

WHITE PAPER

DATABASE SCALABILITY AND CLUSTERING

As application architectures become increasingly dependent on distributed communication and processing, it is extremely important to understand where the bulk of the database processing occurs: on the server, on the client, or split between client and server.

Sponsored by Versant Corporation

INTRODUCTION

There are numerous products on the market today that can be considered “databases,” where a database — when loosely defined — is a product that provides a means of storing data persistently and retrieving that data again at a later point in time. Opinions as to what truly constitutes a database can vary, but they generally agree that a database must address issues that are far more involved than simple storage and retrieval of data.

Generally, a database must address at least two of the following issues:

PERSISTENCE

Storage and (random) retrieval of data

CONCURRENCY

The ability to support multiple users simultaneously (lock granularity is often an issue here)

DISTRIBUTION

Maintenance of relationships across multiple databases (support of locality of reference, data replication)

INTEGRITY

Methods to ensure data is not lost or corrupted(features including automatic two-phase commit, use of dual log files, roll-forward recovery)


AVAILABILITY

Support of 24x7 operation (on-line maintenance, fault tolerance, and other features.)

SCALABILITY

Predictable performance as the number of users or the size of the database increases.

If a product only addresses the issue of persistence, then it can safely be placed in the category of “persistent storage managers.” This category ordinarily includes products that utilize flat files, VSAM, ISAM, or data I/O streaming techniques.



As the number of issues addressed by a database product increases, the more “high-end” that database product is perceived to be, and the more suitable that product is to play a major role in the development and deployment of large-scale, mission critical applications.

As application architectures become increasingly dependent on distributed client/server communication and processing, it is extremely important to understand where the bulk of the database processing occurs: on the server (server-centric architecture), on the client (client-centric architecture), or is the processing relatively evenly split between client and server (balanced client-server architecture).

Along with these vastly different database architectures are two types of data clustering: physical, which directly affects database server performance, and logical, which directly affects client-side application performance. The effect of data clustering on application performance and scalability is directly impacted by the architecture of the underlying database. In fact, the clustering models utilized on both client and server dictate not only how data is retrieved from disk by the server, but also how that data must be transported from server to client. The way that data is transported to the client directly impacts how the application will scale as the quantity of data accessed by the client increases (e.g. transactions become larger and more involved), the size of the database increases, or both. Additionally, since the client-side (logical) clustering model dictates how data is transported from server to client, the clustering model can also have an positive or negative impact on scalability from the perspective of application concurrency as a result of the locking model utilized by a particular database. Simply put, the wrong combination of clientside clustering and locking models can cause application throughput to degrade dramatically as the number of client applications (i.e. database users) increases

This white paper addresses the issue of data clustering and how optimal clustering can benefit performance by improving application scalability. The subject of relational databases is visited briefly, but the emphasis of this article is on client/server architectures built around an Object-Oriented Database Management System (ODBMS).

CLUSTERING DEFINED

First, we must understand what clustering is. A dictionary might describe clustering as “a grouping of related items held together.” For a database, this might be reworded to say, “a grouping of related items stored together for efficiency of access and resource utilization.” In a typical implementation, clustering is simply the collocation of data on the same physical disk page.

SERVER-CENTRIC DATABASES

In a relational database management system (RDBMS), the notion of data clustering is straightforward. Sequential rows of a particular table are stored together on disk, page-by-page. This makes particular sense with the relational architecture as most database activity is based on some form of sequential data access of a table. The physical clustering of data in this environment is extremely important to database performance since the architecture of RDBMS systems is heavily server-centric. (“Server-centric” implies that the majority of actual database operations occur on the server proper, even in a distributed client/server environment.)

This physical collocation of data on disk allows for optimal access of data within a particular table because only data from that table is stored on that particular page. The benefits of clustering in an RDBMS are mostly realized by the server because it performs all of the direct access to disk and virtually all database operations. This clustering plays no real part in client operation as only data requested by the client is actually passed there from the server. In an RDBMS, a query that is not able to benefit from clustering will be slower, but client operation will be unaffected because even in this case only requested data is passed from server to client. Actual data pages remain on the server.

By way of example, in an optimally clustered case, a single query that resulted in 10 rows being returned to the client

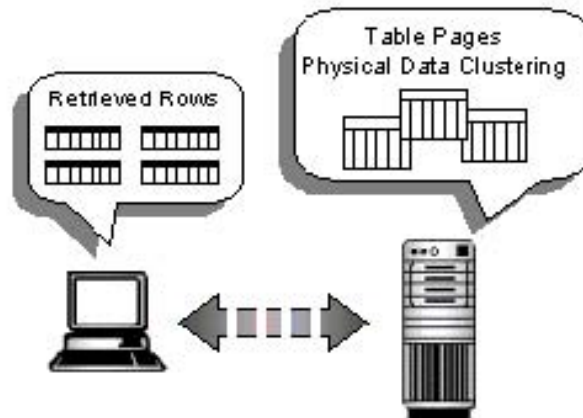


Fig. 1 – Server Centric Technology (RDBMS)

might cause the server to retrieve only a single page from disk. A query that was not able to take advantage of clustering at all might retrieve 10 pages from disk while still returning only 10 rows to the client. In either case, the same amount of data is returned to the client although the server itself must do far more work in the suboptimal case than it does in the optimal case.

CLIENT-CENTRIC DATABASES

In contrast to relational systems, most—but not all—Object-Oriented Database Management Systems (ODBMS) are actually very client-centric in nature. As with an RDBMS, all disk activity occurs on the server. Unlike an RDBMS, however, most database activity occurs within the client application itself. Pages are retrieved from disk and passed directly from server to client for processing. In these systems, even such basic database operations as queries and index maintenance are actually client-side operations. The server simply reads and forwards pages to the client, and then waits for those pages to be modified and returned for update to disk.



Figure 2 – Client Centric Architecture (ODBMS): Physical = Logical Clustering

Looking at this another way, the benefit of clustering is largely felt by the client in these systems because clustering is a function of data content and these servers only know about pages of data. The server benefits only because the more optimally the data is clustered, the lower the number of pages that must be retrieved from disk. More important, however, is that with optimal clustering, only the absolute minimum number of pages required from the server are returned and mapped into client memory.

There can be a problem with this client-centric approach to clustering, however. Optimal clustering is achieved only when data relevant to a particular query is collocated physically on disk. As a particular data element can only reside in a single place within the database, it seems unlikely that the physical location of that element will be optimal for all patterns of access. Therefore, unlike a relational database that might return 10 rows from a query whether clustering was optimal or not, these systems return as many pages as the server must retrieve to satisfy a request.

Rather than returning the ODBMS equivalent of 10 rows (as were returned in the previous RDBMS example), a client-centric architecture might return one page in the optimal case while 10 pages might be returned and mapped into client address space in the suboptimal case.

To build upon the example a little, assume that each row (or object) returned occupies 100 bytes, thus allowing approximately 40 objects to be stored per disk page. Discounting the overhead of the network communication protocol and object instantiation in the client, the optimal case requires one (4 KB) page to be retrieved from disk, and 1000 bytes (10 objects * 100 bytes), or roughly 1 KB to be returned to the client application over the network, consuming 1 KB of network bandwidth and 1 KB of client memory in the process. The worst-case, suboptimal scenario would require that 10 pages be retrieved from disk and that those same 10 pages be returned over the network and mapped into the client's address space. Given that the client's query again only requested 10 objects (at 100 bytes each), and that 40 KB were returned and mapped into the client's address space to satisfy the query (10 pages * 4 KB/page == 40 KB), we see that 98%

$$\frac{(40\text{KB returned} - 1\text{KB required})}{40\text{KB}} \times 100 = 98\%$$

40KB

of the network bandwidth required to transport this data and 98% of the memory required to map the requested data was superfluously consumed (e.g. unnecessarily wasted) in the process. Upon initial reflection, one might wonder the significance of this point; 40 KB is not that much memory or bandwidth anymore. This is where the notion of application and database scalability comes into play. If the 10 rows were to become 10,000, and the 10 pages were to become 10,000, then using the same formula as above, we see that in the optimal case,

$$\begin{aligned} \text{memory required} &= \text{bandwidth required} \\ &= 10,000 \text{ objects } 100 \text{ bytes / object} \\ &= 1,000,000 \text{ bytes} \\ &= 1 \text{ MB of memory or bandwidth consumed} \end{aligned}$$

and in the worst case,

$$\begin{aligned} \text{memory required} &= \text{bandwidth required} \\ &= 10,000 \text{ pages } 4\text{KB} / \text{page} \\ &= 40 \text{ MB of memory or bandwidth consumed} \end{aligned}$$

again resulting in approximately 98% (in this case 39 MB) of superfluously consumed memory and network bandwidth. At a 10-megabit/second rate of transfer, this results in (best case) an extra 31 seconds of time elapsed to complete the request. Couple this with the fact that the extra 39 MB might require the operating system or database to swap out previously mapped pages (and the potential for thrashing if those swapped pages are subsequently accessed again in short order), it is clear that the scalability of page-based, client-centric databases must be carefully scrutinized with all potential patterns of access in mind to ensure that the database will scale sufficiently to meet the needs of the application.

Obviously, the impact of poor physical clustering for a particular pattern of access is minimal if transactions are small or there is sufficient memory available on the client machine to allow a significant portion of the database (the more, the better) to be pre-loaded and retained in memory prior to suboptimal accesses. Of course, this assumes that there are few users trying to access this pre-loaded data if it is subject to modification. Otherwise, the locking protocol would have a severe impact on concurrent access (i.e. number of users) to the data, thus impacting overall scalability.

CHANGING ACCESS PATTERNS

This client-centric approach to clustering may be appropriate (or even optimal) for the primary application under development. Typically the application's data storage model is developed based on a preconceived notion of a predictable access pattern. This preconception will then influence decisions about how data should be collocated on disk.

The problem with this is that there is rarely a single pattern of access that is applicable across the entire application domain under development. While things may be predictable for the primary application as originally designed, the chosen collocation (clustering) of data for that application may be completely inappropriate for other secondary applications used for analysis and reporting. Additionally, changing requirements over time might eventually render the initial clustering model obsolete.

For example, let us assume that the primary application caused the data to be structured on disk for optimal access “from the top.” (e.g. Employees are stored physically with their departments, and access to employees is via navigation through a department.) Another reporting tool wants to access only department data. To access all departments, all employee data that is collocated on a page that contains department information must be transported from server to client in addition to the requested department data.

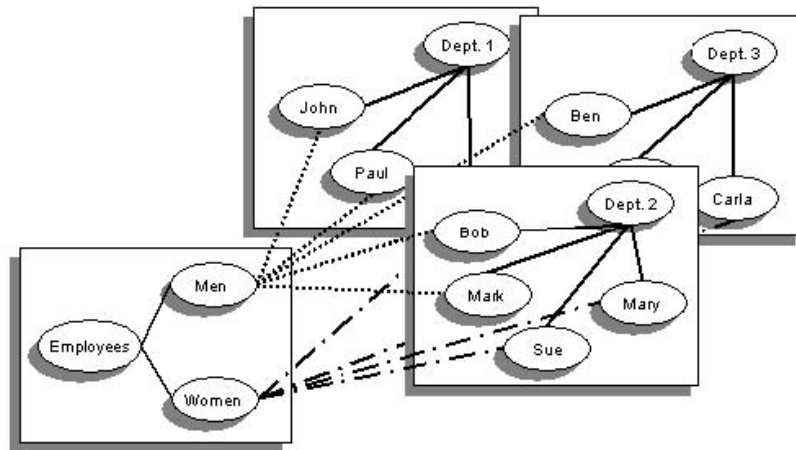


Fig. 3 – Good logical clustering for access to employees via department. Inefficient logical clustering for access to department information without employees. Inefficient logical clustering for access to employees by gender.

TWO TYPES OF CLUSTERING

Analysis of the previous section shows that there are in fact two types of clustering that exist in a client/server database architecture: client-side and server-side. The server-side clustering can be referred to as physical clustering because benefits are realized only when data for a particular disk access are physically collocated on the same disk page. From the server's perspective, client-side clustering can be thought of as logical clustering because benefits are achieved only when client-side resources are optimally utilized for a particular transaction. These resources typically include network bandwidth and process virtual memory.

RELATIONAL DATABASES REVISITED

Looking again at server-centric (relational) databases, this differentiation of server-side and client-side clustering clearly makes sense. Optimal physical clustering results in the retrieval of the smallest possible number of pages from disk. The logical mapping of this data into a client's address space is by definition optimal, and is only suboptimal if the client inadvertently requests more data than it actually needs (e.g. through a query that is too general).

When the query is exactly optimal, then the logical mapping of data can be considered optimal even when physical clustering for that particular access is less-than-optimal. Reading the preceding paragraph, it sounds as if the relational model is optimal, at least from a data clustering perspective. The relational model makes a clear distinction between logical and physical clustering, and thus should be the database model of choice, correct? In truth, the answer is not so straightforward. While it is true that some applications may demand a relational database because of the completely random and ad hoc nature of data access coupled relatively simple data types and relationships, there are

many other applications that find the relational model far too inflexible to solve their problems efficiently. This rigidity inhibits an RDBMS from efficiently modeling complex relationships and various object-oriented concepts such as inheritance.

OBJECT-ORIENTED DATABASES REVISITED

In the case of object-oriented databases, the benefit of physical clustering is much the same as with server-centric databases. The higher the degree of collocation for a particular request (query), the more optimally the server engine can behave; fewer pages must be read from disk to satisfy the query.

There is also another advantage to ODBMSs. Unlike relational databases which cluster rows of a particular table together on a disk page, ODBMSs can store dissimilar but related objects together on a page as anticipated access patterns demand. Closer examination of the example in the section on client-centric databases can help to clarify this statement. In a relational database, if an analyst wanted to find the names of the employees in a particular department a join must be performed: the “department” table must be scanned for the ID of each of the employees in the department and those IDs must then be joined with the “employee” table to find the names associated with the IDs. This results in the access of one or more data pages in each of the tables, as well as a fair amount of additional processing to actually perform the join operation. An object database will allow the department and its employees to be collocated on the same page (or pages if they won’t all fit), with the department maintaining a direct relationship with its employees. As a result, there is less disk I/O performed and less overall processing—employee names are retrieved via direct navigation rather than a join operation.

In the client-centric approach taken by most ODBMSs (where the server retrieves and distributes pages to the clients) physical clustering and logical clustering can be viewed as being one-and-the-same. When a particular access cannot rely on physical clustering for optimal data access (as in the example where the reporting application wanted to list employees by ID),

more pages than are actually necessary to contain the data requested are returned to the client, consuming more client memory and network bandwidth than the requested data actually requires. Therefore, it is clear that in client-centric architectures, when physical clustering proves to be suboptimal for a pattern of access, logical clustering in these cases is always less than optimal as well. By definition, the page-based approach of retrieval and transport is returning more data to the client than it actually requested.

The preceding paragraphs clearly indicate that a client-centric architecture will not scale well in environments that cannot be certain of optimal physical clustering for all patterns of access. In fact, in situations where physical clustering breaks down, client-centric databases can greedily consume far more system resources than should be acceptable, with performance being negatively impacted to a significant degree—particularly as the size of a database and its transactions increase.

BALANCED CLIENT-SERVER DATABASES

There is at least one ODBMS, Versant, whose model for database activity is more evenly divided between client and server than other ODBMS products. In this architecture, the server engine itself is aware of the content and location of individual objects, in much the same way as a relational database understands the location and content of individual rows in its tables. Consequently, database operations such as queries and index maintenance occur in the server proper, while other activities are left for the client application to manage.

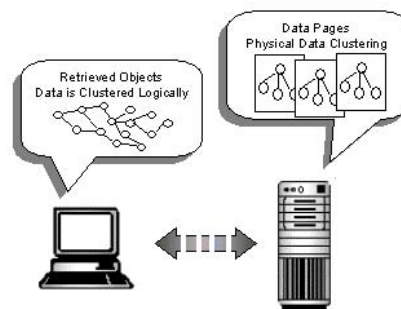



Fig. 4 – Balanced Client-Server Clustering



In this balanced architecture, the ODBMS server benefits from physical clustering of related—but possibly dissimilar—objects. As the server in this case is “object aware,” only those objects that satisfy the request are actually returned to the client from the server (much as a relational database only returns the rows requested). In this architecture, and unlike the clientcentric architecture, logical clustering on the client is different than, and in fact completely unrelated to the physical clustering that exists on the server.

Building again on the previous RDBMS example, whether a client requests one object or ten, only those objects that are specifically requested by the client are actually transported from the server. The amount of data transported is completely independent of the number of pages that must be retrieved from disk on the server to satisfy the request.

What this means is that even if a query is unable to benefit from clustering that exists on the server, only data that is specifically requested is transported to the client. A query that is suboptimal from the server’s perspective has no impact on client performance once the query returns; client resources (memory and network bandwidth) are conserved. Consequently, queries from (secondary) applications that cannot take advantage of the data model will still perform well.

Unlike their client-centric cousins, balanced client-server databases utilize network bandwidth and client process memory optimally, regardless of the impact that physical clustering may or may not have had on the server. The balanced client-server architecture is the only ODBMS architecture that truly exploits the differences between physical data clustering on the server and logical data clustering on the client. Thus, balanced client-server databases generally prove to be the most scalable variety of ODBMS.

SUMMARY

There are several generic database architectures: server-centric (typically relational), page-based client-centric ODBMS, and balanced client-server ODBMS. The impact of data clustering varies with the architecture based on the division of labor between client and server. Regardless, there are two types of clustering, both of which provide benefit to client and/or server when clustering minimizes utilization of limited system resources. Physical clustering benefits the server in all architectures. As disk I/O is among the slowest of all operations that an application can undertake, good physical clustering can be a tremendous benefit for server operations by minimizing I/O required. However, data can only be physically clustered in one way and it is not pragmatic to assume that all database accesses will be optimal based on that clustering. Good logical clustering conserves precious client resources such as memory and network bandwidth. In page-based client-centric architectures, logical and physical clustering are one-and-the-same. As physical clustering becomes suboptimal for a particular access, then so goes the logical clustering as well. In a balanced client-server architecture, good logical clustering is always achieved, regardless of how optimal physical clustering is for a particular access.

As it turns out, optimal logical clustering on the client is a key to maintaining scalability as a database and its associated transactions grow. Client-side logical clustering based on disk pages simply cannot yield scalable results for all patterns of access that an application domain may have. To maintain scalability regardless of a pattern of access, a database must differentiate page-based physical clustering on the server from logical clustering on the client, conserving resources and only transporting information that was specifically requested by the client.