

# Central vs. Distributed Systems

By Donald W. Larson

[dwl Larson@mac.com](mailto:dwl Larson@mac.com)

<http://www.timeoutofmind.com>

Systems architecture is the framework in which individual systems are fitted together to create a larger system. The resulting architecture needs to be valid from an operational and technical viewpoint. This paper attempts to provide some factors of centralized versus distributed systems. No information is presented here for the backup processes of either.

Information presented below is compiled from 3 separate sources and are not meant to be exhaustive in their perspectives of centralized versus distributed systems. **I usually took the words as published and I have not, in most cases, rephrased the information.**

- Introduction to Client/Server Systems, by Paul E. Renaud, published by John Wiley & Sons, Inc., 1993
- Client/Server Strategies, by David Vaskevitch, published by IDG Books Worldwide, 1993
- Information Technology in Action, edited by Richard Y. Wang, published by PTR Prentice Hall, 1993

Generally it makes sense to centralize data if vast portions of it are frequently updated or if all users access all data equally and need to see the most up-to-date data values. The term "client" refers to the requester of information, usually a user's computer in reference to the "server" or provider of the information.

Distributed data exploits the fact that data accesses tend to reflect a locality of reference. A particular workgroup is more likely to access some data more often than others.

In either case, the idea of a database revolves around two assumptions:

- All information is available all the time; nobody is ever told that some information is unavailable
- Any given piece of information is controlled by the database so that only one person at a time can actually change it. If several people want to change exactly the same piece of information, the database ensures that they are queued so that they change it one at a time, and each one sees the results of all changes made before his/hers.

# Central vs. Distributed Systems

## I. Distributed Database Management System -Requirements

- A. Location Transparency
  - 1. Queries access distributed data without knowing the location of the data
- B. Performance Transparency
  - 1. A query optimizer must determine the best path to the data
- C. Copy Transparency
  - 1. Multiple copies might optionally exist. If a site is down, the query is automatically routed to another server.
- D. Transaction Transparency
  - 1. Transactions that update multiple sites behave exactly as others that are local.
- E. Fragmentation Transparency
  - 1. Placing pieces of the data at different sites
- F. Schema Change Transparency
  - 1. Changes to the schema are propagated to other servers
- G. Local DBMS Transparency
  - 1. Different brands of databases provide the same heterogeneous service (Open Systems)

# Central vs. Distributed Systems

## II. Distribution of Resources

### A. Location of Data

1. Replicated (multiple identical copies)
  - a) Many applications experience far more queries than updates. Any updates must be propagated to all other servers. Data locking and server synchronization are non-trivial.
2. Partitioned (divided among several locations)
  - a) Data requested of one server may actually be provided by another server transparent to the request of the user. Same update issues as 1a.
3. Reorganized (derived or summarized from other locations)
  - a) Workgroup servers maintain data at a detail level. Other workgroups access this data at a higher probably summarized level. Reorganized schemas lend themselves to creating multiple, hierarchical architectures beyond 2 tiers. There is an inherent time lag between the detail and summarized levels of information.
4. Cached (partial replication)
  - a) Workgroup servers "borrow" data from the central server temporarily. All caches reduce the need for queries to the central server. Updates can be read-only causing all updates to be done on the central server. Write-through updates occur at the workgroup server and send the updates to the central server. Write-back updates also occur at the workgroup server but the central server is notified that its copy is invalid; the central server is only updated when another workgroup server references the same data.

# Central vs. Distributed Systems

## III. Reducing Data Movement

One of the objectives to distribute data is to minimize moving it (network bandwidth) around the network. Affinity analysis (not explained here) can help identify which data exhibit localized interactions. There are three ways to conserve network bandwidth:

- A. Avoid transferring data at all
  - 1. Use stored procedures to avoid sending records to the client for processing. Also replicate static data on the client to keep references to that data local. Server notifies clients when data has been updated. All client updates sent to server.
- B. Avoid sending data unnecessarily
  - 1. Compress data before sending it
  - 2. Send back smaller pieces of the data expecting the client to access smaller portions as needed.
  - 3. Track transfer failures to resend only the failed portions
- C. Make more efficient use of communications channel
  - 1. Use read-ahead caches anticipating the clients need ahead of time

## IV. Network Efficiency

A measure of how much user data is transferred in proportion to the network overhead involved. A rough measure ignoring the effect of network errors, can be calculated by:

$$E \approx \frac{M}{M + O}$$

where M = message size in bytes

O = overhead required to send the message

= header bytes + ACK size + (P \* network delay \* network speed)

P = number of packets sent

All efficiencies are affected by high-level protocols such as Ethernet and TCP/IP. Lower byte-count messages are less efficient than higher byte-count messages.

# Central vs. Distributed Systems

## V. Multi-server Data Flows

Whenever more than one tier is involved, servers must communicate with one another. Ideally the client should be unaware of any server communication. If the client needed to know about every server, the dynamic changes of server-to-server design would affect every client!

### A. Peer-to-Peer Data Flows

1. The hierarchy of servers can be exploited by temporarily turning the workgroup server into a surrogate client to access the other servers.

### B. Broadcast Data Flows

1. If the amount of server-to-server interaction is small, a broadcast-style of peer-to-peer communication can be appropriate. Each peer considers the request and only the correct peer responds.

### C. Filtered Data Flows

1. In some applications, a server has the data needed to fulfill a client request but needs to route the result through another node before it arrives at the client.