DBMS.next: is the next DBMS revolution looming?

Guy Harrison, Quest Software

For the first time in over 20 years, there appears to be chinks forming in the relational model's dominance within the database management systems (DBMS) market. The relational database management system (RDBMS) of today is increasingly being seen as an obstacle to the IT architectures of tomorrow, and – for the first time – credible alternatives to the relational database are emerging. While it would be reckless to predict the demise of the relational database as a critical component of IT architectures, it is certainly feasible to imagine the relational database as just one of several choices for data storage in next generation applications.

The last DBMS revolution

The relational database architecture became dominant throughout the 1980s in conjunction with the rise of minicomputer and client-server architectures. Client-Server applications were revolutionary in their ease of use, functionality, development and deployment costs. The relational database also made it easier for business to access and leverage DBMS data. Prior to RDBMS, data extracts and reports required specialised programming, wheras SQL allowed data to be extracted by a wider audience. Business Intelligence, reporting tools and the data warehouse entrenched the value of data and helped the relational database achieve almost total dominance by the mid-nineties.

The failed OODBMS revolution

However, from an application developers point of view, the relational model was not quite so ideal. The RDBMS came to prominence during the same period as Object Oriented (OO) programming. While relational database represented data as a set of tables with regular row-column structure, OO represented data in objects that not only associated behaviours, but which also had complex internal structure. The disconnect between these two representations has been described as an "impendence mismatch" that reduces application cohesiveness and leads to longer development cycles and more fragile software implementations.

In an attempt to resolve this disconnect, the Object Oriented Database Management System (OODBMS) was proposed. In an OODBMS, application data was stored as objects that matched the objects used in the programming language.

However, OODBMS failed to have a significant impact. One inhibiting factor was the reluctance of business to revert to the pre-relational scenario in which business reporting services could only be provided by highly trained programmers. The other key factor was the explosion of the internet and e-commerce, which powered the growth of applications whose needs for scalability and reliability could not be met by the emergent OODBMS systems.

Object Relational Mapping

With the relational database entrenched in internet application architecture, application architects solved the impendence mismatch by establishing Object-Relational Mapping (ORM) frameworks which allowed Object Oriented access to relational data. The Java EJB and Hibernate frameworks are the best known examples of ORM. ORM is ubiquitous in the world of Java and becoming

increasingly common in .NET (LINQ) and in Open Source application frameworks such as Ruby on Rails.

Enter utility computing

The internet gold-rush and the global Y2K resulted in almost budget-less funding for computer hardware, software and staffing. However, since the bursting of that bubble, IT has been subjected to unrelenting pressure to reduce cost. Quite a few IT trends of recent times have been accelerated by this economic imperative including increased server automation, free and open source software and reducing margins for software and hardware vendors.

However, it has been clear for some time that the allocation and utilization of computing resources will remain inherently inefficient as long as each application uses dedicated hardware resources. Typically, each significant application runs on dedicated hardware and this hardware is sized to match peak applications requirements. Of-peak, these resources are wasted.

The utility computing concept introduced the idea of allowing computing resources to be allocated on demand in much the same way the power company makes make electricity available on-demand to consumers. Such an approach would need to provide for only the peak aggregate requirement rather than the much higher aggregate of all individual peak requirements.

Grid computing provides a model for establishing the utility computing model on physical hardware. Virtualization provides a partially complementary vision of allocating resources to a variable number of virtual servers from a pool of physical servers – essentially from a grid. Cloud computing extends this concept to the provision of virtualized resources on-demand across the internet.

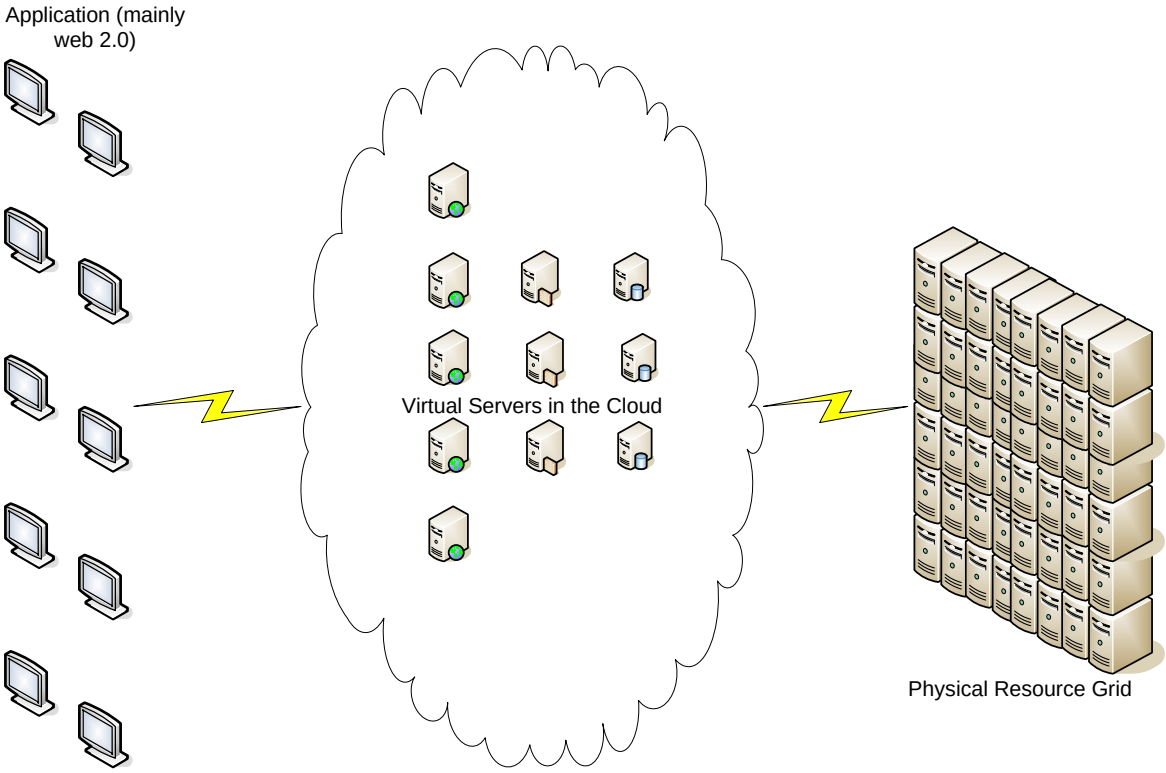

Figure 1 Grids, VMs in the cloud

RDBMS gets in the way again

Most components of modern applications can be deployed to a virtualized or gridded environment without significant disruption. Web servers and applications

servers all scale out using shared-nothing clustering and hence resources can be added or removed from these layers simply by starting or stopping members of the cluster.

Unfortunately, it's much harder to cluster databases. In a shared nothing database cluster, data must either be replicated across the cluster members, or partitioned between them. In either case, adding a machine to the cluster requires data to be copied or moved to the new node. Since this data shipping is a time consuming and expensive process, shared nothing databases are unable to be dynamically and quickly provisioned on demand.

Shared disk databases – Oracle's Real Application Clusters (RAC) in particular – have the potential to scale on demand, since an instance can be added to the cluster without the need to shift any data. RAC technology has legitimate advantages in this respect, but as yet no widely used RAC provisioning solution exists: instances are not created or removed based on demand but instead must be provisioned manually. RAC is also relatively high-maintenance, expensive and proprietary and hence less attractive to those trying to push down licensing and maintenance costs. No other credible shared-disk database architecture is apparent.

Cloud Databases

Cloud computing environments – such as those offered by Amazon (EC2), Google App Engine and (shortly) Microsoft – promise to allow consumers to deploy Web 2.0 applications on a highly scalable infrastructure totally provisioned and managed by the Vendor.

For the vendors seeking to create utility computing clouds or those trying to establish massively parallel but redundant and cheap data-access systems (such as Google), relational databases are untenable. These vendors need a way of managing data that was almost infinitely scalable, inherently reliable and cost-effective.

Google's *BigTable*[1] solution was to develop a relatively simple storage management system that could provide fast access to Petabytes of data which potentially distributed across thousands of machines.

Physically, BigTable resembles a B+-tree index organized table in which branch and leaf nodes are disturbed across multiple machines. Like a B+-tree, nodes "split" as they grow and – since nodes are distributed – this allows for very high scalability across large numbers of machines.

Data elements in BigTable are identified by a primary key, column name and (optionally) a timestamp. BigTable supports column-oriented as well as key-oriented lookups.

Amazon's SimpleDB[2] is conceptually similar to BigTable and forms a key part of Amazon's Web Services Cloud computing environment.

At time of writing the details for Microsoft's cloud computing offering – which includes SQL Server Data Services[3] (SSDS) – were less clear, but SSDS is appears to be at least conceptually similar to BigTable and SimpleDB.

---

[1] http://tinyurl.com/yooofv

[2] http://tinyurl.com/23l97d

[3] http://www.microsoft.com/sql/dataservices

ORM-CloudDB synergies

The chasm between the database management capabilities of Cloud databases and mainstream relational databases is huge. Consequently, it's easy for relational database zealots to dismiss their long term potential.

However, for applications following the ORM and Model-View-Controller (MVC) design patterns, these cloud databases can provide the core data management functionality required by the application. Furthermore, they can provide this functionality with compelling scalability and economic advantages. In short, they exhibit the familiar signature of a disruptive technology[4] – one that provides adequate functionality together with a compelling economic advantage.

Challenges for the CloudDB

Cloud Databases to face significant challenges before they can feasibly enter the mainstream. These include:

- **Transactional support and referential integrity**. Applications using cloud databases are largely responsible for maintaining the integrity of transactions and relationships between "tables". One might anticipate that this challenge will be met by the establishment of middleware layers that mediate between application logic and the CloudDB layer.
- **Complex data accesses**. The ORM pattern excels at single row transactions – get a row, save a row, etc. However, most non trivial applications do have to perform multi-table accesses for which SQL excels. The middleware layer alluded to in the previous point could potentially provide routines to ease these operations.
- **Business Intelligence.** Application data has value not only in terms of enabling application functionality, but also has value in its own right to the business. The dilemma of the pre-relational database – in which valuable business data was locked inside of impenetrable application data stores – is not something that applications owners will be prepared to tolerate. Consequently, mechanisms will be required to allow CloudDB data to be accessed for BI purposes with the ease to which business has become accustomed.

End of the One Size Fits All DBMS?

Round about the same time as the emergence of the Cloud Databases, a few researchers were investigating the fitness for purpose of modern RDBMS. In 2005 Mike Stonebraker (relational database pioneer, creator of Ingres and Postgres) and colleagues published papers arguing that for every significant application type, a customized database design could deliver at least a ten times improvement in performance[5] compared to today's One Size Fits All (OSFA) relational database.

Stonebraker et al followed up with concrete designs for database systems optimized for Data Warehousing (C-Store[6]) and OLTP application processing (H-Store[7]).

---

[4] http://en.wikipedia.org/wiki/Disruptive_technology

[5] http://nms.csail.mit.edu/~stavros/pubs/osfa.pdf

[6] http://www.mit.edu/~dna/vldb.pdf

[7] http://www.vldb.org/conf/2007/papers/industrial/p1150-stonebraker.pdf

C-Store: Next generation Data Warehouse

The C-Store model is based primarily on optimizing the DBMS for retrieving groups of columns rather than groups of rows. In today's relational database, it's inexpensive to retrieve all columns for a row, but very expensive to retrieve all the row values for a particular column. C-Store inverts this dynamic by optimizing for column retrievals.

Partially as a consequence but also as designed, C-Store is optimized for read efficiency over write efficiency. Physically, C-Store organizes data by projections (slices of a table that include specific rows) that are maintained in an optimal sort order. C-Store employs shared nothing clustering to allow for scalability and parallelism across machines.

C-Store support standard SQL and support for read consistency and transactions. Commercial implementations of C-Store exist such as MonetDB and Vertica.

H-Store: OLTP rewrite

H-Store is described as a "complete re-write" of the OLTP DBMS.

Disk technology remains the biggest bottleneck for DBMS systems and consequently H-Store employs memory-based storage. Persistence is guaranteed through replication across multiple machines and – of course – in-memory data can be backed up to a more persistent media. But in the normal course of events, disk IO is not a factor in H-Store performance.

H-Store employs a hierarchical data model. While hierarchical organization is less flexible (and arguably less "correct") than the relational model, it allows for highly optimized partitioning and shared-nothing clustering, which in turn allows for scale out across large numbers of machines. In this respect the H-Store approach is very similar to the CloudDB approach.

H-Store radically simplifies the concurrency model employed by relational database to avoid many of the overhead and contention issues that arise. Each H-Store instance is single-threaded which simplifies locking and latching, although multiple instances can be deployed on a single machine to take advantage of multiple CPUs.

Transactions are made more atomic than in the relational database model by being encapsulated into a single stored procedure call, rather than as a collection of separately executed SQLs. This ensures that transaction durations are minimized (no think-time or network time within transactions) and further reduces locking issues.

The H-Store proposal abandons SQL in favour of a model that has more in common with pre-relational database languages or with ORM-based approaches. In fact, Stonebraker et al suggest the Ruby on Rails ActiveRecord ORM framework as the favoured approach.

Stonebraker et al, claim that H-Store has delivered 80-times improvements in throughput on TPC-C benchmarks.

The possible future

The emergence of CloudDBs, together with the C-Store and H-Store proposals has allowed me – for the first time since the 1990s – to imagine a future in which the relational database of today has been depreciated in favour of new approaches.

The very simple CloudDB approach, and the more sophisticated H-Store approach seem to represent the extreme ends of a architectural continuum that could satisfy a most OLTP requirements. H-Store and CloudDB both offer scalability,

economy and performance, with H-Store layering on transactional support and other features attractive to mission critical applications.

The need to ensure that valuable business data does not become lost in the cloud can be provided by the establishment of C-Store oriented data warehouses which are the ultimate recipients of data required for BI and OLAP purposes.

None of which is to suggest that today's OSFA relational database is about to undergo a rapid disruption. But for the first time in a long time, it's possible to imagine a world in which the relational database is not the database king.

Guy Harrison is a chief architect for database solutions at Quest Software, and is a recognized expert with over 20 years experience in application and database administration, performance tuning and software development. Guy is the author of Oracle SQL High Performance Tuning *(Prentice Hall),* Oracle Desk Reference *(Prentice Hall) and* MySQL Stored Procedure Programming *(O'Reilly with Steven Feuerstein). He is chief architect of Quest's Spotlight® family of diagnostic products and has contributed to the development of other Quest products, such as Foglight® and Quest Central®. For more information, visit http://www.guyharrison.net.*