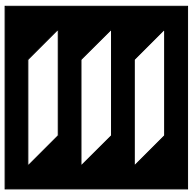

Parallel
Applications
Centre

ESPRIT DBInspector

Deliverable D4.3.0
Report of Development of a Parallel Database: Codes of Best Practice

Report PAC/DBI/R03 Issue 1

Paul Allen

Parallel Applications Centre

18 September 1996

Contents

1 Introduction

This document is Deliverable D4.3.0 of the DBInspector project. It follows D4.2.0 which describes the performance evaluation tests performed at PAC. This document takes a step back and proposes a code of best practice for parallelisation of databases which are similar to the DBInspector Financial Flows Archive. Section 4 describes generic database tuning issues which are of relevance across all relational database management systems (RDBMSs). Section 5 relates to Oracle specific issues, across all platforms.

2 Parallel Database Systems Overview

2.1 Hardware

2.1.1 Components

2.1.1.1 CPU

The CPU is the basic engine of the parallel machine, and machines are usually described by the number of CPUs they contain and the specifications of those CPUs. While this can give an approximate indication of the power of a machine, there are a number of other important factors; memory, disk and interconnect must also be considered.

There are a number of leading CPU families in the Parallel Unix marketplace (e.g. SPARC, MIPS, Intel, IBM RISC, Power PC, DEC Alpha, HP). Performance of the CPU is broadly characterised by the clock speed (in MHz); other factors such as the amount of on-board cache and word lengths can also affect performance.

2.1.1.2 Memory

The amount of memory is important to the performance of the system. Memory is becoming cheaper and larger memory sizes are becoming more common. On large machines GByte databases can be held entirely in memory, increasing performance substantially. Memory is also becoming faster to access, but the rate of increase is much slower than that of CPUs. For this reason, the use of fast on-board processor cache memory is becoming increasingly important.

2.1.1.3 Disk

Databases are all about storing persistent data, and the hard disk is central to any database. Parallel databases are designed for performance, and are generally targeted at large databases. Disk capacity is on the increase. Where one Gbyte drives used to be the norm, 2 Gbyte and now 4 GByte drives are becoming increasingly common with 9 Gbyte drives readily available. However, disk access rates have not been rising so dramatically, and 6 MBytes/s is a ballpark figure for a modern disk. Access rates can be improved with the use of disk caches - memory used by the disk controller, usually using a least recently used (LRU) algorithm to cache frequently accessed data.

Redundant arrays of inexpensive disks (RAID) can be used to increase the performance and reliability of a single drive. A RAID is a collection of disks, under the control of a single

controller (or multiple controllers in advanced systems). RAID devices can be configured in a number of levels; the levels that are mostly applicable to database systems are levels 0,1,5 and 10. RAID 0 has no redundancy and stripes the data over the available disks; this can give good performance but no extra reliability. RAID level 1 contains full mirroring but no striping, giving reliability and a boost in multiple read performance if both halves of the mirror can be read concurrently. RAID 5 is a compromise with parity checks providing redundancy at a 25% overhead; this gives a level of reliability, but decreases write performance. RAID 10 is a combination of levels 0 and 1, provide full mirroring with striping.

A good controller can make use of elevator algorithms and look-ahead. Reading from a disk requires a seek time in which the head moves to the required part of the disk, a rotational latency in which the head waits for the disk to rotate until it is correctly positioned, and an access time, proportional to the amount of data read. If access is completely random, the seek time can dominate and performance is far lower than a simple linear scan of the disk. Elevator algorithms work by sorting a series of accesses at the controller, according to their positions on the disk, thus reducing the overall seek time. Look-ahead algorithms work in conjunction with disk caches, taking advantage of head positioning to read more data than actually requested into the disk cache with a high likelihood of use in subsequent processing.

2.1.1.4 Interconnect

At the heart of a parallel machine is a form of interconnect which allows communication between all parts of the system. It is the interconnect that dictates the scalability of the system; once the interconnect is saturated, adding further components (CPU, memory, disk etc.) will not increase performance (except for cases where components can reduce the interconnect usage, such as on-board cache memory - however it is not always possible to make use of such components).

Interconnect itself can be scalable (such as mesh, switched networks, hypercubes) or non-scalable (such as the bus). A scalable interconnect provides many paths through which data can flow; by adding further paths the overall bandwidth can be increased (provided that these extra paths can be utilised). A bus is driven by a system clock; the bandwidth can not be increased without upgrading the entire interconnect. As the interconnect lies at the heart of the machine, this is not generally possible and so once a bus interconnect is saturated, the entire machine must be replaced.

3 General Architecture Issues

3.1 Parallel Hardware

There are two principal architectures for parallel computers. These are shared memory processing (SMP) (also known as symmetric multi-processing) and distributed memory processing (also known as massively parallel processing or MPP).

3.1.1 Shared Memory Processing (SMP)

SMP is characterised by a number of processors, each of which has access to a common memory address space (Figure 3-1). All memory accesses (apart from those to on-chip CPU cache) and I/O go through a common interconnect. Each processor can access every memory and I/O resource, thus simplifying application development and management. The interconnect is typically a bus or multiple buses. It is this interconnect that acts as a

bottleneck in large configurations, restricting scalability. The introduction of on-board cache can reduce the load on the interconnect; however, this can result in further contention as demand is made for data in the caches of off-board CPUs. This memory contention is similar to disk contention `pinging' of MPP database systems, as described in Section 3.1.2. A scheduling mechanism, giving each process an affinity to a particular processor can help to alleviate the problem of non-local cache access.

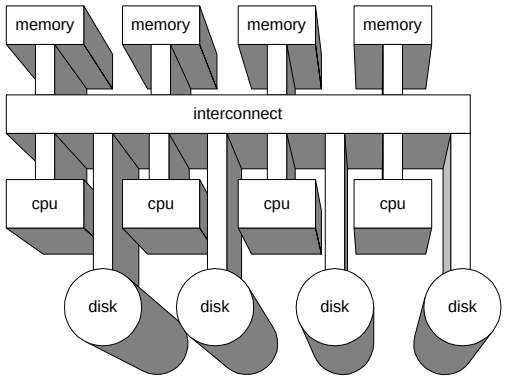

Figure 3-1 Shared Memory Architecture

Non uniform memory access (NUMA) is an emerging technology that replaces the single large bus philosophy of SMP systems. There are two varieties of NUMA: cache-only memory architecture (COMA) and cache-coherent NUMA (CC-NUMA). Both technologies use a processor board with a number of processors, a local bus and local memory as the basic component. COMA uses the local memory as a large cache. CC-NUMA distributes the main memory of the machine across the memory on each board, and maintains a consistent memory image. It can be thought of as a hardware layer that makes an internal MPP-like architecture appear to the application and the OS as an SMP system. The advantages of NUMA systems are that they reduce the amount of traffic on the global interconnect, increasing the overall memory bandwidth and power of the system. Figures of tens of Gbytes memory bandwidth are possible for large systems.

3.1.2 Massively Parallel Processing (MPP)

MPP is characterised by a number of processors, each with their own memory component, communicating via a high-speed interconnect. This interconnect typically has a lower bandwidth than the SMP interconnect as it only has to deal with I/O and inter-processor traffic. Usually the interconnect bandwidth is not fixed for the architecture but scales with the number of processors.

MPP architectures can be divided into two categories: shared nothing (Figure 3-2) and shared disk (Figure 3-3). This distinction can be made at both the hardware and software level. That is to say the hardware may be either shared nothing or shared disk, while the database software may also be shared nothing or shared disk. It is possible to implement a software layer above shared nothing hardware to make it appear like a shared disk system to software that requires it.

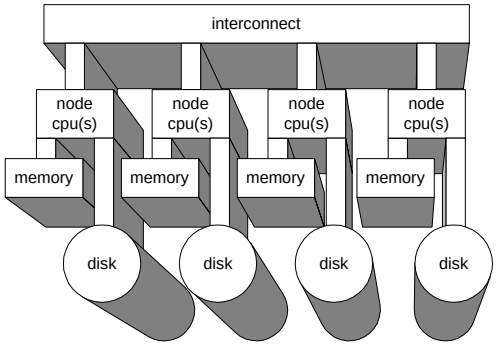

Figure 3-2 MPP Architecture (shared nothing)

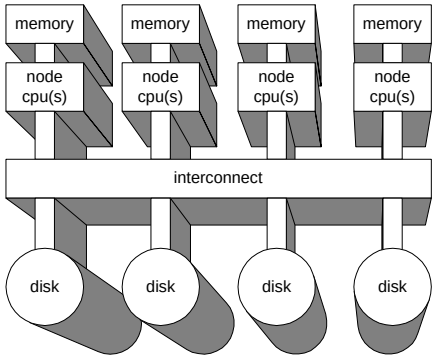

Figure 3-3 MPP Architecture (shared disk)

MPP platforms tend to be well suited for DSS applications due to the potentially high aggregate disk bandwidth and the data can be suitable partitioned to make use of it. MPP systems are not usually suited to OLTP due to the fact that is often necessary for many users to have exclusive access to all of the data. However, this is not necessarily true in all cases - some OLTP applications are suitably partitionable and work adequately on MPP.

3.1.3 Performance

This section contains a few rules of thumb that can be used when sizing systems. These rules are very approximate, but can give a useful indication to the expected ballpark size requirements.

Each disk can be expected to provide about 6 Mbytes/s per disk.

Each processor can be expected to provide 5-10 Mbytes/s per CPU, depending on the power of the processor, and the efficiency of the RDBMS and OS running on the processor. This figure will be lower if complex manipulation is required as well as scanning.

It can be assumed that there is a ratio of memory to I/O bandwidth of about 10-20, when running a relational database. It can also be assumed that the memory access bandwidth of a machine is limited by the interconnect bandwidth, rather than the total aggregate memory module bandwidth. These assumptions can be used when making comparisons between SMP and MPP interconnects, or when assessing the I/O potential of an SMP machine. For example, an SMP with a bus of 2 Gbytes/s could expect to be able to scan data at 100-200 Mbytes/s using a relational database. The exact figures would, of course, be case-specific.

3.2 Parallel Database Software

3.2.1 RDBMS

Relational databases are based on the relational model, developed by E.F Codd in 1970. This model centres on the relations between entities in the databases. By focusing on the relations inherent in the data itself, the database may be considered in logical terms, distinct from the actual layout of the data. This separation of the logical and physical layers is an important concept in relational database theory. Therefore, in a pure relational model, the data is laid out according to the relationships within the data, independent of the access path to the data. This approach increases the flexibility of the system, as it makes no assumptions on the details of the application.

Relational databases are implemented using relational database management systems (RDBMSs). Parallel relational databases are used to obtain maximum performance. There are two types of parallel relational database architectures: shared disk and shared nothing. These architectures are discussed in this section. Note that the categorisation into shared disk and shared nothing is at a software level, and is subtly different to the hardware categorisation. Both RDBMS architectures may be implemented on all hardware architectures.

3.2.1.1 Shared Disk

A shared disk database architecture is shown in Figure 3-4. All processors run an instance of the database. Each instance has access, through the interconnect, to the entire database which is spread over a number of disks. The figure shows one particular table (shaded) which is striped in four equal parts over the four disks.

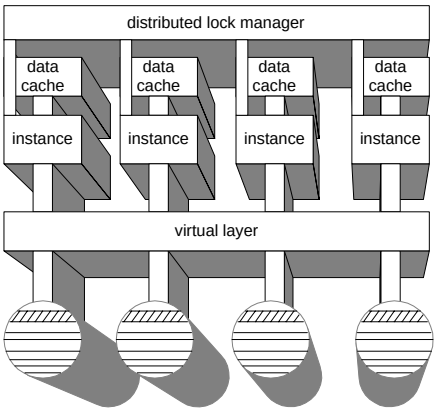

Figure 3-4 Shared Disk Database

It should be noted that shared disk databases can be implemented on any hardware. On SMP hardware, there is only one node with all disks connected to the main system bus. Thus, the shared disk nature comes for free, and no lock manager is required. A shared disk database can also be implemented on a shared nothing hardware, in which case a software layer must provide a logical shared disk to the RDBMS. The performance of this layer is key to the performance of the database on the particular platform.

As all processors can simultaneously access all data in the database, some form of control must be imposed to ensure consistency and prevent corruption. This is achieved with a system of lock management that co-ordinates data access.

An advantage of shared disk databases is that they can still operate when only one processor is on-line. This is not the case for shared nothing databases. A second advantage relates to database join operations. The problems of data location (described in Section 3.2.1.2) are avoided, and the database may make use of cached data in the memory of each node. The principal disadvantage is that the interconnect can become a bottleneck, particularly when implemented as a software layer. If the RDBMS has no concept of 'local data', it cannot take advantage of the high aggregate bandwidth of a shared nothing hardware machine.

Oracle is the main example of a shared disk database. Oracle 7.3 contains local data affinity, whereby each node has knowledge of what data is attached to local disks. This enables the RDBMS to take full advantage of shared nothing hardware; in effect it implements shared nothing access to a shared disk database.

One of the main issues associated with OLTP applications is that of contention for disk blocks that are resident in database caches (i.e. memory), resident on different nodes to the process in question. This requires a number of disk transfers, and is known as a `ping' because it results in disk blocks 'pinging' backwards and forwards between nodes.

3.2.1.2 Shared Nothing

The architecture of a shared nothing database is shown in Figure 3-5. An I/O process exists on each node, accessing the local disks on that node. In contrast to the shared disk architecture, each node 'owns' a fraction of the database. A control process (labelled 'c' on the diagram) controls the execution of each query. On many shared nothing databases, the control process for a given query may be placed on any of the nodes in the system. A shared nothing database can be implemented on any sort of hardware. On SMP or hybrid machines, many 'virtual nodes' may be placed on each node. However, unless shared memory is used for inter-process communication for processes on the same node, effective use will not be made of the hardware.

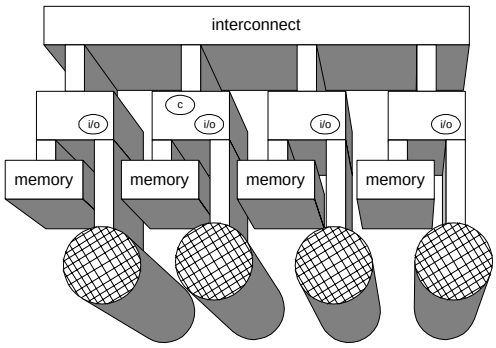

Figure 3-5 Shared Nothing Database

Shared nothing databases are dependent on a good data distribution, i.e. an equal amount of data on each node. This ensures that the processing workload is well balanced and therefore that the parallelism is efficient. Some databases are naturally partitionable in this way and good scalability is possible. For others, some tables may be heavily skewed, resulting in degraded performance.

Data distribution is often achieved using hashing techniques, although this is not the only method. Hashing involves passing one of the columns of a table through a 'hashing' function which generates a pseudo-random key. This key is then used to assign each row to

a node. If the data column has sufficient values, the rows will statistically tend towards a even distribution.

The shared nothing data distribution approach leads to extremely good performance for full table scans. Joins can pose more of a computational problem, however. If the join key has been used for distribution for each table of the join, the corresponding rows will be located on the same node, and there will not be a problem. If, as is sometimes unavoidable, this is not the case, one or more of the tables involved in a join must either be temporarily redistributed or duplicated for the query.

Shared nothing architectures can be differentiated by their integration approach. At one end of the sale, Sybase maintain their sequential database at each node, with the parallel functionality coming from a higher level layer (Sybase MPP, formerly called Navigation Server). This approach will ensure good capability between the sequential and parallel systems. At the other end of the scale, Informix have a redesigned, integrated product. This approach will tend to have lower overheads.

OLTP applications are generally less well suited to shared nothing databases than shared disk databases. There are two layers of processing (co-ordinators and servers), and so communications overheads must be taken into account. Additionally, in order to maintain consistency for complex queries, some form of two-phase commit protocol must be implemented.

3.2.2 MDDBMS

Multi-dimensional databases (otherwise known as on-line analytic processing, or OLAP) are a comparatively new technology designed to address the needs of a wide variety of users. An MDDB presents the data as a multi-dimensional 'cube', with one dimension for each database property (e.g. customer, data etc.). The user may segment ('slice and dice') this cube, choosing any collection of dimensions for comparison. The response time of such systems is interactive and immediate; the choice of dimensions does not affect the response time.

MDDB, or OLAP, is more than simply a storage technology. MDDBs generally include a selection mechanism for specifying data (similar to SQL in capability, but generally graphically based and easier and more intuitive to use). Graphical displays in the form of graphs, charts etc. and other report generation facilities are also included, along with meta-data to allow a business-oriented view of the data rather than a database-oriented view.

An MDDB can be implemented as a separate (to the RDBMS) package. Extracted subsets from the database are uploaded onto a separate platform (typically a PC). There are limits to the size of the MDDB that may be produced in this way (small numbers of Gbytes). The data may be stored as a large cube, or some more advanced sparse matrix form. Many MDDBs allow a number of cubes with different dimensions for each cube.

Alternatively, an MDDB can be implemented by storing the data in a standard RDBMS and placing a higher layer allowing a multi-dimensional view, business meta-data and selection and presentation tools. This is known as relational OLAP or ROLAP.

3.3 Application Software

Relational databases accept SQL, and so at the RDBMS level the application can be treated as a number of SQL statements. Actual applications will also contain further processing written in a 3GL or 4GL, but that need not concern us in this section.

Basic read-only queries may be either indexed (for interactive access) or non-indexed (requiring some form of scan). As an example of the range of query types, consider the following:

Indexed queries do not require access to all data items in a table. For example "What is the profitability of customer X?" or "What is the profitability of customer class Y? (where Y is a small subset of all customers)". Indexed queries are less intensive than scans, and the main factor influencing machine size is the number of concurrent users.

Aggregated or cached queries require access to all data items (a full table scan). However, this may be done once, or at regular intervals (e.g. monthly) with the aggregated or cached tables used by subsequent queries. For example, a cached query could be "Compute the profitability of all customers (storing the results)". An aggregated query could be "Summarise the average profitability of all customers, by customer class". With aggregated and cached queries, the major factors influencing machine size are the amount of aggregation/caching required and the available batch window.

Full table scans may be required on-line, in a 'pseudo-interactive' environment. For example, if the user wants to experiment with different customer segmentation criteria, acting on the entire customer database. The major factor influencing machine size becomes the required response time for the query. For example, if 5 minutes was deemed acceptable to scan a 10Gbyte rentals file, the required configuration would need to be able to scan at 34 Mbytes/s.

With OLTP applications, there are many issues to be considered associated with update and insert performance. However, for DSS and data warehouse applications, these issues are generally easily resolved.

More complex queries are possible, generally involving joins between a number of different tables. These queries can be broken down into components involving indexed accesses and full table scans. For shared nothing architectures, table redistribution for joins can also be a significant factor.

4 Relational Database Tuning

4.1 Introduction

Parallel Database systems are primarily involved with the identification and retrieval of data in large databases. This data is generally stored on disk, and so in simplistic terms it is basically an i/o problem: how to read and write data to and from disk as fast as possible.

There are a number of components of an RDBMS platform and these have been discussed in Section 2.1.1. In an ideal world, each of these components will be utilised 100%, doing useful work for the user, with no degradation in response. In practice, this is an impossible demand. The system will not be as well balanced as this, and at least one of the resources will be acting as a bottleneck. Furthermore, there is a relationship between the throughput of a system and the response times of the queries. As the throughput increases and the

system resources become saturated, an individual query has a smaller share of a given resource, and the response time increases. This increase is gradual at first, but as saturation approaches 100% the response time increases dramatically, and becomes highly unpredictable. For this reason, it is desirable in an interactive system to keep resource usage down at a level short of 100% (e.g. 80%). Resource usage for batch systems should also be kept lower than 100% as full saturation frequently causes database operational problems (the details of which are specific to the RDBMS in question).

The general principles behind relational database tuning are simple. The first area to concentrate on is the method by which the query is executed. The following questions can be asked. Is this the most efficient method, or is there another way of arriving at the same answer with fewer operations? Are there any redundant operations being carried out that can be dropped?

After the execution method has been validated, the second area to concentrate on is the presence of bottlenecks in the system. A bottleneck is a resource that is saturated, delaying the queries that are attempting to access it. A bottleneck will severely restrict the performance of the system, preventing other resources from being fully utilised. In extreme cases, one resource could dictate the entire performance, with all other resources lying idle.

Tuning is an iterative process. Once a bottleneck has been detected, and corrective action taken, the tuning process should recommence. It is quite possible that another resource that was previously under-used has now become a bottleneck. It is also possible that, in improving certain aspects of the system's performance, other aspects have degraded. These issues are discussed further in this section.

4.2 Initial Considerations

4.2.1 Buy the right size box

This may seem obvious, but there have been many instances where this step has been ignored. The effect of this is over-tuning to compensate for insufficient hardware. This generally leads to unpredictable and erratic responses with much stress placed both on the machine and its administrator.

It is also important to ensure that the hardware configuration is well-balanced and particular care must be taken here. The initial configuration may be sized based on the database size in Gigabytes. It is obvious that this cannot be used as the sole sizing criteria and that the processing power of the machine must be sufficient to cope with this amount of data. However, there are other issues that must be taken into account; for example buying a small quantity of large capacity disks would be the cheapest solution but may not provide enough disk resource to be used in parallel and hence may prove to be a system bottleneck. These issues are dealt with in more detail in subsequent parts of this section.

4.2.2 Know what you're tuning for

Tuning is a word with a wide scope and covers a multitude of performance characteristics. It is not a linear phenomena - it is not possible to take a given hardware platform and database combination and to tune them to an optimal state. Not only does the tuning depend on the workload that will be directed at the system, it also depends on which performance criteria should be emphasised. There are a number of criteria which can be used when tuning a system: the two most common criteria are minimised response time and maximised throughput. For example, an OLTP transaction designed to validate a credit-card transaction

to be executed on demand by a clerk may have severe response time constraints, whereas the processing of accounts may be achieved overnight with the requirement that a certain number of transactions be completed within the given batch window. These two requirements (minimising response time and maximising throughput) are very often conflicting. As the throughput is increased, resources tend to saturate and response times also increase.

Even within a particular criteria, such as minimising response times, there are number of decisions to be made. A system could be tuned to minimise the average response time (with maximum response times possibly much higher), tuned to minimise the maximum response time, or tuned on a percentile basis (e.g. 90% of the transactions completing within a certain response).

It is important to know which criteria a system is to be tuned against. This can have important effects on the entire operational process.

Some tuning techniques are generally applicable and will always increase performance (these tend to be along the lines of 'not doing something stupid'.) However, many tuning techniques will increase the performance of one aspect of the system at the cost of decreasing the performance of another aspect.

4.3 General Parallel Database Principles

There are two ways a database can use parallelism: inter-query parallelism and intra-query parallelism.

With inter-query parallelism, an individual query is *not* executed in parallel. However, one query will only use a fraction of the machine resources, and many queries may be executed in parallel. Hence, query response time will not benefit from parallelism, but the overall throughput will improve.

With intra-query parallelism, an individual query may be executed in parallel. Hence, the response time for the query will be dramatically reduced, consuming more of the machine's resource than the non-parallel case. Performance will degrade as the number of concurrent users is increased.

4.4 Tuning Approach

4.4.1 Logical and Physical Design

Database design can be divided into two components: the logical database design, and the physical database design. The first stage in implementing a database involves creating a data model. This is a diagramatic representation of the data, showing the various entities involved, and the relationships between them. There are many standard methodologies for performing this task.

The logical design is then produced from the data model. This involves specifying the tables which will make up the database, together with their primary and foreign keys for join operations.

The physical design dictates how the tables will be laid out over the disks and what the physical representations of each column will be. The primary keys are mapped on to

primary indexes (if applicable for the particular RDBMS used) and decisions made about what indexes should be used.

Once the physical design has been implemented, the system is ready for tuning.

4.5 Tuning Methods

4.5.1 Access Optimisation

4.5.1.1 Logical database design

The principal factor in determining the response time of a full table scan on a given platform is, generally speaking, the size of the table in bytes (as noted in Section 4.5.2.1). Therefore the response time depends both on the number of rows and on the length of each row. A table may be split up vertically into a number of smaller tables, each with the same number of rows and the same primary key, but with differing columns. This is useful if a significant number of queries access a small percentage of the table (for example two or three columns), or if the rows contain large data items (e.g. binary or multimedia information).

The disadvantage of this technique is that if information is required from both halves of the split table, a join must be performed with a corresponding overhead. This can be avoided by storing the full version of the table as well as the partial tables (i.e. duplicating data). This, however, causes a greater update overhead and requires more storage space.

DBInspector note

The table t_sgn002 may be applicable for splitting depending on the nature of the critical queries. If there is a strong performance requirement for full table scan queries that access a small number of columns (for example indicator generation queries that access only the transaction amounts), it may be beneficial to split the table. However, if these generations are achieved easily using batch processes and indexed retrievals are placing the main load on the machine, the technique will not be necessary.

4.5.1.2 SQL optimisation

Often, there are several SQL specifications that will retrieve the same data (logically equivalent queries). This is especially true for complex SQL commands. Usually, the different SQL specifications will have different response times and resource usage; some will be more efficient than others. SQL optimisation is about finding the optimum SQL statement specification for a given query.

Most databases provide some sort of explain plan which can be extremely useful in the tuning phase. The explain function takes a standard SQL statement and displays the output of the optimiser. For simple queries this may not be particularly useful, but for complex joins it can be invaluable in understanding how the system is performing the query.

Once an explain plan has been obtained, it can be analysed. This analysis centres on determining which parts of a complex query are using scans, and which parts are using indexes. For shared nothing databases, optimisation is also necessary for join commands. The optimum methods will depend on the nature of the data, the particular query and the desired performance

4.5.1.3 Avoiding contention

Contention is an important issue in an OLTP environment where many processes may access the same data. To maintain consistency, some form of locking is required and this causes other processes to wait for the locks to be released, resulting in performance degradation. The solution to the contention problem is to partition the data such that concurrent processes access their own data. This many not always be possible.

Contention is generally not as prevalent in DSS systems.

4.5.2 I/O Tuning

There are three principal methods for I/O tuning:

- Increasing the I/O bandwidth. This is useful for full table scans - see Section 4.5.2.1.
- Increasing the number of I/O's per second. This is useful for OLTP and indexed access - see Section 4.5.2.2.
- Reducing the number of I/O's (cacheing) - see Section 4.5.3.

4.5.2.1 Full Table Scans

The main advantage of parallel database technology is for processing full table scans of large tables. Indeed, it can be argued that performing full table scans, together with subsequent processing, is their *only* advantage, as indexed access and updates do not generally benefit greatly from parallelisation. The rate at which a DBMS can scan a table is therefore of paramount importance.

In a parallel system, it is extremely important that the table is spread over a large number of disks in order to increase the number of disk spindles that are being accessed in parallel. While RAID devices offer a performance increase over individual disks, they also tend to have a higher capacity, and thus it is possible to configure an equivalently sized system with fewer RAID devices than corresponding individual disks. This can lead to a disk bottleneck, as the inherent disk parallelism has been reduced. The same principal applies to the use of high capacity disks.

Generally, the performance of a full table scan is linearly dependent on the size of the table in bytes. The size in bytes is directly translated to disk blocks, and hence the amount of data that must be read from disk and processed by the system. Thus both the vertical (number of rows) and horizontal (number and size of columns) dimensions of a table influence performance.

Performance can sometimes be increased by using raw partitions rather than OS file systems. For Oracle Parallel Server this is mandatory as OS file system caches can interfere with database consistency.

4.5.2.2 Transactions and Indexed Queries

Transactions and indexed queries are generally associated with operational systems, rather than the data mining approach adopted by the DBInspector environment, and as such they remain outside the scope of this project. Performance is increased by the use of multiple parallel disks, disk caching and efficient controller algorithms (elevator and look-ahead algorithms).

4.5.3 Memory Tuning

Memory has much higher access rates than disk, and can be used as a cache to increase performance. There are two main places where this can be utilised: database level and disk level. Some form of cache management is required at each level.

Database level cache can be highly integrated with the RDBMS and can make use of database level algorithms such as specifying individual tables or indexes to be placed in the cache. Disk level cache exists at a lower level , and the cache management does not have knowledge of the content of the data. Simple LRU and look-ahead algorithms are the most common techniques used.

5 Oracle Tuning

5.1 Oracle Architecture

5.1.1 Introduction

If Oracle is run on a sequential or SMP machine, the user starts up an *instance* of Oracle. This instance consists of a number of background processes which manage the instance, and an allocated area of memory, known as the System Global Area (SGA). The details of an Oracle instance are discussed in Section 5.1.4.

5.1.2 Parallel Server

The Oracle Parallel Server Option (known as OPS) allows Oracle to make use of an MPP machine. Each node of the MPP machine contains an instance of Oracle, with its own SGA etc. Oracle has a shared disk architecture and each node of the machine must be able to access all the disks.

With the Parallel Server Option on its own (without Parallel Query), the instances do not talk directly to each other, and there is no intra-query parallelisation. The system does of course benefit from inter-query parallelism and many different queries may be executed concurrently on different nodes.

With many nodes having write access to the data, some form of arbitration is necessary to maintain consistency and prevent corruption. This is provided by the distributed lock manager (DLM). The DLM is generally implemented by the hardware vendor from a template provided by Oracle. For update intensive applications, the DLM can become a bottleneck and special attention must be given to its tuning.

5.1.3 Parallel Query

Oracle Parallel Query (OPQ) is separate to Parallel Server, and may be used either independently (e.g. for an SMP node) or in conjunction with OPS. OPQ allows the database to use intra-query parallelism. A number of slave processes are spawned, each process acting on part of the query. Co-ordinator processes collate the data from the slaves and present them to user.

Oracle uses a dynamic strategy for partitioning work amongst the slave processes. Initially the slaves all take relatively large portions of the query. However, the portions are not the largest possible: for n slaves, each slave will take less than $1/n^{th}$ of the data, and so there will

still be work left after the first slave has finished its unit of work. The slave will then take the next portion of work from the heap. The portions of work become progressively smaller as time progresses. This method attempts to perform dynamic load balancing, by minimising the chances of one slave spending a lot more time on a query than the other slaves, thus dominating the response time.

5.1.3.1 Tuning issues

If parallel query is going to be an important activity on a particular database installation, it is beneficial to pre-start the query slaves. A number of slaves are started at instance start-up, and these are used as required by parallel queries. This approach saves on the query slave start-up time at the beginning of a query. The Oracle tuning parameters parallel_min_servers and parallel_max_servers are used to achieve this.

For a given parallel query, both the number of parallel query slaves within a node, and the number of nodes can be set (with the hints 'degree' and 'instances' respectively). As the number of slaves per node increases, the parallel performance also increases, until eventually the system saturates and performance tails off. The saturation point will depend on the number of processors in the node. A figure of two slaves per processor is a good starting point for near optimum performance. For the specific implementation used on the NCR 3600 during the project, it was found that three slaves per processor gave optimum performance, although this was not significantly greater than two slaves per processor. Deliverable 4.2.0 contains further details on this work.

Each parallel query slave runs as a separate process; therefore running many parallel query slaves requires many processes to be executed concurrently. Provision must be made for this within the OS and Oracle. There is a Unix kernel tuning parameter which sets the maximum number of concurrent processes (NPROC) and this must be set at a suitable level. In addition, the Oracle tuning parameter 'processes' must be set.

The Oracle tuning parameter 'sort_area_size' is used to set the size of memory used for sorting. This unit of memory is taken for each degree of parallelism within the instance; so for a sort_area_size of 2M and a query of degree 16, 32M will be required. It is important that the sort_area_size is made large enough; insufficient sort space can cause performance degradation and database hanging.

Parallel queries can also use the shared pool, and it may help to use a large 'shared_pool_size' when running highly parallel queries.

5.1.4 Within an Instance

5.1.4.1 Data Layout

Within Oracle, data is stored in tablespaces. A tablespace corresponds to one or more files in the underlying file system. Everything contained within a tablespace is maintained internally by Oracle.

Tablespaces consist of segments; there are four different types of segment: tables, indexes, temporary segments and rollback segments. Temporary segments are used internally by Oracle during complex queries. Rollback segments are used to hold rollback information (see Section 5.1.4.2).

Each segment consists of a number of extents. These extents are of variable size, and may or may not be the same size, either within a tablespace or even within a segment. The extents are an integer number of Data Blocks (referred to as blocks). The block size is fixed for a given database (at creation time) and cannot be changed without recreating the database. The block size must be n x 1024 bytes, where n is generally a power of 2.

5.1.4.2 Rollback Segments

One requirement of a database transaction is that it be atomic, i.e. it either succeeds completely, or has no action at all. There can not be an intermediate state. In order to achieve this, the system keeps track of an executing query using rollback segments. These are stored in the same way as data segments; however it is conventional to store rollback segments in their own rollback tablespace.

The rollback segments provide a complete record of what data blocks have been modified in the course of the query. If the query fails, the rollback segments are used to re-create the state of the affected tables at the start of the query. Rollback segments are treated as circular buffers by Oracle. If a given query requires so much rollback information that it wraps a rollback buffer, the query will immediately fail and rollback from the point it has reached.

Each instance only controls the rollback information for its own transactions, independently of all other instances. It is therefore wise to give each instance its own rollback tablespace and place it, if possible, on a locally attached disk for higher performance.

5.1.4.3 System Global Area (SGA)

Each Oracle instance contains an area of memory for housekeeping information and the data cache known as the System Global Area (SGA). The major components of the SGA are:

Data block buffer: this contains the data cache and is described in Section 5.2.3.1.

Redo log buffer: this is a circular buffer that contains redo information. Redo information is generated for each committed query and allows the database to recover from a checkpoint or a crash.

Shared pool: this is a shared area of memory which is used by multi-threaded server sessions and for shared SQL.

SQL library cache: this is used to hold parsed SQL queries to save the parsing time on subsequent query submission. It is only useful when many similar interactive queries are being executed (e.g. an OLTP application). For DSS applications, the time taken to parse the SQL is a small percentage of the overall response time.

For shared memory machines it is recommended that the SGA is placed in one shared memory segment. The maximum shared memory segment size should therefore be set large enough to accommodate the SGA. This is typically set as a kernel tuning parameter (SHMMAX on SVR4 systems).

5.2 Tuning Methods

5.2.1 Access Optimisation

As mentioned in Section 4.5.1.2, Oracle provides a method of breaking down a complex query into a series of steps. This can be achieved using the "explain plan" SQL statement, or by using the SQL trace utility. The trace utility is activated by setting the sql_trace Oracle tuning parameter, which causes a trace file to be generated for each executed query. The trace files can then be processed using the tkprof command to generate an explain plan. This method has an advantage over the simpler "explain plan" method, as it provides information about the actual query executed, including volume sizes and (if timed_statistics is set) response times. However, setting sql_trace does place a slight overhead on the system, and should be turned off once the tuning process is complete.

For queries (or parts of queries) that are highly selective, it is beneficial to use an index for better performance. However, as the proportion of the table that is returned increases, the response time increases and the performance advantage given by the index decreases. At a certain level, the response time will be equal to that of a full table scan without an index. This level will be less than the entire table (typically about 20%) due to the overhead of the index.

With parallel query, it may be beneficial to use a full table scan for smaller selections. This is because the ability to perform the query in parallel decreases the response time to below that of the indexed query. However, it must be realised that this will consume more of the machines resource, and concurrent queries will be affected.

5.2.2 I/O Tuning

5.2.2.1 Data I/O

The most important aspect of Oracle I/O tuning is to balance the data distribution, and workload distribution, over many disks. This enables the disks to be read in parallel, increasing the total disk bandwidth. Oracle provides a method of using multiple files to specify a tablespace. However data is not actually striped across the disks using this method; standard Oracle inserts will fill each file one by one. A better distribution can be obtained by using a direct parallel load and manually partitioning the file and assigning each partition to a file. However, the best method is to use an external method of striping the data, such as RAID or software virtual disk layer.

A RAID box can be used as a kind of "super disk". This will tend to have a higher performance than a single disk drive; the performance will be limited either by the rate of input and output to the RAID itself (e.g. the performance of the disk controller(s)), or by the physical connection of the RAID. However, there is a limit to the capacity that can be obtained from a single RAID.

For high disk bandwidth, it is beneficial to use some form of software virtual disk layer. This makes a number of disk devices appear as a single disk and can be configured to stripe the data across the disk automatically. The increase in disk concurrency will outweigh the small performance overhead of the extra layer. The DBInspector work at PAC was carried out using the NCR Disk Array Plus product.

There is a Unix limit of 2Gbytes per file, and so larger tables must use the Oracle multiple file method (using virtual striped files). This procedure is not necessary with 64-bit Oracle, which is available for 64-bit platforms (currently only DEC Alpha machines).

For best performance, data should be placed in raw partitions, to minimise file system overheads. With Oracle Parallel Server, this is obligatory, as file system caches can interfere with consistency. Raw disk partitions have a disadvantage of being slightly harder to manage and backup. On Unix systems, backup can be done using the Unix `dd` command. A couple of useful techniques may be used to ease the administration workload. Firstly, fixed size partitions can be used. This allows easier planning and the possibility of moving partitions without requiring complete disk re-partitioning. At PAC, partitions of 10M, 100M and 500M were used (actually 10.25M, 100.25M and 500.25M to allow space for the fixed tablespace overhead: 0.25M was allowed as this corresponded to the disk cylinder boundaries). Secondly, the raw disk files can be accessed through an indirect Unix link. This is acceptable, performance and consistency-wise, and allows some rearrangement flexibility. It is also obligatory if nodes of a Parallel Server configuration have different access addresses to the disks (as was the case at PAC), as Oracle requires that all nodes see a common file name for a given disk file.

If the data is constantly being updated with records being added and deleted, it will eventually become fragmented. Fragmented data takes up more space than necessary and requires more disk seeks, and therefore causes a performance degradation. This can be alleviated by an export of the data, followed by an import.

5.2.2.2 Other I/O

Redo logs should, if possible, be placed on their own, separate, disks. This is because, in the course of normal operation, redo I/O is write-only and sequential. A redo disk write must take place when a commit occurs (it is the only disk activity that *must* take place, data writes can be deferred), and so it is essential that no contention occurs at this point.

It is sometimes beneficial to separate data and indexes to different disks. This can allow higher performance for indexed accesses, as index accesses and data accesses can be performed concurrently.

Oracle does not currently manage temporary space very well. Temporary tablespaces are required to provide extra temporary storage space for the processing of large queries. Oracle does not appear to use this very efficiently, and often it is necessary to have more temporary space than would have been thought necessary, particularly for large queries involving sorting. For the DBInspector work carried out at PAC, a 2Gbyte temporary tablespace was used. This is almost as large as the 3Gbyte database.

5.2.2.3 Tuning Parameters

db_block_size: For OLTP, the Oracle tuning parameter db_block_size should be relatively small. Using smaller blocks means that the data cache will contain more blocks for a given memory size, and the cache hit rate will be correspondingly higher. A size of 2K is common. For DSS, the db_block_size should be larger. Using larger blocks means that a large table will be contained in fewer blocks, and so the overhead associated with accessing each block will be less significant. A size of 8K is common, although larger sizes may also be used (if allowed by the particular Oracle port).

db_file_multiblock_read_count: This Oracle tuning parameter dictates how many consecutive blocks are read in one unit of I/O. For scanning large tables, it should be made as large as possible to decrease overheads. This may decrease performance with tables smaller than the unit of I/O (db_block_size x db_file_multiblock_read_count). With older versions of Oracle, the full table scans would stream through the cache, using db_file_multiblock_read_count cache blocks. New versions (Oracle 7.2 on) allow direct I/O which allows full table scans to by-pass the cache.

'virtual shared disk stripe size': This parameter should be defined as an integer multiple of the unit of I/O. Preferably it should be significantly larger than the unit of I/O (e.g. 32-64 times larger). Unfortunately, the maximum stripe possible using NCR's DiskArray+ software, used in the project, was 256K. Ideally a figure of 4-16M would have been used.

5.2.3 Memory Tuning

5.2.3.1 Instance Level

A large part of the Oracle SGA contains the data cache. This is an area of memory where data and index blocks are kept on a least recently used (LRU) basis. The data cache is sized by defining the number of blocks. The total size is therefore the number of blocks multiplied by the size of each block.

Full table scan operations generally do not use the cache, although an individual table may be placed in the cache if it is small enough. The cache is mostly useful for indexed accesses, such as OLTP or complex interactive DSS. Often, there is a higher performance advantage gained from having index blocks in the cache rather than data blocks, as there is a higher chance of getting a cache hit. However, the user does not currently have that degree of control over cache entries.

For applications that are extremely dependent on the size of the cache (especially indexed applications), it is generally sensible to make the data cache as large as possible. The exact size will depend on other memory requirements such as the DLM and other applications. If the cache is too large, then the entire node will be short of memory and will start to page to disk, degrading performance. For optimum cache size for a given configuration, the cache can be increased slowly until paging occurs, and then turned down to just below the limit. This approach will require continuous monitoring subsequently, as any increase in users or application workload will bring the memory usage above the machine capacity and paging will occur. For applications that are not so dependent on the size of the cache, such as those involving many full table scans, it is sensible to keep a relatively modest cache to reduce the risk of paging.

6 DBInspector Observations

During the DBInspector project, PAC installed the Financial Flows Archive, supplied by UIC, and performed a number of performance tests, based on two sets of queries: Deliverables D3.0.0 and D3.1.0. These tests, and their results, are detailed in Deliverable D4.2.0. During the performance testing of the DBInspector queries on the Oracle environment, it was found that significant changes were made by Oracle to the Oracle 7 product. These are summarised as follows:

Oracle 7.0. Inter-query parallelism is supported, but not intra-query parallelism. This was the current version at the time of production of the technical annex.

Oracle 7.1. Selects are performed in parallel, but no other database operations. Parallel sorting operations are performed poorly. This was the current version at the start of the project.

Oracle 7.2. Better performance for parallel sorts. Create table as select can be performed in parallel. This was the current version towards the end of the project, and the final version to be installed at PAC.

Oracle 7.3. Node affinity, resulting in more efficient full table scans. This was the current version at the end of the project, but was not installed at PAC due to resourcing reasons.

The key conclusions of the work done by PAC, as detailed in Deliverable D4.2.0 are as follows:

- For a given (low) number of processors, SMP is more efficient than MPP. Hence, to gain significant advantage from MPP, either the individual nodes should be high powered SMP nodes, or a large number of nodes should be used.
- The NCR 3600 platform used for the PAC tests had a maximum of 4 nodes (as such, the PAC machine was maximally configured), with a relatively low specification SCSI bus acting as the hardware shared-disk interconnect.
- The Oracle RDBMS was subject to a number of substantial changes during the course of the project, which reduced the effectiveness of detailed study of a particular version.
- The relative immaturity of the intra-query functionality of Oracle meant that it performed well on SMP nodes, but the MPP functionality lacked robustness and the MPP performance was a little disappointing.
- Oracle 7.3 will contain data affinity so that each node has a concept of 'local data'. In effect, this allows shared nothing access to a shared disk system. If implemented correctly, this will result in a significant performance improvement, realising the potential of MPP.
- Indexes can be used on the main financial flows table to allow efficient access.
- A parallel full table scan can result in a faster response time than an indexed access, although it will use a greater proportion of the machines resources. The relative performance of a full table scan and an indexed access will depend on the fraction of the table accessed, and the specification of the machine.
- It was discovered that caching data in temporary tables was an important technique for each of the analysis methods: statistical analysis, neural networks and visualisation.
- Effective management of the production and maintenance of the temporary tables is key to a successful system.
- The Transform Control Format (TCF) was produced in order to achieve this management.

6.1 The First Set of Queries

The first set of DBInspector queries were issued as Deliverable 3.0.0. These consisted of thirteen indicators, each of which were applied to an individual month of the data set. The first step of the indicator production involved the creation of a temporary table, aggregating the key data columns over financial organisations, municipalities etc. One temporary table would be created for each month. All indicators for that month would then be created from

the temporary table. The creation of the initial temporary table is the only part of the process that accesses the original financial flows archive, and this was by far the most time-consuming query.

An index on the date column can be used to improve the response time of this query. This is particularly useful if there are a large number of indicator base tables that need to be created at the same time (for example, if a new indicator is developed, or if a large amount of incoming data forces the existing indicators for many months to be re-calculated).

6.2 Recommendations

The optimum database layout and tuning decisions will be dependent on the access patterns of the user community. However, the following general recommendations can be made.

- Put all the transactions in one table. De-normalising the data into smaller tables (for example, one table per year) introduces artificial divisions in the data that do not produce sufficient performance advantages to justify them.. Indexing can provide fast access to subsets of the data.
- Put indexes on common access columns - for example: date, province, municipality, organisation
- Do not use indexes when selecting greater than 20% of the transactions table. It is faster and more efficient to do a full table scan.
- If a query is retrieving a relatively small number of transactions (e.g. less than 10000 rows), it is best to use an index. For larger retrieval sizes, using Oracle PQO may give a faster response time. This will, of course, use more of the machine resource. The level at which PQO becomes viable will depend on the amount of parallelism obtainable from the hardware platform. If the parallelism gain on the machine is x, the level at which PQO becomes viable is $(20/x)$ percent of the table. For example, if PQO gives a factor of 10 in performance over a standard full table scan, it may be worth using PQO when selecting more than 2% of the table.
- Table columns should be made large enough to hold aggregated data. For example, the transaction value column should be large enough to hold the maximum total transaction value (potentially across several years of transactions), rather than merely the maximum value for a single row. This will enable the Oracle construct "create table as select" to work.
- Temporary tables can be used to hold cached, aggregated or other pre-processed data for rapid subsequent processing.
- Virtual attribute columns should be added if there is a sufficient need, and pre-processing used to set the values. For example, a column for (itot/nopztot) could be created if this expression was used frequently.
- A large database cache is not necessary if most of the queries are either full table scans or ad-hoc queries (as appears likely). A large database cache would be necessary if there are many users accessing the same tables via indexes.
- A large database block size is necessary to increase the full table scan I/O rate.
- A large 'db_multiblock_read_count' Oracle parameter is necessary to increase the full table scan I/O rate.

- Oracle PQO uses many Unix processes, and it is therefore necessary to ensure that the relevant Oracle and Unix parameters are set to cope with this, as described in Section 5.1.3.1.

7 Conclusions

The components that comprise a parallel machine have been described, together with the principal architectures for parallel hardware and parallel relational database systems. Generic tuning issues for parallel relational database systems have been described, as well as principal tuning techniques for Oracle. Finally, some specific recommendations for the implementation of the DBInspector database have been made.