Load Balancing

- In a network of autonomous nodes there is a large probability that at least one node is idle while tasks are queued at some other node.
- The aim of load balancing is to spread the load among all the available processors.
- Load sharing is a better term since the aim is not to have the loads exactly the same on all machines, but to avoid strong imbalance.

Load Balancing

- The goals of load balancing are:
  - To maximise throughput and minimise individual response times.
  - Provide a fair service to all users.
- So the load balancing system should be **transparent** to the user.
- Transparency is a vital aspect of distribution in general and load balancing in particular.
- The main disadvantage of these schemes is that imbalances can occur where some servers get overloaded.

Job Migration

» This involves moving an active process or job from one machine to another.
  – The process migrates to a target machine.
  – Transferring a sufficient amount of the state of a process from one machine to another.
  – We must know in advance that the job is going to run for a good while more, in order to make the effort of migration worthwhile.

Job Migration

- In practice, this is impossible, but we can usually guess that the longer a job has been running the longer it will continue to run.
- When migrating, we must also handle:
  - outstanding I/O,
  - re-direct TTY I/O,
  - problems which arise if the job is expecting reply messages from other processes.
- All of this requires a fairly sophisticated and transparent underlying system.
  - Using JVMs can simplify this e.g. Hadoop framework

Why Migrate?

» Load sharing
  – Move processes from heavily loaded to lightly load systems.
  – Load can be balanced to improve overall performance.
» Communications performance
  – Processes that interact intensively can be moved to the same node to reduce communications cost.
  – May be better to move process closer to where the data resides when the data is large.

Fine Grained Load Balancing

- Some load balancing systems use a fine granularity of load balancing:
  - So that load balancing decisions are made more often, and the system is less unstable.
- Most parallel machines reassign processes to individual processors for each time slice.
  - This can only be efficiently done if it is a shared memory machine.
- The most common granularity in a distributed system made up of multiple servers is on a per *job* or per *request* basis.

Fine Grained Load Balancing

- Each request is treated as a separate job, which can be executed on any available Server.
- The load can be spread much more evenly and can adapt to variations in the system.
- This is widely considered to be the most useful form of load balancing in a distributed system, and is used in many practical systems.
- It requires a load balancing system to decide which server should be sent each new job.

Server / Source Initiation

» Load balancing systems may be divided into two broad categories:

- Server initiated systems, where an under loaded system come looking for jobs.
- Source initiated systems, where overloaded servers try to farm off jobs to other servers.
  - Server initiated systems do not work well in systems which do not support migration, as an overloaded system must prevent jobs from starting in the hope that an under-loaded system will come calling.
  - This is because a job cannot be moved once started.

Load Balancing Algorithms

- It is possible to break most load balancing algorithms into three distinct policies:
  - The *transfer policy* decides which jobs are eligible for remote execution, and under what local conditions they will be transferred.
  - The *information policy* decides what load index (measurement of load) is used, and what information is available about the load on other hosts.
  - The *placement* or *location policy* decides which remote node is to be used for execution of the job.
- The algorithms used for these tasks are mainly independent of each other.

Transfer Policy

- The very first step in load balancing is to decide if a job should be considered for remote execution or done locally.
- There is no point in load balancing small jobs, as the overhead incurred in sending it to another server is frequently more than the actual cost of executing the job.
- The major problem, of course, is that it is not possible, in general, to predict the runtime of a job.
  - Although frequently, we can make a good guess.

Transfer Policy

- For example:
  - cc, nroff, TeX, LaTeX, mod2 -- probably big
  - ls, who -- probably small
  - a.out __ ????
- Also, there is no point in load balancing if the local load is light anyway:
  - So the Transfer Policy will normally include a threshold load index below which the job will be executed locally regardless of the state of other nodes.
- There are also some jobs that have to be executed locally - e.g, mount, shutdown, etc.

Information Policy

» There are two major parts to this:

- What **load index** to use and,
- How to disseminate load information among the hosts in the system.

» **Load Index:**

- Lots of different measurements may be used:
  - e.g. number of processes in the ready queue,
  - use of memory,
  - amount of I/O, etc..
  - The CPU cost of actually calculating the load can get excessively high - so it is likely that a simple system will be used.

Information Policy

- For example, Unix systems typically calculate the load based on the number of processes in the ready queue averaged over the last 1, 5, and 15 minutes (try the **w** command).
  - Note, however, that this is significantly less useful on high speed CPU machines ...
- The instantaneous value is too volatile, so the figure has to be averaged over a period of time.
  - This means that the load is always slightly out of date, and reacts slowly to change.
  - This can contribute to a condition known as **flooding** where a system suddenly gets overloaded ...

Information Policy

» **Load Dissemination:**

- To make an informed load balancing decision, a server must have some information about which other servers are lightly/heavily loaded.
- One obvious method is for a server to periodically broadcast its own load…
- However, this can be expensive in terms of the number of messages sent as the size of the system grows, and, more importantly, the burden placed on the servers to handle and process all the incoming packets.

Information Policy

- Other techniques include: partial broadcast to a subset of servers, e.g., nearest neighbours; centralised information gathering: and passing a vector of loads around a ring.
- The system must also decide how often the information is to be updated.
- *Probing* is an example of a completely different approach to handling load information, which works better in some situations.
- A common probing algorithm is to pick n nodes at random, and to poll each for their loads.

Location Policy

- Once it has been decided that a job should be considered for remote execution, it is up to the location or placement policy to decide at which server it should be done.
  - Note that the location policy may decide to run the job at the Local node.
  - If the local load is the lowest of the available nodes after taking the (often significant) cost of remote execution into account.
- Placement algorithms are broadly divided into two categories: *static* and *adaptive..*

Location Policy

- Static algorithms use no dynamic information and have fixed rules for assigning jobs, e.g., 80% of jobs go to node A and 20% to node B.
  - Therefore, Static algorithms do NOT react to changes in the system, and frequently provide bad performance as a result.
- Adaptive algorithms use dynamic information (such as the loads) to attempt to react to changes in the system and provide better service.
- The location and information policies in real systems are frequently closely linked.

Location Policy

» Flooding

– May happen when a server is identified as being lightly loaded by a number of other servers.
  - Each sends a job to this lightly loaded machine at the same time, without taking into account the load being placed on it by other servers.
  - The lightly loaded machine will quickly become *saturated* and the execution time of programs sent to it will rapidly climb.

– Contributing factors:
  - Out of date load information.
  - Location policies of different nodes not co-ordintated.

Location Policy

- Possible solutions:
  - Keep the load information *perfectly* up to date - this is impossible!
  - Don't send a job directly to a server, but first ask it if it will accept the job. (or send the job to it, but allow it to reject the job, or find another server to handle it).
  - In this fashion, a lightly loaded server can accept the first few jobs offered, and then refuse the rest.
- Both of these methods involve increasing the number of messages sent.
  - An alternative is to introduce a random element into the location algorithm, e.g.. find n lightly loaded servers, and randomly send the job to one of these.

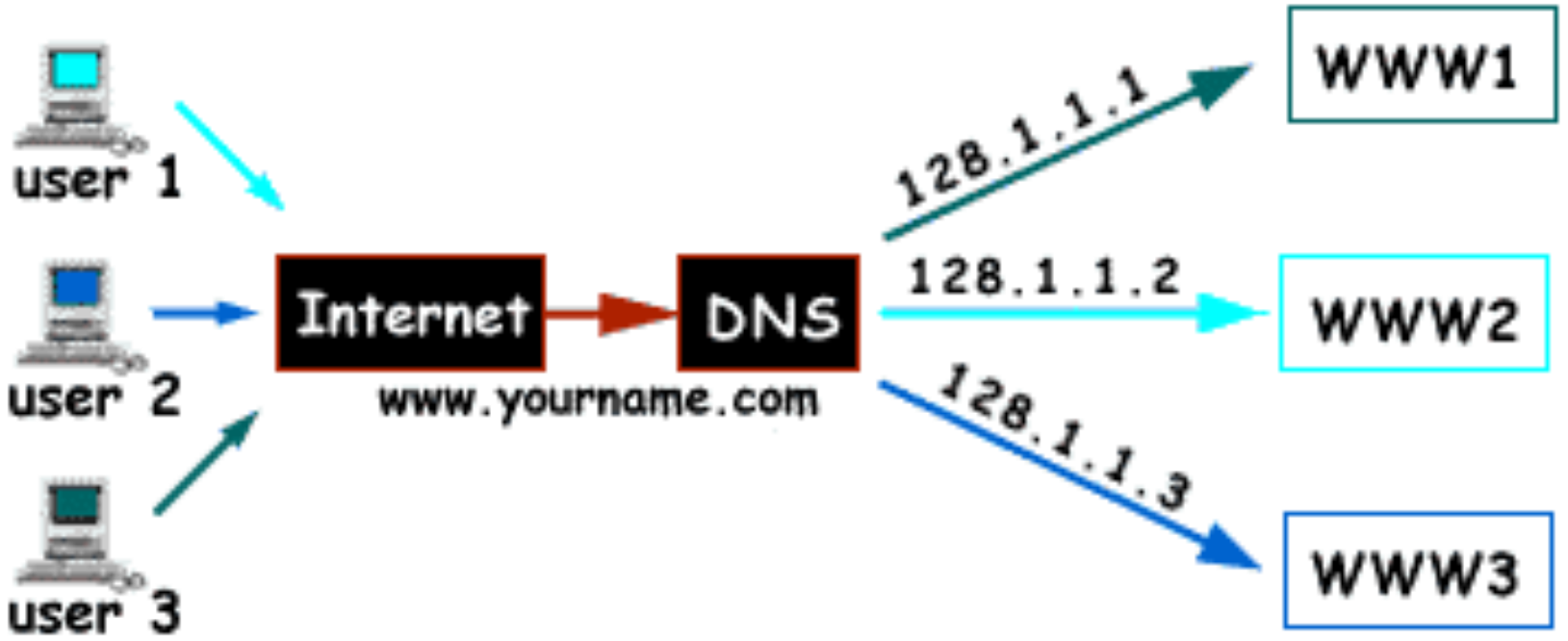
Other Design Issues

- As with all distributed system components, we can provide load balancing either by layering it on top of the OS or modifying the OS.
  - The first is potentially more general, while the second is potentially more efficient.
- Load balancing software can be broken down into 2 major layers:
- Load/job exchange layer:
  - When and where to send jobs.
- Remote execution layer:
  - Mechanism for remote execution.
  - Handles transparency.

DNS Load Balancing

- Configure a domain in the Domain Name System (DNS) such that client requests to the domain are distributed across a group of servers.
  - A domain can correspond to a website or another service that is made accessible via the Internet.
  - It facilitates faster access to a service by providing several IP addresses for a single domain name, which routes traffic between two or more servers.
  - It uses round-robin access to the list of server IP addresses returned by DNS.
- Can have issues related to caching of results and reliability of an individual server goes down

DNS Load Balancing



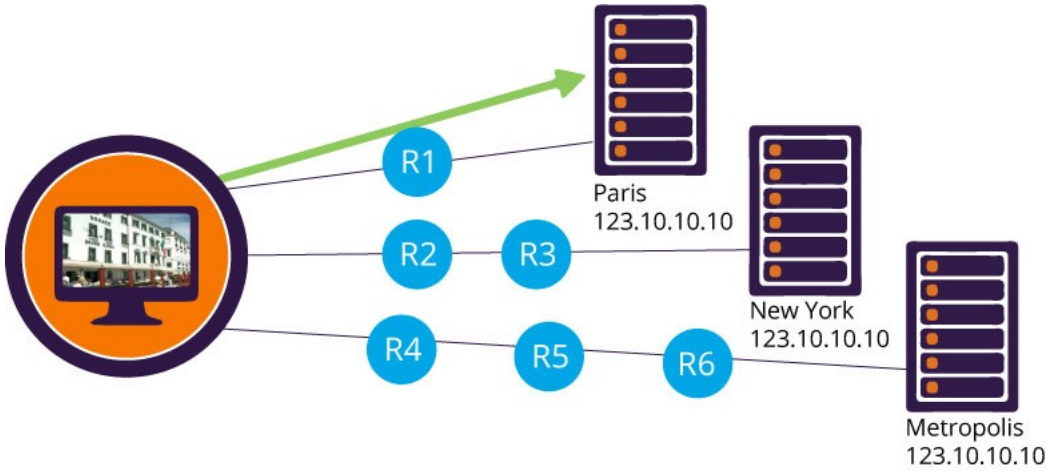
BGP Anycast

- Using Anycast, a collection of servers share the same IP address and send data from a source computer to the server that is topographically the closest.
  - This helps cut down on latency and bandwidth costs, improves load time for users, and improves availability.
  - It is important to remember that topographically closer does not inherently mean geographically closer, though this is often the case.
  - Great for load balancing and reliability for large CDN providers that have server resources in different parts of the world.

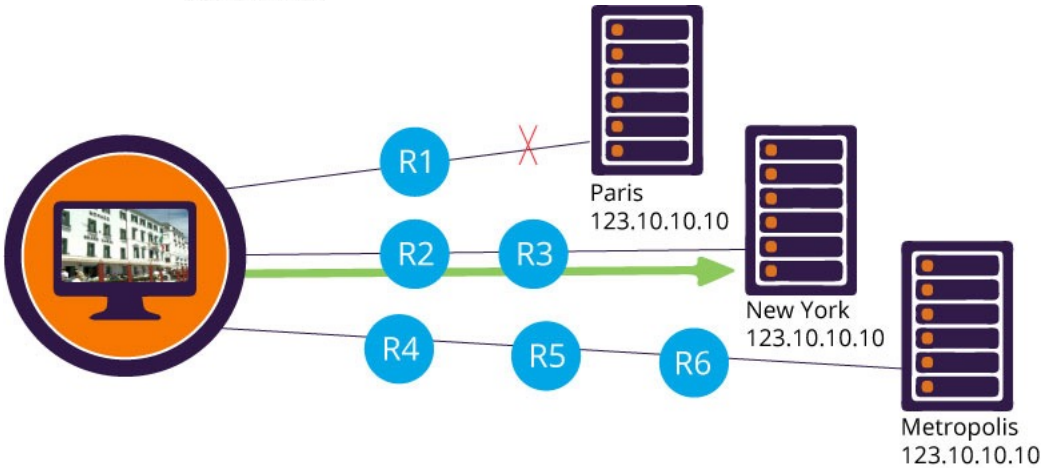
BGP Anycast

- Anycast is linked with the BGP protocol which ensures that all of a router's neighbors are aware of the networks that can be reached through that router and the topographical distance to those networks.
  - The main principle of anycast is that the same IP address range is advertised in the BGP messages of multiple routers at different locations
  - As the IP range propagates across the Internet, routers become aware of which of their neighbours provides the shortest path to the advertised IP address
  - Apart from large CDNs, this is also used to ensure to reach DNS root servers around the world

BGP Anycast



If the Paris server goes down internet traffic will automatically find the Next best path



Case Study: LB Unix Shell

- Study of Load Balancing Performance.
  - » Case Study based on adding load balancing capabilities to a Unix Shell.
  - » If the local load isn't lower that the transfer threshold, T, the modified C shell tries to transfer the job to a remote linux server.
  - » It reads a file that contains the list of jobs types (cc, tex, . . .) it is to threat like this.
  - » Five information and location policies:

Case Study: LB Unix Shell

1. DISTED: each server periodically broadcasts its load index to all other serverss (if it is significantly different from the previous value).
   - If a server decides to try to send a job remote, it uses its list of load indices to select the server that appears to have the lightest load.
   - Job is done locally if all servers are heavily loaded.
2. GLOBAL: each server periodically sends its load index to a master server, which then periodically broadcasts load index list to all servers.
   - In other ways it is similar to DISTED.

Case Study: LB Unix Shell

3. CENTRAL: The load index list is collected at a master server, and all placement requests are sent to it.
   - This is like DISTED but all decisions are made by the master server.
4. LOWEST: If it decides to try to send a job remote, it randomly probes n servers, and then selects the server with the lightest load.
5. RANDOM: pick a server at random and send the job to it (no retransfer of jobs).

Case Study: LB Unix Shell

Main Results:

	Resp Time	Improv.	Std Dev.	Imp.
NoLB	**53.3**	**-**	**90.1**	**-**
DISTED	**36.4**	**31.7%**	**50.6**	**43.8%**
GLOBAL	**32.6**	**38.9%**	**43.6**	**51.7%**
CENTRAL	**33.7**	**36.8%**	**48.5**	**46.8%**
LOWEST	**31.8**	**40.3%**	**42.8**	**52.5%**
RANDOM	**39.9**	**25.2%**	**62.0**	**31.2%**

Case Study: LB Unix Shell

- All algorithms show significant improvements over not using load balancing.
- The Standard deviation was measured since it has a major effect on the user.
- Note that in adaptive load balancing the system is taking advantage of short term imbalances in server loads, whereas static load balancing takes advantage of long term load (/performance) imbalances.
- They found that the effect of load balancing increased as the system load increased.

Case Study: LB Unix Shell

» **Varying the parameters:**

- Load Exchange Period:
  - Found that there was an optimum (about 5 secs).
  - Any shorter put too high a load on the system.
  - Any longer meant that the information was out of date and poor decisions, and even flooding, could anise.
  - There was significant benefit from load balancing even if the period was a long as 60 secs.
- Local Load Threshold:
  - Too low implied unnecessary transfer of jobs: too high meant less load balancing.
  - Values of 1 to 2.5 worked well.

Case Study: LB Unix Shell

- Probe Limit (value for n)
  - Too low implied too similar to RANDOM.
  - Too high implied unnecessary probing.
  - Figures around 4 worked well.
- Immobility Factor: (the percentage of total CPU time that is consumed by the immobile jobs - jobs in the "don't transfer" list).
  - Even at 70% load balancing was still worthwhile.
- Fault tolerant and dynamically re-configurable load balancing are provided by modern cloud computing frameworks.