Parallel Server is designed to run on multiple nodes, which share a disk farm. There are multiple instances of Oracle (each with it's own System Global Area and Log devices) running, one per node. These nodes share the database via some kind of shared disk interconnect(e.g. dual ported SCSI disks), and are synchronized via the use of a Distributed Lock Manager, or DLM. This approach requires a dedicated interconnect for DLM traffic, ideally faster than Ethernet given the traffic and impact of additional latency.

Oracle Parallel Server (OPS) and Clusters are typically sold as a high availability option for a database server. This high availability is accomplished through the ability to access the database from any node, even if one node has failed. If one node suffers a failure the other node will continue functioning, and users connected to the failed node can reconnect to the live node. There are important configuration issues to consider here, e.g. if you are planning to implement a full failover system you may need to have enough headroom for both user communities on both nodes in order to maintain the throughput/response time you had before a system failed. While the High Availability functionality is a valid selling point for Clusters and OPS, some vendors will attempt to sell this as a scalability solution. This is particularly relevant to Cray with Sun's PDB product. While Sun may claim that a Sun PDB built from 2 20 processor SC2000s is equivalent to a 40 processor CS6400, the truth is that the scaling realized from Clusters and OPS has been in the 10-30% range. This scaling is dependent on how well the data can be partitioned to minimize sharing of actual data blocks between nodes, something that is quite difficult for most OLTP applications (see mention of block pinging, below)

Informix XPS (Version 8.0) is designed to run in loosely coupled environments, without the requirement for shared disk. It is a true 'shared nothing' system, in that it maintains separate data sets which can be globally accessed via a single transaction or query. The benefit here is in the ability to distribute work evenly between several nodes in an MPP/distributed system, with each node running in parallel. While this may work for certain transactions where data is properly distributed and access patterns are known and predictable, this architecture can be problematic for either randomly accessed or volatile data. If a user runs a query which focuses on a particular data set which was not anticipated, this could cause the query to run exclusively on the node where the pertinent data resides. Informix XPS is built around the notion that queries will access data according to the distribution/fragmentation algorithm. If a query does not take advantage of the distribution algorithm specified, the expected parallelism will not be realized.

Like Informix XPS, Sybase Navigation server is designed to run in loosely coupled environments. However, rather than having location reference in various data dictionaries, Sybase routes transactions to their appropriate servers via an 'intermediary' server, or Navigation server. This technology was developed jointly by Sybase and ATT-GIS(formerly NCR), and was never publicly benchmarked. Most sources indicate that it does not perform well, and Sybase seems to have stepped back from their belief that this would be their answer to scalability problems.

What really works???

With the variety of systems architectures, database implementations, and application types, it's apparent that there are a number of different options and combinations for parallel database processing. Some applications may work with some system implementations while others will not. How does one determine what will and will not work, and how does one make buying decisions given the quantity and complexity of all this information?

It is important to understand that application types will greatly impact how the various different database products run on the different architectures. For the sake of simplicity we'll focus first on DSS and OLTP on Shared Disk/Cluster architectures, and then move on to both application types and Shared Nothing and shared everything. From there we'll analyze how one finds the best compromise for their particular needs.

As discussed above, Clusters and Oracle Parallel Server require a shared disk, with multiple nodes running multiple instances of Oracle. Each instance/node runs transactions or queries concurrently, depending on he Distributed Lock Manager to ensure that the database remains consistent in spite of updates from multiple nodes. This may work in cases where the data being accessed by the user base is not shared (and subsequently requests for data do not clash), but unfortunately this is rarely the case for the types of enterprise applications where clusters are deployed.

As mentioned above, Oracle Parallel Server requires the running of multiple Oracle ‘instances’, which run against a common database. Updates to data are mediated by Distributed Lock Managers, which run on each node. As mentioned above, the occurrence of ‘block pinging’, which happens with concurrent updates adds both overhead and latency to update oriented transactions in Oracle Parallel Server installations. Block pinging occurs when one node requires a block which is already in the cache, or SGA, of the other node, and that particular block is ‘dirty’, or has been updated. The requesting node cannot complete it’s read until the node which is holding the dirty block has written it out to disk. In the meantime, both nodes have to pass messages regarding the state of the block, adding both latency and interconnect traffic and overhead to the transaction. This appears to be one of the key issues pertaining to scalability of Oracle Parallel Server, and has not yet been perfected by any vendor. The only way to avoid block pinging is to partition the database such that users from specific nodes do not require data being used by other nodes. This is generally difficult to do, as one of the primary features and benefits of relational systems is access to shared data.

The issue of block pinging is primarily a problem for OLTP applications. However, there are numerous issues pertaining to DSS workloads in shared disk/OPS environments which impact performance significantly. As mentioned above, Oracle handles parallel query by dispatching work among query servers or ‘slaves’, using operating system level processes to run parallel tasks. In order to do this efficiently the query optimizer and/or query coordinator must submit work to query slaves evenly to ensure that their tasks can be performed in parallel and without clashing. This becomes increasingly problematic when the dispatching occurs across nodes, which is a requirement of parallel query in Oracle Parallel Server environments. Additionally, the basic function of parallel query is one in which some number of query servers are ‘producers’, and they read data off of disk, while some number are ‘consumers’, and they process (sort, join) data which comes off of disk. Data that is read by the ‘producer’ query servers is then passed off to the ‘consumers’, and the data will be passed from server to server based on a hash scheme which will go across nodes. Once again the issue of traffic and added latency lead to degraded performance. One other relevant issue pertaining to Oracle Parallel Server systems is the fact that they are designed around a certain particular architecture, namely a shared disk farm. For systems which do not inherently support this (e.g. IBM SP2) this presents a problem, as the vendor must then provide some layer to simulate a shared disk environment (in the case of IBM this is done via a virtual file system) which will typically have some performance impact on the IO subsystem for all nodes.

Shared Nothing systems take a very different architectural approach to managing database systems than do Shared Disk/Shared Everything systems in that they actually run separate databases or data ‘sets’, and spread these data sets across the nodes in the system. This is actually not a very new idea, as the notion of distributed databases managed with Two Phase Commit has been around for many years. What is new is that some Open Database vendors are supporting and selling this concept, and basing next generation architectures on it. The selling point for Shared Nothing systems is their theoretical potential for scaling. Given that they do not share a bus or share access to resources such as disks or memory they should scale infinitely. However, scaling in commercial applications is dependent on how the underlying software scales, and herein lies the problem for Shared Nothing systems.

Shared Nothing systems have typically not been widely sold as OLTP solutions. This is primarily due to the fact that the update oriented nature of OLTP workloads requires 2 phase commit support, and no Open Systems vendor has been able to come up with a high performance approach to 2 phase commit. The nature of 2 phase commit is to pass messages to ensure that an update can occur across nodes while maintaining data integrity (e.g. an update may span nodes, and in order to maintain atomicity in a transaction the transaction must complete on all nodes or on none at all). The latency and complexity which accompany this message passing make OLTP workloads and shared nothing architectures very difficult to implement, and for the moment restricts OLTP workloads to Shared Everything/Shared Disk systems.

Shared Nothing systems appear to be more suited to DSS applications, given their read only nature. Data is distributed among nodes (typically based on some key value), and when a query is executed it is routed to the appropriate nodes. This seems to be a reasonably sound approach if all queries are keyed off of the distribution key, however it presents a large problem for queries which don't utilize the expected/defined key as a qualifier. For example, if the distribution key is month, and all sales data is distributed among the nodes based on month, this distribution may work well for queries which require even access to all months. However, for doing quarterly or month end closing this could be problematic, as the data may be isolated to 1 or a few of the available nodes. Given that the majority of queries executed in DSS systems are ad-hoc and subsequently unpredictable, this presents a sizable problem for Shared Nothings systems in DSS environments.

Given that Shared Everything Systems/SMPs are the underlying foundation of most Shared Nothing and Shared Disk implementations, it should come as no surprise that commercial applications function quite well on these systems. In OLTP environments they do not suffer the impact of 'block pinging', as there is only a single instance of the RDBMS running, and there is no need to synchronize updates between instances. Since all data is resident on a single node, there is no need for 2 Phase Commit to ensure that updates are atomic. And in the realm of DSS, all query routing occurs locally, so data does not have to pass over an interconnect from producer to consumer. Additionally, given that all data is local, there is no partitioning problem as one sees with MPPs in Shared Nothing environments. All data is local, and given the shared processors/memory queries cannot bottleneck on a single node if the query does not follow the predefined access patterns.

There are still numerous issues required to ensure high performance in commercial applications on shared everything systems. As mentioned above, the Operating System, RDBMS, and Hardware Platform must all provide the appropriate scalability to ensure that the applications perform sufficiently. In the case of the Cray CS6400/Starfire series, the underlying OS is Solaris 2.x. Industry benchmarks have proven that this is the most scalable general purpose Operating System on the market, and Cray has enhanced the OS significantly to add value and extend the scalability. Among these enhancements are memory and processor partitioning, improvements to memory management, ability to address more memory/processors/busses, and more highly parallelized data structures within the operating system kernel. In the area of RDBMS scalability, Cray's Strategic Applications Engineering organization works closely with vendors such as Oracle and Informix to identify and address scalability issues in their products. These issues include isolating particular locks/latches within the RDBMS, determining methods to improve query optimization, and enhancing how the DBMS products utilize the base system IO. As far as the base Hardware platform, Cray has utilized it's years of experience in highly parallel systems, coupled with the availability of commodity components such as SPARC/Solaris/SBUS devices to produce the highest bandwidth SMP system on the market today.

Conclusions

While System and Database Vendors are aggressively working to develop next generation systems based on Clusters and possibly MPPs, these product combinations still have severe limitations. It will most likely be 2 to 3 years before these products reach an appropriate level of maturity to realistically address today's business problems for the commercial world. From both a performance and manageability point of view, there is a great deal more work that must go into these products.

Given that the key point of these products is to address the perceived limitations of the more traditional Shared Everything/SMP machines, it seems to make sense that one would look at the more sensible approach of improving on high end SMP systems rather than pursuing radically different, immature architectures such as Clusters/MPPs.

In the future it's clear that the lines will blur between Shared Everything/Shared Disk/Shared Nothing architectures. As the software products mature to take advantage of new technologies, the underlying systems will have capabilities to address multiple architectures (via Domains, non-uniform memory architectures, etc.). However, that capability is a ways off, and given the availability, scalability and manageability requirements of commercial systems it only makes sense to trust known, proven technology which can be leveraged in the future to capitalize on new technologies.