Architecture of Enterprise Applications IV Performance Improvement

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
e-mail: chen-hp@sjtu.edu.cn

Agenda

- Performance Improvement
  - Memcached
  - Cluster
  - Reverse Proxy

Memcached

- Free & open source, high-performance, distributed memory object caching system, generic in nature, but intended for use in speeding up dynamic web applications by alleviating database load.

- Memcached is an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering.

- The latest stable memcached release is v1.4.15

Memcached

- Example

```
function get_foo(foo_id)
        foo = memcached_get("foo:" . foo_id)
        return foo if defined foo

        foo = fetch_foo_from_database(foo_id)
        memcached_set("foo:" . foo_id, foo)
        return foo
end
```

Memcached

- Example

```
$ telnet localhost 11211
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
get foo
VALUE foo 0 2
hi
END
stats
STAT pid 8861
(etc)
```

Memcached

- Design Philosophies
  - Simple Key/Value Store
  - Smarts Half in Client, Half in Server
  - Servers are Disconnected From Each Other
  - O(1) Everything
  - Forgetting Data is a Feature
  - Cache Invalidation is a Hard Problem

Memcached


```
$MEMCACHE_SERVERS = array(
   "10.1.1.1", //web1
   "10.1.1.2", //web2
   "10.1.1.3", //web3
);



$memcache = new Memcache();
foreach($MEMCACHE_SERVERS as $server){
   $memcache->addServer ( $server );
}
```

Memcached


```
$huge_data_for_front_page = $memcache->get("huge_data_for_front_page");
if($huge_data_for_front_page === false){
   $huge_data_for_front_page = array();
   $sql = "SELECT * FROM hugetable WHERE timestamp >
              lastweek ORDER BY timestamp ASC LIMIT 50000";
   $res = mysql_query($sql, $mysql_connection);
   while($rec = mysql_fetch_assoc($res)){
      $huge_data_for_frong_page[] = $rec;
   }
   // cache for 10 minutes
   $memcache->set("huge_data_for_front_page", $huge_data_for_front_page, 0, 600);
}

// use $huge_data_for_front_page how you please
```

Standard Protocol

- Clients of memcached communicate with server through TCP connections.
  - A UDP interface is also available
  - A given running memcached server listens on some (configurable) port; clients connect to that port, send commands to the server, read responses, and eventually close the connection.

- There are two kinds of data sent in the memcache protocol: text lines and unstructured data.
  - Text lines are used for commands from clients and responses from servers.
  - Unstructured data is sent when a client wants to store or retrieve data.

Standard Protocol

- There are three types of commands.
  - Storage commands (there are six: "set", "add", "replace", "append" "prepend" and "cas") ask the server to store some data identified by a key.
  - Retrieval commands (there are two: "get" and "gets") ask the server to retrieve data corresponding to a set of keys (one or more keys in one request).
  - All other commands don't involve unstructured data. In all of them, the client sends one command line, and expects (depending on the command) either one line of response, or several lines of response ending with "END" on the last line.

Storage commands

- <command name> <key> <flags> <exptime> <bytes> [noreply]\r\n
- cas <key> <flags> <exptime> <bytes> <cas unique> [noreply]\r\n

- <command name> is "set", "add", "replace", "append" or "prepend"
  - "set" means "store this data".
  - "add" means "store this data, but only if the server *doesn't* already hold data for this key".
  - "replace" means "store this data, but only if the server *does* already hold data for this key".
  - "append" means "add this data to an existing key after existing data".
  - "prepend" means "add this data to an existing key before existing data".

Storage commands

- <key> is the key under which the client asks to store the data
- <flags> is an arbitrary 16-bit unsigned integer (written out in decimal) that the server stores along with the data and sends back when the item is retrieved.
- <exptime> is expiration time
  <bytes> is the number of bytes in the data block to follow, *not* including the delimiting \r\n.
- <cas unique> is a unique 64-bit value of an existing entry.

- After this line, the client sends the data block:
- <data block>\r\n
  - <data block> is a chunk of arbitrary 8-bit data of length <bytes> from the previous line.

Retrieval commands

- get <key>*\r\n
- gets <key>*\r\n
  - - <key>* means one or more key strings separated by whitespace.

- After this command, the client expects zero or more items, each of which is received as a text line followed by a data block. After all the items have been transmitted, the server sends the string

  "END\r\n"

  to indicate the end of response.

- Each item sent by the server looks like this:
- VALUE <key> <flags> <bytes> [<cas unique>]\r\n

  <data block>\r\n
  - <key> is the key for the item being sent
  - <flags> is the flags value set by the storage command
  - <bytes> is the length of the data block to follow, *not* including its delimiting \r\n
  - <cas unique> is a unique 64-bit integer that uniquely identifies this specific item.
  - <data block> is the data for this item.

Hashing and Clustering

- **Hashing**
  - All clients should be able to hash keys across multiple servers.
- **Consistent Hashing**
  - Most clients have the ability to use consistent hashing, either natively or via an external library.
- **Key of item in Memcached**
  - E.g. “classname” + “id”
  - Blog1208
- **Hashing Function**
  - Key mod #nodes

Clustering

- A large-scale system typically:
  - Has many user, potentially in many different places
  - Is long-running, that is, required to be “always up”
  - Processes large numbers of transactions per second
  - May see increases in both its user population and system load
  - Represents considerable business value
  - Is operated and managed by multiple persons

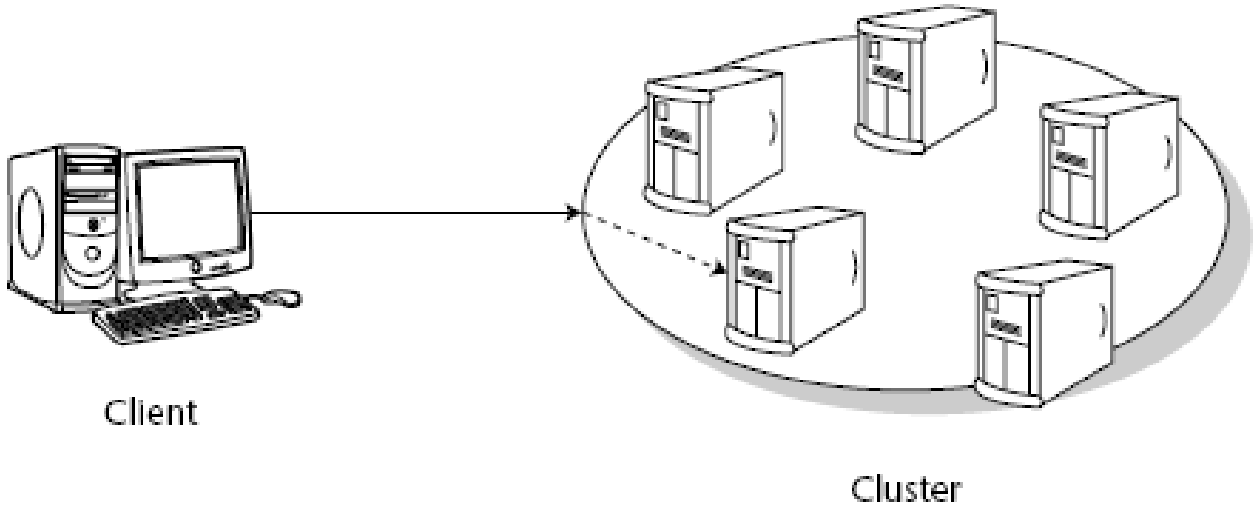
- Essential requirements on large-scale systems are often summarized by the following three properties(RAS):
  - Reliability
  - Availability
  - Serviceability
  - Scalability

Clustering

- Clustering addresses many of the issues faced by large-scale systems at the same time.
- A cluster is a loosely coupled group of servers that provide unified services to their clients.
- The client's view of the cluster is a single, simple system, not a group of collaborating servers. This is referred to as a single-system view or single-system image.
- Computers in a cluster are called nodes.

Clustering

- Clustering can be a very involved technology, potentially encompassing group communication and replication protocols, and network components such as load balancers and traffic redirectors at different layers in the protocol stack.



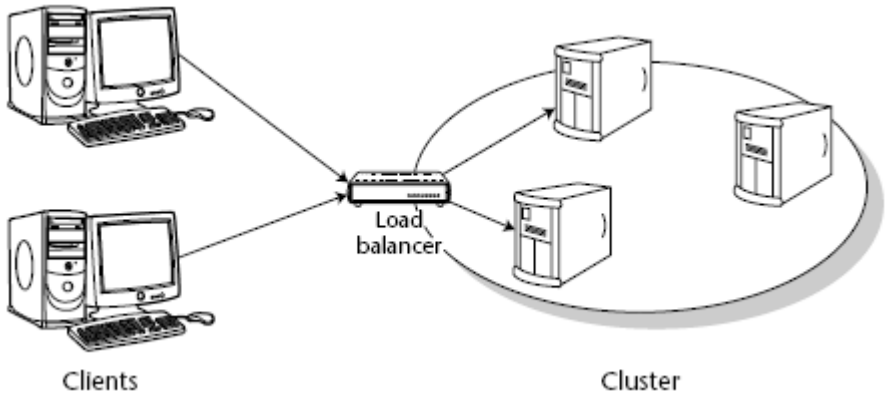
Clustering

- The main principle behind clustering is that of redundancy.
  - Reliability
    - Remove single points of failure
  - Availability
    - Overall availability is $1-(1-f\%)^n$
  - Serviceability
    - More complex than a single application server
    - But we could get ability for hot upgrade
  - Scalability
    - It is cheaper to build a cluster using standard hardware than to rely on multiprocessor machines.
    - Extending a cluster by adding extra servers can be done during operation and hence is less disruptive than plugging in another CPU board.

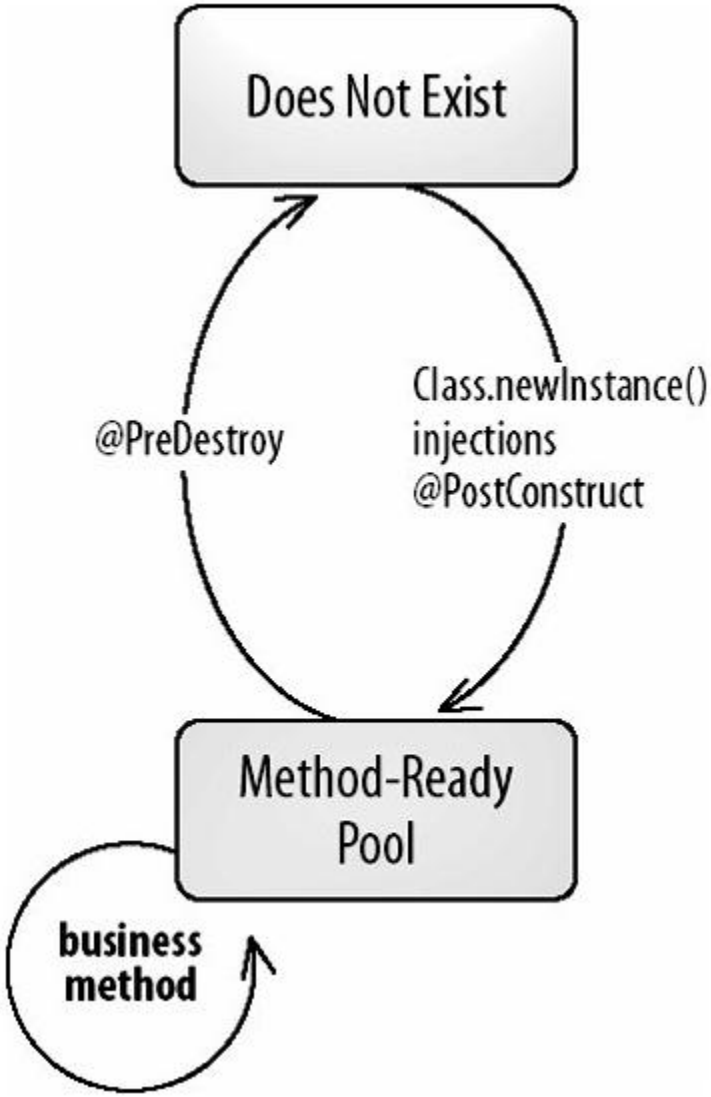
Load balancing and Failover

- Load balancing means distributing the requests among cluster nodes to optimize the performance of the whole system.
  - The algorithm that the load balancer uses to decide which target node to pick for a request can be systematic or random.
  - Alternatively, the load balancer could try to monitor the load on the different nodes in the cluster and pick node that appears less loaded than others.

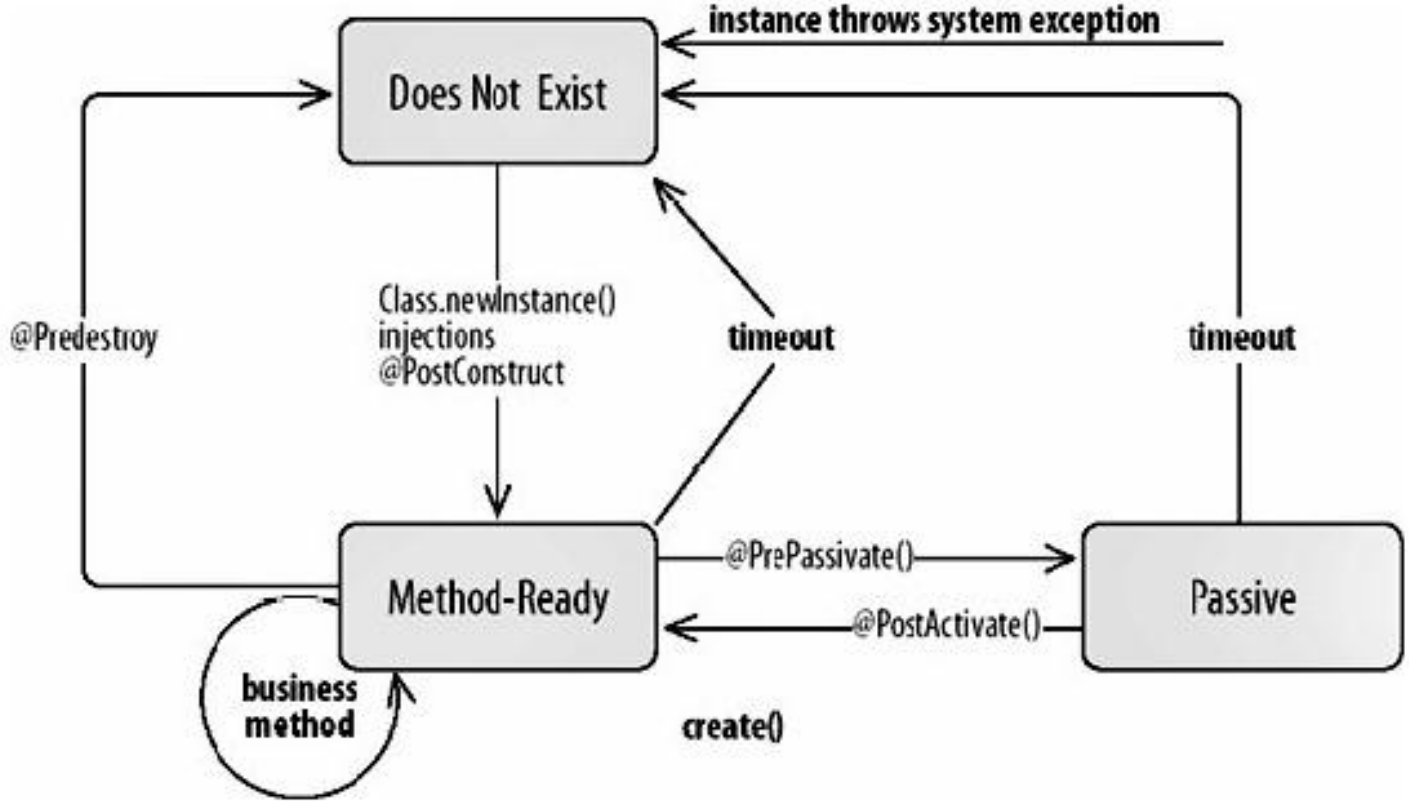
- An important feature for Web load balancers is session stickiness, which means that all requests in a client's session are directed to the same server.



Life Cycle of a Stateless Session Bean

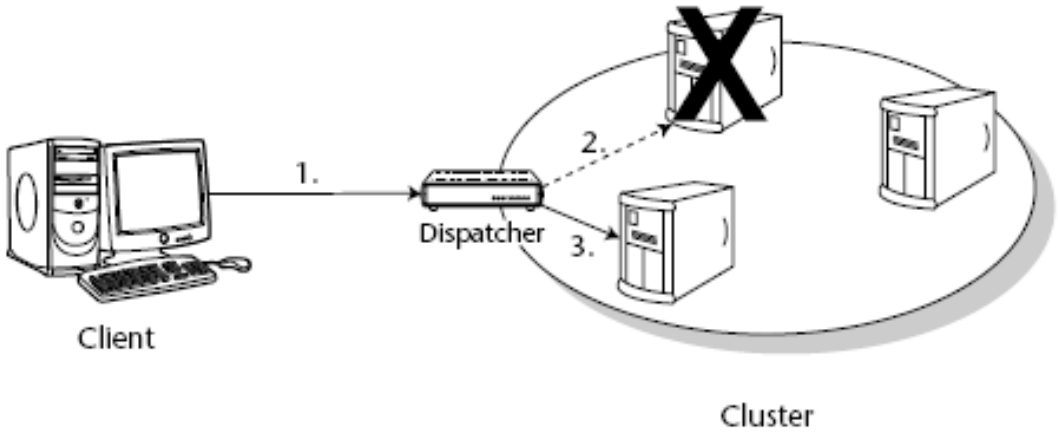


Life Cycle of a Stateful Session Bean



Load balancing and Failover

- For a cluster to provide higher availability to clients that a single server, the cluster must be able to failover from a primary server to another, secondary server when failures occur.
  - Request-level failover. It occurs when a request that is directed to one node for servicing cannot be serviced and is subsequently redirected to another node.
  - Session failover. If session state is shared between clients and servers, request-level failover may not be sufficient to continue operations. In this case, the session state must also be reconstructed at server node.



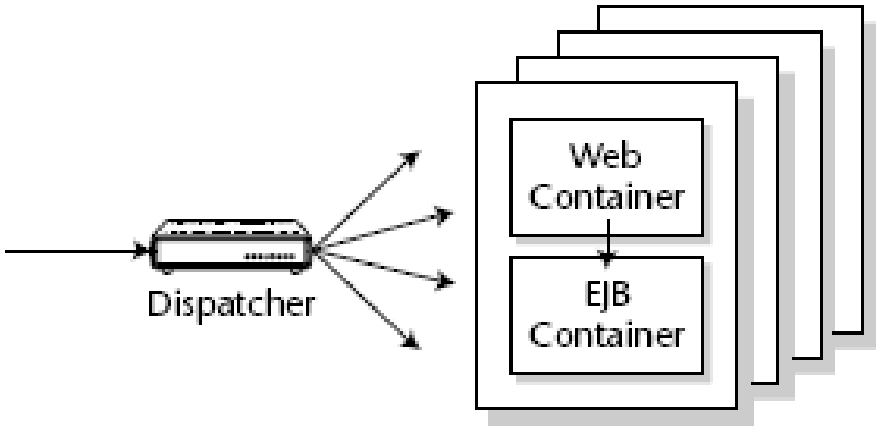
The concept of idempotence

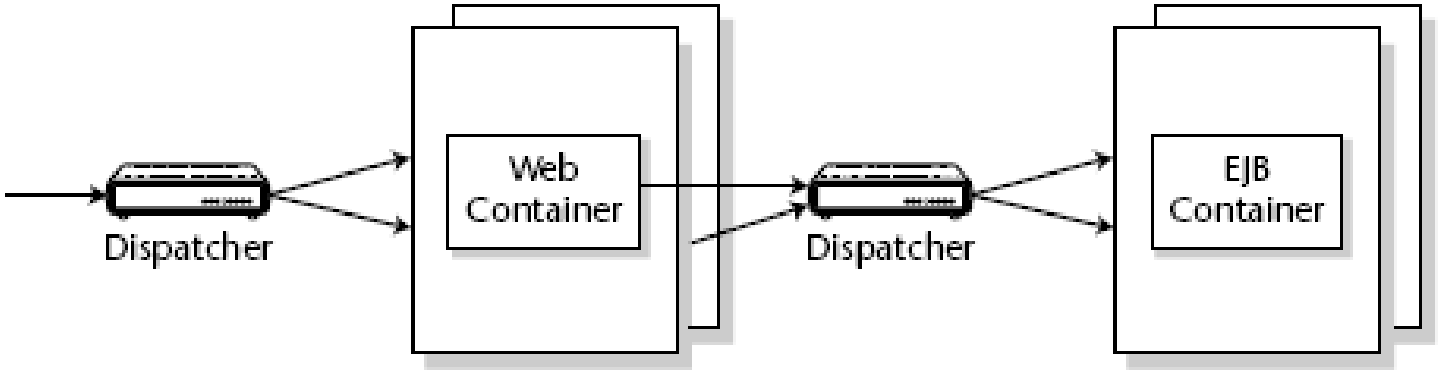
- An idempotent method is one that can be called repeatedly with the same arguments and achieves the same results each time.
  - HTTP GET
  - Generally, any methods that alter a persistent store based on its current state are not idempotent, since two invocations of the same method will alter the persistent store twice.

- A failed request could have occurred at one of three points:
  - After the request has been initiated but before method invocation on the server has begun to execute.
  - After the method invocation on the server has begun to execute, but before the method has completed.
  - After the method invocation on the server has completed but before the response has been successfully transmitted to the remote client.

Multi-tier applications

- In a Web-based system, the following configurations are possible:
  - Collocated architecture
  - Distributed architecture





Multi-tier applications

REin
REliable, INtelligent & Scalable Systems

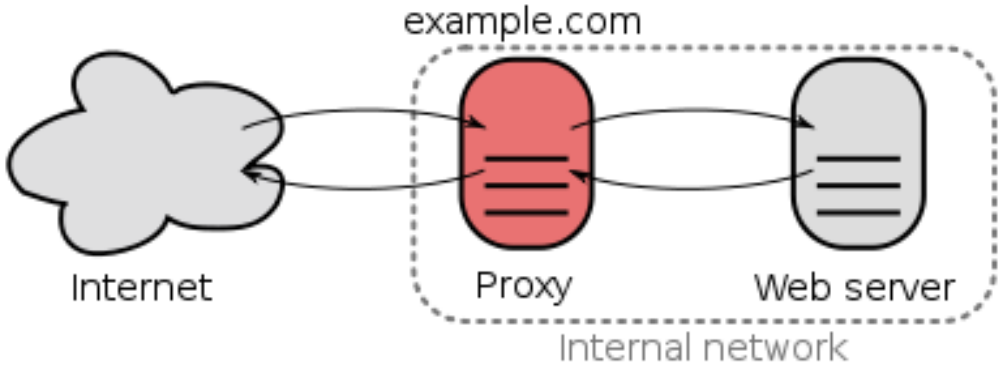
FEATURE	COLLOCATED	DISTRIBUTED	WINNER?
Reliability	High	Low	Collocated
Availability	High	Low	Collocated
Serviceability	High	Low	Collocated
Network efficiency	No sockets	More marshalling overhead	Collocated
Efficient use of hardware	High	Low	Collocated
Security	No firewall	Firewall	Distributed
Serving quick Web requests that do not involve EJB components	Web servers are competing for hardware resources with the application server	Web servers are dedicated	Distributed
Conflicts over responsibility	High	Low	Distributed
Loading balancing	Dispatcher	Dispatcher	Equal

Clustering on Multicore Nodes

- How should we deploy app servers on a multicore node?
- Essentially, an instance of app server is a single process
  - Unless it is implemented in a parallel way
- To cluster multiple instances of app server running on the node
  - To modify the ports used in app server
  - Or to run them individually in Virtual Machines
- For example
  - A Tomcat cluster in a single multicore node

Reverse Proxy

- In computer networks, a **reverse proxy** is a type of proxy server that retrieves resources on behalf of a client from one or more servers.
  - These resources are then returned to the client as though they originated from the reverse proxy itself.



Uses of Reverse Proxies

- Reverse proxies can hide the existence and characteristics of the origin server(s).
- Application firewall features can protect against common web-based attacks.
  - Without a reverse proxy, removing malware or initiating takedowns, for example, can become difficult.
- In the case of secure websites, the SSL encryption is sometimes not performed by the web server itself, but is instead offloaded to a reverse proxy that may be equipped with SSL acceleration hardware.
- A reverse proxy can distribute the load from incoming requests to several servers, with each server serving its own application area.

Uses of Reverse Proxies

- A reverse proxy can reduce load on its origin servers by caching static content, as well as dynamic content.
- A reverse proxy can optimize content by compressing it in order to speed up loading times.
- In a technique known as "spoon feeding", a dynamically generated page can be produced all at once and served to the reverse-proxy, which can then return it to the client a little bit at a time.
- Reverse proxies can be used whenever multiple web servers must be accessible via a single public IP address.

Apache Reverse Proxy

- A reverse proxy, by contrast, appears to the client just like an ordinary web server.
  - No special configuration on the client is necessary.
  - The client makes ordinary requests for content in the name-space of the reverse proxy.
  - The reverse proxy then decides where to send those requests, and returns the content as if it was itself the origin.

- A typical usage of a reverse proxy is to provide Internet users access to a server that is behind a firewall.
  - Reverse proxies can also be used to balance load among several back-end servers, or to provide caching for a slower back-end server.
  - In addition, reverse proxies can be used simply to bring several servers into the same URL space.

Apache Reverse Proxy

- A reverse proxy is activated using the ProxyPass directive or the [P] flag to the RewriteRule directive. It is **not** necessary to turn ProxyRequests on in order to configure a reverse proxy.

```
ProxyRequests Off

<Proxy *>
Order deny,allow
Allow from all
</Proxy>

ProxyPass /foo http://foo.example.com/bar
ProxyPassReverse /foo http://foo.example.com/bar
```

References

- http://memcached.org/
- Rima Patel Sriganesh, Gerald Brose, Micah Silverman: Mastering Enterprise JavaBeans 3.0 4th Edition
- http://en.wikipedia.org/wiki/Reverse_proxy
- http://httpd.apache.org/docs/2.0/mod/mod_proxy.html#forwardreverse

Thank You!