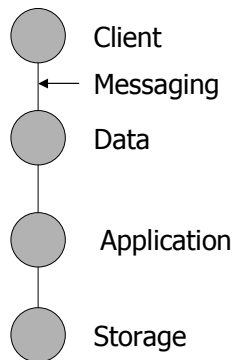


A Comparison of Distributed and Peer-to-peer Computing Architectures

A number of computing architectures are emerging in an effort to provide support for distributed transactions. Some of these architectures are based on traditional transactional computing models. Others more closely resemble network models. To be viable, a distributed computing architecture must ensure both data and transactional integrity at all levels.

The basic components of a transaction are:

Traditional Client-server Architecture

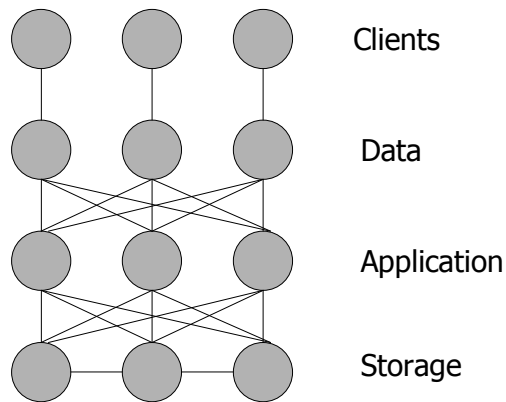


Note: In all diagrams, the placement of a line denotes a messaging component.

This is an example of a traditional client-server architecture. A client inputs data to and receives output from an application on a given computer. The application in turn stores the data in a database. In this way the integrity of both the data and transaction is preserved. Any transaction whether centralized or distributed must be able to maintain the integrity of both the data and the transaction regardless of how many participants, business objects, applications and databases are involved in the transaction.

A fully distributed network computing model would therefore be able to maintain both data and transactional integrity and to manage multiple instances of all components regardless of where those components physically reside on the network. Such a model would look like the following:

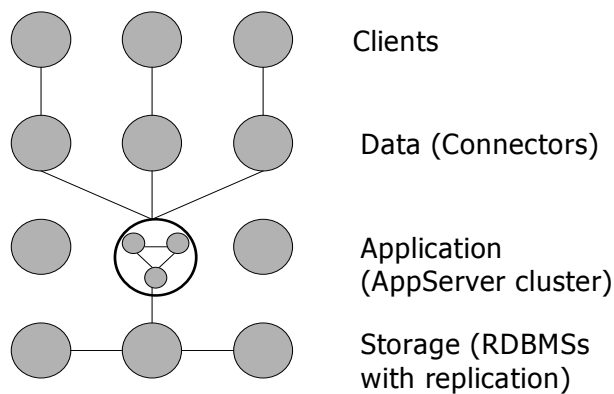
Fully Distributed Computing Architecture



This model describes an architecture in which clients or users interact with the system in the usual manner. The difference is that the data (or application objects), once in the system, can be placed or moved anywhere at anytime in a secure, reliable manner. This is the promise of distributed, network computing. This is also the type of architecture that is necessary to support future generations of IP and Internet based services and applications. These services and applications require that transactional objects be stored and routed over the network in much the same way that IP packets are stored and routed.

Other types of architectures exhibit certain characteristics of this distributed computing model. Enterprise Application Integration (i.e., EAI) and Business Process Management (i.e., BPM) applications based on application servers and relational databases have distributed data, but have not been able to globally distribute either the application components or the storage components:

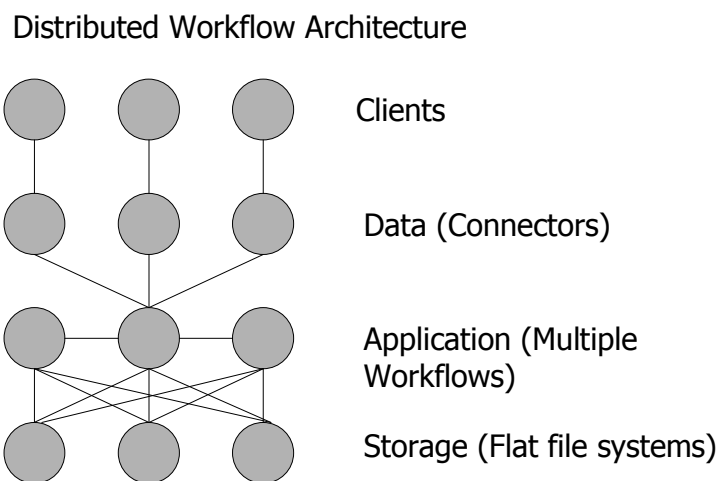
Current EAI Architecture



These architectures use a message bus as the primary mechanism for the distribution of data. The problem is that, no matter how widely distributed the data is, the application and database is confined to a single physical location. The farther away from the AppServer that the data is, the longer the transaction takes. And since AppServers and relational databases do not easily scale across multiple computers, the greater the volume of transactions, the more expensive it is to implement and scale the application and database components.

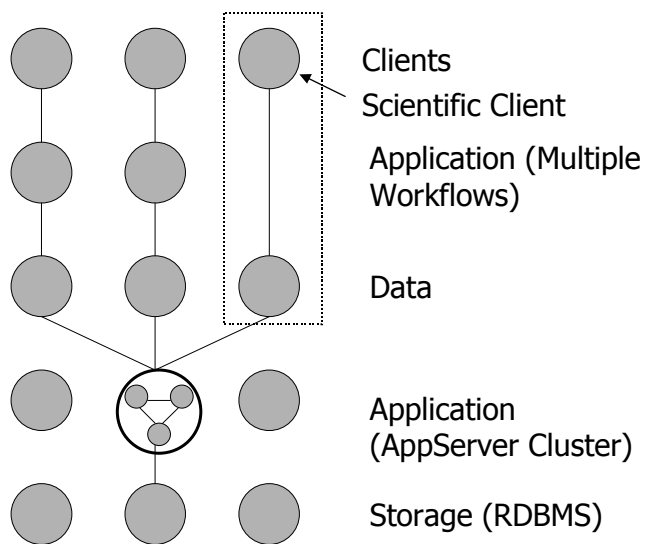
Perhaps one of the biggest problems associated with the use of centralized databases is not technical but psychological in nature. Business models such as eMarketplaces require businesses to engage in cooperative transactions that may include business partners and competitors. No business wants to have its critical data co-mingled with that of other businesses. The use of relational databases in these environments requires that each participating business either agree to put its data in the same database as other businesses, or force the provider of the services to create a separate instance of the database for each participant.

Other approaches to distributed computing have concentrated on the need to distribute application components as workflows. These approaches have had limited success in that they have not been able to resolve issues related to object level security, referential integrity of the data, and the distribution of physical data stores. In short, they are not transactional:



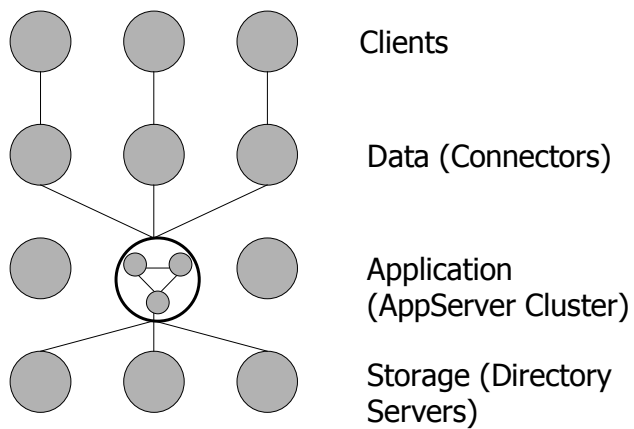
Another method used to distribute processing components is the peer-to-peer model. Peer-to-peer architectures normally provide a means of sharing files among participants and dividing large, calculation intensive problems into smaller batch processes that can be distributed among multiple PCs. Again, they are not transactional.

Peer-to-Peer Architecture



Finally, some solutions use directory servers to enable limited storage of some business objects:

Directory-enabled Architecture

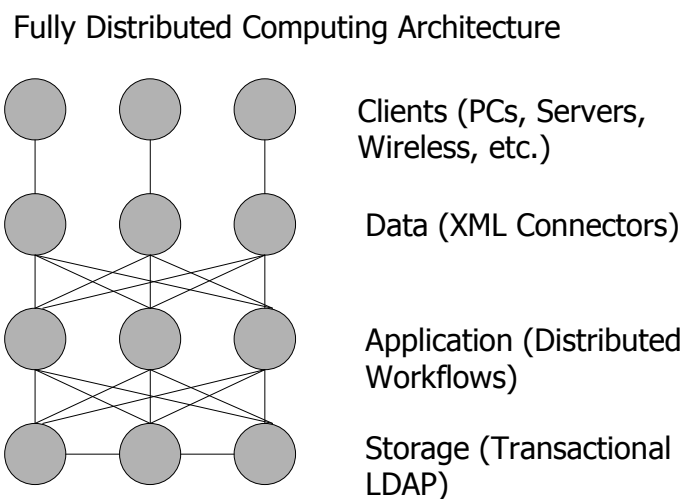


Often these directory-enabled architectures are designed to solve user-based security and policy management.

Whether it involves Web Services or other Internet protocols or specifications, a fully distributed computing environment must meet the following requirements:

- Componentization and mobility of objects, transactions, applications and data storage
- Security of every component regardless of its physical location
- Referential integrity of all data regardless of its state or physical location
- The ability to support changes in any component at any time without affecting the integrity of ongoing transactions
- The ability to support multiple instances of any or every component
- The ability to view any or all components at any degree of detail in real time
- The ability to support failover and recovery of components in real time
- The ability to enable multiple devices such as servers, PCs, SANs and wireless appliances to participate in the data acquisition, application logic and storage of complex transactions
- The ability to integrate with existing systems (i.e. ERPs, AppServers, RDBMSs, etc.) in a heterogeneous computing environment

Added to this is the requirement to support mobile transactions, allowing portions of the transactions to be maintained in one or more mobile devices. This means that the various components must be small enough to fit on wireless devices having limited memory and intelligent enough to support synchronization of both on-line and off-line transactions.



An example of a fully distributed computing architecture might contain any combination of client devices, Internet based interfaces for data as well as programs, stateless services that include fully distributed workflows (i.e., state machines) and fully distributed and secure storage for all software and data components.