20TB on a Linux Cluster Today: How to Build a Multiterabyte Data Warehouse, Using Linux and RAC

An Oracle White Paper
November 2004
# 20TB on a Linux Cluster Today: How to Build a Multiterabyte Data Warehouse, Using Linux and RAC

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>3</td>
</tr>
<tr>
<td>Getting the Right Throughput on your Hardware</td>
<td>4</td>
</tr>
<tr>
<td>Configuring the hardware</td>
<td>6</td>
</tr>
<tr>
<td>Switch</td>
<td>6</td>
</tr>
<tr>
<td>HBA</td>
<td>7</td>
</tr>
<tr>
<td>Balance between switch capacity, number of disk arrays and HBAs</td>
<td>7</td>
</tr>
<tr>
<td>Storage</td>
<td>7</td>
</tr>
<tr>
<td>Before Installing Oracle consider dd</td>
<td>11</td>
</tr>
<tr>
<td>Oracle Configuration</td>
<td>13</td>
</tr>
<tr>
<td>Asynchronous IO</td>
<td>13</td>
</tr>
<tr>
<td>Memory</td>
<td>15</td>
</tr>
<tr>
<td>Data Compression</td>
<td>15</td>
</tr>
<tr>
<td>Tuning the Interconnect Protocols</td>
<td>16</td>
</tr>
<tr>
<td>Specifying the Interconnect Protocol</td>
<td>17</td>
</tr>
<tr>
<td>Tuning the Interconnect Protocols</td>
<td>18</td>
</tr>
<tr>
<td>Conclusion</td>
<td>19</td>
</tr>
</tbody>
</table>
Introduction

An emerging trend in the industry is the deployment of a “Grid” of industry standard commodity server running the Linux operating system for applications that have traditionally been deployed on high-end proprietary SMP systems. Many decades of development has gone into making these SMP systems scalable and reliable. To be successful a Grid based solution must be able to achieve the same level of performance and reliability as the SMP system it is replacing and be able to do this at a lower total cost of ownership.

By following simple guidelines for configuring a data warehouse, one can avoid unreasonable expectations and misunderstandings about what Linux and commodity boxes can do and cannot do. Expectations should come from an understanding of the technology, not from a personal interpretation of the marketing buzz. Some of the common misunderstandings are

“I need 6.5GB of I/O throughput and 150 concurrent users; make it happen on a $2 million Linux solution”

“If I wire the system together with 2 Gigabit cables I will get 2 Gigabytes of throughput”

“I only need 10 disks to store my 1 TB database now that we have 130 GB disk drives!”

This paper will help guide system engineers to create reliable and performant Linux RAC systems for a data warehouse applications. The paper is organized in three sections. The first sections supplies guidelines setting up hardware. Starting with a simple model on how to determine throughput requirements it continues with guidelines on how to meet these requirements. Before installing Oracle on a configured system we recommend to measure IO throughput with a generic IO testing tool. The second section shows how to measure IO throughput using the widely available dd tool. In the third section we focus on the software side of the system setup. This includes guidelines to configure Oracle and other system components.
Getting the Right Throughput on your Hardware

Gigabit vs. Gigabyte

A common confusion in discussions between customer, engineers, and software/hardware vendors is the use of the Bits vs. Bytes in component specification. Customers and Software vendors tend to communicate their requirements in Bytes/Second while hardware vendors tend to measure their hardware in Bits/Second. This can yield a lot of confusion and, in some cases, to badly configured systems. The usual convention is for megabit to be written as Mbit (lowercase “b”) and megabyte to be written as MB (upper case “B”).

It is generally accepted that 1 Gb/s equals 128 MB/s. The following table shows a list of system components with their typical performance in MB/s.

<table>
<thead>
<tr>
<th>Component</th>
<th>Throughput Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bits</td>
</tr>
<tr>
<td>16 Port Switch</td>
<td>8 * 2Gbit/s</td>
</tr>
<tr>
<td>Fibre Channel</td>
<td>2Gbit/s</td>
</tr>
<tr>
<td>Disk Controller</td>
<td>2Gbit/s</td>
</tr>
<tr>
<td>GigE NIC</td>
<td>2Gbit/s</td>
</tr>
<tr>
<td>1Gbit HBA</td>
<td>1Gbit/s</td>
</tr>
<tr>
<td>2Gbit HBA</td>
<td>2Gbit/s</td>
</tr>
</tbody>
</table>

Determining the IO demand

One of the first steps in sizing a system is to determine the amount of IO that is needed for the application.

There is no general answer to the question of how much IO your system requires since it heavily depends on the workload running on the system. Two approaches to sizing a system have been successfully used by the authors to determine the IO demand of the application. One is to define a lower and upper bound of the IO throughput, the other is to size the IO according to the number of CPU’s used in the system.

Queries used in Data Warehouse Applications can generally be divided into two categories: IO bound and CPU intensive queries. Given the examples below it is necessary to determine which type of queries dominates the intended workload. IO intensive queries define the upper boundary of the IO throughput, while CPU intensive queries define the lower boundary. An extreme example of an IO intensive query is the following parallel table-scan of lineitem.

---

1 This performance is not the theoretical maximal performance as specified in the manual. This number indicates what we have observed on real systems running Oracle under DSS workloads.
SELECT store_name, AVG(sales)
FROM store_sales s, time t
WHERE ss_sold_date_sk = t_time_sk
    AND t_time_year between 1999 and 2002
GROUP BY store_name;

This query spends very little CPU time processing rows, the majority of the elapsed
time is spent on IO. When run exclusively\(^2\) on the system it can drive about 200
MB/s using today’s CPUs (3 GHz Xeon processor).

On the other hand, consider the following, more complex query. In addition to
performing a table-scan it also performs aggregations.

SELECT store_name, returnflag,
    linestatus,
    sum(quantity),
    sum(extendedprice),
    avg(quantity),
    avg(extendedprice),
    avg(discount),
    count(*)
FROM store_sales
GROUP BY returnflag, linestatus
ORDER BY returnflag, linestatus;

Due to the aggregations this query spends more CPU time processing rows than
reading rows, thus challenging the IO subsystem far less. In single-user mode this
query reads about 50 MB/s. A typical system’s workload will fall anywhere
between the two extreme cases. To determine the throughput requirements of a
system, it is necessary to analyze whether the workload contains more IO or CPU
intensive queries. Keep in mind that the throughput numbers for these two
queries are for single user queries running against one table at a high degree of
parallelism. More complex queries use join operations, which show phases of IO
and CPU activities. Typical systems operate in multi-session mode, where multiple
concurrent users issue queries. Also, another typical workload can be found during
ETL operations. During these operations write performance is also to be
considered.

Another approach is to size the throughput requirements solely on the number of
CPU's that are attached to your system. On an average data warehouse system the
rule of thumb is that today's CPUs (3.0 GHz Xeon) can consume a sustained rate
of about 200MB/s of disk IO\(^3\).

\(^2\) Being the only query running on the system
\(^3\) Running in IO intensive queries in parallel
Configuring the hardware

There are many components involved in completing a disk IO (from the disk into memory). Each component on the IO path, starting with the disk controller, connections, possibly a switch and the Host Bus Adapter (HBA) need to be orchestrated to guarantee the desired throughput.

Figure 1 shows a conceptual diagram of a 4 node RAC system. 4 servers, equipped with 2 host bus adapters (HBA) are connected to 8 disk arrays using 2 8-port switches. Note that the system needs to be configured such that each node in the RAC configuration can access all 4 disk arrays.

![Figure 1: Conceptual System Diagram](image)

to deliver 800 MB/s. Half of the IO throughput gets routed through Switch 1, the other half gets routed through Switch 2, hence the total switch throughput is 1.6 GB/s. Each system has 2 HBAs, each capable of delivering 200 MB/s. An IO intensive operation running across all 4 nodes can read in 1.6 GB/s (4 nodes * 2 HBAs * 200 MB/s). Using this system a 1 Terabyte table can be scanned in 625 seconds.

Switch

There is not much to configure in the switch. Nevertheless it is an element on the critical path and the maximal throughput needs to be in-line with the aimed
throughput. Some switches are organized in Quads. These switches are optimized for packets sent between ports connected to the same quad. Since IO activity never occurs between HBAs, each quad should have an equal number of HBA and disk array connections to maximize total switch throughput.

**HBA**

Each node will have one or more HBA's. An HBA is a fiber channel host bus adapter. Each HBA is capable of approximately 200 Mb/sec. Each node should have as many HBA as necessary to satisfy the calculated IO demand. Additional software may be required to take advantage of multiple HBA's. This software will control load balancing over the HBA’s and failover capabilities.

**Balance between switch capacity, number of disk arrays and HBAs**

In order to ensure that all components on the IO path are of equal capacity, you need look at the total capacity of all HBAs, the switch and the disk arrays. At the same time you need to keep system limitations in mind. The number of slots for HBAs, and Network Interface Cards (NICs) are limited, so are the number of ports on a switch and the number of controllers in a disk array. Note, that some systems provide one or two on board NICs which can be used for the public network and heartbeat. Also, to increase HBA capacity, one can use dual port HBAs.

Let's assume one has a 16 port fibre-channel switch. Therefore, the total number of disk controllers and HBAs, one can attach to this configuration cannot exceed 16. Furthermore, the total number of HBAs should always be the same as the number of disk controllers providing that there are sufficient slots for HBA's. The ideal configuration is having 8 HBAs and 8 disk controller yielding a total throughput of 8x200MB=1.6GB/s. It is important that the switch’s capacity is also at least 1.6GB/s. If we reduce the number of HBAs to 4 we can attach more disk arrays, but the total throughput will still be determined by the 4 HBAs, which is about 800MB/s.

**Storage**

Configuring a storage subsystem for a Linux grid environment is no different than for any other platform. It should be simple, efficient, highly available and very scalable but it does not have to be complicated. One of the easiest ways to achieve this is to apply the S.A.M.E. methodology (Stripe and Mirror Everything). S.A.M.E. can be implemented at the hardware level or by using ASM (Automatic Storage Management) or by a combination of both. This paper will only deal with implementing S.A.M.E using a combination of hardware and ASM features.

From the hardware perspective, one should build in redundancy by implementing mirroring at the storage subsystem level. Group the disks into hardware disk
groups. Then create LUNs striped across all of the disks in a hardware disk group. The stripe size should be one-megabyte to ensure the full bandwidth of all disks is available for any operation. If it is not possible to use a 1MB stripe size ensure the number of disk in the hardware disk group multiplied by the stripe size is greater than or equal to 1 MB.

ASM is a new feature introduced in Oracle 10g, which provides filesystem and volume manager capabilities, built into the Oracle database kernel. To use ASM for database storage, you must first create an ASM instance. Once the ASM instance is started, create one or more ASM disk groups. A disk group is a set of disk devices, which ASM manages as a single unit. A disk group is comparable to a LVM’s (Logical Volume Manager) volume group or a storage group. Each disk in the disk group should have the same physical characteristic such as size or speed as ASM spreads data evenly across all of the devices in the disk group to optimize performance and utilization. If the devices have different physical characteristic it is possible to create artificial hotspots or bottlenecks. ASM uses a 1-megabyte stripe size.

ASM can also increase database availability by providing the ability to add or remove disk devices from disk groups without shutting down the ASM instance or the database that uses it. ASM automatically rebalances the files across the disk group after disks have been added or removed. Remember when using RAC extra disks are required for OCR (Oracle Cluster Registry) and the voting disk. These disks cannot be part of ASM. It’s recommended that if you use ASM on Linux you use the ASM library driver and associated utilities to configure the devices that will be include in the ASM disk groups. For more information on the ASM library driver please see the Oracle 10g Real Application Clusters Installation and Configuration Guide or the Oracle technical white paper Oracle Database10g Automatic Storage Management ASMLIB Storage Management Interface.

There is a potential pit fall that should be avoided when using hardware striping and ASM, which is having the ASM stripes overlap the hardware stripe. For instance, consider the following diagram of a disk subsystem with 16 disks, D1 to D16 in a single hardware disk group HG1.

![Diagram of HG1 - 16-disk hardware disk group]

*Figure 2: HG1 - 16-disk hardware disk group*
Four LUN’s A, B, C, D are created on hardware disk group HG1. Each LUN is striped over all disks in the hardware disk group with a 1M stripe size.

An ASM disk group g1 is created using LUN A, B, C and D. In the diagram below the black box represents ASM’s 1MB stripe. The first 1MB is allocated from LUN A, physical disk D1. The second 1MB is allocated from LUN B physical disk D1. The third is allocated from LUN C physical disk D1. The fourth from LUN D physical disk D1. The fifth from LUN A disk D2 and so on. The first 16 MB will actually be allocated from 4 different slices of the same 4 physical disks (D1, D2 D3 & D4), which will cause increased disk contention and increased disk head movement. Both of which will result in poor I/O performance.

The best way to avoid this disk contention would be to create 4 hardware disk groups HG1, HG2, HG3, HG4.
Each hardware disk group would have 2 LUN’s A1 B1, A2 B2, A3 B3 and A4 B4.

The ASM disk group g1 would be created using LUN 1A, 2A, 3A and 4A. The first 1MB would be allocated from LUN A1, physical disk D1. The second from LUN A2 physical disk D2. The third from LUN A3 physical disk D3. The fourth from LUN A4 physical disk D4. The fifth from LUN A1 disk D5 and so on. That way the frist 16 MB will actually be allocated from 16 different physical disks.
Before installing Oracle consider dd

Before installing Oracle on a system one should consider running a simple IO test to make sure that one gets the throughput from the IO system that one expect. A tool that is widely available on Linux systems is dd. The dd tool, located in /bin copies files by reading one block (size is adjustable) at a time from an input device and writing it to an output device. Blocks are read sequentially from the input device. Here are the relevant options for dd:

bs=BYTES: Read BYTES bytes at a time (= 1 block)
count=BLOCKS: copy only BLocks input blocks
if=FILE: read from FILE instead of stdin
of=FILE: write to FILE instead of stdout
skip=BLOCKS: skip BLocks bs-sized blocks at start of input

We are not interested in the actual data returned from dd, so the output device should be set to /dev/null. The number of blocks to be read depends on the time you would like to run the IO test. The larger the value of the “count” option the longer dd runs. A typical command line for dd looks like this.

```
dd bs=1048576 if=/raw/data_1 of=/dev/null count=1000
```

In order to emulate the disk IO behavior of Oracle with the dd tool, the block size should be set to database block size x multiblock_read_count. A table scan is usually performed in parallel with many processes reading different portions of the table. This behavior can be emulated by running concurrently copies of dd. Be aware that if multiple copies of dd read the same data, they can take advantage of disk or disk array caches. This should be avoided since it distorts from the real IO capabilities of the disk array. Most likely Oracle will not take advantage of disk array caches during a table scans. Hence, if there are multiple input devices each dd should be assigned different input device. If this is not possible, one can assign each dd a portion of an input device by assigning an offset into the file using the skip option (see above). For instance if there is only one large device of 1GB and one would like to run 5 copies of dd, one can divide the device into 5 parts of 200MB each and use the following command lines to start 5 copies:

```
dd bs=1048576 count=200 if=/raw/data_1 of=/dev/null
dd bs=1048576 count=200 skip=200 if=/raw/data_1 of=/dev/null
dd bs=1048576 count=200 skip=400 if=/raw/data_1 of=/dev/null
dd bs=1048576 count=200 skip=600 if=/raw/data_1 of=/dev/null
dd bs=1048576 count=200 skip=800 if=/raw/data_1 of=/dev/null
```

Each of the dd commands above reads 200MB, performing 200 IOs of 1MB each. The first dd starts at block 1, the second at block 200, the third at block 400, etc.

The maxim throughput per node should be measured first, before measuring the throughput of the entire system. The disk throughput can be measured with many different tools, including vmstat. One can use the following command to measure the throughput:

```
dd bs=1048576 if=/raw/data_1 of=/dev/null
```
vmstat 3

3 is the number of seconds between the vmstat snapshots. The time between snapshot can be varied. However, smaller intervals than 3 seconds can yield inaccurate results since, if the system is cpu bound, the vmstat process might get scheduled at irregular times yielding incorrectly reported spikes in throughput.

The output of this command will appear similar to this:

```
procs                      memory      swap          io     system         cpu
r  b   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy id wa
3  0      0 6431196  56632 1317684    0    0 463397    23 2603   790 32  4 64  0
4  0      0 6431196  56632 1317684    0    0 441733    11 2612   837 31  3 66  0
3  0      0 6431048  56632 1317684    0    0 447437     5 2504   736 32  3 64  0
```

The column of interest is bi (blocks in). The unit of measure for this column is blocks, which are usually 1 MB. To measure the throughput of one node, the number of concurrent copies of `dd` should be increased on this node incrementally until there is no increase in disk throughput. The test should be repeated by increasing the number of concurrently executing copies of `dd` on each node in the system to measure the total system throughput.

One should not be surprised if the maximum throughput achieved on the single node test is higher than the maximum throughput achieved on each of the nodes in the multi-node test. This is because adding more nodes to a system adds more contention in the switch and storage array. However, the closer the per node throughput comes to the single node throughput, the better balanced the system is.

The throughput measured with a `dd` test will serve as a target IO rate that can be achieved with Oracle. The IO rate achievable with Oracle should be about 85% of the IO rate achievable with `dd`, because Oracle is doing other operations in executing the query besides pure IO's. Figure 8 shows the result of throughput test with `dd` and Oracle. In this test we vary the number of `dds` from 1 to 9. As the graph shows the disk throughput increases in steps of about 80MB/s for the first 6 `dds` before it flattens out at a maximum of about 480 MB/s. In the test with Oracle we increased the degree of parallelism from 1 to 9 of a table that is striped across as many disk as the raw devices of the `dd` test. As one can see performance follows very closely that of the `dd` test. However, the total throughput flattens out at about 420 MB/s, at about 85% of the maximal `dd` throughput.
Disk Throughput

![Disk Throughput graph](image)

Figure 8: dd and Oracle IO test

Oracle Configuration

Asynchronous IO

Asynchronous IO can dramatically increase IO throughput. Instead of issuing one IO at a time and waiting for it to return as in synchronous IO, Oracle issues multiple IOs. While the IOs are returning from the disk subsystem, the Oracle process can work on other tasks, such as computing aggregates or performing join operations. Currently Oracle defaults to synchronous IO. On Linux asynchronous I/O can be used on RAWIO, EXT2, EXT3, NFS, REISERFS filesystem.4

By default, the DISKASYNCHIO parameter in the parameter file (init.ora) is set to `true` to enable asynchronous I/O on raw devices. If you are running an Oracle database on a system that supports kernel asynchronous I/O and that is certified by Oracle to use asynchronous I/O, perform the following steps to enable asynchronous I/O support:

As the Oracle user, change directory to the `$ORACLE_HOME/rdbms/lib` directory and enter the following commands:

```
make -f ins_rdbms.mk async_on
make -f ins_rdbms.mk oracle
```

If you receive the "/usr/bin/ld: cannot find -laio" error, then the system does not support kernel asynchronous I/O and you must enter the following command to restore the Oracle instance to a usable state: `make -f ins_rdbms.mk async_off`

4 On Linux, Automatic Storage Manager (ASM) uses asynchronous I/O by default.
Parameters in the init.ora file need to be changed, add following lines to the appropriate init.ora file:

Parameter settings in init.ora or spfile.ora for raw devices:

disk_asynch_io=true (default value is true)

Parameter settings in init.ora file or spfile.ora for filesystem files:

disk_asynch_io=true
filesystemio_options=asynch

It is possible to check if Oracle is issuing asynchronous IO’s by tracing a reader parallel server. For instance, while performing a full table scan with the following query:

```
SELECT /*+ full(store_sales) */
count(*)
FROM   store_sales;
```

Execute the following command:
```
strace -p `ps -ef | grep p001 | awk '{print $1}'`
```

Among other systems calls you will see either:

a. synchronous calls

```
read(3, "783|37567799|213483944|2004-10-1"..., 4096) = 4096
```

b. asynchronous calls

```
io_submit(0xb6a93000, 0x1, 0xbfff236c)  = 1
nio_submit(0xb6a93000, 0x1, 0xbfff236c)  = 1
io_getevents(0xb6a93000, 0x1, 0x400, 0xbffeb794, 0xbffeb774) = 2
```

If you enable the Oracle binary for asynchronous IOs and you turn asynchronous IOs on as indicated above on a system that does not support asynchronous IOs, performance might degrade substantially because for each IO Oracle will first try to issue a asynchronous call. After realizing that this calls fails Oracle issues a synchronous call.

For optimal performance one should also issue the following commands to increase the values for two system parameters:
```
echo 1048576 > /proc/sys/fs/aio-max-size
```

add "fs.aio-max-size=1048576" to /etc/sysctl.conf so it is set after every reboot
Memory

One of the largest concerns when implementing a database system on 32-bit Linux has always been how much memory could be addressed. Oracle 10g out of the box can only have a 1.7GB of shared memory for its SGA on a generic 2.4.x 32-bit Linux kernel. However, it is possible to increase this size, using one of the following options: change the memory mapped_base, using a shared memory file system or implement hugetlbfs. For more information about these option please see Appendix C of the *Database Administrator's Reference 10g Release 1 (10.1) for UNIX Systems: AIX-Based Systems, hp HP-UX PA-RISC (64-bit), hp Tru64 UNIX, Linux x86, and Solaris Operating System (SPARC)*.

However most data warehouses don’t need a large SGA. Data warehouse applications are dominated by queries scanning a lot of data, performing join, large sort and bitmap operations at a high degree of parallelism. Memory for hash joins and sort operations is allocated out of the PGA (Program Global Area), not the SGA. PGA memory is not bound by the 1.7GB SGA. A 4 CPU system running a degree of parallelism of 8 uses typically less than 3.2GB of PGA. To tune the PGA please refer to the *Oracle® Database Performance Tuning Guide 10g Release 1 (10.1)*.

Data Compression

So far we have talked about increasing physical throughput, that is, how to increase the throughput of bits through the various pieces of hardware. However, if the hardware configuration is set up properly to maximize IO performance, yet the desired performance cannot be achieved one can consider other techniques to minimize the database’s IO requirements. The most important feature is table compression.

By increasing the number of rows per block, table compression can increase table-scan performance beyond hardware limits. For instance, let’s assume that without compression on average 200 rows fit into one block. With a compression ratio of 50 percent (which is not very much as we have shown in many papers), the number of rows per block increases by 100 to 300 rows. On this compressed table, the database is now reading 50% more rows with same number of IO’s. The overhead of reading a compressed row compared to a non compressed row is virtual zero since decompressing only requires one more pointer lookup in memory.

Table compression can also significantly reduce disk and buffer cache requirements for database tables. If a table is defined “compressed” it will use fewer data blocks on disk, thereby, reducing disk space requirements. Data from a compressed table is read and cached in its compressed format and it is decompressed only at data access time. Because data is cached in its compressed form, significantly more data can fit into the same amount of buffer cache (see Figure 9).
The space savings (SS) are therefore defined as:

\[
SS = \frac{\#_{\text{non-compressed blocks}} - \#_{\text{compressed blocks}}}{\#_{\text{non-compressed blocks}}} \times 100
\]

The compression factor mostly depends on the data content at the block level. Unique fields, (fields with a high cardinality) such as primary keys cannot be compressed, whereas fields with a very low cardinality can be compressed very well. On the other hand, longer fields yield a larger compression factor since the space saving is larger than for shorter fields. Additionally, if a sequence of columns contains the same content, the compression algorithm combines those columns into a multi column compression element for even better compression results. In most cases, larger block sizes increase the compression factor for a database table as more column values can be linked to the same symbol table.

**Tuning the Interconnect Protocols**

The Interconnect is the network that connects individual machines to form a cluster. When talking about the Interconnect we refer to both, the actual wire and the protocol that runs over this wire. On Linux, Oracle supports Ethernet (Fast Ethernet, GigE) and Infiniband as wires connecting machines. The protocols that can be run on Ethernet are TCP/IP and UDP. On Infiniband one can run TPC/IP, UDP and uDAPL. The performance of Fast Ethernet is about 10MB/s while GigE is about 100MB/s and Infiniband is about 1GB/s. The protocols, too, have different performance characteristics. TPC/IP being the easiest and most commonly used protocol in today’s IT infrastructure is also the slowest protocol with the largest CPU overhead. UDP has been available for quite some time and has less CPU overhead. uDAPL being the latest protocol available with Oracle has the least CPU overhead.

In general the Interconnect is being used for three different purposes in Oracle systems. As part of the Oracle’s Cluster Ready Services (CRS) each node periodically sends a heartbeat to any other node notifying them that it is still alive. It is necessary that these heartbeats reach all other nodes because otherwise the cluster might fall apart (split brain). This is why Oracle recommends designating a dedicated network for the Interconnect, so that the heartbeat messages are reliably transmitted.

The Interconnect is also being used for cache fusion. Cache Fusion is used when accessing the buffer cache in a RAC environment. With Cache Fusion a block that is requested by node A and currently cached on Node B is sent directly using the Interconnect from Node B to Node A.

The largest demand for interconnect traffic in data warehouse applications comes from inter process communication (IPC). When performing join, aggregation or load operations involving multiple nodes it might be necessary to re-distribute data.
and send control messages from one node to another. Processes, which are also called Parallel Servers, communicate with each other using the Interconnect. The amount of interconnect traffic depends on the operation and the number of nodes participating in the operation. Join operations tend to utilize the interconnect traffic more than simple aggregations because of possible communication between Parallel Servers. The amount of interconnect traffic can vary significantly depending on the distribution method. Which distribution method is used can be found in the PQ Distrib column of the query plan. Cases where one side of the join is broadcasted or both sides are hash-distributed result in the largest interconnect traffic. Partial Partition Wise Joins in which only one side of the join is redistributed result in less interconnect traffic, while Full Partition Wise Joins in which no side needs to be redistributed result in the least interconnect traffic.

The amount of interconnect traffic also depends on how many nodes participate in a join operation. The more nodes participate in a join operation the more data needs to be distributed to remote nodes. For instance in a 4-node RAC cluster with 4 CPU each to maximize load performance with external tables one needs to set the DOP to 32 on both the external and internal tables. This will result in 8 Parallel Servers performing read operations from the external table on each node as well as 8 Parallel Servers performing table creation statements on each node. On the other hand if there are 4 users on average on the systems issuing queries, it is very likely that each user’s query runs locally on one node reducing the number of remote data distribution to zero.

**Specifying the Network for IPC**

The network being used for IPC can be specified in the node specific init.ora file with the `cluster_interconnects` parameter. This parameter specifies the network interface address that all the other nodes will use to communicate to this node.

**Specifying the Interconnect Protocol**

At node startup time a message is printed in the alert.log file indicating which interconnect protocol is being used. For instance for uDAPL the line looks like this:

```
cluster interconnect IPC version: Oracle UDAPL ...
```

Switching between interconnect protocols is easy. Assuming the Interconnect Protocol you wish to switch to is supported on the current wire. You will need to shut down any Oracle instance before performing the following operations:

```
cd $ORACLE_HOME/rdbms/lib
make -f ins_rdbms.mk ipc_udp  (for UDP)  or
make -f ins_rdbms.mk ipc_tcp  (for TPC/IP)  or
make -f ins_rdbms.mk ipc_ib   (for uDAPL)
```
The expert user can also manually perform the transition from one interconnect protocol to another. The only difference in the environment is the interconnect library `libskgx10.so`. The makefile only removes the current `libskgx10.so` and copies the desired library into `libskgx10.so`.

<table>
<thead>
<tr>
<th>Library Name</th>
<th>Interconnect Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>libskgx10.so</td>
<td>The one being linked in by Oracle</td>
</tr>
<tr>
<td>libskgxpd.so</td>
<td>Dummy, no interconnect protocol (for single instance)</td>
</tr>
<tr>
<td>libskgxpu.so</td>
<td>UDP</td>
</tr>
<tr>
<td>libskgxpt.so</td>
<td>TCP</td>
</tr>
</tbody>
</table>

**Tuning the Interconnect Protocols**

In order to minimize the overhead of every message sent across the Interconnect, the message size should be set to a large value. Depending on the Interconnect protocol message sizes up to 64k are allowed. However, the larger the message size the more memory being used to buffer messages. Also, very large messages sizes tend to overemphasize the overhead that originates from sending control messages with the large message size. Tests for data warehouse benchmarks have shown that message sizes between 16k and 32k perform best.

All components of the hardware configuration should support Jumbo frames (NIC’s, switches and storage). Larger frame sizes reduce server overhead and increase throughput. Standard Ethernet frames sizes have been 1500 bytes since it was created in 1980. Jumbo frames extends Ethernet frame sizes to 9000 bytes. It is also possible to use Jumbo Frames with UDP over IP. However jumbo frames cannot be used at all unless every component can support them.

In case of UDP it is possible to use multiple networks for IPC traffic. Simply add multiple IP addresses to the `cluster_interconnect` parameter separated by “;” and Oracle will load balance between the different IP addresses. Furthermore it is recommended to tune the send and receive buffers to 256k.

In case of UDP over Infiniband it is recommended to add the following OS parameters to tune UDP traffic.

In `/etc/modules.conf`

Specify options: `ipoib IpoibXmitBuffers=100`  
`IpoibRecvBuffers=2000`
Conclusion

For a Grid based solution to be successful it has to be able to achieve the same level of performance and reliability as the SMP system it is replacing and be able to do this at a lower total cost of ownership.

This paper outlines the four main steps that to ensure you build a reliable and performant Grid based solution.

- Determine the required throughput for the system
- Build the grid to meet the require throughput remembering it is only as fast as it’s slowest element.
- Using operating system utilities such as dd to test that the grid can deliverer the required throughput.
- Configure Oracle

Using an example of a SAN storage solution this paper guides the reader through each of the above steps and provides hands-on examples for building a successful Linux RAC system.