On Benchmarking the ORACLE Parallel Server on nCUBE2

S. Coticoglou, M. Podgorny and A. Choudhary

Northeast Parallel Architectures Center, Syracuse University
Syracuse, NY, 13244-4100

Abstract

Massively Parallel Processor architectures (MPP) have been proposed as a promising alternative to mainframe computers for the largest database and transaction processing tasks. An example of this trend is the implementation of ORACLE parallel database server on several massively parallel machines like Meiko, nCUBE2, and KSR. While many claims have been made for the performance of ORACLE on MPP platforms, few studies have been done to test the actual performance advantages and disadvantages. A benchmark study of the performance and scalability of the ORACLE Parallel Server v6.2 on nCUBE2 is reported in this paper. The advantages of the stripe file system and the gigacache are also studied and displayed in the benchmark results.

1 Introduction

Massively Parallel Processor systems like nCUBE2, CM-5, IPSC/860 while successful in the computation intensive areas of business and science, have not managed till now to penetrate the data processing market. This due to the lack of effective commercial available software to harness MPP’s scalability for Database Management Systems (DBMS). Recently, this situation changed with the availability of ORACLE Parallel Database Server on several massively parallel machines like Meiko [7], nCUBE2 [8], and KSR [12]. The potential market for these systems is very large with applications in industries such as banking, financial services, insurance, information services, telecommunications, direct marketing and transportation.

However, few studies have been performed to test the performance advantages and disadvantages. The only published numbers are those for nCUBE2 running an industry standard benchmark, the TPC-B [10]. An nCUBE2 with 64 processors and 48 I/O nodes managed to achieve a rate of over 1000 transactions (tps) at a capital cost of $2,500 per tps.

In this paper we present an experimental evaluation of the performance and scalability of the ORACLE Parallel Server v6.2 on nCUBE2. The current implementation of the Parallel Server on nCUBE2 allows multiple independent processors access to a single shared database, and guarantees that any necessary coordination will occur when two processors attempt to simultaneously access a single item of data. The database files can be spread over several disks attached to different I/O nodes so that file operations can occur in parallel. An additional cache, called gigacache can be activated to exploit the enormous cache capabilities of the processing nodes. The main purpose of our experiments was to measure the impact of the following three factors in the performance and scalability of the ORACLE parallel server: 1) single node performance, 2) ORACLE stripe file system performance, and 3) gigacache performance.

The rest of the paper is organized as follows. In Sections 2 and 3 we describe the architecture of nCUBE2 and the ORACLE Parallel Server, respectively. The performance study and the results are contained in Section 4. Our conclusions are described in Section 5.

2 An overview of nCUBE2

The nCUBE2 is a massively parallel system that can have as many as $2^{13}$ (8192) processing and $2^{10}$ (1024) I/O nodes. Each processing node is an independent fully functioning computer composed primarily of a proprietary 64-bit CPU operating at 7.5 MIPS, 64-bit floating point unit operating at 2.5 MFLOPS, a network communication unit, a memory control unit, and 1 to 64 MB of main memory. The network communication unit provides 14 bidirectional, cut-
through routed DMA channels which support the hypercube interconnection scheme. These channels support transfers of data and control messages between any two nodes at 2.2 MBytes/sec. Cooperation between processing nodes takes place by means of message passing. The processing nodes are allocated in subcubes which operate independently of each other.

The I/O nodes have the same CPU and memory configuration as the processing nodes, and are primarily used to control disks, tape drives, network communications and other peripheral resources. Each processing node can be directly connected to up to one I/O node through a dedicated DMA channel.

The nCUBE2 uses a host computer as its primary user interface for system administration and starting programs on nCUBE2 nodes.

3 The Architecture of ORACLE Parallel Server on nCUBE2

The ORACLE Parallel Server on nCUBE2 allows multiple ORACLE instances to execute concurrently against a single shared database. An ORACLE instance is a complete copy of the ORACLE Relational Database Management System (RDBMS) with its own buffer space and background processes. The ORACLE Parallel Server uses an external lock manager service, called distributed lock manager (DLM), to coordinate the access to shared data, and to ensure data consistency across instances. The DLM may consist of several processes running on different processing nodes, scaling with the number of instances. The function of the DLM is quite orthogonal to ORACLES's data locking mechanism. ORACLE supports a row level locking mechanism to prevent destructive interaction between users connected to the same instance.

ORACLE on nCUBE2 provides a stripe file system to exploit the parallel I/O capabilities of nCUBE2. The ORACLE Stripe Files system (OSF) allows to create striped files across several disks attached to different I/O nodes, so that file operations can occur in parallel. Striped files are defined in volumes. A volume is a logical entity that consists of multiple physical partitions on different disks. Each volume has its own file system. The most important characteristic of a volume is the stripe width. The stripe width is defined as the number of physically contiguous blocks in a single partition of a volume. The stripe width specifies how the logical address space maps to the physical address space.

The ORACLE Parallel Server also provides an optional cache, called gigacache, which is accessible by all instances. The gigacache isolates the instances from the I/O subsystem by caching large portions or even the whole database in the core memory of one or more processing nodes. These nodes operate as gigacache nodes. The data in gigacache are organized using a data partitioning scheme. In particular, a hash function is used to map units of striped data to gigacache nodes.
Applications access an ORACLE instance through an interprocessor communication mechanism known as two-task architecture. In the two-task architecture, each application process starts a corresponding shadow process at the same node with the instance. The shadow process performs various database functions on behalf of the invoking process. The two-task architecture allows the applications to run anywhere in the system, even on workstations connected to nCUBE2 via ethernet.

4 Performance Study

The system under test was the ORACLE Parallel Server v6.2 running on an nCUBE2 with 32 processing nodes, and 32 MBytes of main memory per node. The I/O subsystem consisted of 16 I/O nodes with 4 MBytes of main memory each. Each I/O node had two disks attached to it through a SCSI-2 controller. All nodes ran the Vertex operating system v3.1. The host to the nCUBE2 was a Silicon Graphics Workstation 4D35.

The Benchmark relations were based on the Wisconsin benchmark [1]. The Wisconsin benchmark has been fairly extensively used to measure speedup and scalability of ad-hoc queries in parallel database systems, including Gamma, Teradata, and Tandem. The Wisconsin relations provide many advantages for this purpose, since they are easy to scale with precise control of data size and value distributions. Each relation consists of tuples that are 208 bytes wide.

We used two different metrics to measure the performance of the ORACLE Parallel Server: response time, and throughput. While response time is an important measure of database system performance, throughput metrics are more suitable for measuring speedup and scalability in a parallel database system. The benchmark tests were divided into three categories: single-instance, multi-instance, and gigacache tests.

4.1 Single-Instance Tests

In single-instance tests, we ran a number of synthetically generated queries on a single processing node. Our goal was to measure the response time of the system as we vary the number of disks in the volumes used to distribute the benchmark relations. We ran the queries on two sets of relations containing 10,000 and 100,000 tuples, respectively. The queries did not employ any indices since our aim was to stress the I/O subsystem as much as possible. In our tests, we used five different volume configurations to distribute the benchmark relations. Figure 2 shows the disk layout for all volumes. The columns represent I/O nodes, and the rows disk partitions. The stripe width for each volume was $I_{max}/N$, where $I_{max}$ is the maximum size of an I/O request generated by a single ORACLE instance, and $N$ is the number of disks in the volume. As a result, multiple disks and I/O nodes were involved per request.

Figure 3 shows the response time of four representative queries as a function of the number of disks in a volume. The response time improves for relations striped across as many as four disks. For volumes with more than four disks the improvement is either very small or non existent. To illustrate this behavior better, Figure 4 shows the relative speedup that was obtained relative to the case where the data are not striped, but reside on a single disk. The speedup was very small even for the best case. This behavior can be explained as combination of two factors. First,
the node processors are not fast enough to generate sufficient requests to the I/O subsystem, making the performance of the system CPU bound. Second the I/O requests generated by the ORACLE instances are very small, 32 KBytes or less. A stripe filesystem can hardly help to reduce the I/O latencies for such small requests.

The relative speedup for different queries is inversely proportional to their CPU intensity: the largest speedup is observed for the I/O intensive select query and the smallest one for the CPU intensive projection query which requires to sort the entire table. The reason for this behavior, is that only the I/O related execution time is subject to potential speedup. The more CPU intensive the query is, the less speedup is expected.

4.2 Multi-Instance Tests

The next set of tests were designed to measure the throughput of the system as the number of active instances and the number of disks in the volumes were varied. The workload consisted of a single select query that each instance ran continuously for twenty minutes. As in the previous tests, there were no indices so each query had to read the whole relation.

We performed two different sets of experiments. In the first set, each instance read from a different table of exactly the same size, while in the second set all instances read from the same table. In both experiments we varied the number of instances from 1 to 24. Again, as in the single instance tests, we experimented with different volume configurations up to 12 disks. The stripe width was fixed 32KBytes. As a result, almost all requests fell within a single stripe unit, and each disk was able to service a different request. This scheme provides good performance for a user workload characterized by a high request rate. The number of blocks read by a single query was larger than the number of available blocks in the buffer pool of a single instance, so there was no impact from buffer pool hits on the actual throughput rates obtained.

Figure 6 shows the results we obtained from the
The size of the database was much smaller than before. The disk latencies were also smaller because of the smaller seek times. As expected, the capacity of I/O subsystem increased and was able to accommodate a larger number of I/O requests. The system becomes I/O bound only when more than 16 instances are active at the same time, for all volume configurations. The most interesting observation from this set of experiments is the exactly same system throughput behavior for volume configurations with more than one disk.

4.3 Gigacache Tests

The final set of tests had the purpose to investigate the influence of gigacache on the performance and scalability of the ORACLE Parallel Server. We performed two sets of experiments. In both cases the size of the database was very small, 2MBytes, and could easily fit in the gigacache. In the first set we repeated the multi-instance tests with the gigacache enabled. Ideally, one would like to see significant improvement in the performance, because the gigacache can isolate the instances from the I/O subsystem. Figure 7 presents the throughput of the system with and without the gigacache enabled. The benefits from the use of gigacache are very small and are limited to the case where there are less than 8 instances active at the same time. For larger number of instances the configuration that distributes the tables over four disks without using gigacache, gives better throughput.

In the second set of experiments, we measured the scalability of the gigacache. We experimented with different gigacache configurations, varying the number of gigacache nodes from 1 to 24. Figure 8 shows that the throughput behavior of the system is almost the same for all configurations, independently how many nodes are used to spread over the cache entries. This behavior is very similar with what we have already observed in previous experiments with regard the scalability of the ORACLE stripe file system.

5 Conclusions

We expect that the presented results will contribute to a better understanding of the performance issues of the Parallel Server v6.2. Although we obtained a significant performance improvement by using volumes of four disks instead of one, the benefits of a larger number of striped disks were minuscule. In the multi-instance tests the maximum capacity of the system was reached at 16 instances. The performance number for the gigacache showed that there is little advantage
of using it, if the number of disks used to spread across
the benchmark tables is sufficient. We anticipate that
some of the bottlenecks uncovered in this study will
have a significant impact on the parallel execution of
individual queries in the next version of the Parallel
Server.

References

[1] Bitton D., DeWitt D. J., and Turbyfill C.,
"Benchmarking Database Systems: A Systematic
Approach", in Proc. of the 1983 Very Large
Database Conference, 1983.

[2] Cesarini F. and Salza S., editors, Database Ma-
chine Performance: Modeling Methologies and

[3] Costicoglou S., Podgorny M., and Choudhary A.,
"Performance Evaluation of the ORACLE Par-
allel Server v6.2 on nCUBE2", Technical report,
Northeast Parallel Architectures Center, SYracuse

[4] DeWitt D. and Gray J., "Parallel Database Sys-
tems: The Future of High Performance Database

[5] Dietrich S., Brown M., Cories-Rello E., and
Wunderlin S., "A Practitioner's Introduction to
Database Performance Benchmarks and Measure-

Database and Transaction Processing Systems,

A Parallel and Scalable Open Systems Platform
for ORACLE", in Gray P., editor, Advanced
Database Systems, 10th British National Confer-

[8] Linder B., "ORACLE Parallel RDBMS on Mas-
sively Parallel Systems", in Sec. Int. Conf. on
Parallel and Distributed Information Systems, San

of Parallel Transaction Processing in Shared Noth-
ing Databases", in Proc. 4th Int. PARLE Conf.,

[10] "TPC-B, Full Disclosure Report for the nCUBE2
using ORACLE v6.2", Oracle Corporation, July

[11] Podgorny M., Costicoglou S., and Mills K., "Par-
allel Database Technology and the Status of OR-
ACLE 7", Technical Report SCCS-373, Northeast

[12] Reiner D. S., "The Kendall Square Query De-
composer", in Sec. Int. Conf. on Parallel and
Distributed Information Systems, San Diego, Jan.
1993.

[13] Turbyfill C., Comparative Benchmarking of Re-
lationl Database Systems, PhD thesis, Cornell
University, 1987.

"Analysis of Database System Architectures Us-
ing Benchmarks", IEEE Transactions on Software