System Models for Distributed and Cloud Computing

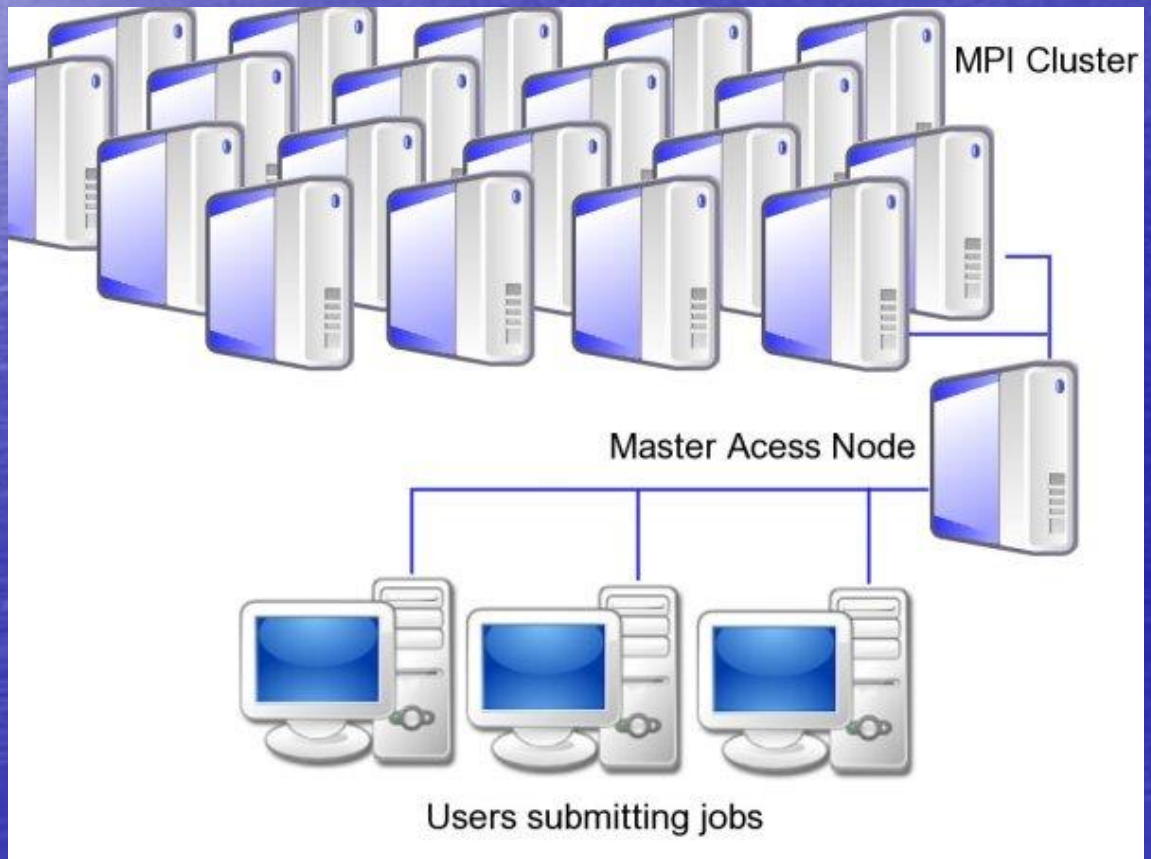
Dr. Sanjay P. Ahuja, Ph.D.

2010-14 FIS Distinguished Professor of Computer Science

School of Computing, UNF

Classification of Distributed Computing Systems

- These can be classified into 4 groups: clusters, peer-to-peer networks, grids, and clouds.
- A computing cluster consists of interconnected stand-alone computers which work cooperatively as a single integrated computing resource. The network of compute nodes are connected by LAN/SAN and are typically homogeneous with distributed control running Unix/Linux. They are suited to HPC.



Peer-to-peer (P2P) Networks

- In a P2P network, every node (peer) acts as both a client and server. Peers act autonomously to join or leave the network. No central coordination or central database is needed. No peer machine has a global view of the entire P2P system. The system is self-organizing with distributed control.

- Unlike the cluster or grid, a P2P network does not use dedicated interconnection network.

- P2P Networks are classified into different groups:

  Distributed File Sharing: content distribution of MP3 music, video, etc. E.g. Gnutella, Napster, BitTorrent.

  Collaboration P2P networks: Skype chatting, instant messaging, gaming etc.

  Distributed P2P computing: specific application computing such as SETI@home provides 25 Tflops of distributed computing power over 3 million Internet host machines.

-

Computational and Data Grids

- Grids are heterogeneous clusters interconnected by high-speed networks. They have centralized control, are server-oriented with authenticated security. They are suited to distributed supercomputing. E.g. TeraGrid.

- Like an electric utility power grid, a *computing grid* offers an infrastructure that couples computers, software/middleware, people, and sensors together.

- The grid is constructed across LANs, WANs, or Internet backbones at a regional, national, or global scale.

- The computers used in a grid include servers, clusters, and supercomputers. PCs, laptops, and mobile devices can be used to access a grid system.

Clouds

- A Cloud is a pool of virtualized computer resources. A cloud can host a variety of different workloads, including batch-style backend jobs and interactive and user-facing applications.

- Workloads can be deployed and scaled out quickly through rapid provisioning of VMs. Virtualization of server resources has enabled cost effectiveness and allowed cloud systems to leverage low costs to benefit both users and providers.

- Cloud system should be able to monitor resource usage in real time to enable rebalancing of allocations when needed.

- Cloud computing applies a virtualized platform with elastic resources on demand by provisioning hardware, software, and data sets dynamically. Desktop computing is moved to a service-oriented platform using server clusters and huge databases at datacenters.

Advantage of Clouds over Traditional Distributed Systems

- Traditional distributed computing systems provided for on-premise computing and were owned and operated by autonomous administrative domains (e.g. a company).
- These traditional systems encountered performance bottlenecks, constant system maintenance, poor server (and other resource) utilization, and increasing costs associated with hardware/software upgrades.
- Cloud computing as an on-demand computing paradigm resolves or relieves many of these problems.

Software Environments for Distributed Systems and Clouds:

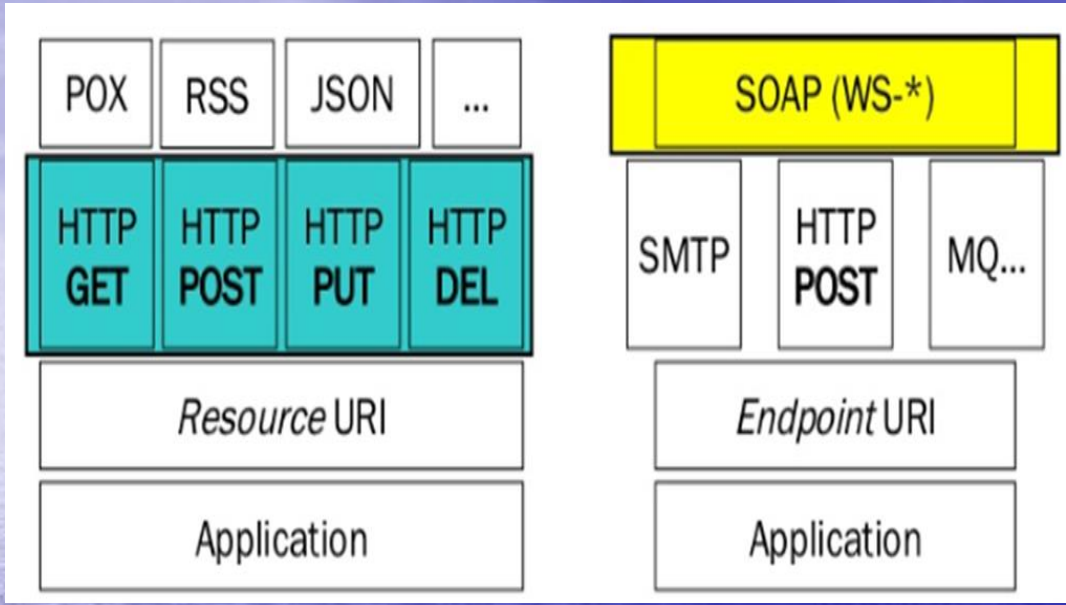
Service-Oriented Architecture (SOA) Layered Architecture

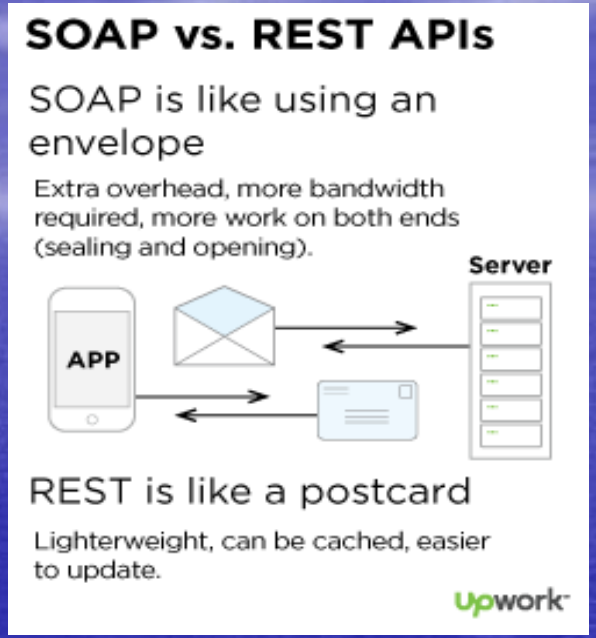
- In web services, Java RMI, and CORBA, an entity is, respectively, a service, a Java remote object, and a CORBA object. These build on the TCP/IP network stack. On top of the network stack we have a base software environment, which would be .NET/Apache Axis for web services, the JVM for Java, and the ORB network for CORBA. On top of this base environment, a higher level environment with features specific to the distributed computing environment is built.

- Loose coupling and support of heterogeneous implementations make services more attractive than distributed objects.

CORBA Stack	RMI Stack	Web Services Stack
IDL	Java interface	WSDL
CORBA Services	RMI Registry	UDDI
CORBA Stubs/Skeletons	RMI Stubs/Skeletons	SOAP Message
CDR binary encoding	Java native encoding - serialization	XML Unicode encoding
IIOP	JRMP	HTTP
RPC or Message Oriented Middleware (Websphere MQ or JMS)		
ORB	JVM	.NET/Apache Axis
TCP/IP/DataLink/Physical		

Service-Oriented Architecture (SOA) Layered Architecture: SOAP vs REST services





- REST supports many data formats, whereas SOAP only allows XML.
- REST supports JSON (smaller data formats and offers faster parsing compared to XML parsing in SOAP which is slower).
- REST provides superior performance, particularly through caching for information that's not altered and not dynamic.
- REST is used most often for major services such as Amazon, Twittter.
- REST is generally faster and uses less bandwidth.
- SOAP provides robust security through WS-Security and so useful for enterprise apps such as banking and financial apps; REST only has SSL.
- SOAP offers built-in retry logic to compensate for failed communications.

Performance Metrics and Scalability Analysis

- Performance Metrics:
- CPU speed: MHz or GHz, SPEC benchmarks like SPECINT
- Network Bandwidth: Mbps or Gbps
- System throughput: MIPS, TFlops (tera floating-point operations per second), TPS (transactions per second), IOPS (IO operations per second)
- Other metrics: Response time, network latency, system availability

- Scalability:
- Scalability is the ability of a system to handle growing amount of work in a capable/efficient manner or its ability to be enlarged to accommodate that growth.
- For example, it can refer to the capability of a system to increase total throughput under an increased load when resources (typically hardware) are added.

Scalability

Scale Vertically

To scale vertically (or **scale up**) means to add resources to a single node in a system, typically involving the addition of CPUs or memory to a single computer.

Tradeoffs

There are tradeoffs between the two models. Larger numbers of computers means increased management complexity, as well as a more complex programming model and issues such as throughput and latency between nodes.

Also, some applications do not lend themselves to a distributed computing model.

In the past, the price difference between the two models has favored "scale up" computing for those applications that fit its paradigm, but recent advances in virtualization technology have blurred that advantage, since deploying a new virtual system/server over a hypervisor is almost always less expensive than actually buying and installing a real one.

Scalability

- One form of scalability for parallel and distributed systems is:
- Size Scalability

  This refers to achieving higher performance or more functionality by increasing the *machine size*. Size in this case refers to adding processors, cache, memory, storage, or I/O channels.
- Scale Horizontally and Vertically

  Methods of adding more resources for a particular application fall into two broad categories:

  Scale Horizontally

  To scale horizontally (or **scale out**) means to add more nodes to a system, such as adding a new computer to a distributed software application. An example might be scaling out from one Web server system to three.

  The scale-out model has created an increased demand for shared data storage with very high I/O performance, especially where processing of large amounts of data is required.

Amdahl's Law

It is typically cheaper to add a new node to a system in order to achieve improved performance than to perform performance tuning to improve the capacity that each node can handle. But this approach can have diminishing returns as indicated by Amdahl's Law.

Consider the execution of a given program on a uniprocessor workstation with a total execution time of *T* minutes. Now, let's say that the program has been parallelized or partitioned for parallel execution on a cluster of many processing nodes.

Assume that a fraction *a* of the code must be executed sequentially, called the *sequential block*. Therefore, *(1 - a)* of the code can be compiled for parallel execution by *n* processors. The total execution time of program is calculated by:

$$a\,T + (1 - a)\,T / n$$

where the first term is the sequential execution time on a single processor and the second term is the parallel execution time on *n* processing nodes.

All system or communication overhead is ignored here. The I/O and exception handling time is also not included in the speedup analysis.

Amdahl's Law

Amdahl's Law states that the ***Speedup Factor*** of using the n-processor system over the use of a single processor is expressed by

$$\text{Speedup} = S = T / [a\,T + (1 - a)\,T / n]$$
$$= 1 / [a + (1 - a) / n]$$

The maximum speedup of n is achievable only when $a = 0$, i.e. the entire program is parallelizable.

As the cluster becomes sufficiently large, i.e. $n \rightarrow \infty$, then $S \rightarrow 1 / a$, an upper bound on the speedup S. This upper bound is independent of the cluster size, n. The sequential bottleneck is the portion of the code that cannot be parallelized.

Example, $a = 0.25$ and so $(1 - 0.25) = 0.75$ then the maximum speedup, $S = 4$ even if one uses hundreds of processors.

Amdahl's Law teaches us that we should make the sequential bottleneck as small as possible. Increasing the cluster size alone may not result in a good speedup in this case.

Amdahl's Law

- Example: suppose 70% of a program can be sped up if parallelized and run on multiple CPUs instead of one CPU.
- N = 4 processors

  $S = 1 / [0.3 + (1 - 0.3) / 4] = 2.105$

- Doubling the number of processors to N = 8 processors

  $S = 1 / [0.3 + (1 - 0.3) / 8] = 2.581$

  Double the processing power has only improved the speedup by roughly one-fifth. Therefore, throwing in more hardware is not necessarily the optimal approach.

System Efficiency

- To execute a fixed workload on *n* processors, parallel processing may lead to a system efficiency defined as:

  System Efficiency, $E = S / n = 1 / [\alpha n + (1 - \alpha)]$

  System efficiency can be rather low if the cluster size is very large.

  Example: To execute a program on a cluster with n = 4, $\alpha = 0.25$ and so $(1 – 0.25) = 0.75$,

  $E = 1 / [0.25 * 4 + 0.75] = 0.57$ or 57%

  Now if we have 256 nodes (i.e. n = 256)

  $E = 1 / [0.25 * 256 + 0.75] = 0.015$ or 1.5%

  This is because only a few processors (4, as in the previous case) are kept busy, while the majority of the processors (or nodes) are left idling.

Fault Tolerance and System Availability

- High availability (HA) is desired in all clusters, grids, P2P networks, and cloud systems. A system is highly available if it has a long Mean Time to Failure (MTTF) and a short Mean Time to Repair (MTTR).
- System Availability = MTTF / (MTTF + MTTR)
- All hardware, software, and network components may fail. Single points of failure that bring down the entire system must be avoided when designing distributed systems.
- High-availability is, ultimately, the holy grail of the cloud. For clouds, availability relates to the time that the datacenter is accessible or delivers the intended IT service as a proportion of the duration for which the service is purchased.
- Adding hardware redundancy, increasing component reliability, designing for testability all help to enhance system availability and dependability.
- In general, as a distributed system increases in size, availability decreases due to a higher chance of failure and a difficulty in isolating failures.

Computing Availability

- **CASE 1**
- Consider that a cloud instance (such as an EC2 instance) over a 6 week timeline has a total uptime of 900 hours and a total downtime of 108 hours with 6 failures in this timeline. Compute the availability.
- Total timeline in hours = 6 weeks * 7 days/week * 24 hours/day = 1008 hours
- MTTF = 900 / 6 = 150
- MTTR = 108 / 6 = 18
- Availability = 150 / (150 + 18) = (150 / 168) = 0.892 or 89.2%

- **CASE 2**
- Same as Case 1 but we have 2 failures instead. Compute availability.
- MTTF = 900 / 2 = 450
- MTTR = 108 / 2 = 54
- Availability = 450 / (450 + 54) = (450 / 504) = 0.892 or 89.2%

Reliability vs. Availability

- Reliability is a measure of the probability that an item will perform its intended function without failures for a specified interval under stated conditions.

- Higher the MTTF, higher the reliability.
- A commonly used measure of reliability:

- Failure Rate (λ) = number of failures / total time

- Calculate the Failure Rate for Case 1 and Case 2.
- Case 1 had a MTTF of 150 and a failure rate = 6 failures / 1008 = 0.00595
- Case 2 had a MTTF of 450 and a failure rate = 2 failures / 1008 = 0.00198
- Even though both cases had the same Availability, case 2 has a better reliability because it has a better MTTF and a lower failure rate.

Reliability vs. Availability

- A piece of equipment can be available but not reliable. For example, a machine is down 6 minutes every hour. This translates into an availability of 90% (i.e. 54 minutes / 60 minutes = 0.9) but a reliability of less than 1 hour.

- **Generally speaking a reliable machine has high availability but an available machine may or may not be very reliable.**

- Something to think about:
- If you are flying do you want the aircraft to have high levels of availability or reliability? Think about it. If the aircraft has poor availability, then this may have an influence on whether the plane departs (and therefore lands) on time. On the other hand, if the aircraft has poor reliability, then this may have an influence on whether the plane lands at all!

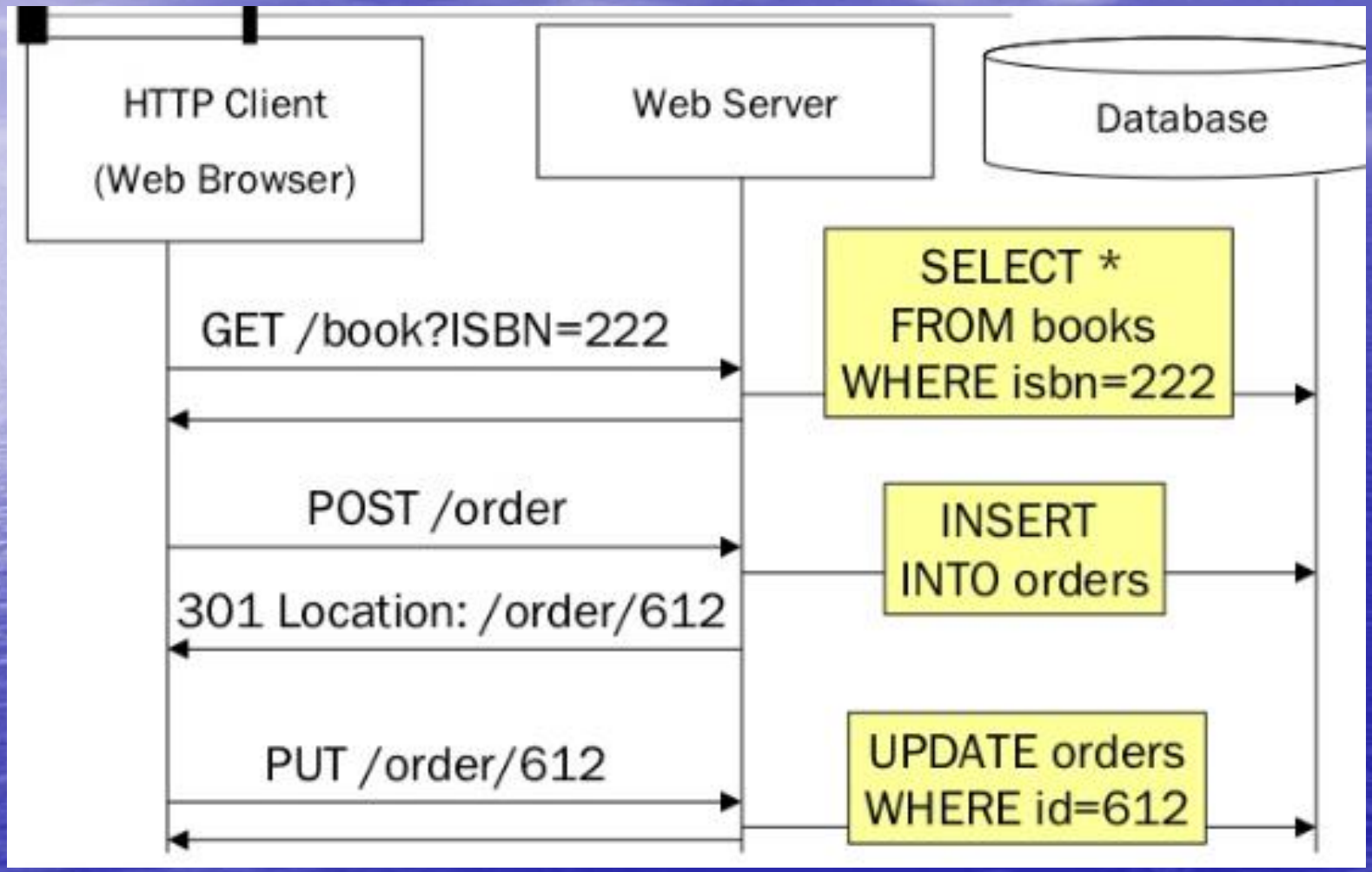
Interpreting Cloud Availability

- Cloud vendors use slang like three 9's, four 9's, or five 9's when talking about the availability of their datacenter or cloud services.

- E.g. 99.9% availability is referred to as three 9's and is an uptime of greater than 99.9%. This corresponds to a downtime per year of 8 hours and 45 minutes per year. This is calculated as:
- (365 × 24) – 0.999 (365 × 24) = 8760 - 8751.24 = 8.76 hours = 8 hours and 45 minutes. This is a downtime of more than 1 working day and can have real financial consequences for the enterprise cloud user.

- Calculate the downtime per year given an availability of four 9's (i.e. an uptime of 99.99%).
- This is calculated as:
- (365 × 24) – 0.9999 (365 × 24) = 8760 – 8759.124 = 0.876 hours = 52 minutes and 30 secs

RESTful Web Service Example



SOAP Web Service Example