In-Q-My Technologies GmbH

Clustering

Advantages of Clusters And Distributed Systems

Definition of a Cluster

A cluster is a parallel or distributed system consisting of independent computers that cooperate as a single system. In the case of the the In-Q-My Application Server Cluster, this group consists of two types of participants – dispatchers and servers. Generally speaking, clusters offer a way to utilize computer resources more productively in comparison to when the same number of machines are working standalone - the total result is greater than the sum of the separate parts.

Cost-efficiency is one aspect of choosing whether to set a cluster or to use standalone machines. Another advantage of clusters is that, from technological point of view, they are more reliable, secure, productive, and deliver better overall service than standalone machines.

The goal of a cluster is to allow distributing of the computing load over the several machines participating in it. These machines communicate with the dispatchers and with each other in order to provide better – faster and more reliable - service to the client. The idea of clusters is that the machines in it represent one unit and clients perceive the cluster as a single entity - not as a group, comprised of several computers. For clients the IP address of the dispatcher is the IP address of the whole cluster. Clients neither see the IP addresses of the servers in the cluster, nor have to know these multiple IP addresses. On the other hand, administrators know the IP addresses of the dispatchers and servers in the cluster.

When connected in a cluster, servers are still physically separated as different machines. The participants in the cluster share disk access and resources that manage data, but they do not share memory or processors.

Among the advantages of the clustering solution provided by the In-Q-My Application Server are scalability of the system, the ability to use machines, which are geographically dispersed, load balancing, fail-over, high availability and easy administration.

Scalability of the system

The ability to "plug-in" additional resources in the system at any time when more computing power is necessary is one of the main advantages of clusters and distributed systems.

Scalability allows to add new participants in the cluster when the load of clients' requests is heavy. It can be done dynamically without having to stop the cluster. This feature gives flexibility and ability quickly to adapt to changing workloads and to provide adequate service to more users when this is necessary. In-Q-My Application Server supports up to 64 participants in the cluster, which allows to serve simultaneously millions of clients' requests.

What is more important, because In-Q-My Application Server is J2EE™ compatible, it is possible the cluster to consist of machines with different hardware architectures and running under various OS – Windows™, UNIX™, Solaris, etc – which makes In-Q-My Application Server really cross-platform. This feature of J2EE™ compatible application servers allows building clusters that meet perfectly well the hardware requirements for more or less powerful, for cheaper or cutting-edge equipment.

Scalability is not only in regard to hardware. The architecture of In-Q-My Application Server gives users the ability to add new services to the already existing ones. This provides opportunities to easily integrate with existing external systems.

Globally Dispersed

In-Q-My Application Server allows the cluster to be globally distributed. The machines, connected in a cluster can belong to different WANs (Wide Area Network) and it is possible even to be distributed globally, i.e. in different countries on different continents. Global distribution of the participants in a cluster is a feature not supported by the most application servers, because their clustering technology uses simple IP multicast communication, which restricts, or at least firmly does not recommend, distributing the participants in a WAN.

Load Balancing

The idea behind load balancing is to send incoming requests for processing to the least busy server in the cluster. Load balancing increases productivity, because it allows to fully utilize the available resources and at the same time to deliver increased application throughput and optimal response time.

Generally speaking, load balancing for a group of servers can be done via hardware or software. Hardware load balancing usually provides less flexibility to the administrator, while software load balancing allows him or her more precise techniques for configuring the system and measuring performance.

The algorithm for load balancing developed by In-Q-My is as follows. At regular intervals, set by the administrator, the servers in the cluster send information to the dispatcher(s) about the level of their resource usage in percentage. When a new request arrives, the dispatcher(s) send it to the least loaded server. The algorithm takes into account not only the percentage of used resources, but also the percentage of free resources, which means that if an extremely big number of requests arrives simultaneously, not all of them will be send only to that server, which is the least busy at the moment, but will be directed to several machines in a way that will evenly distribute the load among them.

This algorithm is simple, yet powerful and functions faultlessly in clusters where servers run on different platforms and/or have different processing capabilities (not equal processing power or not equal amount of memory) as well as in homogeneous clusters where all servers have equal processing power.

Fail-Over Recovery

Fail-over recovery means that if one of the participants in the cluster is temporary or permanently unavailable, for instance because of a hardware crash, its functions are undertaken by another participant. As a result, requests are automatically redirected to a working server in the cluster and users and applications are not aware of what is happening, because the system continues to process requests properly.

Fail-over recovery applies not only to the servers in the cluster, but to dispatchers as well. If there are two or more dispatchers in the system and one of them fails, the incoming requests are redirected to the IP of another dispatcher.

Fail-over recovery is provided for data, too. Data is replicated (mirrored) on other participants of the cluster and this ensures that even in case of a crash there will be no information loss. Having not just the data, but its latest update reflecting its state after a transaction has been committed or rolled back on a server in the cluster, is vitally important and bearing this in mind In-Q-My has developed a distributed database which allows synchronization of the cached data distributed in the cluster.

The fail-over algorithms are based on the communication protocols for access to the system which are developed by In-Q-My. The fail-over algorithms prevent a single point of failure that affects the whole system. This leads to improved performance and higher availability.

High Availability

One of the most irritating messages a user sees is “The server could be down or is not responding.“ The expectations of users are that the server is up and running twenty-four hours a day, seven days a week. An unavailable server is a disaster for e-business. The losses due to downtime cannot be measured only in terms of revenues or missed profit. They damage the image of the company and ruin the confidence in it.

High availability is bound with scalability and fail-over recovery. These are the two factors (besides the hardware parameters of the participants in the cluster, of course) that predominantly influence the availability of the system. High availability has several aspects. Even in cases when the number of participants in the cluster is very big, and even when fail-over recovery and load balance algorithms are perfect, it is possible in theory billions of requests to arrive simultaneously. This poses risks for overloading of the system. In a situation like the above mentioned, it is more important first to finish processing of already received requests, than to receive new ones. The temporary denial to receive new requests for processing is not a high availability trade-off; it is simply provided to guarantee that under extreme circumstances the system will still continue to function without compromising security and the reliability of processed information.

Easy Administration

One of the greatest advantages of clusters is that they are easily administered. Through the Visual and Console administration facilities, In-Q-My Application Server allows integrated administering of the machines in the cluster instead of administering standalone servers. The presence of a “central point of command” leads to reduced costs for administration and training in terms of money, time and overall efficiency. Integrated administration creates a complete view of all the critical components and gives better control over total performance. It also allows fine-tuning of particular components/nodes on one or more of the participants in the cluster, which is a prerequisite for an adequate reaction in critical situations.

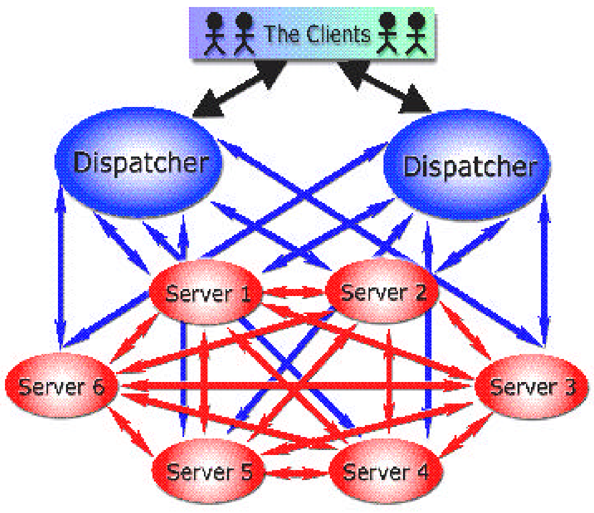
In-Q-My Application Server provides means for remote administration of the cluster, too. The administration tools allow secure monitoring both at start-up and during runtime. The runtime administration facilities give information about the status of execution of requests.

The cluster-level log files are another instrument useful for administering a cluster. In-Q-My provide cluster-level log files, which allow to monitor what has happened in the various modules of the system and which give administrators the freedom to choose what kind of information to be logged, where to log it, and in how much detail.

Cluster Architecture

Cluster components

There are two types of cluster components that participate in In-Q-My Application Server Cluster. As mentioned above, these two kinds of participants are dispatchers and servers.



The Participants in In-Q-My Application Server Cluster

Dispatchers

The function of the dispatcher(s) in the cluster is to receive requests from clients and distribute them to the servers in the cluster for processing. After requests are processed, the results are returned back to the dispatcher and from there to the client. The route passed by requests differs depending on their complexity and the resources they have to access. The general scheme is discussed in more detail in the chapter about the Route of the Requests.

What is important to note is that dispatchers do not process requests. One might wonder what their role is – only to forward requests to the servers and the responses back to the clients? In the hierarchy in clusters this role is crucial, especially in regard to load balancing and failover.

If a considerable number of requests is expected, dispatchers must be more powerful machines in order to be able to handle the requests without becoming the bottle-neck in the system, especially in regard to the fact that each dispatcher communicates with each of the servers in the cluster, as is seen from the picture above. But this does not necessarily mean that the dispatcher(s) must be the most powerful machine(s) in the cluster. On the other hand, recommended (hardware) configurations for the cluster depend also on the complexity of applications that are to be deployed and how much processing is to be done, as it will be discussed next.

Servers

The role of servers is to process client requests. What is important to note is that servers do not communicate directly with clients. Each of the servers in the cluster communicates with all other servers and dispatchers, as the picture shows. The fact that each server communicates with all other servers means that there is replication and synchronization of data, which leads to faster and more secure processing of the requests and executing the business logic.

The business logic of applications (EJBs, JSPs, Servlets, etc) is hosted on servers. In case of complex applications that require a lot of processing and/or communication, servers can become a bottle-neck for the system if their number or processing power is insufficient.

Cluster Configurations

The cluster solutions, which In-Q-My Application Server provides are several. Depending on the goals set for a particular system – complexity of applications, expected number of requests, available hardware, etc - its administrator should choose how to configure the cluster. There is not a general recommendation about the number of the participants in the cluster, the ratio between dispatchers and servers, what services and components to deploy on which machines, etc. The main guiding light is the overall performance and resource usage of the system. For instance, when the workload is not heavy, it is not reasonable to have many dispatchers and/or many servers, because in this case the percentage of their resource usage will be low and the system will not be efficient. In other cases the administrator(s) of the cluster need to configure a cluster comprised of as many machines as possible, because the requests are arriving "in bulk".

The recommended cluster configurations for application servers can be divided into two groups – with only one or with more than one dispatchers. Needless to say, the presence of at least one dispatcher and one server is necessary in order a cluster to be set. For very small systems this configuration - one dispatcher and one server - could be the right choice.

The first possible configuration is **a single dispatcher – n servers**. As stated above, the minimal possible number of servers in the cluster is one. This configuration allows building a stable and reliable system, but provides no fail-over and load balancing (obviously, if the only server crashes, there is nowhere else to redirect the requests to), which are among the main advantages of clustering and large-scale distributed systems.

More common are configurations, which involve two, three or more servers. Again, all client requests are received by the dispatcher, but in this configuration the dispatcher's role is to redirect the requests to the least busy server, where they are processed. This provides better fail-over, load balancing, and scalability, which leads to higher availability and better overall performance. The resources of the system are used better, because of the concurrency in processing of requests. The administration tools, provided by In-Q-My Application Server, make it an easy job to manage such a cluster configuration.

The "one dispatcher-many servers" configuration is suitable in cases when the number of arriving requests is medium, which allows all of them to be accepted by the dispatcher without turning it into the bottle-neck of the system. Still, the only dispatcher is a potential single point of failure. It is recommended for this configuration the applications and services to be homogeneously distributed over the servers in the cluster, because this allows better scalability.

The second possible configuration is **m dispatchers - n servers** (m < n). In this configuration it is not mandatory all dispatchers and servers to be on the same LAN (Local Area Network). They can belong to different WANs (Wide Area Network) and it is possible even to be distributed globally, i.e. in different countries on different continents.

The m dispatchers - n servers configuration unleashes the full potential of the clustering feature of In-Q-My Application Server, because it allows more requests to be received and processed. Again, there is no general recommendation about the ratio between the dispatchers and the servers in the cluster, nor about how to distribute the services and the applications on the machines in the cluster.

Often it is better to distribute homogeneously those applications and services, which allow it. This affords real scalability and fail-over – the risks for a single point of failure are reduced further. In terms of scalability the improvement is clear - a new participant, no matter if a server or a dispatcher, can be plugged into the system at any time. It has only to register with the group and joins the cluster.

One of the issues that need to be considered in this configuration is overall performance. "By default" the overall performance of an m dispatchers - n dispatchers cluster improves in comparison to the other possible configurations. But it requires proper installation and configuration of the system, proper development of the applications, proper deployment. Another potential drawback is out of the reach of In-Q-My Application Server and the reason is the operation system(s), under which the cluster runs. It is obvious that not all operation systems are equally good as server platforms, although In-Q-My Application Server is a cross-platform server and this fact provides more flexibility in choosing the operation system(s) to run the cluster on.

Logical Modules in the Cluster

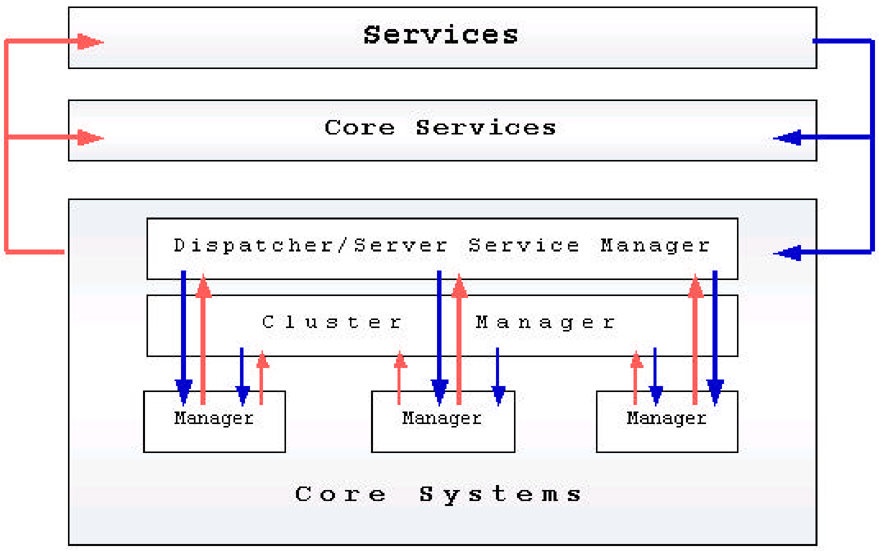
A cluster can be discussed in many aspects. One of them – the role of the participants in it – was the subject of the previous section. Another approach is to look at the modules or the separate parts that integrate the pieces of code into a system.

There are three types of logical modules in the structure of In-Q-My Application Server:

- core systems (managers);
- core services;

- (additional) services.

These three types of logical modules build every server and dispatcher in the cluster. The list of core systems, core services and (additional) services differs depending on the type of the participant (a server or a dispatcher). The picture below illustrates the modules on a separate machine and the data stream of communication among them. Communication is bi-directional, as the red and blue arrows identify it.



Logical Modules

Managers

Managers are the very base of the servers and dispatchers in the cluster. Core systems are located on each of the machines in the cluster and are not distributed. Managers are the first to be started, when a server/dispatcher starts.

The managers included in In-Q-My Application Server are:

- Framework. This is the manager that initializes the rest of the managers. In a way, it is a container for the managers.

- Managers, which have special functions in regard to clusters – Cluster Manager and Dispatcher (respectively Server) Service Manager.

- Log Manager, Thread Manager, Ports Manager, Timeout Manager, Memory Manager, Connections Manager, Classloader Manager. Their roles are explained in more detail in the technical documentation of In-Q-My Application Server.

Different managers perform different tasks. The two managers that are (most) related to the cluster are Cluster Manager and Dispatcher/Server Service Manager.

As its name implies, Cluster Manager is directly related to the cluster. This is the module in In-Q-My Application Server, which starts the cluster. If Cluster Manager on the first dispatcher node does not start, the whole cluster will not start. If Cluster Manager on a server node does not start, this element will not the join the cluster. The property files of Cluster Manager allow to set its properties.

Dispatcher Service Manager runs only on dispatchers and Server Service Manager runs only on servers. None of the managers on one machine communicates with its counterpart (the same manager) on another machine, no matter if a server or a dispatcher, not even Dispatcher Service Manager or Server Service Manager, besides Cluster Manager. All the communication in the cluster passes through the Cluster Managers of the servers and dispatchers, as it is shown in the picture on next page.

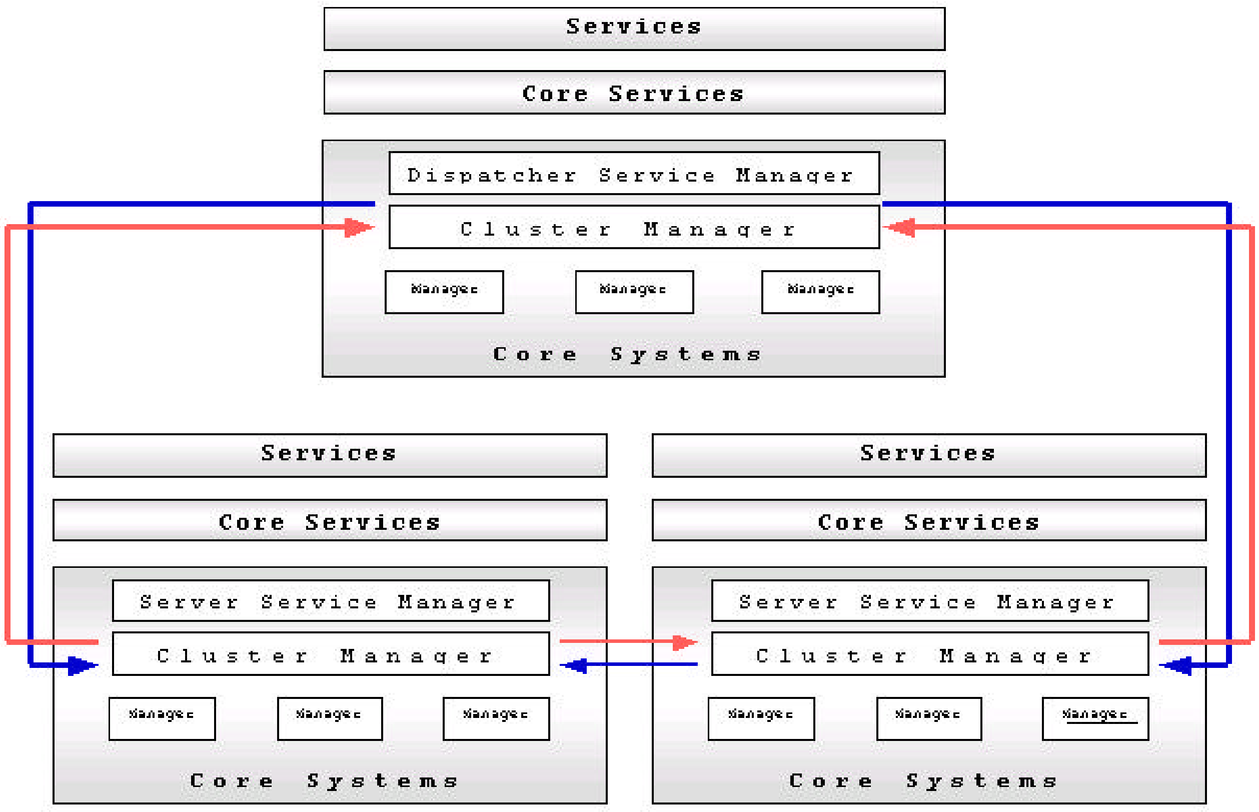
Core Services

Core Services are the second module in the kernel of the server. They provide the basic functionality of the server/dispatcher. On each of the machines in the cluster, Core Services are started by Service Manager, as seen from the picture above. Core Services in In-Q-My Application Server are: AI Service, SSL Service, Log, Deploy, Httptunneling, Crypt, P4, etc. The list of Core Services on servers and dispatchers is different. For instance, Admin, SecurityManager, DBMS, etc are Core Services on servers but quite logically are not run on dispatchers at all, because the role of dispatchers in clusters does not suppose the dispatcher to manage users or to take care of security, for instance.

(Additional) Services

Services are these logical modules in the architecture of the cluster, which extend its functionality. They are not part of the kernel of the server and the system could work properly without them, but the specified service will not be provided to clients. Again, the list of additional services on server and dispatcher nodes differs. It differs for the different servers and dispatchers (Server1, Server 2, etc.), too, because it reflects the role of the particular participant. Some of the additional services are: telnet, http, ejbentity, ejbsession, servlet_jsp, dbpool. It is up to the administrator to choose which services to start on which machine and this decision is made depending on the needs of the system (for instance what kind of applications are to be deployed in the cluster and on a particular machine).

Functionality and configuration of (additional) services is discussed in more detail in the documentation of In-Q-My Application Server.



The Stream Of Communication In The Cluster

This picture presents the logical modules in the architecture of In-Q-My Application Server and how they communicate with each other. For clarity, only the communication between the different participants in the cluster is presented here. The data stream of communication between the logical modules for each of the participants is given in the previous picture. It does not change when the machines are connected in a cluster.

The Route of Requests

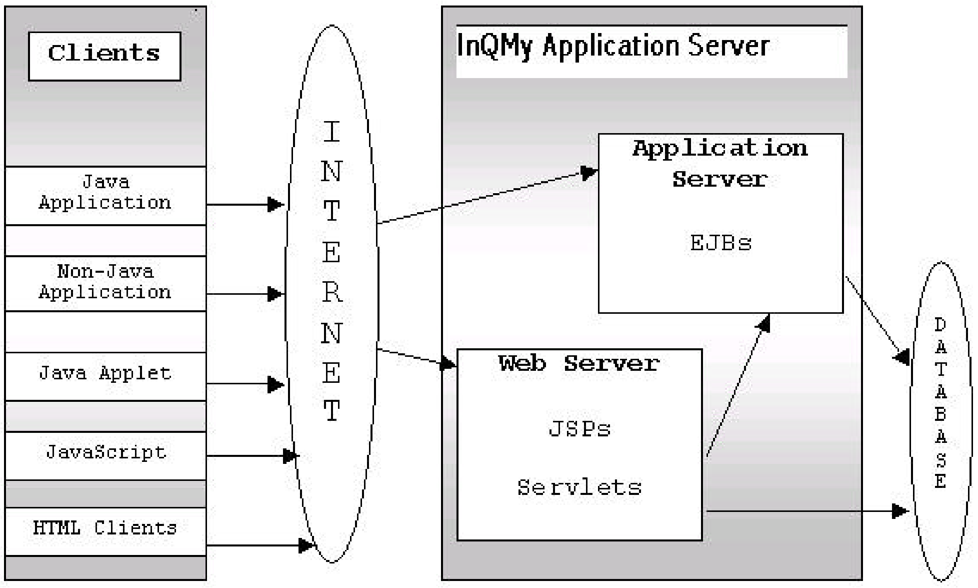
Application servers and multi-tier architecture

It is easier to understand the logic of the route of processing requests by In-Q-My Application Server, if this process is discussed in connection with the role of Web application servers and multi-tier architecture.

The role of Web application servers is to provide the platform for deploying and running sophisticated Web applications in addition to serving static pages. The occurrence of application servers' technology a couple of years ago led to the advent of multi-tier architecture, which was an answer to the limitations imposed by traditional two-tier (client-server) architectures.

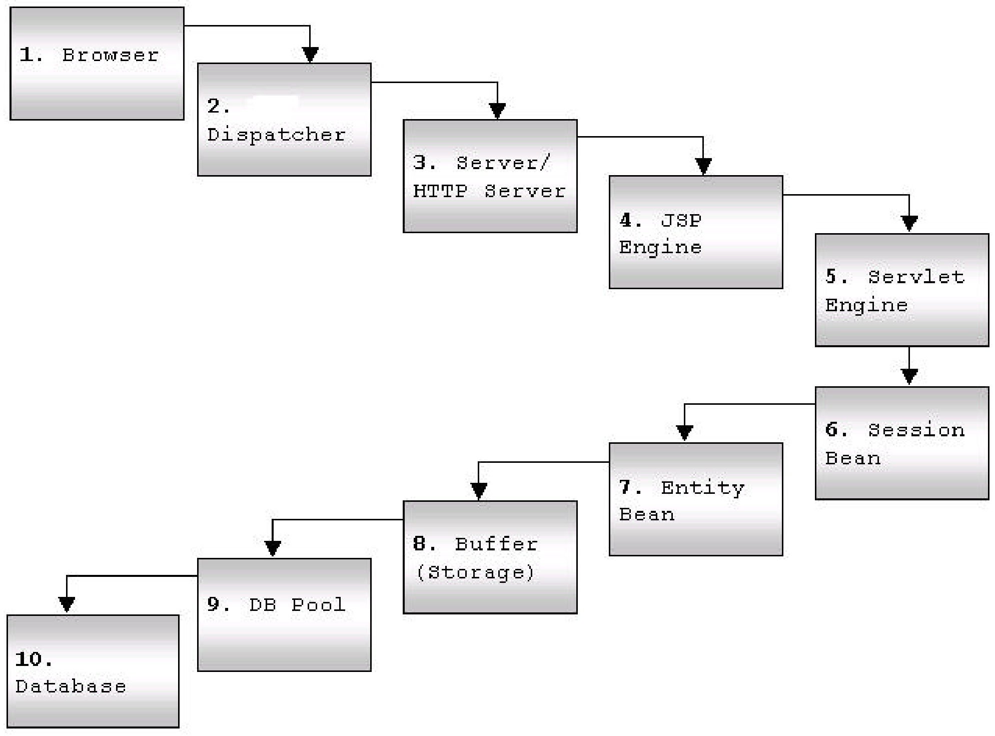
In multi-tier architectures clients do not communicate directly with the database, but send their requests to the application server, which then makes the connection with the database.

The next picture shows the general route of requests in multi-tier architectures and the role of In-Q-My Application Server:



Multi-Tier Architectures And The Place Of Web Application Servers In Them

The next picture presents in more detail what is happening to the request inside the application server.



The Route Of Requests

The aim of this picture is to present in a simplified manner what happens to requests from the moment they are sent by the client to the dispatcher till the results are delivered back to the client. When looking at this picture, one should bear in mind that:

a) The route of requests shows the longest possible route a request could travel. It is not meant to imply that every request passes along the whole route. Actually, usually requests do not pass through all of these steps or they follow the order (1-10) but skip some of the steps and go directly to a next step. For instance, if it is a JSP request, there is no need to go to the bean container in order to compile a servlet and to return it to the client.

b) On deliberately, in the picture there is no cluster. The picture shows only one server besides the dispatcher. The reason is visually to simplify the presentation of what kind of processing happens to a request in the system, no matter on which of the many servers in the cluster. If the picture were to present several servers, as the actual situation in clusters is, the picture would contain n JSP Engines, n Servlet Engines, n Bean Containers, etc, distributed on several servers. In this case it will be more difficult for the user to understand the idea. It is more important to show the route in general and the different stages a request passes, than in which container or engine on which server what is done in particular, especially in regard to the fact that this depends almost entirely on the way the cluster is configured. And above all, presenting the route as “distributed” among the servers might suggest that this is the recommended configuration – a JSP Engine on Server 1 accesses the Servlet Engine on Server 4, which in turn passes it to the EJB Container on Server 45, etc. Once again it should be stated, that there is no recommended configuration what to be deployed where – it is up to the administrator, and above all to the particular application.

c) The path for delivering the results of processing is along the same route but in reverse order, i.e. step 10, 9, 8, etc. and because of that it is not presented separately.

1. The browser (in this case, but it can be another client application) sends a request to the dispatcher(s) of the cluster.

2. The dispatcher(s) receive the requests and depending on its type, direct it to a server to be processed. In the case of a browser, which sends an HTTP request using the HyperText Transfer Protocol, the dispatcher sends it to a server on which the HTTP Service of In-Q-My Application Server, is running. If the expected number of requests for static HTML pages is considerable, at this stage it is possible to add an Apache Web server and to send the request to it. The idea is to leave all dynamic HTML pages, which require more serious processing before being delivered to the client, to be handled by In-Q-My Application Server.

3. A server receives the request. This stage allows distributing the request among the servers in the cluster. The corresponding service of In-Q-My Application Server receives the request. Depending on the type of the request, it can be directed by the dispatcher not only to HTTP Service but also to EJB Session Service, EJB Entity Service, Deploy Service, etc.

4. This stage allows distributing the request among the servers in the cluster, too. If it is a HTML request and there is a JSP in it, which must be processed before returned to the client, the request is passed to the JSP engine.

5. The servlet engine compiles the JSP into a servlet. The servlet can either be returned to the client, or if the servlet uses beans, it goes further along the route. This stage allows distributing the request among the servers in the cluster, too.

6. If the client is an application that doesn't need HTTP and processing of HTML pages, JSPs and Servlets, on step 3 the request is sent to the corresponding EJB service of In-Q-My Application Server and steps 4 and 5 are skipped. This stage does not allow distributing the request among the servers in the cluster, i.e. a given session bean can be processed only on the server, where it was received. But for faster overall performance it is advisable session beans to be replicated in the cluster.

7. At this stages entity beans are processed. In the case of entity beans with container managed persistence they can be distributed among the servers in the cluster. There is always one server, which is responsible for all the transactions with a given bean. EJBs with bean managed persistence make direct directions with the database through the DB Pool.

8. The role of the buffer (storage) is to establish connections with the pool, to lock connections, transactions, bean instances, etc. It is closely related to entity beans.

9. DB Pool – holds connections to the Database (10) that are to be used only for the needs of In-Q-My Application Server. The DB Pool is located only on one machine and is not replicated in the cluster. The binding to the DB Pool is global and it is accessed via the Naming Service.

10. This could be any external database, a driver for which is supported by In-Q-My Application Server – Oracle, SAP-DB, etc. This is the last stage in processing the request by an application server.

Bottlenecks Along This Route And In Multi-tier Web Applications In General

The bottlenecks for the faultless functioning of an In-Q-My Application Server cluster can be expected to occur on every transition from one step of the route to the next, as well as in the connections between the tiers. Usually bottlenecks are due to problems external to the cluster. Generally speaking, the cluster is functioning but bottlenecks reduce its performance and efficiency and threaten scalability, load balancing or fail-over.

Among the factors for occurrence of bottlenecks are:

- not enough processing power of some or all of the machines in the cluster or on clients' side
- improper configuration and administration of the machines in the cluster
- not carefully thought of development and deployment of the applications running in the cluster.

As this list implies, the reasons for bottlenecks occurrence can be in any of the tiers of multi-tier architecture.

Bottlenecks In The First Tier - Clients And The Connection To Internet

This source of trouble has nothing to do with In-Q-My Application Server clustering solution but since it also deteriorates the overall efficiency of the system, it must be mentioned.

Usually bottlenecks in the first tier are due to underpowered machines, which can not meet adequately the necessary processing requirements at this stage. Sometimes there is one more reason for bad performance - the applications and components, that are deployed on In-Q-My Application Server contain a considerable amount of business logic to be executed by clients (applets, scripts, forms) and this clutters clients' machines. One possible solution to this situation is to rewrite applications and components in a way that allows part of the logic to be executed on the servers in the cluster, not by the client.

The second possible bottleneck is the Internet connection of the client. If the bandwidth of the connection is low this could also be a potential problem for thick applications. One of the possible solutions is upgrading the bandwidth, and the other is again modifying the applications and components in a way, which will make them thinner. Another possible solution is mirrowing (even geographic) the servers in the cluster.

Bottlenecks In Dispatchers And Servers

Basically, the dispatchers and servers in the cluster could become a bottleneck, if they are not properly configured for the number of requests and for the complexity of the applications that are running in the cluster. One possible reason could be the inadequate ratio between servers and dispatchers – either the number of dispatchers is not enough for the number of requests, or the servers (their total number or the number of servers, configured for particular services) are not enough and are flooded by the requests. The situation gets worse if the requests require more complex processing before being returned to the client.

The above-mentioned problem is more serious in clusters, which use hardware for load balancing, because usually hardware does not allow such a fine-tuning of the system. Some competitors' clusters, which use load balancing hardware in place of a software dispatcher, do not provide load balancing and failover for clustered servlets and JSPs. The presence of dispatchers in In-Q-My Application Server cluster solution provides more flexibility in configuring the system and tools for administering it, too.

Sometimes the number of requests turns dispatcher(s) into a bottle-neck due to external factors. Not all OS are equally good as platforms for Web application servers. Under certain circumstances the operational system the dispatcher is running on could impose limits on the number of requests, which can pass through a socket.