Oracle Real Application Clusters and Industry Trends in Cluster Parallelism and Availability

James Hamilton, JamesRH@microsoft.com
Architect, Microsoft SQL Server

December 2004

1. Introduction

The Oracle Real Application Cluster (RAC) feature is frequently offered as a potential solution to many of the problems faced by those deploying mission critical, data-centric applications. As is the case with most important discussions, this one is being driven by many factors, some technical and some not, but all relevant to those making a database (DB) technology decision. This paper explores some of the issues driving the discussions surrounding RAC, investigates alternative technologies, and discusses some related trends within the database management system (DBMS) development community.

Before continuing, let's review the author's background and biases since they have some bearing on the discussion that follows. James Hamilton is an architect on the Microsoft SQL Server development team, where he has led many teams within the server including the SQL Language Compiler, Query Optimizer, Query Execution Engine, DDL, metadata, and Catalog management groups, Security, XML, Client and networking protocol teams, the Full-Text Search development group, and the Common Language Runtime (CLR) integration effort. Prior to joining SQL Server, James spent 11 years at IBM where he was Lead Architect on DB2 UDB, helping to ship many releases. Having helped customers successfully deploy DB2 Parallel Edition (now a feature of DB2 Enterprise Edition), some will correctly point out that the author has some biases towards shared nothing DB parallelism over shared disk-based solutions, which is the core technology on which RAC is based. Certainly some of that concern is well founded, but fortunately, this topic is much more complex than one that can be resolved through simple architectural biases and preferences. All potential parallel clustering technologies have both advantages and disadvantages, and each has successful deployment examples that can be referenced. This paper will focus on what problems are being addressed by RAC, the alternatives available from Oracle and its competitors, and compare and contrast the different approaches.

The RAC solution, some aspects of its implementation, its brand name, and how it is sold have evolved in the decade since the technology was originally conceived as Oracle Parallel Server. [MORL02, YDNR02] 10G RAC is aimed squarely at solving two of the most important problems facing those deploying data-centric applications: 1) application availability and 2) affordable performance. These requirements are important to data management customers, and consequently, Oracle and the industry as a whole offer features to address these requirements based on application design, requirements, and the deployment scenario.

2. Application Availability

Before looking at the more exotic availability techniques, it's worth first reviewing a few core engineering tenets. Application availability, especially when focusing on unplanned downtime, comes from a few general principles applied to all aspects of overall system design: simplicity, redundancy, and isolation.

Simplicity

Any product or feature designed to improve overall application availability needs to be simple to administer since operations and administrative error remains, by many measures, the single largest source of application downtime. For example, a recent Oracle RAC white paper reported that administrative error was the driver of 36% of the unplanned downtime experienced by a typical server side system [ORCL02]. David Patterson confirmed this result in his study, *A Simple Way to Estimate the Cost of Downtime*, where he found that human error was responsible for 53% of the downtime [PATT01].

System complexity leads to administrative errors, and even where errors aren't the eventual result, the database administrator (DBA) time consumed by excessive system complexity is often substantial. These day-to-day fire fights are often what prevent an administrative team from being able to focus more on long term planning and improving the overall operations infrastructure. On this last point, that of administrative

complexity, there is considerable variability between the major DBMS providers, and this is worth studying closely. However, ignoring these variations between competitors and just looking at the Oracle offerings, RAC is far more complicated than their single node offering [FORE02]. A long time Oracle consultant reports that a typical, well managed, single node Oracle system can easily achieve 99.9% reliability whereas the same application workload running on a two node Oracle RAC, with the same level of administrative investment, will typically achieve lower availability, often as low as 98% [YDNR02].

There is much room for debate on these reported results since they are all a product of the workload and the quality of the DBAs managing the respective systems. However, there is little debate that complexity either consumes high quality DBA time or leads to availability problems, and often both.

Operational and administrative simplicity is the strongest driver of application availability.

Redundancy

Components will fail, and there must be sufficient redundancy throughout the system to survive these inevitable component failures in hardware, software, and administration. Any highly reliable system must be composed of redundant components since avoiding single points of failure is the only reliable way to achieve application availability in the presence of inevitable failures. Hardware and software components will fail and, without sufficient redundancy in the system, component failure yields to loss of system availability.

Over the last 15 years, considerable agreement has emerged on the best technology for database data redundancy: log shipping among independent database nodes. IBM IMS Remote Site Recovery was one of the first products, to implement this technique. The first public description of it known by the author was by Burkes and Treiber at the 1989 High Performance Transaction Systems Workshop in Asilomar, CA. [BURK89]. Oracle, DB2, and SQL Server all have similar, log shipping-based solutions with Oracle offering DataGuard [ORCL01], IBM providing DB2 Log Shipping [DB203], and Microsoft offering SQL Server 2000 Log Shipping [SQLS01] and the SQL Server 2005 Database Mirroring feature [SQLS02]. Database Mirroring will be used as an example to discuss some of the advantages of this approach to database availability. Looking at Figure 1, it can be seen that with Database Mirroring (log shipping), the primary node and the secondary node share no resources with the only connections between the two systems being the low-level transaction log format. It should be noted that logging and recovery, the components that interact with the transaction log, are the most tested and trusted code paths in a relational database system.

In a log shipping configuration, the two systems are maintaining 100% independent hardware, software, and copies of the data, and failure of any hardware or software component will not render the data unavailable.

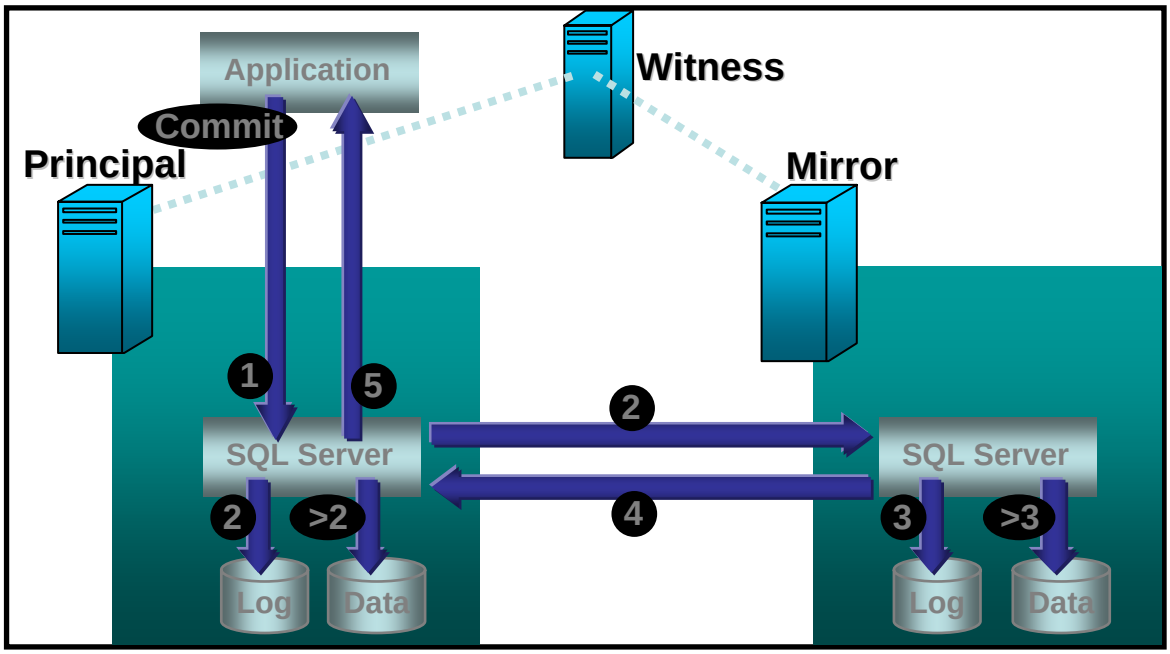

Figure 1 Database Mirroring

The Oracle RAC system (Figure 2) below lacks the degree of redundancy found in the log shipping design shown above and all nodes in the cluster share the storage sub-system. With RAC, the database compute nodes are redundant and interchangeable which is a good thing. However, there is only one copy of the database itself which is to say that if part of the database gets damaged, data will become unavailable and the entire cluster may be brought down.

Data redundancy is easy to achieve in any RAC deployment by using a RAID-based storage subsystem, but this only provides protection against physical storage system failures. If there is corruption, data damage or loss of data integrity logically "above" the storage subsystem, the SAN will dutifully redundantly store this damaged data in multiple copies and the data will no longer be available to any node in the cluster. However, with a log shipping solution such as DataGuard or Database Mirroring, the data is protected against these failures since the redundant copy is made logically above the storage subsystem, the HBA, the device drivers, and the operating system.

The key point to keep in mind is that a SAN can redundantly store multiple copies of the data that it receives but can provide little protection against failures of other software and hardware components in database, operating system, and storage stack.

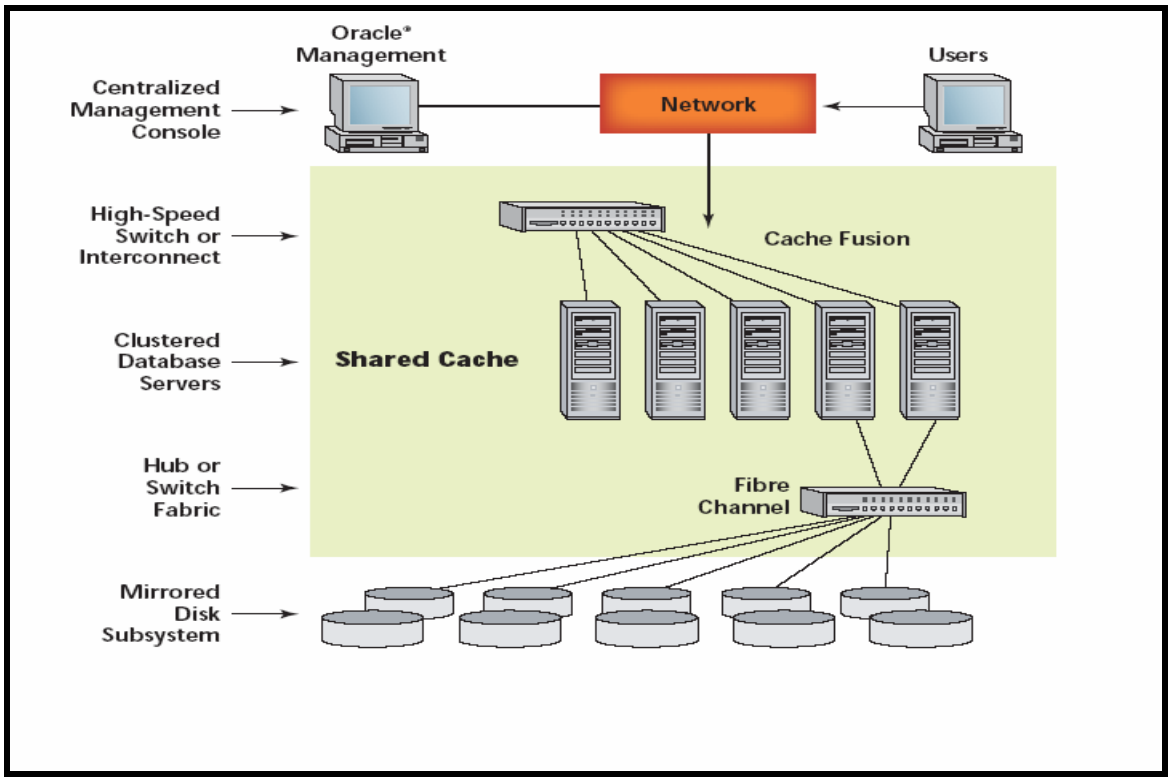

Figure 2 Oracle RAC architecture [INTL02]

In summary, an Oracle RAC configuration has compute node redundancy. (operating DB nodes can take over from those that fail.) It typically is deployed with physical database redundancy by using a RAID-based storage subsystem. There is, however, only a single logical copy of the database, and this is the most important resource in the system. In the next section on isolation, the failure modes that would render the data unavailable in RAC due to this single point of failure are looked at in more detail and these single failure points are what makes us uncomfortable in depending upon a shared, concurrent access disk subsystem to achieve high availability in applications requiring very high degrees of reliability. SAN systems provide excellent administrative characteristics and good quality data protection but they are not a full solution for database availability. Those customers deploying RAC alone without augmenting the solution with Data Guard or other availability techniques leaves the system unprotected against a broad class of increasingly common system failures.

The near complete redundancy of log shipping is available from all major DBMS providers including Oracle, and it's a more appropriate availability solution.

Isolation

Looking at Figure 1, it can be seen that single points of failure are avoided in the log shipping-based solution. The two database systems don't share any hardware or software components in the storage stack and only communicate via the transaction log being shipped between them. This is one of the key differentiators of log shipping from shared disk-based solution like RAC, which shares a single storage subsystem among all the attached database nodes (Figure 2).

A few examples illustrate the negative availability tradeoffs inherent in any availability design dependent upon a disk subsystem concurrently shared by all DB nodes (serial, non-concurrent sharing of a SAN storage fabric by multiple nodes does not suffer from most of these issues). Concurrent, shared disk access from multiple nodes can allow a single Heisenbug (a rare, timing dependent fault in the DBMS storage engine) that damages a database page to irreparably damage all copies of this page stored on the shared RAID device. Since this resource is shared by all nodes in the cluster, all DBMS nodes will lose access to the data on this page at the same time. And, since this fault happened above the disk subsystem level, the

RAID subsystem will have potentially created multiple copies of this damaged database page. All the copies in the storage subsystem will now have the same page corruption, the entire cluster has lost availability to this data, and administrative intervention with the possibility of extended downtime is the likely outcome. Fortunately, storage engine bugs like the one described here are rare. Moving down the stack will show the risk of failure climbing to increasingly uncomfortable levels. For example, consider a situation where the DBMS node successfully writes out the page, but it gets damaged by any of the millions of lines of operating system code between the DBMS and the storage system device driver. In this case, the outcome is the same: all copies of the data are damaged and all nodes lose access to the damaged page.

Assuming that neither the operating system nor the database suffers a fault, there are still many opportunities for catastrophic data damage and consequent loss of availability in systems such as RAC dependent upon concurrent, shared access to storage. The major SAN providers now develop, maintain, and enhance in excess of a million lines of storage subsystem microcode. This is to say that the storage hardware is now mostly software, and this subsystem is no more immune from failure than any other component of the system including the DBMS. Consider the possibility that the SAN controller or its firmware fails, the SAN experiences any form of unrecoverable communications loss, or the disk subsystem hardware or software experiences a serious software bug. Note that the majority of SAN providers employ a write-in-place buffer management policy which leaves the system open to lost page integrity in the presence of partial writes caused by issues in the storage network fabric and some parts of the SAN system. Any of these events can again yield the same result: catastrophic loss of data availability to both the primary compute node and in all secondary nodes across the RAC cluster. This is an unmasked error where recovery will require non-trivial downtime, and if it's a recurring failure, it may be very difficult and time consuming to isolate and correct.

RAC is sufficiently complex both internally and from an operations perspective that unusual events and error conditions can bring down the entire cluster. Fortunately, these cluster-wide failures are uncommon, but when they occur, problem determination can be very difficult, often requiring special skills and hours (or even days) to fully work through. On many failures, the root cause may never be found [YDNR02].

Isolation and fault containment is a required component in any highly available system.

Availability Summary:
The original design point for RAC when the technology was first conceived and implemented nearly a decade ago [MORL02] was multi-node scalability. So it should not be surprising that RAC suffers from single points of failure that make it a poor choice as a primary availability mechanism. It simply wasn't the design point. RAC is a scalability feature and is most applicable when application performance characteristics require a multi-node cluster DBMS. It is not the best technology choice to achieve the availability goals of high value applications where availability is a primary system design point. Oracle and other DBMS providers offer better choices and they should be used.

3. Affordable Performance

Performance and database scalability will always be a dominant issue for data-centric applications. Database scalability remains a core application developer concern since synchronous persistent state scaling is so difficult (many applications relax full ACID semantics to make this issue easier to address). The ability to scale applications is bounded by our ability to scale the management of the shared application state. If the application was truly stateless, or if the application state was completely non-shared, linear application scaling could be easily achieved. To scale such an application, one would merely add more instances of it. Unfortunately, most useful applications have considerable state and much of this state needs to be shared between different instances of the application for those that can support multiple application instances. This shared state is usually stored in relational database management systems (RDBMS), and therefore it is DBMS scaling that poses some of the largest challenges for application designers.

Application designers employ many tricks and techniques to reduce the database bottleneck, but these techniques require effort, they tend to constrain how the application can be written, and they can potentially bring additional administrative and/or operational costs. As a consequence, there has always been a strong desire to depend upon a DBMS infrastructure provider to solve this problem, freeing the application developer to focus on the application rather than upon cluster parallelism.

These issues in scalable performance are being addressed industry-wide, from two separate directions:

1. Hardware advancements and database scaling improvements.
2. Mid-tier caching and scaling.

3.1 Hardware Advancements and Database Scaling Improvements

Hardware continues to advance quickly, and single-node database performance has been improving at roughly a Moore's law pace over the last decade. In 1994, the best single-node TPC-C performance in the industry was 1,470 tpmC (IBM RS6000 PowerServer R24), whereas all three major database providers are now able to produce performance results of nearly 1000 times this level a decade later. At the same time that an almost three order of magnitude improvement in performance has been seen, a corresponding reduction in the cost of each transaction by nearly three orders of magnitude has also been seen. Ten years ago, the leading system, an IBM product, was running $666.12/tpmC, whereas the best results today are grouped around $5/tpmC.

Hardware advances continue at this pace, and Intel expects that this will continue for more than another decade [INTL03].

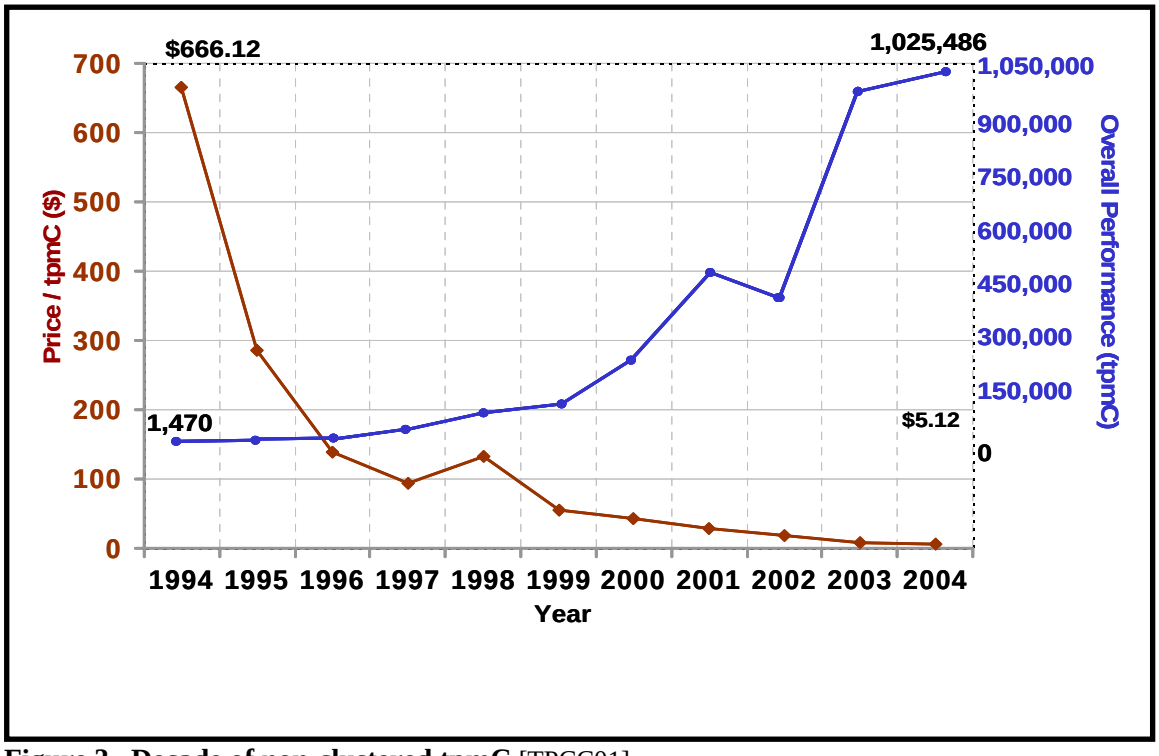

Figure 3 Decade of non-clustered tpmC [TPCC01]

In the discussion above, TPC-C results were employed only because the data is widely available and the results are comparable between the large database management company offerings. However, most customer applications that the author has worked with are clearly far more complex than TPC-C. Nonetheless, there is no debate that the combination of hardware advancements and improvements in

DBMS technology from all the major providers has led to impressive improvements over the last decade. For example, the author helped deliver a 256 node shared nothing database system to a customer in the mid 90's, and in a recent meeting with this same customer, that monster was compared against a modern single node system. In all dimensions except floor space, weight, and power consumed, it was far less capable than a contemporary single node system available.

The conclusion drawn from this is the majority of OLTP workloads can be run successfully on high scale, single node systems, but this isn't an argument that all workloads need to be hosted that way. There are other factors at play, the most interesting of which centers around higher scale systems, which tend to cost disproportionately more than multiple lower scale systems. That is to say, eight 4-ways are typically cheaper than a single 32-way. So, although single node systems are often able to support the workload, there remain good arguments in favor of clustered deployments. The argument here is: "a single node *could* host the entire workload at acceptable performance levels, but the hardware required to host it is more expensive than the commodity priced hardware that could be used if a cluster-based DBMS architecture was employed." Outside of high-end decision support and real time data warehousing workloads, the majority of applications can be hosted on a single system but, since these mainframe-class systems are relatively expensive, most of us in the industry focus the discussion on "affordable performance" which is really a more meaningful discussion from an engineering perspective. In reality, it is total solution cost that matters the most to customers.

There are two reasons to consider single system image DBMS clusters of which Oracle RAC is one example: 1) the workload can't be hosted on a single node DBMS, and 2) it may be less expensive to use a commodity multi-node system. Unfortunately, here as in many technology choices, there is no clear answer.

Looking at Figure 4, it can be seen that administrative and operational costs tend to dominate the hardware and software costs. This is becoming increasingly apparent each year with hardware costs falling, software costs ranging from constant to falling, system complexity rising, and people costs increasing.

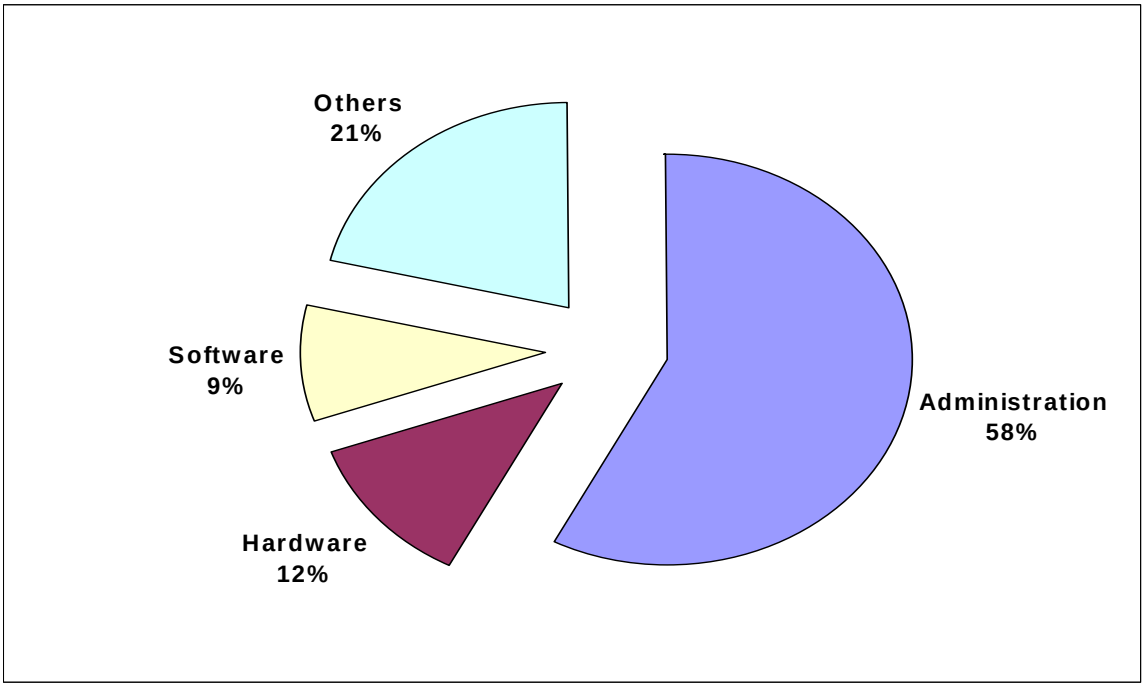

Figure 4 IT spending by category [TCOS01]

Administrative and operational costs continue to dominate, and this trend is expected to continue if not accelerate over the next decade. Two conclusions fall out from this data: 1) complexity should be avoided, and 2) hardware costs are falling and are becoming an ever smaller component in the cost of deploying a data-intensive application.

It is a bit cliché but it remains as true today as when it was coined some years back: application workloads that can be hosted on a single system, should be. Nonetheless, there are times when the DBMS needs to be scaled over multiple systems. There are at least two fundamental approaches available to scale DBMS workloads: 1) depend upon a single system image, cluster-based DBMS such as Oracle RAC or 2) employ one or both mid-tier caching and data partitioning. Delegating the heavy lifting to the DBMS provider has an obvious appeal in that it sounds simple but it is rarely that simple. This paper will investigate these options in more detail, discuss the potential costs and engineering trade-offs that may be required when selecting a cluster-based DBMS, and contrast these against some of the advantages and disadvantages of the caching and data partitioning approach.

Cluster DBMS license premium: There are high costs associated with implementing a cluster-based DBMS architecture. The clearest summary that the author has seen is the following:

> *Oracle Enterprise Edition costs US$40.000,- per cpu or US$800,- per named user plus (NUP), as it's called now. RAC costs 50% on top of that, which means US$60.000,- and US$1200,- per cpu or per NUP.* [YDNR02]

In addition to the original purchase price, there is an annual maintenance charge. So, although RAC can enable the use of commodity hardware, this is only achievable by substantially growing the software licensing bill. It's a bit ironic to have to pay US$60,000 per CPU to be able to use lower cost, commodity hardware.

Performance impact: James Morle of Scale Abilities investigated the overhead of running a RAC system in his paper *Unbreakable* [MORL02]. In this paper, he benchmarks an order entry application running under non-RAC Oracle and the same workload under a single node RAC deployment on the same hardware. What he found was an 18% overhead in moving to RAC running the same workload on exactly the same hardware. A well known supporter from Oracle in an amusing but informative discussion explains that RAC is like an amplifier. [ASKTOM04] What he means is that RAC makes a bad application much worse, but can also make a good application better. Tom goes on to say:

> I*'ve seen developers say "well, if the shared pool is the problem because we didn't bind, we'll use RAC -- we'll have two shared pools, problem solved". bzzzt -- you have two shared pools to keep consistent with each other dependency wise now, hard parsing -- if it was causing a problem in a single node will just get worse in multi-node.*

Morel makes a similar observation [MORL02] in noting that "you should bank on putting considerable thought into configuration, tuning, and operation of the cluster."

Many applications *can* run on RAC without change, but most need application investment and tuning to get acceptable performance. This isn't really a RAC problem so much as a general reality with clusters: getting an application to run well on a cluster, no matter which DBMS produced it, is typically going to require application development investment. Admittedly, this is in conflict with some of the marketing literature, but it is absolutely consistent with the experiences of those who have done multiple cluster deployments.

Non-commodity hardware: RAC depends upon shared disk support. This is technically possible to implement using multi-initiator SCSI, but in the words of "Oracle RAC Best Practices on Linux" [ORCL03], you actually can run RAC

> *…with a dual-ported shared SCSI drive, and a 10 mb/s ethernet network between them. This is instructive in terms of understanding the absolute minimum requirements to allow RAC to function, but is of little use for more than demonstration purposes.*

RAC requires special purpose storage subsystem hardware, and this hardware comes at a premium. As shown in Figure 4 above, total hardware costs are about 12% of the total cost of ownership of the system. Looking just at the hardware costs, it's the disk component that dominates, with disks representing an ever increasing component of the hardware costs. IDC (see Figure 5) reports that 75% of the hardware costs of large database deployments are invested in disk subsystems. Jim Gray reported similar results in the Terraserver with 78% of the original Terraserver hardware costs being the storage subsystem [GRAY04].

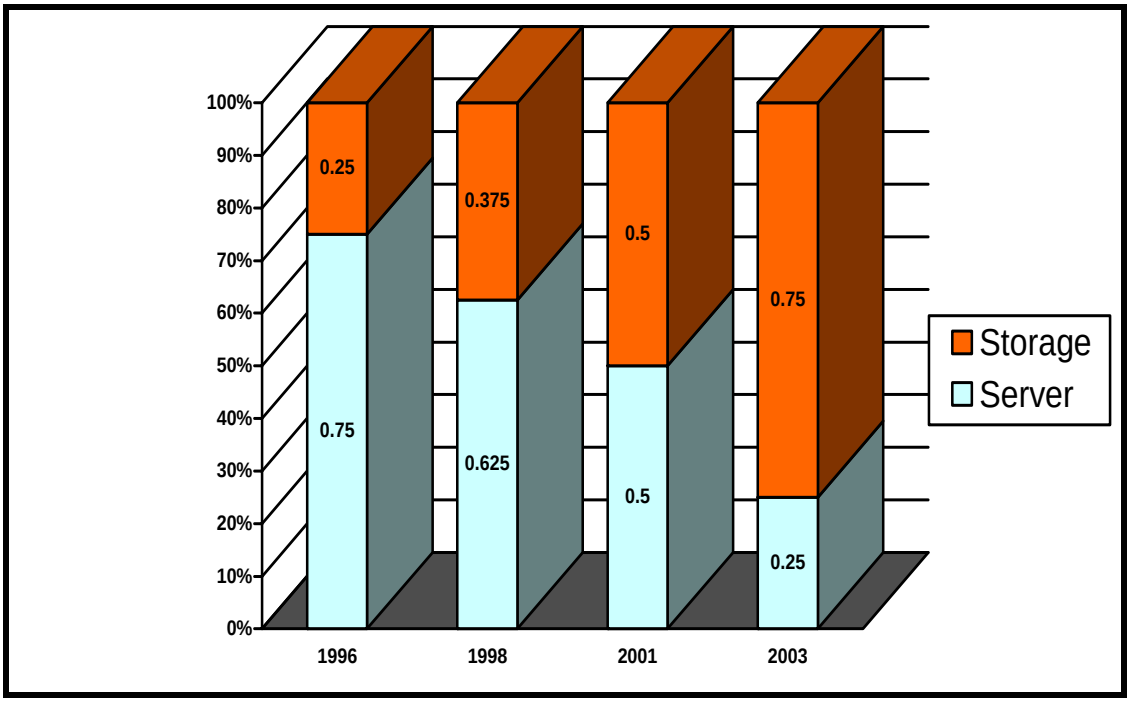

Figure 5 Storage as a percentage of hardware spending [IDC01]

Summarizing these findings, disks form roughly 75% of the hardware expense on large system deployments leaving system hardware (the non-disk hardware component) at only 25% of the hardware expense. Figure 4 shows that the hardware expense is about 12% of the total system cost. So, the disk component of the hardware cost is around 9% and the non-disk component is at 3% of the total system cost. This is to say that RAC is helping to reduce costs through using commodity priced components for 3% of the total system cost. These savings are not invisible, but they do not appear large enough to be the sole driver of a system decision. They need to be weighed against the other costs and constraints incumbent in using RAC.

When considering this data, two factors become quickly clear: 1) the RAC focus on commodity hardware is only addressing 3% of the overall cost problem, and 2) RAC brings the additional software costs discussed in the previous section and additional administrative costs that will be further investigated in the next section.

Administrative penalty and complexity: Operational and administrative costs increase nearly linearly with each computer system added to the solution, which is why operations staffs are increasingly consolidating workloads on fewer systems when availability requirements allow. Smaller scale systems are often less expensive. However, the cost advantage of commodity systems, when weighed against the additional operational costs of more systems in the solution, can end up being a disadvantage for many workloads. This point is further emphasized by findings in the previous section: the largest component of the total system cost is administration (58%) and the smallest component is the systems (non-disk hardware) component (3%).

Where commodity hardware can be used, it should be used. However, all the costs in a given application deployment must be fully understood to ensure that a tightly coupled, multi-node configuration is an overall total cost of ownership improvement. The author has seen workloads that were better hosted on a single system, but has also seen workloads where a cluster was the better choice. The right answer is an application specific one. However, there are other aspects of administrative complexity that need to be considered. The first, and by far the most important, is that 70% of all downtime [GART01] is caused by administrative error. It's easy to assume that more educated administrators are the needed ingredient, but the dominant factor really is system complexity. There is a near linear relationship between system complexity and the risk of administrative error. System complexity is the driver of downtime.

System designers, whether they are DBMS engine developers or application architects responsible for high scale system deployment, develop an increasing fear of complexity as they gain experience. Complexity yields brittle systems that are hard to understand, hard to deploy, and hard to maintain. In fact, one of the key skills that the author looks for when interviewing senior systems designers is a combination of humility and fear. The author is most interested in working with engineers that are fully capable of designing and working with the most complex systems, yet whenever possible, they work hard to avoid it. Complexity, where it can be avoided, should be.

Debugging a tightly coupled, multi-system DBMS can be extraordinarily difficult, and when things go wrong, the problem may not find solution until escalated all the way back to the developers of the system and this can take weeks.

> *I've seen many clusters that just froze for no apparent reason in my time. It's always possible to make the OS or Cluster software dump a trace/log file when it happens.*
> …
> *Then the files (often with sizes measured in GB) are shipped to the vendor and some months later they will report back that it wasn't possible to pinpoint the exact reason for the complete cluster freeze or crash, but that this parameter was probably a bit low and this parameter was probably a bit high.*
>
> *That's what always happens. I have never – really: never – seen a vendor who could correctly diagnose and explain a hanging cluster or a cluster that kept crashing.* [YDNR02]

Oracle RAC is a very complex system software component. One detailed analysis performed by a 10 year Oracle veteran lays this out fairly simply:

> *One way of looking at availability is this: If you have a standalone Unix box it will usually give you 99.9% availability over a year (some say 99.5, some say 99.9). It just runs. And so does Oracle usually. If you have a two-node Unix cluster the availability over a year drops to 98%.*
>
> *Yep, quite surprising, but the two main reasons are that the increased complexity (extra layers of code, extra hardware, etc.) introduced with clusters and RAC will cause some additional downtime – and that it just takes longer to boot a cluster, startup RAC, and perform some other management tasks.* [YDNR02]

Cluster-based DBMS are complex software systems, and as a consequence, require additional care when installing, deploying, upgrading, and debugging. The failure of the Orbitz RAC deployment [EWK04] is an example of a deployment where something went seriously wrong and substantial, fairly heavily publicized downtime was the result. Although the actual cause of the downtime was never fully understood (this is often the case with cluster failures), the solution of moving back to a single node appears to have completely solved the problem and draws some question as to whether the application needed RAC in the first place. In this example, there haven't been major failures reported subsequently. It seems ironic that a single node system could be more available than a multi-node, and clearly this shouldn't be the case. Yet, this pattern is seen repeatedly. Complex systems are more difficult to make robust.

All major database management systems are imperfect, and all suffer occasional failures, so redundancy is important, single points of failure need to be avoided, and most important, complexity must be eliminated from systems that need to be continuously available. If something goes wrong, and it's difficult to debug, the downtime can mount quickly and uncontrollably. It's almost impossible to recover from a week of downtime in the online data processing world.

Summary: In Section 2, availability solutions were investigated, and the conclusion was made that log shipping was a better availability solution that a single system image cluster solution like RAC. Fortunately, all major DBMS providers including Oracle offer this technology. In section 3.1, performance and scaling advances were looked at, and it was concluded that many database workloads can be efficiently supported on a single system. For those workloads that do require multiple DBMS systems, there are at least two options: 1) depend upon a single system image DBMS such as Oracle RAC, or 2) depend upon mid-tier caching and other scaling techniques. In the section above, the costs associated with the tightly coupled, cluster-based solutions like RAC including the license premium, the potential cost impact of commodity hardware, and the administrative complexity that comes from this solution were looked at more deeply.

In the next section, alternatives to scaling DBMS workloads across multiple back-end systems are looked at. What needs to be understood, is 1) whether a multi-node solution is required and 2) if a multi-node solution is required, is a shared disk cluster the best performing, simplest to administer, and least cost solution, or are other techniques more appropriate.

3.2 Mid-Tier Caching and Scaling Techniques

Reflecting on the costs, robustness, and performance constraints implied by depending upon tightly coupled clusters as enumerated above, it is worth fully considering all possible options. Generally, there are at least two main approaches used to scale applications not depending upon tightly coupled DB clusters: 1) mid-tier caching, and 2) data portioning. Both these architectures can be used to scale a database workload over multiple commodity hardware systems to support application scaling while avoiding mainframe cost structures and avoiding the complexity and single points of failure inherent in tightly coupled systems. Further, avoiding the natural tendency to move to a tightly coupled, single system image database management system dramatically decreases the probability that a single administrative mistake, error, or system bug could bring down a substantial portion or the entire system. In fact, it is this increased robustness coupled with the increased administrative flexibility that drives the very highest scale workloads to adopt these techniques.

eBay, for example, uses partitioning with caching for their mission critical OLTP system rather than depending upon RAC, even though they are an Oracle customer and can easily choose to run either the single node or RAC solution.

Let's look at the alternatives more closely.

Alternative architectures - mid-tier caching: This is a common technique, and many applications have been built with custom mid-tier caches. For example, in standard deployments, SAP depends upon a custom mid-tier data caching layer. Other applications build upon main memory caches or main memory databases of which TimesTen is amongst the best known. Many J2EE suppliers offer framework-based solutions as does Microsoft with the .NET Framework. The caching support in the .NET Framework is unique in that SQL Server 2005 is tightly integrated with it, and as monitored data is changed in the database, cache invalidations are sent from the data-tier to the cache keeping it up-to-date. In this approach, the actual queries that were used to load the cache can be registered for notifications in SQL Server. As the data changes, SQL Server sends cache invalidations to the mid-tier. SQL Server 2005 restricts the notifications almost exclusively to the cases where the query result changes. Many techniques track changes at a table granularity rather than those scoped to a specific query within a table, and as a consequence, can over notify. Keeping the over-notification to a minimum and generating notifications

directly from SQL Server where the data change is happening significantly improves the cache invalidation efficiency.

A related approach of DB offload that is equally effective for those workloads where it can be applied is to replicate data from a transactional server to a read-only reporting server.

Caching is a useful tool that can yield extremely robust and scalable application architecture for workloads where the technique is appropriate. In fact, caching is employed by nearly all of the high scale e-commerce sites as one component of their application scaling strategy.

Alternative architectures - partitioned systems: Another approach that is often employed to scale the data storage tier is to partition the database. Most very high scale systems evolve through two forms of partitioning, if they weren't written to be fully partitioned from the beginning. The first form of partitioning is functional partitioning. Systems that have been functionally partitioned separate the data from different components of the application into different databases. For example, the customer information system might be stored in a different database than the billing system. Functional partitioning is typically fairly easy to achieve and is often quite effective so many applications never move beyond this solution.

The second form of partitioning is range-based or hash-based partitioning, and although this requires a little more work to implement, it is highly effective and provides considerable application flexibility once implemented.

Range-based partitioning works by allowing users to break up a large collection of data (usually in a single table) into several smaller, more manageable chunks. This is done by identifying an appropriate partition key and specifying the ranges of data based on that partition key that each chunk would hold. These chunks are called partitions and the full set of partitions is collectively known as a range partitioned table.

The following diagram (Figure 6) provides a simple but very commonly implemented example of range partitioning.

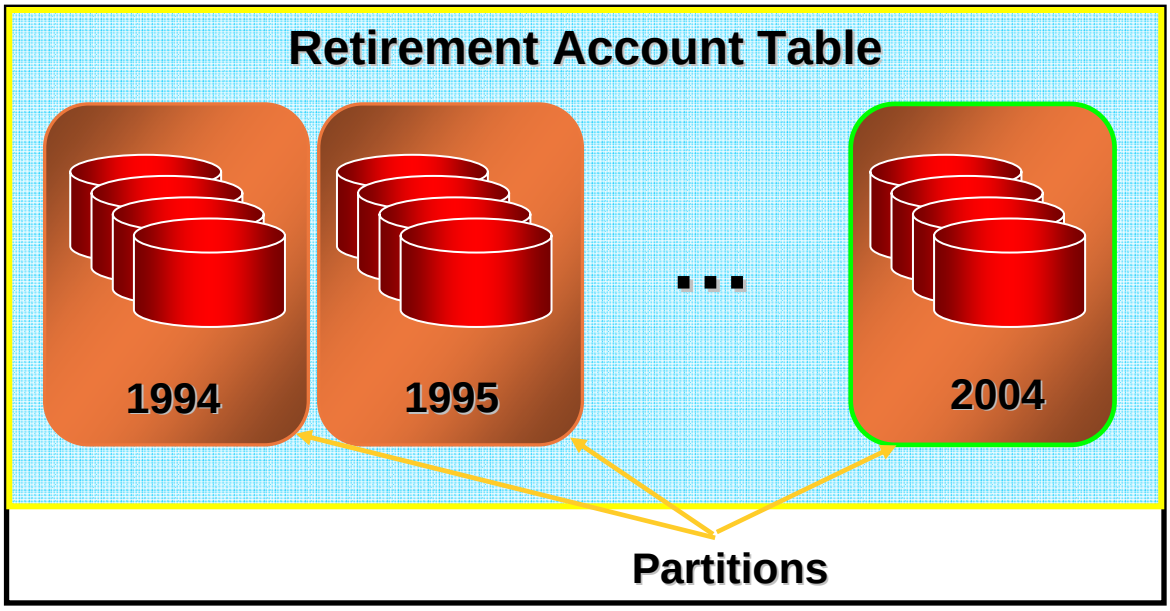

Figure 6 Range-based partitioning

In this example, a single table contains retirement account data stored by a tax agency. This table contains line items for each person in the country that has a retirement account, and each person's account may have many transactions every year based on contribution levels, retirement plans, etc. This is a classic scenario

where range partitioning can be useful. Using year as the partition key, this large table is separated into several smaller partitions, each of which can be managed independently, and each can possibly be stored on independent servers. Maintenance operations can also be performed on partitions independently, hence minimizing any potential performance impact to the system as a whole.

Another form of partitioning that works well with many workloads is hash-based partitioning. This approach is in very common use in high scale e-commerce sites. Hash-based partitioning allows the table to be spread over a potentially large number of database back-ends and has the advantage of tending to smooth out query skew and update hot spots. Using this technique, some data dependent key is selected to be used by the mid-tier to determine which partition is being operated upon. In well written systems, this partitioning information isn't hard coded in the mid-tier and is instead loaded from the DB at mid-tier startup time (see Figure 7).

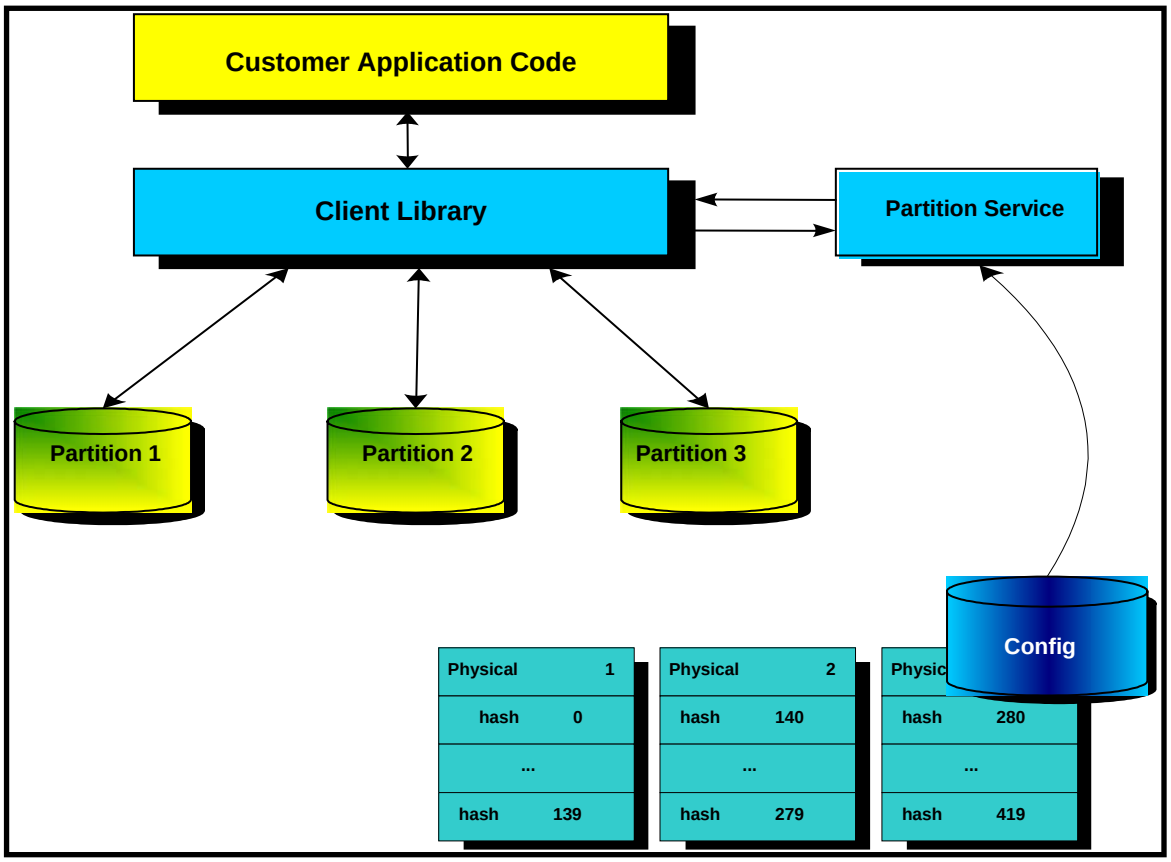

Figure 7 Hash-based partitioning

The model shown in Figure 7 allows a workload to evenly spread over multiple servers, and it allows partition placement reconfiguration by updating the configuration information stored in the DB and reloading the partitioning information into the mid-tiers.

Two further small advances are needed to make this system both dynamic and quickly adaptable. The first is to support online reconfiguration. This way partitions can be brought offline, moved, and brought back online without any loss of system availability. It turns out this can be easily handled by introducing two states for each partition: 1) offline or 2) online with a location. Each partition is registered in the partition DB in one of these two states. Each mid-tier checks for configuration changes at a tunable interval. If they can't load the configuration data, they go offline. If there are no configuration changes (just use a configuration generation number), they do nothing. If there are configuration changes, they update their routing tables. Since each mid-tier is guaranteed to update its configuration data every N minutes, a partition can be brought offline within N minutes. Using this technique, partitions can now be managed

and moved between servers independently. The granule of data movement is still fairly large, however, so one more refinement is still needed.

The problem with the architecture described so far is the following: if you have 8 servers and therefore 8 partitions and decide to update the system from 8 to 10 servers, it's not possible to fully spread the workload over all 10 servers. The solution here is to over-partition as shown in Figure 8.

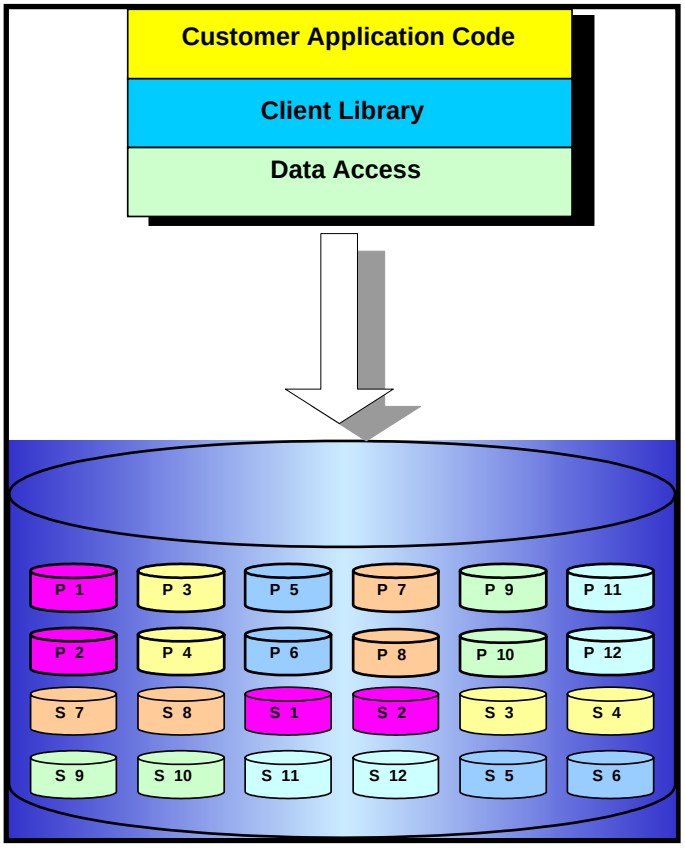

Figure 8: Administrative control through over-partitioning

With over-partitioning, instead of dividing up tables into 8 partitions as was done in our example above, a thousand partitions is used instead. There is nothing magical about using a thousand partitions. Any number will do as long as the number of partitions is much larger than the possible number of servers that could be provisioned. The number of partitions should be sufficiently large that the business impact of bringing a partition offline for maintenance operations is minimal. With over-partitioning, if a new set of servers are added, the data can be quickly and easily redistributed over the new servers. If a system fails, the data can be spread over the remaining servers without a substantial impact on the performance of the remaining servers.

Supporting both hash-based partitioning and range-based partitioning is useful in that hash gives good flat data distribution and range gives some administrative advantages. Both partitioning strategies are useful, although with the customers that the author works with, they often elect to initially only implement hash.

Reporting is implemented by defining views that span all servers and aggregating results across all partitions in each table.

This technique brings a constraint in that cross-server operations need to be minimized. It is wise to choose a partitioning strategy where highly related partitions are stored on the same servers, and there is minimal

cross-server update traffic. Clearly this does constrain the application somewhat, but in return, what is achieved is tremendous administrative flexibility. Given that during the life of most applications, the amount spent on operations and administration is usually orders of magnitude more than that spent in originally writing the application, this is usually a good tradeoff and this is exactly the architecture employed by the highest scale applications in the MSN network.

Using hash-based partitioning with over-partitioning, an application can be hosted initially on one server. If it is popular, it can be hosted without application change across multiple servers. Each server can mutually protect each other using log shipping, so there is no loss of availability on any system, database or hardware failure. Systems can be upgraded independently, one at a time. Rolling upgrades can be done without constraint even between DB versions and even when there are metadata changes between these versions. Because each system has full administrative autonomy, there is complete isolation of failures, administrative errors, etc.

This approach builds upon single system databases rather than their more complex, clustered brethren, and as a consequence, is a more robust solution. Since each database is 100% independent, all potential failures are contained. There exist no failure modes that take down the entire system, and diagnosing problems doesn't require very rare and very expensive clustered database specialists. Nor does it require complex software that costs $60,000 per CPU.

It turns out that this technique is exactly the implementation architecture used by some clustered database management systems in the market today. It is the backbone on which many high scale e-commerce sites are built. It is somewhat more work to implement this solution initially. The combination of fault isolation, administrative node autonomy, and flexibility to quickly grow or shrink as workloads change, make it an ideal solution for highly dynamic workloads where the availability risk of tightly coupled clusters is unacceptable and the mainframe-like software costs of clustered DBMS systems are unaffordable.

3.3 Affordable Performance Summary

The author has long believed that scalable DBMS clusters solve some application problems particularly well and yet they are often over-zealously offered to solve application problems where they aren't the most cost effective solution. Many application workloads can be hosted very cost effectively on a single SMP system with low DBMS software and maintenance costs and with minimal administrative complexity. Where a single DBMS system doesn't offer sufficient headroom, there exist both application-based solutions built upon partitioning and/or caching and DBMS hosted solutions like Oracle RAC. Both approaches can be used to address the data-tier scaling problem. The appropriate choice can only be made by understanding the constraints that each design alternative brings, and make the appropriate cost/complexity/application constraint tradeoff. Many of the tradeoffs are outlined above. It's useful to note that most very high scale e-commerce systems choose the partitioning and caching route in that their systems are already so large and complex that they needed to choose the simplest and most reliable data-tier architecture possible.

4. Summary

Shared disk cluster database management systems such as Oracle RAC are being discussed as a potential solution to the application scaling and robustness problem. In this paper, it is argued that the best solutions for availability have no single points of failure and support geo-clustering. RAC, with millions of lines of shared software between the DBMS and the disk offering many single points of failure, is less suitable as an availability solution and is better used as a multi-node scale-out solution. The most robust availability solutions are based upon log shipping and each of the three major DBMS providers, including Oracle, provide systems based upon this approach. Oracle Data Guard, IBM Log Shipping, and Microsoft SQL Server Log Shipping and Database Mirroring are all good availability solutions. Shared disk clusters, such as Oracle RAC, are neither the most economic nor the most effective approach to achieve database availability.

The original design point for Oracle RAC, when the technology was first conceived nearly a decade ago under a different name, was multi-node scale-up and that continues to be where this technology is best applied. For robustness, if RAC is used, we recommend that the scaling solution be combined with log shipping (e.g. Data Guard) to achieve high availability.

When working through whether multi-system clusters are appropriate for a given application, the complete cost equation needs to be understood. Several factors need to be kept in mind: 1) the disk subsystem cost is the dominant hardware cost component on large scale database hardware systems, and RAC requires non-commodity disk subsystems in any production deployment, 2) administration costs dominate hardware costs by a considerable margin on large scale deployments, and the complexity of the back-end DBMS system will influence these costs, and 3) database software costs are higher when using multi-node clusters, and single system image clusters like RAC come at a substantial premium above this point. That's not to say that multi-system DBMS solutions are not an appropriate solution for scaling a workload. They are still the right approach for some applications.

When multi-node database solutions are needed to achieve the goals of the application, there are two general approaches that can be employed. One approach is to delegate completely to the DBMS, and depend upon a shared disk cluster DBMS like Oracle RAC. Another approach is to depend upon data partitioning with data directed routing and/or mid-tier caching. The latter requires additional application design investment, but once this is made, offers more robustness, lower cost, and greater application flexibility. Some of the application design approaches are discussed that can be employed to achieve node-independence in the data-tier. They show that, for many applications, partitioning is the most scalable and most affordable solution to achieving application scaling. Global scale e-commerce systems requiring near continuous availability, where scaling requirements are difficult to predict at deployment time, and other high scale systems typically employ partitioning to achieve scalability and node-autonomy, and log shipping to achieve their availability requirements.

This paper focuses on two important attributes of high scale, data intensive applications: 1) application availability and 2) affordable performance. The original design point for RAC was multi-node scalability and it remains a less than ideal choice to address application availability. Fortunately, all major DBMS providers including Oracle offer technologies better suited to achieve this goal. We recommend these alternatives be used. Focusing on affordable performance where RAC can be considered, it has been shown that there exist more cost effective architectural alternatives that should be considered when deploying high scale data-intensive application workloads.

References

[ASKTOM04] Ask Tom Q & A
http://asktom.oracle.com/pls/ask/f?p=4950:8:9669274154404307949::NO::F4950_P8_DISPLAYID,F4950_P8_CRITERIA:22006637216777

[BURK89] Burkes, D., AND Treiber, K. 1989. Design approaches for real time recovery. Presentation at the 3rd International Workshop on High Performance Transaction Systems (Pacific Grove, CA., Sept.).

[DB203] DB2 Log Shipping
http://www-106.ibm.com/developerworks/db2/library/techarticle/0304mcinnis/0304mcinnis.html

[EWK04] If Oracle RAC Crashed Orbitz, Can We Trust 10G?
http://www.eweek.com/article2/0,1759,1429002,00.asp

[FORE02] Oracle 9i RAC Adoption Rate Is Slow, Unlikely To Change Soon
http://www.forrester.com/go?docid=19456

[IDC03] IT Spend Survey
http://www.idc.com/

[INTL03] Intel Scientist Finds Wall for Moore's Law
http://news.com.com/2100-7337-5112061.html?tag=nefd_lede

[INTL02] Intel, Deploying Oracle9i Real Application Clusters on Intel Architecture-based Servers
http://www.intel.com/ebusiness/pdf/affiliates/wp024201.pdf.

[GART01] Gartner Viewpoint: Microsoft outages hold universal lessons
http://news.com.com/2009-1001-251651.html?legacy=cnet&tag=owv

[GRAY04] TerraServer Cluster and SAN Experience, J. Gray, T. Barclay, July 2004
http://research.microsoft.com/research/pubs/view.aspx?tr_id=771

[MORL02] Unbreakable, James Morle, Scale Abilities, Ltd.
www.oaktable.net/getFile/36

[ORCL01] Oracle Data Guard Overview
http://www.oracle.com/technology/deploy/availability/htdocs/DataGuardOverview.html

[ORCL02] "Technical Comparison of Oracle9i Database vs. IBM DB2 UDB: focus on High Availability", Feb. 2002
http://www.oracle.com/technology/products/oracle9i/pdf/CWP_9iVsDB2_HA.PDF.

[ORCL03] Oracle RAC Best Practices on Linux
http://otn.oracle.com/tech/linux/pdf/RAC_best_practices.pdf

[PATT01] Patterson, D. 2002. A Simple Way to Estimate the Cost of Downtime. Regular LISA Paper, LISA Conference 2002

[SQLS01] Log Shipping in SQL Server 2000
http://www.microsoft.com/technet/prodtechnol/sql/2000/maintain/logship1.mspx

[SQLS02] An Overview of SQL Server 2005 Beta 2 for the Database Administrator – Extending High Availability to all Database Applications
http://www.microsoft.com/technet/prodtechnol/sql/2005/maintain/sqlydba.mspx#EGAA

[TCOS01]
IDC: Windows 2000 Versus Linux in Enterprise Computing: An Assessment of Business Value for Selected Workloads, Jean Bozman, Gal Gillen, Charles Kolodgy, Dan Kusnetzky, Randy Perry, David Shiang, Oct2002.
Giga: Budgeting for IT: Average Spending Ratios, Julie Giera, Aug2002
Forrester: IT Spending: The Real Opportunity Is in Human Capital, Craig Symons, Mar2003.
Gartner: Management Update: Enterprises Should Assess How Their IT Spending Stacks Up, Barbara Gomolski, Aug2003.
Meta: How Does Your IT Organization Measure Up to Current Industrywide Spending Performance Metrics?, Jed Rubin, Nov2003

[TPCC01] Transaction Processing Performance Council TPC-C Benchmark
http://www.tpc.org/

1. IBM RS 6000 Power Server R24 c/s; IBM DB2 for AIX 2.1; IBM AIX 3.2.5; 1-CPU; 1,470 tpmC; US$666.12 /tpmC; Available 12/15/1995
2. IBM Power 4+; IBM DB2 UDB 8.1; IBM AIX 5L V5.2; 32-CPUs; 1,025,486 tpmC; US$5.43 /tpmC; Available 08/16/04

[YDNR02] You Probably Don't Need RAC
http://www.miracleas.dk/WritingsFromMogens/YouProbablyDontNeedRACUSVersion.pdf