Much Ado About Shared-Nothing

Michael G. Norman
Bloor Research, Bletchley, England*

Thomas Zurek, Peter Thanisch
Dept. of Computer Science, Edinburgh University, Scotland†

Abstract

In a 'shared-nothing' parallel computer, each processor has its own memory and disks and processors communicate by passing messages through an interconnect. Many academic researchers, and some vendors, assert that shared-nothingness is the 'consensus' architecture for parallel DBMSs. This alleged consensus is used as a justification for simulation models, algorithms, research prototypes and even marketing campaigns.

We argue that shared-nothingness is no longer the consensus hardware architecture and that hardware resource sharing is a poor basis for categorising parallel DBMS software architectures if one wishes to compare the performance characteristics of parallel DBMS products.

1 Introduction

A parallel computer has a 'shared-nothing' architecture if each processor has its own memory and its own disk sub-system; processors communicate by passing messages through an interconnect. In the 1980s, several 'brand leader' parallel database machines had shared-nothing architectures and, in 1992, DeWitt and Gray were able to assert that

> *a consensus on parallel and distributed system architecture has emerged. This architecture is based on a shared-nothing hardware design.* [DeWitt and Gray, 1992]

DeWitt and Gray's assertion has been quoted or paraphrased many times and the alleged existence of this 'consensus' has been used to justify several research projects (simulations, analytical modelling, prototype building) in the academic world and as a basis for at least three marketing campaigns for commercial DBMS products (namely DB2/6000 Parallel Edition ('PE'), Sybase Navigation Server and Informix Online XPS). You will come across the phrase in the white papaers and even the sales literature of several parallel database vendors.

In our view, shared-nothingness is no longer the consensus hardware architecture. Although it has become more common to think of shared-nothingness as a software architecture for DBMS products, we also argue that distinctions based on resource sharing are not

*Tel. + 44 131 553 3313, Fax + 44 131 555 1178, Email: mgn@makespan.demon.co.uk

†King's Buildings, JCMB, Edinburgh EH9 3JZ, Scotland, Tel. + 44 131 6505133, Fax + 44 131 6677209, Email: {tz,pt}@dcs.ed.ac.uk

a useful way to categorise DBMS architectures if one wishes to understand the DBMS's performance characteristics, particularly if one is looking at differentiating between the general-purpose parallel DBMS products now on the market.

In section 2, we give the traditional categorisation of parallel architectures - hard and soft - in terms of hardware resource sharing. In section 3, we examine the extent to which shared-nothingness can be said to be the consensus - hard or soft. Concluding remarks are in section 4.

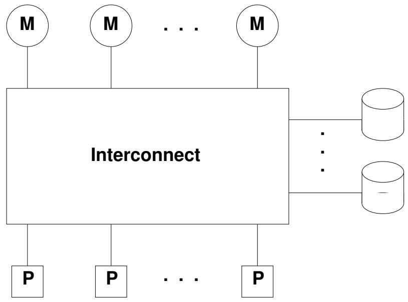

Figure 1: Shared-Memory Architecture

2 The Traditional Categorisation for Parallel Databases

Traditionally, architectures for parallel DBMS were categorised by the way in which processors share hardware resources like disk devices and memory [Stonebraker, 1986]. This categorisation was used to facilitate discussion about appropriate parallel *hardware* architectures for DBMS. Many researchers participated in the ensuing discussion; see e.g. [Hua et al., 1991], [DeWitt and Gray, 1992], [Bergsten et al., 1993], [Valduriez, 1993b], [Baru et al., 1995]; many others base their arguments on it. We summarise their conclusions and briefly describe the architectural categories, namely *shared-memory* (SM), *shared-disk* (SD) and *shared-nothing* (SN).

The following basic arguments and characteristics are often given for these architectures. These are not necessarily arguments in which we place much credence.

Shared-Memory In a *shared-memory* system, all disks and all memory modules are shared by the processors; see Figure 1. There is a global address space for main memory and all disks are equally accessible by all processors. It is said that SM is simple for the DBMS vendors to program because of the global address space in main memory. Load balancing can be arranged relatively easily because each processor has equal access to all disks. Interprocessor communication is fast (and incurs low overhead) as they can cooperate via main memory. However, system costs are high, in particular for the interconnect (large number of components connected). Conflicting accesses to main memory can decrease the performance. It is also argued that access to main memory is the reason why SM-architectures do not scale up well: [Bhide and Stonebraker, 1988] showed that beyond a certain number of processors, access to main memory can become a bottleneck that limits the system's processing speed. Thus SM systems are limited to a small number (2̃0-30) of processors [Valduriez, 1993a], *cf.* [Baru et al., 1995].

Shared-Disk In a *Shared-disk* system, each processor has its private memory, but access to disks is shared by all processors; see Figure 2. It is argued that the costs for SD system are relatively low as the interconnect could be a bus system based on standard technology. It is also argued that load balancing is relatively easy for the same reason as in the

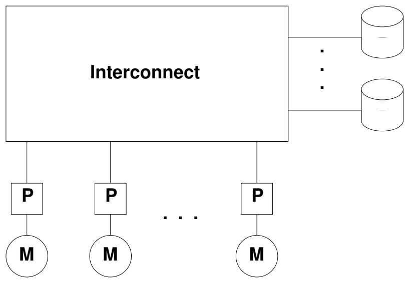

Figure 2: Shared-Disk Architecture

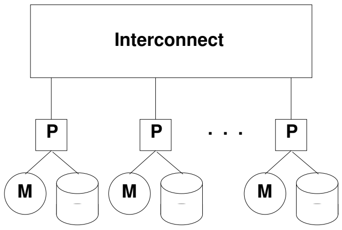

Figure 3: Shared-Nothing Architecture

SM case, and that the availability of data is higher than in an SN system (see below) as a crash on one processor does not result in the data of a particular disk being unavailable. Much of the down-side of SD systems is said to relate to an increase in complexity, caused, e.g., by cache coherency control mechanisms needed to maintain consistent disk pages in the processors' individual caches. This limits the scalability of a SD system. Finally, access to shared disks might result in a bottleneck due to limited bus capacity.

Shared-Nothing In a *shared-nothing* system, each processor has its own private memory and disks and it acts as a server for the data on its disks; see Figure 3. It is said that the costs of an SN system are low because it can be constructed from commodity components. Availability is also often considered to be a serious problem (see following section). Data skew can cause serious load imbalance. Load imbalance can also be caused by the the execution of operations being predetermined by the data placement on the disks and the necessity to avoid huge data shipping through the network to other processors.

3 The Arguments for Shared-Nothing

Initially the architectural discussion described in the previous section was only related to the parallel hardware. In recent years it has become more a discussion around the optimal software architecture for parallel DBMS. Nevertheless, in this section, we question the suitability of shared-nothing as a hardware architecture. We also discuss, where appropriate, the value of shared-nothingnes as a software architecture.

The advantages that DeWitt and Gray [DeWitt and Gray, 1992] claimed for shared-nothingness are as follows.

1 "*Partitioning allows multiple processors to scan large relations in parallel without any exotic I/O devices. [...] Each memory and disk is owned by some processor that acts as a server for that data.*"

Most commercially-available parallel hardware platforms currently use standard commodity disk components, typically single SCSI disks or SCSI RAID devices. The problem with the basic shared-nothing architecture is the availability and fault tolerance in the case that a processor fails. The data on its corresponding disk is then unavailable. In practice, SN systems use multiply-attached disks and redundancy mechanisms like data replication to guarantee availability. This results in the problem of keeping multiple copies of a data item consistent. In practice, however, both SN and SD systems use the same algorithmic mechanisms and the same hardware features to achieve availability. The distinction between the approaches is often simply whether the software systems involved in guaranteeing redundancy are provided within the operating system (shared-disk) or DBMS (shared-nothing).

2 "*The other architectures move large quantities of data through the network. The shared-nothing design moves only questions and answers through the network.*"

This considers only the ideal case of tables being *collocated*, i.e. ideally placed for processing. When describing IBM DB2/6000 PE, [Baru et al., 1995] describe techniques for join (the most expensive operator), distinguishing between collocated, direct, broadcast and repartitioned joins as the necessary strategies. The latter three cause at least one table (or intermediate result) to be shipped through the network. Typically it will be the intermediate result that gets shipped through the network, which will tend to be the larger of the two inputs to the join whenever one large table is joined with several small tables.

It is interesting to note that some parallel DBMS products are advertised as being shared-nothing, but in reality, each server can and does access each disk controller. For example, Informix DSA's servers retrieve metadata from any part of the system in order to discover how a table has been partitioned.

3 "*Shared-nothing architectures minimize interference by minimizing resource sharing.*"

This is a non-sequitur. Shared-nothing architectures simply *determine* interference by specifying that certain operations will occur at certain points in the parallel architecture. In practice this will tend to *increase* interference since the DBMS is unable dynamically to change the location of these operations.

The interference that is minimised by a shared-nothing system is the usage of the interconnect. In today's commercial hardware architectures (unlike earlier ones) aggregate interconnect bandwidths are extremely high, and scale up at least linearly as processors are added to the system. Aggregate interconnect bandwidths are typically ten times higher than aggregate disk bandwidths, so it is actually difficult to saturate the bandwidth of the interconnect; there is simply nowhere that the data could have come from.

4 "*The systems demonstrate near-linear speedup and scale-up on relational queries.*"

In practice, the ideal, i.e. linear, scale-up is not achieved. Notions such as 'speedup' and 'scale-up' come from scientific parallel computation, but have no simple applicability to parallel database processing. Speeding up by plugging in more nodes is rarely a simple matter when you are running real workloads. In general, it will require re-partitioning the data, and re-configuring the machine. According to Teradata's staff "Determining the optimal clique configuration mix [...] at this time, is a black art that involves an application specific complex trade-off between fault-tolerance, price and performance" [Witkowski, 1993].

In practice, shared-nothing rarely scales up or speeds up linearly because the interconnect becomes saturated beyond a certain volume of communication.

During the life-cycle of a parallel computer, there are other, practical considerations: processor clock rates double every two years. Therefore one can replace the old processors by new, faster ones and improve the quality of the components in this way. Various studies have demonstrated that this strategy leads to superior performance wherever it is possible; see, e.g.,[Hua et al., 1991]. Essentially, the trick is to be as un-parallel as possible in order to minimise interprocessor communication overhead.

5 "*Teradata systems may have over 1,000 processors.*"

This may be physically feasible, but in practice, Teradata's own staff state that "The YNET bandwidth is not scalable to the maximum 1000 processor board configuration" [Witkowski, 1993]. YNET is a saturable network, and this type of limitation does not apply in the case of newer networks such as BY-Net. However, it is to be expected that other issues will similarly limit the scalabilty of the new AT&T 5100 machine.

4 Conclusions

If there is a convergence of parallel hardware, it is towards a hybrid architecture, comprising two levels. Such systems are either SM (SMP) nodes (inner level) combined in an SN manner [Hua et al., 1991] (see figure 4) or SM (SMP) nodes (inner level) combined in an SD manner (outer level) [Valduriez, 1993b]. These hybrid architectures combine the advantages of the previously discussed types and compensate for their respective disadvantages [Hua et al., 1991]. New versions of traditionally SN-machines are converging towards the hybrid type, for example Tandem's ServerNet-based machines, the Teradata/AT&T/GIS 5100 or future developments in IBM's RS/6000 SP.

Developments in technology have meant that distinctions based on hardware resource sharing are no longer particularly useful if you

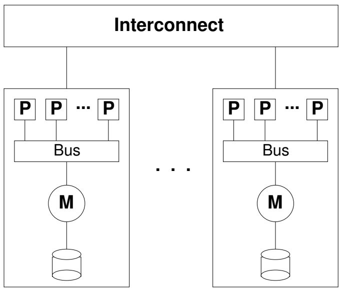

Figure 4: Hybrid, two-level architecture

want to gain some insight into the relative performance of parallel DBMSs.

When preparing our recent analysis of commercially-available parallel DBMS implementations [Norman and Thanisch, 1995], we found that the simple shared-disk vs shared-nothing distinction was not helpful in differentiating between products. We found it necessary to develop a new generic model of parallel DBMS architectures in order to categorise parallel DBMS products. This model is based on the way that processes and threads are organised to cooperate in transaction and complex query processing.

References

Baru, C., Fecteau, G., Goyal, A., Hsiao, H., Jhingran, A., Padmanabhan, S., Copeland, G., and Wilson, W. (1995). DB2 Parallel Edition. *IBM Systems Journal*, 34(2):292–322.

Bergsten, B., Couprie, M., and Valduriez, P. (1993). Overview of Parallel Architectures for Databases. *The Computer Journal*, 36(8).

Bhide, A. and Stonebraker, M. (1988). A Performance Comparison of Two Architectures for Fast Transaction Processing. In *Proc. of the 4th International Conference on Data Engineering, Los Angeles, CA, USA*, pages 536–545.

DeWitt, D. and Gray, J. (1992). Parallel Database Systems: The Future of High Performance Database Systems. *Communications of the ACM*, 35(6):85–98.

Hua, K., Lee, C., and Peir, J.-K. (1991). Interconnecting Shared-Everything Systems for Efficient Parallel Query Processing. In *Proceedings of the 1st International Conference on Parallel Distributed Information Systems, Miami Beach, FL, USA*, pages 262–270.

Norman, M. and Thanisch, P. (1995). *Parallel Database Technology: An Evaluation and Comparison of Scalable Systems*. The Bloor Research Group, UK. ISBN 1-874160-17-1.

Stonebraker, M. (1986). The Case for Shared Nothing. *IEEE Data Engineering*, 9(1).

Valduriez, P. (1993a). Parallel Database Systems: open problems and new issues. *Distributed and Parallel Databases*, 1(2):137–165.

Valduriez, P. (1993b). Parallel Database Systems: the case for shared-something. In *Proc. of the 9th International Conference on Data Engineering, Vienna, Austria*, pages 460–465.

Witkowski, A. e. a. (1993). NCR 3700 – the next-generation industrial database computer. In *Proceedings of the 19th International Conference on Very Large Data Bases, Dublin, Ireland*, pages 230–243.