Operating System Extensions for the Teradata Parallel VLDB

John Catozzi – Senior Teradata Architect
NCR
17095 Via Del Campo
San Diego
CA 92127, USA
John.Catozzi@NCR.com

Sorana Rabinovici – Senior Teradata Architect
NCR
100 N Sepulveda Blvd
El Segundo
CA 90245, USA
Sorana.Rabinovici@NCR.com

Abstract

This paper describes the new architecture for supporting the Teradata commercial VLDB on several new operating environments. We start with an overview of the Teradata database software architecture, specifically the Parallel Database Extensions (PDE) that serve as a layer between the NCR Unix OS and the database software. We then describe the challenges of implementing an Open PDE for several new platforms, (Microsoft Windows 2000, Linux, HP-UX, and Microsoft Windows XP). Finally we show some performance data to compare the original NCR Unix version with the new Open Version of PDE for Teradata using published one-terabyte TPC-R data.

1. Introduction

Teradata is the leading commercial VLDB for datawarehousing and decision support applications. It is a parallel database originally implemented on a proprietary operating system and hardware platform from 1980-1994. Version 2 of Teradata was created on NCR's MP-RAS SVR4 Unix platform. That version (presented at VLDB '95) embodies a collection of Unix kernel-mode operating system extensions known as the Parallel Database Extensions (PDE) for Teradata. Packaged up as a standard "package add" feature, these extensions allow the former proprietary Teradata Database to run on NCR's standard Unix platform. Recently these Parallel Database Extensions were re-architected in an open manner to allow the deployment of the Teradata Database on multiple open platforms. The first implementation of this new Open PDE was for the Microsoft Windows NT 4.0 (now Windows 2000) platform. This version was first deployed in the marketplace in 1998. Today there are dozens of systems running on this Microsoft Windows 2000 version. This Open PDE is now being further ported to multiple platforms including the 64 bit versions of Linux, HP-UX, Solaris, and the 32 and 64-bit versions of the Microsoft Windows XP operating system. The challenge in architecting Open PDE was in making the porting process affordable and the results maintainable, without compromising the world class performance achieved by the original specific MP-RAS implementation.

Proceedings of the 27th VLDB Conference,
Roma, Italy, 2001

2. MPP Platform for Teradata

Teradata is a shared-nothing parallel database which runs primarily on massively parallel processor (MPP) shared nothing hardware. On the MPP platforms sold by NCR this MPP hardware consists of four-way standard Intel server nodes with typically two gigabytes of memory each connected by the Bynet, NCR's high speed interconnect. The collection of nodes is divided into small groups called cliques that enjoy shared access to disk arrays. Figure 1 shows a typical small system with two cliques of four nodes each. This size system would usually support about 1.5 – 6.0 terabytes of storage. The disk sharing capability is only used in the case of a node failure. In normal operation, unlike some clustered database systems, each node is solely responsible for a subset of the disk farm. The disk storage is divided up into many "vdisks" or virtual disks which contain the data for a single partition of the database. This vdisk is managed by a collection of processes known (for historical reasons) as an Access Module Processor Virtual Processor (AMP vproc). The tasks that comprise this AMP vproc are the only ones which ever access this partition of the database. If a node were to fail, the AMP vproc would be moved to another

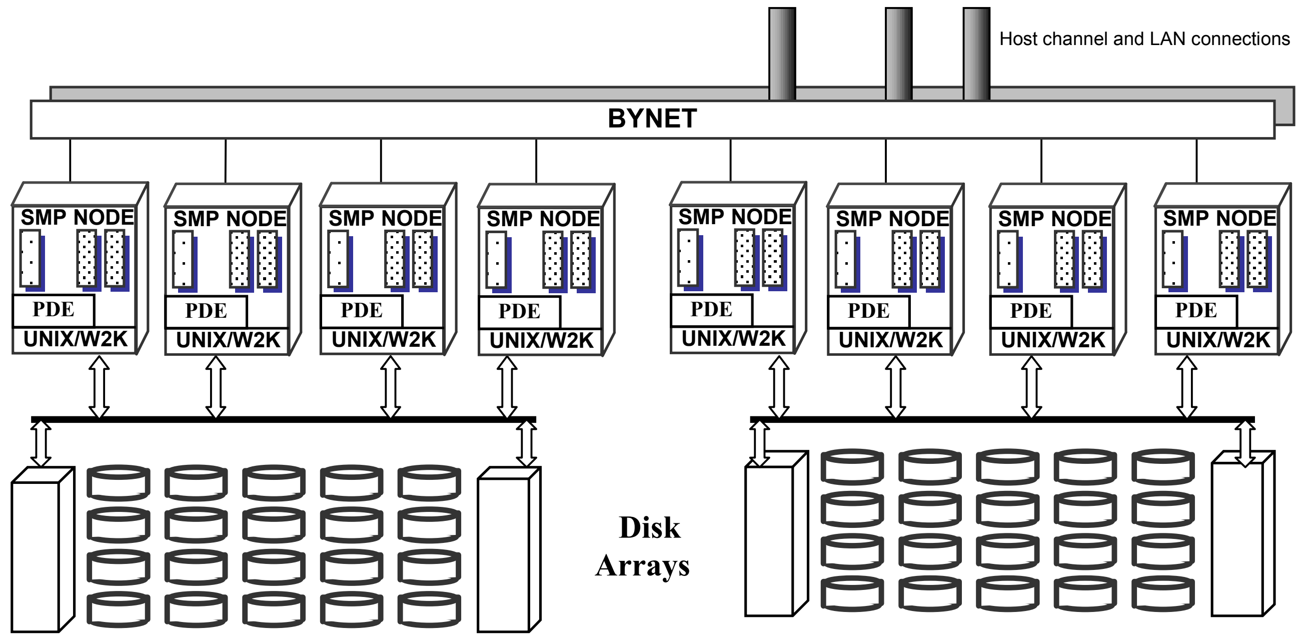

Figure 1. NCR MPP Platform for Teradata

node of the clique where it can still access the same storage.

3. Architecture of Teradata Database

The Teradata Database software is really two distinct collections of code. The first, comprising the database engine, consists of three million lines of "C" code which are the same for all hardware and operating system environments. A second set of software called the Parallel Database Extensions, or PDE, provides the actual interface logic between the database and the platform software and hardware. This relationship is shown in Figure 2.

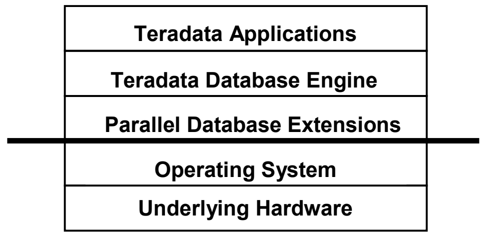

Figure 2. Teradata Software Layers

4. Features of Parallel Database Extensions

Teradata achieves its unmatched performance and much of its parallel nature through a set of operating system extensions called Parallel Database Extensions. These enhancements to the base operating system extend and parallelize the standard OS environment, providing the over 400 services used by the Teradata Database. These services consist of functions such as memory management, database data block access and cache management, messaging system, process management (start up, shut down, abort processing), priority scheduling, and the whole parallel infrastructure that allows the Teradata Database to run on a collection of one, four, or hundreds of independent SMP nodes as if it was all one system.

5. MP-RAS Implementation of PDE

In order to achieve the highest performance level possible, the original MP-RAS version of PDE was coded completely as kernel mode extensions to the NCR MP-RAS SVR4 Unix operating system. These extensions were built upon the SVR4 extensible objects of scheduling class and segment driver. This architecture is described more fully in "OS Support for VLDBs: Unix Enhancements for the Teradata Data Base", VLDB'95, Proceedings of 21st International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland. This implementation was very specific to SVR4 Unix and NCR's MP-RAS operating system in particular, and is not readily portable to other environments.

6. Open PDE

The desire to run Teradata on other operating environments, and specifically, Microsoft Windows NT, led in 1998 to the re-architecting of PDE in a more open, portable manner. In this new architecture for PDE, some of the code runs in kernel mode, installed as a "driver", while much of the code operates in user mode. The new PDE is also implemented with a layered approach to enhance portability.

The outer layer provides the actual interface to the database software and implements the services required. It is mostly unchanged from one operating system environment to another. The lower layer comprises the primitive functions, or "bricks", out of which the upper layer is built. These functions are implemented in the most performant fashion for each operating system. They include routines for implementing the low-level user-mode and kernel-mode multi-processor locks, memory "zone" allocation schemes, offset queues, wait objects, names, and daemons. In the following sections, several of these areas are described along with the problems they presented and the solutions used.

6.1 User-mode Vs. Kernel Mode

In the original MP-RAS implementation, the entire PDE was coded as kernel mode extensions to the underlying Unix operating system. This was easy and natural to do, since the original proprietary Teradata Operating System which was being replaced by PDE was all kernel mode and the target Unix operating system was owned by NCR and readily available to the PDE developers. In re-architecting PDE for portability with several target platforms this decision was resurfaced. The implementation of at least some of the PDE code in user mode was a fairly easy decision. User-mode code is generally easier to debug and maintain, and if underlying OS services are not required, can be made to perform as well as or better than a kernel-mode equivalent. Whether or not **any** of the Open PDE code would be placed inside the kernel was a much larger debate. In the end, the desire for maximum performance won out over the ease of implementation with several of the PDE sub-systems being placed in a kernel mode "driver". There are two major performance advantages allowed a kernel mode implementation that dictated its use for a few of the major sub-systems. One of these is the ability inside the kernel of providing an efficient low-level spin lock for access to shared context that must be protected from simultaneous access by the multiple CPUs of an SMP complex. There is simply no efficient means of providing this protection in user-mode. The other clear advantage to coding a service in kernel mode is when the routine must make two or more calls to the underlying operating system kernel-mode code to accomplish the desired action. If the service is already operating in kernel-mode, the repeated transitions from user-mode to kernel-mode and back can be avoided.

6.2 Processes Vs. Threads Vs. Fibers

In the original MP-RAS implementation of PDE the individual tasks of the database software were implemented as complete heavyweight processes, each with its own memory map and kernel context including registers values. The reason for this was simple: NCR's MP-RAS Unix does not have any other construct such as lightweight processes (threads) or no-weight processes (fibres). While re-architecting PDE, we looked at how to best instantiate the Teradata Database tasks and after some considerable investigation decided to use a combination of multiple heavyweight processes and kernel threads.

The Teradata database tasks can be readily placed into groups that have a large amount of shared context. Each of these groups is instantiated as a process, with the individual members of the group being the separate threads of the process. As true kernel threads, each has its own kernel context, and thread local storage. However, the threads all share the process' memory map. This makes sharing of context automatic while still allowing for private data, albeit at a reduced level of protection from wild stores by other threads. Sharing of data between different processes is accomplished through some form of shared memory mechanism as available on each operating environment.

A typical 4-way, 2 GB node running Teradata will have about 80 processes with around 2000 total threads. Only 10 of the processes with about 120 threads each handle the vast bulk of the database work. These 10 processes are each responsible for one partition of the database, with the 120 threads within each process handling the simultaneous operations for one or more queries involving that partition of data. All of the other processes are for handling administration, start-up, shut-down, aborting, debugging, tracing, error logging, and other seldom-occurring events where the performance requirements are not so stringent.

An argument was put forward that if the over-head of context switching of processes was an undo burden than we might be better off with instantiating the database tasks as fibres rather than threads. Fibres are completely instantiated in user mode. Fibres are lightweight entities that all share not only the same memory map but utilise the same kernel context. Because of this, only one fibre at a time can be executing. The slightly reduced context switch time from not leaving user mode does not overcome the obvious drawback of not executing in parallel, thus precluding their use for implementing the Teradata Database.

6.3 Implementation of PDE Primitives

To give an idea of how PDE primitives are architected in as portable a manner as possible, we describe here one of those functions. PDE needs the ability to perform thread synchronisation actions. In order to do this PDE defines an entity called a synchronisation event. A synchronisation event can be any arbitrary happening that a thread desires to wait for or be notified of. After establishing the event object we define the operations upon the object such as create, init, wait, wakeup, reset, and destroy. A structure is created to hold the components of the synchronisation event's context, most of which are completely system independent. We then look at the facilities available in each of the operating systems of interest with which we can cause the necessary behaviour. In this case we used Solaris conditional variables, Linux wait queues, and Windows events. The portion of the function that is unique to each operating system is then coded to interact with the underlying OS and act upon the commonly defined object.

The rest of PDE then uses the synchronisation event functions for all of its thread synchronisation needs. The rest of the code need not be aware of the different way the function is implemented for each operating system.

In a similar manner PDE has implemented primitive functions for a vast array of activities including: raw I/O capability, memory locking, shared memory support, inter-processor locking, and process and thread management.

7.0 Performance of Open PDE

After creating all the primitive functions, the rest of PDE is coded using these operations in order to supply the hundreds of services required to support the Teradata Database software. After getting it all to work, the next question is: How well does it perform compared to the specific, dedicated version on MP-RAS? The first implementation of Open PDE was for Microsoft Windows NT 4.0. Surprisingly, Teradata running on this version of Open PDE performed within 10% of the then current MP-RAS Unix version. The Windows version of Open PDE has now been in production since 1999 and has since been converted to Windows 2000. Several potential problem areas were identified using the many debugging and performance-monitoring features built into PDE. We present here two of these areas along with the solutions implemented.

7.1 SMP Locking Protocol for Global Resourses

One very important aspect of performance for programs running on an SMP computer is the locking protocol that is used to protect global resources from improper updates by threads running simultaneously on the various CPUs. It is very easy for locks on resources used by many threads to be a major bottleneck. Upon investigation, it was found that several of our global resources such as the free disk space structures were being accessed enough that their global locks were bottlenecks. To solve this problem we created a general method for reducing the collisions on a single lock by partitioning the resource into *n* separate pools, each with its own lock. A central index cell is incremented to decide which pool to acquire a resource from, and the resource control element has an index value pointing to the appropriate pool for later return. In this way the contention for a single lock is eliminated, resulting in some cases, more than an order of magnitude less spin lock contention.

7.2 Eliminating Data Copying

Another common source of performance degradation in any system handling a large amount of data is unnecessary copying of data from one buffer to another. In Teradata a buffer is often broadcast to all vprocs over the Bynet as part of table duplication. Upon receipt by the node, the buffer must be made available to one process in each of the vprocs on the node. In order to accomplish this without copying the data several times we utilize a daemon that supplies the Bynet with an available queue of offsets into a special buffer area. This buffer area is contained in a shared file. As each message arrives, the offset is mapped into the shared spaces of the destination processes and the offset is placed on each receiver's mailbox. Upon awakening, each process "sees' the buffer in his own space at the specified offset.

7.3 Performance of MP-RAS Vs. Windows 2000

Figure 3 is a table showing the query times for each of the queries in published one terabyte TPC-R benchmarks. Most of the queries run in about the same or better time in the new Open PDE implementation on Microsoft Windows 2000 as they did on the dedicated MP-RAS Unix version. (Query 21 and query 22 show vastly improved times due to a new feature in the database that was added between the two benchmark runs.)

TPC-R Query #	MP/RAS Time	Windows Time
Q1	0.5	0.5
Q2	11.7	11.0
Q3	156.7	164.0
Q4	27.2	23.4
Q5	0.9	0.9
Q6	0.5	0.6
Q7	1.1	1.1
Q8	1.3	1.5
Q9	491.2	483.8
Q10	585.8	641.4
Q11	185.2	193.4
Q12	0.5	0.5
Q13	1460.2	1571.8
Q14	0.5	0.5
Q15	178.2	187.1
Q16	180.1	184.8
Q17	26.0	29.2
Q18	48.8	43.2
Q19	815.6	720.7
Q20	384.0	419.4
Q21	2490.8	1084.1
Q22	214.4	67.3

Figure 3. TPC-R Comparison Data

8.0 Summary

We have demonstrated that with care it is possible to architect a portable set of OS extensions for vastly different operating systems such as Solaris, Linux and Windows 2000 that will perform as well as the original dedicated extensions on NCR's MP-RAS Unix.