HIGH PERFORMANCE COMPUTING

Prepared By: Dr. Prakash Chandra Jena

EINSTEIN ACADEMY OF TECHNOLOGY & MANAGEMENT
(Managed by Udayanath Educational & Charitable Trust, Bhubaneswar)




AICTE

BPUT

SCTE & VT

www.eatm.in

Department of Computer Science and Engineering
Einstein Academy of Technology & Management
Baniatangi , Khurdha , Odisha-752060

CONTENTS

Lecture 39	Task Consolidation
Lecture 40	Introduction to Cloud Sim, Cloudlet, Virtual Machine and its Provisioning, Time and Space-shared Provisioning

DISCLAIMER

This document does not claim any novelty; originality and it cannot be used as a substitute for prescribed textbooks. The information presented here is merely a collection of knowledge base including research papers, books, online sources etc. by the committee members for their respective teaching assignments. Various online/offline sources as mentioned at the end of the document as well as freely available material from internet were helpful for preparing this document. Citation is needed for some parts of this document. The ownership of the information lies with the respective authors/institution/publisher. Further, this study material is not intended to be used for commercial purpose and the committee members make no representations or warranties with respect to the accuracy or completeness of the information contents of this document and specially disclaim any implied warranties of merchantability or fitness for a particular purpose. The committee members shall not be liable for any loss or profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

Introduction to Cluster Computing [1]

The essence of Pfister's [2] and Buyya's [3] work defines clusters as follows:

A cluster is a type of parallel and distributed system, which consists of a collection of inter-connected stand-alone computers working together as a single integrated computing resource.

Buyya defined one of the popular definitions for Grids at the 2002 Grid Planet conference, San Jose, USA as follows:

A Grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed 'autonomous' resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements.

Buyya [1] propose the clouds definition as follows:

A Cloud is a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resource(s) based on service-level agreements established through negotiation between the service provider and consumers.

The computing power of a sequential computer is not enough to carry out scientific and engineering applications. In order to meet the computing power, we can improve the speed of processors, memory and other components of the sequential computer. However, computing power is limited when we consider very complex applications. A cost-effective solution is to connect multiple sequential computers together and combine their computing power. We call it parallel computers.

There are three ways to improve performance as per Pfister [1].

	Non-Technical Term	**Technical Term**
1	Work Harder	Faster Hardware
2	Work Smarter	More Efficiently
3	Get Help	Multiple Computer to Solve a Task

The evolution of various computing or systems is as follows.

Year	**Computing**
1950	Multi-Processors Systems
1960-80	Supercomputers
1988	Reconfigurable Computing
1990	Cluster Computers
1998	Distributed Computing
2000	Grid Computing
2006	SOA and Web Services, Deep Computing, Multi-Core Architecture, Skeleton Based

	Programming, Network Devices
2008	Cloud Computing
2009-15	Heterogeneous Multicore General-Purpose computing on Graphics Processing Units (GPGPU), APU, Big Data

There are two eras of computing as follows [3].

1. Sequential Computing Era
2. Parallel Computing Era

The computing era is started with improvement of following things [3].

1. Hardware Architecture
2. System Software
3. Applications
4. Problem Solving Environments (PSEs)

The components of computing eras are going through the following phases [3].

1. Research and Development
2. Commercialization
3. Commodity

A cluster connects a number of computing nodes or personal computers that are used as servers via a fast local area network. It may be a two-node system that connects two personal computers or fast supercomputer. However, the supercomputers may include many clusters.

According to the latest TOP500 list [4] (i.e., November 2014), the best 10 supercomputers are as follows.

RANK	**SITE**	**SYSTEM**	**CORES**	**RMAX (TFLOP/S)**	**RPEAK (TFLOP/S)**	**POWER (KW)**
1	National Super Computer Center in Guangzhou China	Tianhe-2 (MilkyWay-2)- TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
2	DOE/SC/Oak Ridge National Laboratory United States	Titan - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
3	DOE/NNSA/LLNL United States	Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
4	RIKEN Advanced Institute for Computational	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660

	Science (AICS) Japan					
5	DOE/SC/Argonne National Laboratory United States	Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432	8,586.6	10,066.3	3,945
6	Swiss National Supercomputing Centre (CSCS) Switzerland	Piz Daint - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x Cray Inc.	115,984	6,271.0	7,788.9	2,325
7	Texas Advanced Computing Center/Univ. of Texas United States	Stampede - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P Dell	462,462	5,168.1	8,520.1	4,510
8	Forschungszentrum Juelich (FZJ) Germany	JUQUEEN - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	458,752	5,008.9	5,872.0	2,301
9	DOE/NNSA/LLNL United States	Vulcan - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	393,216	4,293.3	5,033.2	1,972
10	Government United States	Cray CS-Storm, Intel Xeon E5-2660v2 10C 2.2GHz, Infiniband FDR, Nvidia K40 Cray Inc.	72,800	3,577.0	6,131.8	1,499

Scalable Parallel Computer Architectures [3, 5]

The scalable parallel computer architectures are as follows.

1. Massively Parallel Processors (MPP)
    - It is a shared-nothing architecture.
    - Each node in MPP runs a copy of the operating systems (OSs).
2. Symmetric Multiprocessors (SMP)
    - It is a shared-everything architecture.
3. Cache-Coherent Nonuniform Memory Access (CC-NUMA)
4. Distributed Systems
    - Each node runs its own OS.
    - It is the combinations of MPPs, SMPs, clusters and individual computers.
5. Clusters

The detailed comparisons of the scalable parallel computer architectures are shown below [3, 5].

Characteristic	MPP	SMP CC-NUMA	Cluster	Distributed
Number of Nodes	100 to 1000	10 to 100	100 or less	10 to 1000
Node Complexity	Fine Grain or Medium	Medium or Coarse Grain	Medium Grain	Wide Range
Internode Communication	Message Passing or Shared Variables for Distributed Shared Memory	Centralized and Distributed Shared Memory (DSM)	Message Passing	Shared Files, RPC, Message Passing and IPC
Job Scheduling	Single Run Queue on Host	Single Run Queue	Multiple Queue but Coordinated	Independent Queues
SSI Support	Partially	Always in SMP and some NUMA	Desired	No
Node OS Copies and Type	N Micro-Kernels Monolithic or Layered Oss	One Monolithic SMP and Many for NUMA	N OS Platform-Homogeneous or Micro-Kernel	N OS Platforms Homogeneous
Address Space	Multiple – Single for DSM	Single	Multiple or Single	Multiple
Internode Security	Unnecessary	Unnecessary	Required if Exposed	Required
Ownership	One Organization	One Organization	One or More Organizations	Many Organizations

Granularity [6] refers to the extent to which a system or material or a large entity is decomposed into small pieces. Alternatively, it is to the extent for which smaller entities are joined to form a larger entity. It is of two types, namely Coarse-Grained and Fine-Grained.

A Coarse-Grained defines a system regards large subcomponents of which the larger ones are composed. A Fine-Grained defines a system regards smaller components of which the larger ones are composed.

The granularity of data refers to the size in which data fields are sub-divided. For example, the postal address of our Department can be recorded with Coarse-Granularity in a single field as follows.

Address = Department of CSE, IT and MCA, VSSUT, Burla, 768018, Odisha, India

The same address can be recorded with Fine-Granularity as multiple fields.

Department Address = CSE, IT and MCA

University = VSSUT

City = Burla

Postal Code = 768018

State = Odisha

Country = India

Note that, Fine-Granularity becomes an overhead for data storage.

Message Passing [7]

- Variables have to be marshaled
- Cost of communication is obvious
- Processes are protected by having private address space
- Processes should execute at the same time

DSM [7]

- Variables are shared directly
- Cost of communication is invisible
- Processes could cause error by altering data
- Executing the processes may happen with non-overlapping lifetimes

Kernel [8] is a program that manages I/O requests from software and translates them into data processing instructions for the CPU and other electronic components of a computer.

A Monolithic Kernel [8] executes all the OS instructions in the same address space in order to improve the performance.

A Micro-Kernel [8] runs most of the OS's background processes in user space to make the OS more modular. Therefore, it is easier to maintain.

Cluster Computer and its Architecture [3]

A Cluster consists of a collection of interconnected stand-alone computers working together as a single computing resource. A computer node can be a single or multi-processor system such as PCs, workstations, servers, SMPs with memory, I/O and an OS. The nodes are interconnected via a LAN.

The cluster components are as follows.

1. Multiple High Performance Computers
2. Oss (Layered or Micro-Kernel Based)
3. High Performance Networks or Switches (Gigabit Ethernet and Myrinet)
4. Network Interface Cards (NICs)
5. Fast Communication Protocols and Services (Active and Fast Messages)
6. Cluster Middleware (Single System Image (SSI) and System Availability Infrastructure)
7. Parallel Programming Environments and Tools (Parallel Virtual Machine (PVM), Message Passing Interface (MPI))
8. Applications (Sequential, Parallel or Distributed)

Cluster Classifications [3]

The various features of clusters are as follows.

1. High Performance
2. Expandability and Scalability
3. High Throughput
4. High Availability

Cluster can be classified into many categories as follows.

1. Application Target
    - High Performance Clusters
    - High Availability Clusters
2. Node Ownership
    - Dedicated Clusters
    - Nondedicated Clusters
3. Node Hardware
    - Cluster of PCs (CoPs) or Piles of PCs (PoPs)
    - Cluster of Workstations (COWs)
    - Cluster of SMPs (CLUMPs)
4. Node OS
    - Linux Clusters (Beowulf)
    - Solaris Clusters (Berkeley NOW)
    - NT Clusters (High Performance Virtual Machine (HPVM))

- Advanced Interactive eXecutive (AIX) Clusters (IBM Service Pack 2 (SP2))
- Digital Virtual Memory System (VMS) Clusters
- HP-UX Clusters
- Microsoft Wolfpack Clusters

5. Node Configuration
    - Homogeneous Clusters
    - Heterogeneous Clusters
6. Levels of Clustering
    - Group Clusters (No. of Nodes = 2 to 99)
    - Departmental Clusters (No. of Nodes = 10 to 100s)
    - Organizational Clusters (No. of Nodes = Many 100s)
    - National Metacomputers (No. of Nodes = Many Departmental or Organizational Systems or Clusters)
    - International Metacomputers (No. of Nodes = 1000s to Many Millions)

Components for Clusters [3]

The components of clusters are the hardware and software used to build clusters and nodes. They are as follows.

1. Processors
    - Microprocessor Architecture (RISC, CISC, VLIW and Vector)
    - Intel x86 Processor (Pentium Pro and II)
    - Pentium Pro shows a very strong integer performance in contrast to Sun's UltraSPARC for high performance range at the same clock speed. However, the floating-point performance is much lower.
    - The Pentium II Xeon uses a memory bus of 100 MHz. It is available with a choice of 512 KB to 2 MB of L2 cache.
    - Other processors: x86 variants (AMD x86, Cyrix x86), Digital Alpha, IBM PowerPC, Sun SPARC, SGI MIPS and HP PA.
    - Berkeley NOW uses Sun's SPARC processors in their cluster nodes.
2. Memory and Cache
    - The memory present inside a PC was 640 KBs. Today, a PC is delivered with 32 or 64 MBs installed in slots with each slot holding a Standard Industry Memory Module (SIMM). The capacity of a PC is now many hundreds of MBs.
    - Cache is used to keep recently used blocks of memory for very fast access. The size of cache is usually in the range of 8KBs to 2MBs.
3. Disk and I/O
    - The I/O performance is improved to carry out I/O operations in parallel. It is supported by parallel file systems based on hardware or software Redundancy Array of Inexpensive Disk (RAID).
    - Hardware RAID is more expensive than Software RAID.
4. System Bus
    - Bus is the collection of wires which carries data from one component to another. The components are CPU, Main Memory and others.
    - Bus is of following types.
        - Address Bus
        - Data Bus
        - Control Bus
    - Address bus is the collection of wires which transfer the addresses of Memory or I/O devices. For instance, Intel 8085 Microprocessor has an address bus of 16 bits. It shows that the Microprocessor can transfer maximum 16 bit address.
    - Data bus is the collection of wires which is used to transfer data within the Microprocessor and Memory or I/O devices. Intel 8085 has a data bus of 8 bits. That's why Intel 8085 is called 8 bit Microprocessor.

- Control bus is responsible for issuing the control signals such as read, write or opcode fetch to perform some operations with the selected memory location.
- Every bus has a clock speed. The initial PC bus has a clock speed of 5 MHz and it is 8 bits wide.
- In PCs, the ISA bus is replaced by faster buses such as PCI.
- The ISA bus is extended to be 16 bits wide and an enhanced clock speed of 13 MHz. However, it is not sufficient to meet the demands of the latest CPUs, disk and other components.
- The VESA local bus is a 32 bit bus that has been outdated by the Intel PCI bus.
- PCI bus allows 133 Mbytes/s.

5. Cluster Interconnects
    - The nodes in a cluster are interconnected via standard Ethernet and these nodes are communicated using a standard networking protocol such as TCP/IP or a low-level protocol such as Active Messages.
    - Ethernet: 10 Mbps
    - Fast Ethernet: 100 Mbps
    - Gigabit Ethernet
    - The two main characteristics of Gigabit Ethernet are as follows.
        - It preserves Ethernet's simplicity which enabling a smooth migration to Gigabit-per-second (Gbps) speeds.
        - It delivers a very high bandwidth to aggregate multiple Fast Ethernet segments.
    - Asynchronous Transfer Mode (ATM)
        - It is a switched virtual-circuit technology.
        - It is intended to be used for both LAN and WAN, presenting a unified approach to both.
        - It is based on small fixed-size data packets termed cell. It is designed to allow cells to be transferred using a number of medias such as copper wire and fiber optic cables.
        - CAT-5 is used with ATM allowing upgrades of existing networks without replacing cabling.
    - Scalable Coherent Interface (SCI)
        - It aims to provide a low-latency distributed shared memory across a cluster.
        - It is design to support distributed multiprocessing with high bandwidth and low latency.
        - It is a point-to-point architecture with directory-based cache coherence.
        - Dolphin has produced an SCI MPI which offers less than 12 μs zero message-length latency on the Sun SPARC platform.
        -

- Myrinet
  - It is a 1.28 Gbps full duplex interconnection network supplied by Myricom [9].
  - It uses low latency cut-through routing switches, which is able to offer fault tolerance.
  - It supports both Linux and NT.
  - It is relatively expensive when compared to Fast Ethernet, but has following advantages. 1) Very low latency (5 μs), 2) Very high throughput, 3) Greater flexibility.
  - The main disadvantage of Myrinet is its price. The cost of Myrinet-LAN components including the cables and switches is $1,500 per host. Switches with more than 16 ports are unavailable. Therefore, scaling is complicated.

Cluster Middleware and Single System Image [3]

Single System Image (SSI) is the collection of interconnected nodes that appear as a unified resource. It creates an illusion of resources such as hardware or software that presents a single powerful resource. It is supported by a middleware layer that resides between the OS and the user-level environment. The middleware consists of two sub-layers, namely SSI Infrastructure and System Availability Infrastructure (SAI). SAI enables cluster services such as checkpointing, automatic failover, recovery from failure and fault-tolerant.

1. SSI Levels or Layers
    - Hardware (Digital (DEC) Memory Channel, Hardware DSM and SMP Techniques)
    - Operating System Kernel – Gluing Layer (Solaris MC and GLUnix)
    - Applications and Subsystems – Middleware
        - Applications
        - Runtime Systems
        - Resource Management and Scheduling Software (LSF and CODINE)
2. SSI Boundaries
    - Every SSI has a boundary.
    - SSI can exist at different levels within a system – one able to be built on another
3. SSI Benefits
    - It provides a view of all system resources and activities from any node of the cluster.
    - It frees the end user to know where the application will run.
    - It frees the operator to know where a resource is located.
    - It allows the administrator to manage the entire cluster as a single entity.
    - It allows both centralize or decentralize system management and control to avoid the need of skilled administrators for system administration.
    - It simplifies system management.
    - It provides location-independent message communication.
    - It tracks the locations of all resources so that there is no longer any need for system operators to be concerned with their physical location while carrying out system management tasks.
4. Middleware Design Goals
    - Transparency
    - Scalable Performance
    - Enhanced Availability
5. Key Service of SSI and Availability Infrastructure
    - SSI Support Services
        - Single Point of Entry
        - Single File Hierarchy
        - Single Point of Management and Control

  - Single Virtual Networking
  - Single Memory Space
  - Single Job Management System
  - Single User Interface
- Availability Support Functions
  - Single I/O Space
  - Single Process Space
  - Checkpointing and Process Migration

Resource Management and Scheduling (RMS) [3]

RMS is the process of distributing user's applications among computers to maximize the throughput. The software that performs the RMS has two components, namely resource manager and resource scheduler. Resource manager deals with locating and allocating computational resources, authentication, process creation and migration whereas the resource scheduler deals with queuing applications, resource location and assignment.

RMS is a client-server system. The jobs are submitted to the RMS environment and the environment is responsible for place, schedule and run the job in the appropriate way.

The services provided by a RMS environment are as follows.

1. Process Migration
2. Checkpointing
    - Taxonomy of Checkpoint Implementation [14]
        - Application-Level
            - Single Threaded
            - Multi Threaded
            - Mig Threaded
        - User-Level
            - Patch
            - Library
        - System-Level
            - Kernel-level
            - Hardware-level
3. Scavenging Idle Cycles
4. Fault Tolerance
5. Minimization of Impact on Users
6. Load Balancing
7. Multiple Application Queue

There are many commercial and research packages available for RMS as follows.

1. LSF (http://www.platform.com/)
    - The full form of LSF is Load Sharing Facility
    - Fair Share
    - Preemptive
    - Backfill and Service Level Agreement (SLA) Scheduling
    - High Throughput Scheduling
    - Multi-cluster Scheduling
    - Topology, Resource and Energy-aware Scheduling

- LSF is a job scheduling and monitoring software system developed and maintained by Platform Computing.
- LSF is used to run jobs on the blade center.
- A job is submitted from one of the head nodes (login01, login02 for 32-bit jobs, login03 for jobs compiled to use 64-bits) and waits until resources become available on the computational nodes.
- Jobs which ask for 4 or fewer processors and 15 minutes or less time are given a high priority.

2. CODINE (http://www.genias.de/products/codine/tech_desc.html)
   - The full form of CODINE is Computing in Distributed Networked Environments.
   - Advanced Reservation
   - CODINE was a grid computing computer cluster software system, developed and supported by Sun Microsystems and later Oracle [12].

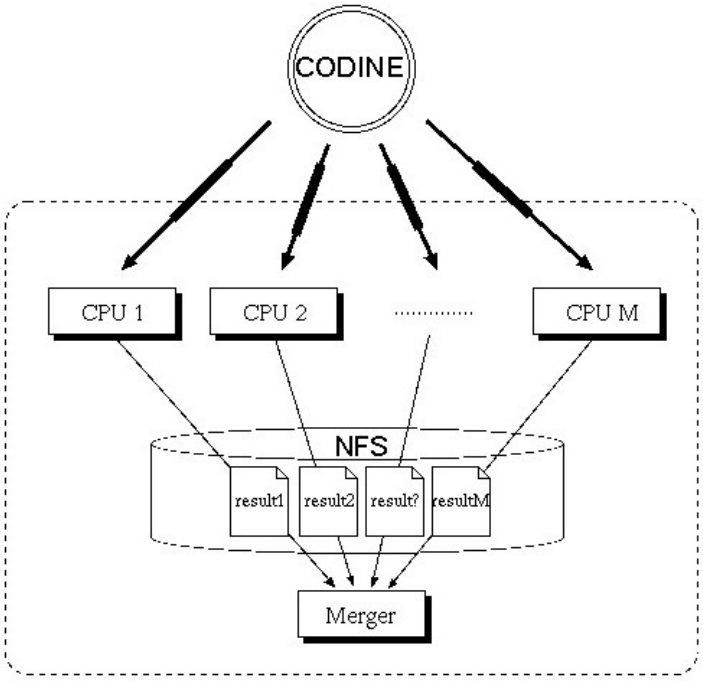


Figure 1 CODINE [13]

Programming Environments and Tools, Applications [3]

1. Threads
    - It is a light-weight process.
    - It is applicable for concurrent programming on both uniprocessor and multi-processors computers.
2. Message Passing Systems (MPI and PVM) [10-11]
    - MPI is a specification for the developers and users of message passing libraries. It is not a library.
    - MPI primarily addresses the message-passing parallel programming model. Data is moved from the address space of one process to that of another process through cooperative operations on each process.
    - The goal of the MPI is to provide a widely used standard for writing message passing programs. The interface attempts to be:
        - practical
        - portable
        - efficient
        - flexible
    - The MPI standard has gone through a number of revisions, with the most recent version being MPI-3.
    - Interface specifications have been defined for C and Fortran90 language bindings:
        - C++ bindings from MPI-1 are removed in MPI-3
        - MPI-3 also provides support for Fortran 2003 and 2008 features
    - Actual MPI library implementations differ in which version and features of the MPI standard they support.
    - MPI runs on virtually any hardware platform:
        - Distributed Memory
        - Shared Memory
        - Hybrid
    - Reasons for using MPI
        - Standardization
        - Portability
        - Performance Opportunities
        - Functionality
        - Availability
    - PVM is a software package that permits a heterogeneous collection of Unix and/or Windows computers hooked together by a network to be used as a single large parallel computer. Thus large computational problems can be solved more cost effectively by using the aggregate power and memory of many computers.

- PVM software is very portable.
- PVM enables users to exploit their existing computer hardware to solve much larger problems at minimal additional cost. Hundreds of sites around the world are using PVM to solve important scientific, industrial and medical problems in addition to PVM's use as an educational tool to teach parallel programming. With tens of thousands of users, PVM has become the de facto standard for distributed computing world-wide.

3. Distributed Shared Memory (DSM) Systems
4. Parallel Debuggers and Profilers
5. Performance Analysis Tools
6. Cluster Administration Tools

Cluster Applications [3]

1. Grand Challenge Applications (GCAs)
    - Crystallographic and Microtomographic Structural Problems
    - Protein Dynamics and Biocatalysis
    - Relativistic Quantum Chemistry of Antinides
    - Virtual Materials Design and Processing
    - Global Climate Modeling
    - Discrete Event Simulation
2. Supercomputing Applications
3. Computational Intensive Applications
4. Data or I/O Intensive Applications
5. Transaction Intensive Applications

Representative Cluster Systems, Heterogeneous Clusters [3, 15]

Many projects are investigating the development of supercomputing class machines using commodity off-the-shelf components (COTS). The popular projects are listed as follows.

Project	Organization
Network of Workstations (NOW)	University of California, Berkeley
High Performance Virtual Machine (HPVM)	University of Illinois, Urbana-Champaign
Beowulf	Goddard Space Flight Center, NASA
Solaris-MC	Sun Labs, Sun Microsystems, Palo Alto, CA

NOW

- Platform: PCs and Workstations
- Communications: Myrinet
- OS: Solaris
- Tool: PVM

HPVM

- Platform: PCs
- Communications: Myrinet
- OS: Linux
- Tool: MPI

Beowulf

- Platform: PCs
- Communications: Multiple Ethernet with TCP/IP
- OS: Linux
- Tool: MPI/PVM

Solaris-MC

- Platform: PCs and Workstations
- Communications: Solaris supported
- OS: Solaris
- Tool: C++ and CORBA

Heterogeneous Clusters

- Clusters deliberately heterogeneous in order to explore the higher floating point performance of certain architecture and the low cost of other systems.

- A heterogeneous layout means automating administration work will obviously become more complex, i.e., software packaging is different.
- Major challenges [16]:
  - Four major challenges that must be overcome so that heterogeneous computing clusters emerge as the preferred platform for executing a wide variety of enterprise workloads.
  - First, most enterprise applications in use today were not designed to run on such dynamic, open and heterogeneous computing clusters. Migrating these applications to heterogeneous computing clusters, especially with substantial improvement in performance or energy-efficiency, is an open problem.
  - Second, creating new enterprise applications ground-up to execute on the new, heterogeneous computing platform is also daunting. Writing high-performance, energy-efficient programs for these architectures is extremely challenging due to the unprecedented scale of parallelism, and heterogeneity in computing, interconnect and storage units.
  - Third, cost savings from the new shared-infrastructure architecture for consumption and delivery of IT services are only possible when multiple enterprise applications can amicably share resources (multi-tenancy) in the heterogeneous computing cluster. However, enabling multi-tenancy without adversely impacting the stringent quality of service metrics of each application calls for dynamic scalability and virtualization of a wide variety of diverse computing, storage and interconnect units, and this is yet another unsolved problem.
  - Finally, enterprise applications encounter highly varying user loads, with spikes of unusually heavy load. Meeting quality of service metrics across varying loads calls for an elastic computing infrastructure that can automatically provision (increase or decrease) computing resources used by an application in response to varying user demand. Currently, no good solutions exist to meet this challenge.

Security, Resource Sharing, Locality, Dependability [3]

Security

There is always a tradeoff between usability and security. Allowing rsh (remote shell) access from the outside to each node just by matching usernames and hosts with each user's .rhosts file is not good as a security incident in a single node compromises the security of all the systems who share that user's home. For instance, mail can be abused in a similar way – just change that user's .forward file to do the mail delivery via a pipe to an interesting executable or script. A service is not safe unless all of the services it depends on are at least equally safe.

Connecting all the nodes directly to the external network may cause two main problems. First, we make temporary changes and forget to restore them. Second, systems tend to have information leaks. The operating system and its version can easily guessed just with IP access, even with harmless services running and almost all operating system have serious security problems in their IP stack in the recent history.

Unencrypted Versus Encrypted Sustained Throughput

Unencrypted Stream	>7.5MB/s
Blowfish Encrypted Stream	2.75MB/s
Idea Encrypted Stream	1.8MB/s
3DES Encrypted Stream	0.75MB/s

Special care must be taken when building clusters of clusters is done. The approach for making these metaclusters secure is building secure tunnels between the clusters, usually from front-end to front-end

If intermediate backbone switches can be trusted and have the necessary software or resources, they can setup a VLAN joining the clusters, achieving greater bandwidth and lower latency than routing at the IP level via the front-ends.

Resource Sharing

Resource Sharing need the cooperation among the processors to ensure that no processor is idle while there are tasks waiting for service.

In Load Sharing, three location policies were studied. They are random policy, threshold policy and shortest policy.

Threshold policy probes a limited number of nodes. It terminates the probing as soon as it finds a node with a queue length shorter than the threshold.

Shortest policy probes several nodes and then selects the one having the shortest queue, from among those having queue lengths shorter than the threshold.

In the Flexible Load Sharing Algorithm (FLS) a location policy similar to threshold is used. In contrast to threshold, FLS bases its decisions on local information which is possibly replicated at multiple nodes. For scalability, FLS divides a system into small subsets which may overlap. Each of these subsets forms a cache held at a node. This algorithm supports mutual inclusion or exclusion. It is noteworthy to mention that FLS does not attempt to produce the best possible solution, but like threshold, it offers instead an adequate one, at a fraction of the cost.

Grid Computing: Introduction to Grid Computing, Virtual Organizations, Architecture, Applications, Computational, Data, Desktop and Enterprise Grids, Data-intensive Applications [17]

Grid computing has emerged as an important new field, distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications and in some cases, high-performance orientation. The term "the Grid" was coined in the mid1990s to denote a proposed distributed computing infrastructure for advanced science and engineering. Considerable progress has since been made on the construction of such an infrastructure, but the term "Grid" has also been conflated, at least in popular perception, to embrace everything from advanced networking to artificial intelligence. One might wonder whether the term has any real substance and meaning. Is there really a distinct "Grid problem" and hence a need for new "Grid technologies"? If so, what is the nature of these technologies, and what is their domain of applicability? While numerous groups have interest in Grid concepts and share, to a significant extent, a common vision of Grid architecture, we do not see consensus on the answers to these questions. The Grid concept is indeed motivated by a real and specific problem and that there is an emerging, well-defined Grid technology base that solves this problem. Grid technologies are currently distinct from other major technology trends, such as Internet, enterprise, distributed, and peer-to-peer computing, these other trends can benefit significantly from growing into the problem space addressed by Grid technologies.

The real and specific problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations. The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data and other resources, as is required by a range of collaborative problem-solving and resource brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules are called a virtual organization (VO). The following are examples of VOs: the application service providers, storage service providers, cycle providers, and consultants engaged by a car manufacturer to perform scenario evaluation during planning for a new factory; members of an industrial consortium bidding on a new aircraft; a crisis management team and the databases and simulation systems that they use to plan a response to an emergency situation; and members of a large, international, multiyear high energy physics collaboration. Each of these examples represents an approach to computing and problem solving based on collaboration in computation and data rich environments.

As these examples show, VOs vary tremendously in their purpose, scope, size, duration, structure, community, and sociology. Nevertheless, careful study of underlying technology requirements leads us to identify a broad set of common concerns and requirements. In particular, the need for highly flexible sharing relationships, ranging from client-server to peer-

to-peer and brokered; for complex and high levels of control over how shared resources are used, including fine-grained access control, delegation, and application of local and global policies; for sharing of varied resources, ranging from programs, files, and data to computers, sensors, and networks; and for diverse usage modes, ranging from single user to multi-user and from performance sensitive to cost-sensitive and hence embracing issues of quality of service, scheduling, co-allocation, and accounting.

Introduction to Cloud Computing, Types: Deployment and Service Models, Characteristics, Applications [18-19]

Computing is being transformed into a model consisting of services that are commoditized and delivered in a manner similar to utilities such as water, electricity, gas and telephony. In such a model, users access services based on their requirements, regardless of where the services are hosted. Several computing paradigms, such as grid computing, have promised to deliver this utility computing vision. Cloud computing is the most recent emerging paradigm promising to turn the vision of computing utilities into a reality.

In 1969, Leonard Kleinrock, one of the chief scientists of the original Advanced Research Projects Agency Network (ARPANET), which seeded the Internet, said: As of now, computer networks are still in their infancy, but as they grow up and become sophisticated, we will probably see the spread of computer utilities which, like present electric and telephone utilities, will service individual homes and offices across the country. This vision of computing utilities based on a service-provisioning model anticipated the massive transformation of the entire computing industry in the 21st century, where by computing services will be readily available on demand, just as other utility services such as water, electricity, telephone, and gas are available in today's society. Similarly, users (consumers) need to pay providers only when they access the computing services. In addition, consumers no longer need to invest heavily or encounter difficulties in building and maintaining complex IT infrastructure.

In such a model, users access services based on their requirements without regard to where the services are hosted. This model has been referred to as utility computing or recently (since 2007), as cloud computing. The latter term often denotes the infrastructure as a cloud from which businesses and users can access applications as services from anywhere in the world and on demand. Hence, cloud computing can be classified as a new paradigm for the dynamic provisioning of computing services supported by state-of-the-art data centers employing virtualization technologies for consolidation and effective utilization of resources. Cloud computing allows renting infrastructure, runtime environments and services on a pay-per-use basis. This principle finds several practical applications and then gives different images of cloud computing to different people. Chief information and technology officers of large enterprises see opportunities for scaling their infrastructure on demand and sizing it according to their business needs. End users leveraging cloud computing services can access their documents and data anytime, anywhere and from any device connected to the Internet. The most diffuse views of cloud computing can be summarized as follows: I don't care where my servers are, who manages them, where my documents are stored or where my applications are hosted. I just want them always available and access them from any device connected through Internet. And I am willing to pay for this service for as a long as I need it.

The definition proposed by the U. S. National Institute of Standards and Technology (NIST):

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

Cloud Types

Most people separate cloud computing into two distinct sets of models:

• Deployment models: This refers to the location and management of the cloud's infrastructure.

• Service models: This consists of the particular types of services that you can access on a cloud computing platform**.**

Deployment Models

A deployment model defines the purpose of the cloud and the nature of how the cloud is located. The NIST definition for the four deployment models is as follows:

1. **Public cloud:** The public cloud infrastructure is available for public use alternatively for a large industry group and is owned by an organization selling cloud services.
2. **Private cloud:** The private cloud infrastructure is operated for the exclusive use of an organization. The cloud may be managed by that organization or a third party. Private clouds may be either on- or off-premises.
3. **Hybrid cloud:** A hybrid cloud combines multiple clouds (private, community of public) where those clouds retain their unique identities, but are bound together as a unit. A hybrid cloud may offer standardized or proprietary access to data and applications, as well as application portability.
4. **Community cloud:** A community cloud is one where the cloud has been organized to serve a common function or purpose. It may be for one organization or for several organizations, but they share common concerns such as their mission, policies, security, regulatory compliance needs, and so on. A community cloud may be managed by the constituent organization(s) or by a third party.

Service Models

In the deployment model, different cloud types are an expression of the manner in which infrastructure is deployed. You can think of the cloud as the boundary between where a client's network, management and responsibilities ends and the cloud service provider's begins. As cloud computing has developed, different vendors offer clouds that have different services associated

with them. The portfolio of services offered adds another set of definitions called the service model.

Three service types have been universally accepted:

1. **Infrastructure as a Service:** IaaS provides virtual machines, virtual storage, virtual infrastructure and other hardware assets as resources that clients can provision. The IaaS service provider manages all the infrastructure, while the client is responsible for all other aspects of the deployment. This can include the operating system, applications, and user interactions with the system.
2. **Platform as a Service:** PaaS provides virtual machines, operating systems, applications, services, development frameworks, transactions, and control structures. The client can deploy its applications on the cloud infrastructure or use applications that were programmed using languages and tools that are supported by the PaaS service provider. The service provider manages the cloud infrastructure, the operating systems, and the enabling software. The client is responsible for installing and managing the application that it is deploying.
3. **Software as a Service:** SaaS is a complete operating environment with applications, management and the user interface. In the SaaS model, the application is provided to the client through a thin client interface (a browser, usually), and the customer's responsibility begins and ends with entering and managing its data and user interaction. Everything from the application down to the infrastructure is the vendor's responsibility.

Examples of IaaS service providers include:
• Amazon Elastic Compute Cloud (EC2)
• Eucalyptus
• GoGrid
• FlexiScale
• Linode
• RackSpace Cloud
• Terremark

Examples of PaaS services are:
• Force.com
• GoGrid CloudCenter
• Google AppEngine
• Windows Azure Platform

Examples of SaaS cloud service providers are:
• GoogleApps
• Oracle On Demand

• SalesForce.com
• SQL Azure

Characteristics

The NIST Definition of Cloud Computing by Peter Mell and Tim Grance has classified cloud computing into the three service models (SaaS, IaaS and PaaS) and four cloud types (public, private, community and hybrid), also assigns five essential characteristics that cloud computing systems must offer:

1. **On-demand self-service:** A client can provision computer resources without the need for interaction with cloud service provider personnel.

2. **Broad network access:** Access to resources in the cloud is available over the network using standard methods in a manner that provides platform-independent access to clients of all types. This includes a mixture of heterogeneous operating systems, and thick and thin platforms such as laptops, mobile phones, and PDA.

3. **Resource pooling:** A cloud service provider creates resources that are pooled together in a system that supports multi-tenant usage. Physical and virtual systems are dynamically allocated or reallocated as needed. Intrinsic in this concept of pooling is the idea of abstraction that hides the location of resources such as virtual machines, processing, memory, storage, and network bandwidth and connectivity.

4. **Rapid elasticity:** Resources can be rapidly and elastically provisioned. The system can add resources by either scaling up systems (more powerful computers) or scaling out systems (more computers of the same kind), and scaling may be automatic or manual. From the standpoint of the client, cloud computing resources should look limitless and can be purchased at any time and in any quantity.

5. **Measured service:** The use of cloud system resources is measured, audited, and reported to the customer based on a metered system. A client can be charged based on a known metric such as amount of storage used, number of transactions, network I/O (Input/Output) or bandwidth, amount of processing power used, and so forth. A client is charged based on the level of services provided.

Applications

Practical examples of such systems exist across all market segments:

1. Large enterprises can offload some of their activities to cloud-based systems. Recently, the New York Times has converted its digital library of past editions into a Web-friendly format. This required a considerable amount of computing power for a short period of time. By renting Amazon EC2 and S3 Cloud resources, the Times performed this task in 36 hours and relinquished these resources, with no additional costs.
2. Small enterprises and start-ups can afford to translate their ideas into business results more quickly, without excessive up-front costs. Animoto is a company that creates videos out of images, music and video fragments submitted by users. The process involves a considerable amount of storage and back end processing required for producing the video, which is finally made available to the user. Animoto does not own a single server and bases its computing infrastructure entirely on Amazon Web Services, which are sized on demand according to the overall workload to be processed. Such workload can vary a lot and require instant scalability. Up-front investment is clearly not an effective solution for many companies and cloud computing systems become an appropriate alternative.
3. System developers can concentrate on the business logic rather than dealing with the complexity of infrastructure management and scalability. Little Fluffy Toys is a company in London that has developed a widget providing users with information about nearby bicycle rental services. The company has managed to back the widget's computing needs on Google App Engine and be on the market in only one week.
4. End users can have their documents accessible from everywhere and any device. Apple iCloud is a service that allows users to have their documents stored in the Cloud and access them from any device users connect to it. This makes it possible to take a picture while traveling with a smartphone, go back home and edit the same picture on your laptop, and have it show as updated on your tablet computer. This process is completely transparent to the user, who does not have to setup cables and connect these devices with each other.

Service-Level Agreement, Virtualization [18-19]

A Service Level Agreement (SLA) is the contract for performance negotiated between you and a service provider. In the early days of cloud computing, all SLAs were negotiated between a client and the provider. Today with the advent of large utility-like cloud computing providers, most SLAs are standardized until a client becomes a large consumer of services.

Caution

Some SLAs are enforceable as contracts, but many are really agreements that are more along the lines of an Operating Level Agreement (OLA) and may not have the force of law. It's good to have an attorney review these documents before you make a major commitment to a cloud provider.

SLAs usually specify these parameters:

- Availability of the service (uptime)
- Response times or latency
- Reliability of the service components
- Responsibilities of each party
- Warranties

If a vendor fails to meet the stated targets or minimums, it is punished by having to offer the client a credit or pay a penalty. In this regard, an SLA should be like buying insurance, and like buying insurance, getting the insurer to pay up when disaster strikes can be the devil's work.

Microsoft publishes the SLAs associated with the Windows Azure Platform components, which is illustrative of industry practice for cloud providers. Each individual component has its own SLA.

Windows Azure SLA: Windows Azure has separate SLA's for compute and storage. For compute, we guarantee that when you deploy two or more role instances in different fault and upgrade domains, your Internet facing roles will have external connectivity at least 99.95% of the time. Additionally, we will monitor all of your individual role instances and guarantee that 99.9% of the time we will detect when a role instance's process is not running and initiate corrective action.

Cross-Ref

- **SQL Azure SLA:** "SQL Azure customers will have connectivity between the database and our

Internet gateway. SQL Azure will maintain a "Monthly Availability" of 99.9% during a calendar month. "Monthly Availability Percentage" for a specific customer database is the ratio of the time the database was available to customers to the total time in a month. Time is measured in 5-minute intervals in a 30-day monthly cycle. Availability is always calculated for a full month. An interval is marked as unavailable if the customer's attempts to connect to a database are rejected by the SQL Azure gateway."

• **AppFabric SLA:** "Uptime percentage commitments and SLA credits for Service Bus and Access Control are similar to those specified above in the Windows Azure SLA. Due to inherent differences between the technologies, underlying SLA definitions and terms differ for the Service Bus and Access Control services. Using the Service Bus, customers will have connectivity between a customer's service endpoint and our Internet gateway; when our service fails to establish a connection from the gateway to a customer's service endpoint, then the service is assumed to be unavailable. Using Access Control, customers will have connectivity between the Access Control endpoints and our Internet gateway. In addition, for both Service Bus and Access Control, the service will process correctly formatted requests for the handling of messages and tokens; when our service fails to process a request properly, then the service is assumed to be unavailable. SLA calculations will be based on an average over a 30-day monthly cycle, with 5-minute time intervals. Failures seen by a customer in the form of service unavailability will be counted for the purpose of availability calculations for that customer."

Some cloud providers allow for service credits based on their ability to meet their contractual levels of uptime. For example, Amazon applies a service credit of 10 percent off the charge for Amazon S3 if the monthly uptime is equal to or greater than 99 percent but less than 99.9 percent. When the uptime drops below 99 percent, the service credit percentage rises to 25 percent and this credit is applied to usage in the next billing period. Amazon Web Services uses an algorithm that calculates uptime based on the following formula:

Uptime = Error Rate/Requests

as measured for each 5-minute interval during a billing period. The error rate is based on internal server counters such as "InternalError" or "ServiceUnavailable." There are exclusions that limit Amazon's exposure.

Service Level Agreements are based on the usage model. Most cloud providers price their pay-as-you-go resources at a premium and issue standard SLAs only for that purpose. You can also purchase subscriptions at various levels that guarantee you access to a certain amount of purchased resources. The SLAs attached to a subscription often offer different terms. If your

organization requires access to a certain level of resources, then you need a subscription to a service. A usage model may not provide that level of access under peak load conditions.

Virtualization

Virtualization is a large umbrella of technologies and concepts that are meant to provide an abstract environment — whether virtual hardware or an operating system — to run applications. The term virtualization is often synonymous with hardware virtualization, which plays a fundamental role in efficiently delivering Infrastructure-as-a-Service (IaaS) solutions for cloud computing. In fact, virtualization technologies have along trail in the history of computer science and have been available in many flavors by providing virtual environments at the operating system level, the programming language level, and the application level. Moreover, virtualization technologies provide a virtual environment for not only executing applications but also for storage, memory, and networking.

Since its inception, virtualization has been sporadically explored and adopted, but in the last few years there has been a consistent and growing trend to leverage this technology. Virtualization technologies have gained renewed interested recently due to the confluence of several phenomena:

- Increased performance and computing capacity
- Underutilized hardware and software resources
- Lack of space
- Greening initiatives
- Rise of administrative costs

Virtualization is a broad concept that refers to the creation of a virtual version of something, whether hardware, a software environment, storage, or a network. In a virtualized environment there are three major components: guest, host, and virtualization layer. The guest represents the system component that interacts with the virtualization layer rather than with the host, as would normally happen. The host represents the original environment where the guest is supposed to be managed. The virtualization layer is responsible for recreating the same or a different environment where the guest will operate. Such a general abstraction finds different applications and then implementations of the virtualization technology. The most intuitive and popular is represented by hardware virtualization, which also constitutes the original realization of the virtualization concept. In the case of hardware virtualization, the guest is represented by a system image comprising an operating system and installed applications. These are installed on top of virtual hardware that is controlled and managed by the virtualization layer, also called the virtual machine manager. The host is instead represented by the physical hardware, and in some cases the operating system, that defines the environment where the virtual machine manager is

running. In the case of virtual storage, the guest might be client applications or users that interact with the virtual storage management software deployed on top of the real storage system. The case of virtual networking is also similar: The guest applications and users —interacts with a virtual network, such as a virtual private network (VPN), which is managed by specific software (VPN client) using the physical network available on the node. VPNs are useful for creating the illusion of being with in a different physical network and thus accessing there sources in it, which would otherwise not be available.

High-Throughput Computing: Task Computing and Task-based Application Models [18-19]

Organizing an application in terms of tasks is the most intuitive and common practice for developing parallel and distributed computing applications. A task identifies one or more operations that produce a distinct output and that can be isolated as a single logical unit. In practice, a task is represented as a distinct unit of code, or a program, that can be separated and executed in a remote runtime environment. Programs are the most common option for representing tasks, especially in the field of scientific computing, which has leveraged distributed computing for its computational needs.

Multithreaded programming is mainly concerned with providing a support for parallelism within a single machine. Task computing provides distribution by harnessing the compute power of several computing nodes. Hence, the presence of a distributed infrastructure is explicit in this model. Historically, the infrastructures that have been leveraged to execute tasks are clusters, supercomputers, and computing grids. Now clouds have emerged as an attractive solution to obtain a huge computing power on demand for the execution of distributed applications. To achieve it, suitable middleware is needed. The middleware is a software layer that enables the coordinated use of multiple resources, which are drawn from a datacenter or geographically distributed networked computers. A user submits the collection of tasks to the access point(s) of the middleware, which will take care of scheduling and monitoring the execution of tasks. Each computing resource provides an appropriate runtime environment, which may vary from implementation to implementation (a simple shell, a sandboxed environment, or a virtual machine). Task submission is normally done using the APIs provided by the middleware, whether a Web or programming language interface. Appropriate APIs are also provided to monitor task status and collect their results upon completion. Because task abstraction is general, there exist different models of distributed applications falling under the umbrella of task computing. Despite this variety, it is possible to identify a set of common operations that the middleware needs to support the creation and execution of task-based applications.

These operations are:
- Coordinating and scheduling tasks for execution on a set of remote nodes
- Moving programs to remote nodes and managing their dependencies
- Creating an environment for execution of tasks on the remote nodes
- Monitoring each task's execution and informing the user about its status
- Access to the output produced by the task

Models for task computing may differ in the way tasks are scheduled, which in turn depends on whether tasks are interrelated or they need to communicate among themselves.

According to the specific nature of the problem, a variety of categories for task computing have been proposed over time. These categories do not enforce any specific application model but provide an overall view of the characteristics of the problems. They implicitly impose requirements on the infrastructure and the middleware. Applications falling into this category are high-performance computing (HPC), high-throughput computing (HTC), and many-task computing (MTC).

High-performance computing

High-performance computing (HPC) is the use of distributed computing facilities for solving problems that need large computing power. Historically, supercomputers and clusters are specifically designed to support HPC applications that are developed to solve "Grand Challenge" problems in science and engineering. The general profile of HPC applications is constituted by a large collection of compute-intensive tasks that need to be processed in a short period of time. It is common to have parallel and tightly coupled tasks, which require low-latency interconnection networks to minimize the data exchange time. The metrics to evaluate HPC systems are floating point operations per second (FLOPS), now tera-FLOPS or even peta-FLOPS, which identify the number of floating point operations per second that a computing system can perform.

High-throughput computing

High-throughput computing (HTC) is the use of distributed computing facilities for applications requiring large computing power over a long period of time. HTC systems need to be robust and to reliably operate over a long time scale. Traditionally, computing grids composed of heterogeneous resources (clusters, workstations, and volunteer desktop machines) have been used to support HTC. The general profile of HTC applications is that they are made up of a large number of tasks of which the execution can last for a considerable amount of time (i.e., weeks or months). Classical examples of such applications are scientific simulations or statistical analyses. It is quite common to have independent tasks that can be scheduled in distributed resources because they do not need to communicate. HTC systems measure their performance in terms of jobs completed per month.

Many-task computing

The many-task computing (MTC) model started receiving attention recently and covers a wide variety of applications. It aims to bridge the gap between HPC and HTC. MTC is similar to HTC, but it concentrates on the use of many computing resources over a short period of time to accomplish many computational tasks. In brief, MTC denotes high-performance computations comprising multiple distinct activities coupled via file system operations. What characterizes

MTC is the heterogeneity of tasks that might be of considerably different nature: Tasks may be small or large, single-processor or multiprocessor, compute-intensive or data-intensive, static or dynamic, homogeneous or heterogeneous. The general profile of MTC applications includes loosely coupled applications that are generally communication-intensive but not naturally expressed using the message-passing interface commonly found in HPC, drawing attention to the many computations that are heterogeneous but not embarrassingly parallel. Given the large number of tasks commonly composing MTC applications, any distributed facility with a large availability of computing elements is able to support MTC. Such facilities include supercomputers, large clusters, and emerging cloud infrastructures.

Frameworks for task computing

There are several frameworks that can be used to support the execution of task-based applications on distributed computing resources, including clouds. Some popular software systems that support the task-computing framework are Condor, Globus Toolkit, Sun Grid Engine (SGE), BOINC, Nimrod/G, and Aneka.

Architecture of all these systems consist of two main components: a scheduling node (one or more) and worker nodes. The organization of the system components may vary. For example, multiple scheduling nodes can be organized in hierarchical structures. This configuration is quite common in the middleware for computing grids, which harness a variety of distributed resources from one or more organizations or sites. Each of these sites may have their own scheduling engine, especially if the system contributes to the grid but also serves local users.

A classic example is the cluster setup where the system might feature an installation of Condor or SGE for batch job submission; these services are generally used locally to the site, but the cluster can be integrated into a larger grid where meta-schedulers such as GRAM (Globus Resource Allocation Manager) can dispatch a collection of jobs to the cluster. Other options include the presence of gateway nodes that do not have any scheduling capabilities and simply constitute the access point to the system. These nodes have indexing services that allow users to identify the available resources in the system, its current status, and the available schedulers. For worker nodes, they generally provide a sandboxed environment where tasks are executed on behalf of a specific user or within a given security context, limiting the operations that can be performed by programs such as file system access. File staging is also a fundamental feature supported by these systems. Clusters are normally equipped with shared file systems and parallel I/O facilities. Grids provide users with various staging facilities, such as credential access to remote worker nodes or automated staging services that transparently move files from user local machine to remote nodes. Condor is probably the most widely used and long-lived middleware for managing clusters, idle workstations, and a collection of clusters. Condor-G is a version of

Condor that supports integration with grid computing resources, such as those managed by Globus. Condor supports common features of batch-queuing systems along with the capability to checkpoint jobs and manage overload nodes. It provides a powerful job resource-matching mechanism, which schedules jobs only on resources that have the appropriate runtime environment. Condor can handle both serial and parallel jobs on a wide variety of resources. It is used by hundreds of organizations in industry, government, and academia to manage infrastructures ranging from a handful to well over thousands of workstations.

Sun Grid Engine (SGE), now Oracle Grid Engine, is middleware for workload and distributed resource management. Initially developed to support the execution of jobs on clusters, SGE integrated additional capabilities and now is able to manage heterogeneous resources and constitutes middleware for grid computing. It supports the execution of parallel, serial, interactive, and parametric jobs and features advanced scheduling capabilities such as budget-based and group based scheduling, scheduling applications that have deadlines, custom policies, and advance reservation.

The Globus Toolkit is a collection of technologies that enable grid computing. It provides a comprehensive set of tools for sharing computing power, databases, and other services across corporate, institutional, and geographic boundaries without sacrificing local autonomy. The toolkit features software services, libraries, and tools for resource monitoring, discovery, and management as well as security and file management. The Globus Toolkit addresses core issues of grid computing: the management of a distributed environment composed of heterogeneous resources spanning different organizations, with all this condition implies in terms of security and interoperation. To provide valid support for grid computing in such scenarios, the toolkit defines a collection of interfaces and protocol for interoperation that enable different systems to integrate with each other and expose resources outside their boundaries.

Nimrod/G is a tool for automated modeling and execution of parameter sweep applications (parameter studies) over global computational grids. It provides a simple declarative parametric modeling language for expressing parametric experiments. A domain expert can easily create a plan for a parametric experiment and use the Nimrod/G system to deploy jobs on distributed resources for execution. It has been used for a very wide range of applications over the years, ranging from quantum chemistry to policy and environmental impact. Moreover, it uses novel resource management and scheduling algorithms based on economic principles. Specifically, it supports deadline- and budget-constrained scheduling of applications on distributed grid resources to minimize the execution cost and at the same deliver results in a timely manner.

Berkeley Open Infrastructure for Network Computing (BOINC) is framework for volunteer and grid computing. It allows us to turn desktop machines into volunteer computing nodes that are

leveraged to run jobs when such machines become inactive. BOINC is composed of two main components: the BOINC server and the BOINC client. The former is the central node that keeps track of all the available resources and scheduling jobs; the latter is the software component that is deployed on desktop machines and that creates the BOINC execution environment for job submission. Given the volatility of BOINC clients, BOINC supports job check pointing and duplication. Even if mostly focused on volunteer computing, BOINC systems can be easily set up to provide more stable support for job execution by creating computing grids with dedicated machines. To leverage BOINC, it is necessary to create an application project. When installing BOINC clients, users can decide the application project to which they want to donate the CPU cycles of their computer. Currently several projects, ranging from medicine to astronomy and cryptography, are running on the BOINC infrastructure.

Task-based application models

There are several models based on the concept of the task as the fundamental unit for composing distributed applications. What makes these models different from one another is the way in which tasks are generated, the relationships they have with each other, and the presence of dependencies or other conditions—for example, a specific set of services in the runtime environment—that have to be met.

Embarrassingly parallel applications

Embarrassingly parallel applications constitute the most simple and intuitive category of distributed applications. The tasks might be of the same type or of different types, and they do not need to communicate among themselves. This category of applications is supported by the majority of the frameworks for distributed computing. Since tasks do not need to communicate, there is a lot of freedom regarding the way they are scheduled. Tasks can be executed in any order, and there is no specific requirement for tasks to be executed at the same time. Therefore, scheduling these applications is simplified and mostly concerned with the optimal mapping of tasks to available resources. Frameworks and tools supporting embarrassingly parallel applications are the Globus Toolkit, BOINC, and Aneka. There are several problems that can be modeled as embarrassingly parallel. These include image and video rendering, evolutionary optimization, and model forecasting. In image and video rendering the task is represented by the rendering of a pixel (more likely a portion of the image) or a frame, respectively. For evolutionary optimization meta heuristics, a task is identified by a single run of the algorithm with a given parameter set. The same applies to model forecasting applications. In general, scientific applications constitute a considerable source of embarrassingly parallel applications, even though they mostly fall into the more specific category of parameter sweep applications.

Parameter sweep applications

Parameter sweep applications are a specific class of embarrassingly parallel applications for which the tasks are identical in their nature and differ only by the specific parameters used to execute them. Parameter sweep applications are identified by a template task and a set of parameters. The template task defines the operations that will be performed on the remote node for the execution of tasks. The template task is parametric, and the parameter set identifies the combination of variables whose assignments specialize the template task into a specific instance. The combination of parameters, together with their range of admissible values, identifies the multidimensional domain of the application, and each point in this domain identifies a task instance.

Any distributed computing framework that provides support for embarrassingly parallel applications can also support the execution of parameter sweep applications, since the tasks composing the application can be executed independently of each other. The only difference is that the tasks that will be executed are generated by iterating over all the possible and admissible combinations of parameters. This operation can be performed by frameworks natively or tools that are part of the distributed computing middleware. For example, Nimrod/G is natively designed to support the execution of parameter sweep applications, and Aneka provides client-based tools for visually composing a template task, defining parameters, and iterating over all the possible combinations of such parameters.

A plethora of applications fall into this category. Mostly they come from the scientific computing domain: evolutionary optimization algorithms, weather-forecasting models, computational fluid dynamics applications, Monte Carlo methods, and many others. For example, in the case of evolutionary algorithms it is possible to identify the domain of the applications as a combination of the relevant parameters of the algorithm. For genetic algorithms these might be the number of individuals of the population used by the optimizer and the number of generations for which to run the optimizer. The following example in pseudo-code demonstrates how to use parameter sweeping for the execution of a generic evolutionary algorithm which can be configured with different population sizes and generations.

```
Individuals = {100, 200, 300, 500, 1000}
Generations = {50, 100, 200, 400}
for each indiv in individuals do
for each generation in generations do
task=generate_task(indiv, generation)
submit_task(task)
```

The algorithm sketched in the example defines a bidimensional domain composed of discrete variables and then iterated over each combination of individuals and generations to generate all the tasks composing the application. In this case 20 tasks are generated. The function generate_task is specific to the application and creates the task instance by substituting the values of indiv and generation to the corresponding variables in the template definition. The function submit_task is specific to the middleware used and performs the actual task submission.

A template task is in general a composition of operations concerning the execution of legacy applications with the appropriate parameters and set of file system operations for moving data. Therefore, frameworks that natively support the execution of parameter sweep applications often provide a set of useful commands for manipulating or operating on files. Also, the template task is often expressed as single file that composes together the commands provided. The commonly available commands are:

- Execute. Executes a program on the remote node.
- Copy. Copies a file to/from the remote node.
- Substitute. Substitutes the parameter values with their placeholders inside a file.
- Delete. Deletes a file.

All these commands can operate with parameters that are substituted with their actual values for each task instance.

Market-Based Management of Clouds [18]

Cloud computing is still in its infancy, and its prominent use is twofold:
(1) complete replacement of in-house IT infrastructure and services with the same capabilities rented by service providers; and
(2) elastic scaling of existing computing systems in order to address peak workloads.

The efforts in research and industry have been mostly oriented to design and implement systems that actually enable business vendors and enterprises to achieve these goals. The real potential of cloud computing resides in the fact that it actually facilitates the establishment of a market for trading IT utilities. This opportunity until now has been mildly explored and falls in the domain of what it is called market-oriented cloud computing.

Market-oriented cloud computing

Cloud computing already embodies the concept of providing IT assets as utilities. Then, what makes cloud computing different from market-oriented cloud computing? First, it is important to understand what we intend by the term market. The Oxford English Dictionary (OED) defines a market as a "place where a trade is conducted". More precisely, market refers to a meeting or a gathering together of people for the purchase and sale of goods. A broader characterization defines the term market as the action of buying and selling, a commercial transaction, a purchase, or a bargain. Therefore, essentially the word market is the act of trading mostly performed in an environment—either physical or virtual—that is specifically dedicated to such activity.

If we consider the way IT assets and services are consumed as utilities, it is evident that there is a trade-off between the service provider and the consumer; this enables the use of the service by the user under a given SLA. Therefore, cloud computing already expresses the concept of trade, even though the interaction between consumer and provider is not as sophisticated as happens in real markets: Users generally select one cloud computing vendor from among a group of competing providers and leverage its services as long as they need them. Moreover, at present, most service providers have inflexible pricing, generally limited to flat rates or tariffs based on usage thresholds. In addition, many providers have proprietary interfaces to their services, thus restricting the ability of consumers to quickly move—and with minimal conversion costs—from one vendor to another.

This rigidity, known as vendor lock-in, undermines the potential of cloud computing to be an open market where services are freely traded. Therefore, to remove such restrictions, it is required that vendors expose services through standard interfaces. This enables full commoditization and thus would pave the way for the creation of a market infrastructure for trading services. What differentiates market-oriented cloud computing (MOCC) from cloud

computing is the presence of a virtual marketplace where IT services are traded and brokered dynamically. This is something that still has to be achieved and that will significantly evolve the way cloud computing services are eventually delivered to the consumer. More precisely, what is missing is the availability of a market where desired services are published and then automatically bid on by matching the requirements of customers and providers. At present, some cloud computing vendors are already moving in this direction; this phenomenon is happening in the IaaS domain—which is the market sector that is more consolidated and mature for cloud computing—but it has not taken off broadly yet. We can clearly characterize the relationship between cloud computing and MOCC as follows:

Market Oriented Computing has the same characteristics as Cloud Computing; therefore it is a dynamically provisioned unified computing resource allowing you to manage software and data storage as on aggregate capacity resulting in "real-time" infrastructure across public and private infrastructures. Market Oriented Cloud Computing goes one step further by allowing spread into multiple public and hybrid environments dynamically composed by trading service.

The realization of this vision is technically possible today but is not probable, given the lack of standards and overall immaturity of the market. Nonetheless, it is expected that in the near future, with the introduction of standards, concerns about security and trust will begin to disappear and enterprises will feel more comfortable leveraging a market-oriented model for integrating IT infrastructure and services from the cloud. Moreover, the presence of a demand-based marketplace represents an opportunity for enterprises to shape their infrastructure for dynamically reacting to workload spikes and for cutting maintenance costs. It also allows the possibility to temporarily lease some in-house capacity during low usage periods, thus giving a better return on investment. These developments will lead to the complete realization of market-oriented cloud computing.

A reference model for MOCC

Market-oriented cloud computing originated from the coordination of several components: service consumers, service providers, and other entities that make trading between these two groups possible. Market orientation not only influences the organization on the global scale of the cloud computing market. It also shapes the internal architecture of cloud computing providers that need to support a more flexible allocation of their resources, which is driven by additional parameters such as those defining the quality of service.

A global view of market-oriented cloud computing

It provides guidance on how MOCC can be implemented in practice. Several components and entities contribute to the definition of a global market-oriented architecture. The fundamental component is the virtual marketplace—represented by the Cloud Exchange (CEx)—which acts as a market maker, bringing service producers and consumers together. The principal players in the virtual marketplace are the cloud coordinators and the cloud brokers. The cloud coordinators represent the cloud vendors and publish the services that vendors offer. The cloud brokers operate on behalf of the consumers and identify the subset of services that match customers' requirements in terms of service profiles and quality of service.

Brokers perform the same function as they would in the real world: They mediate between coordinators and consumers by acquiring services from the first and subleasing them to the latter. Brokers can accept requests from many users. At the same time, users can leverage different brokers. A similar relationship can be considered between coordinators and cloud computing services vendors. Coordinators take responsibility for publishing and advertising services on behalf of vendors and can gain benefits from reselling services to brokers. Every single participant has its own utility function, that they all want to optimize rewards. Negotiations and trades are carried out in a secure and dependable environment and are mostly driven by SLAs, which each party has to fulfill. There might be different models for negotiation among entities, even though the auction model seems to be the more appropriate in the current scenario. The same consideration can be made for the pricing models: Prices can be fixed, but it is expected that they will most likely change according to market conditions.

Several components contribute to the realization of the Cloud Exchange and implement its features. There are three major components:

• Directory. The market directory contains a listing of all the published services that are available in the cloud marketplace. The directory not only contains a simple mapping between service names and the corresponding vendor (or cloud coordinators) offering them. It also provides additional metadata that can help the brokers or the end users in filtering from among the services of interest those that can really meet the expected quality of service. Moreover, several indexing methods can be provided to optimize the discovery of services according to various criteria. This component is modified in its content by service providers and queried by service consumers.

• Auctioneer. The auctioneer is in charge of keeping track of the running auctions in the marketplace and of verifying that the auctions for services are properly conducted and that malicious market players are prevented from performing illegal activities.

• Bank. The bank is the component that takes care of the financial aspect of all the operations happening in the virtual marketplace. It also ensures that all the financial transactions are carried out in a secure and dependable environment. Consumers and providers may register with the bank and have one or multiple accounts that can be used to perform the transactions in the virtual marketplace.

This organization, constitutes only a reference model that is used to guide system architects and designers in laying out the foundations of a Cloud Exchange system. In reality, the architecture of such a system is more complex and articulated since other elements have to be taken into account. For instance, since the cloud marketplace supports trading, which ultimately involves financial transactions between different parties, security becomes of fundamental importance. It is then important to put in place all the mechanisms that enable secure electronic transactions.

Market-oriented architecture for datacenters

Datacenters are the building blocks of the computing infrastructure that backs the services offered by a cloud computing vendor, no matter its specific category (IaaS, PaaS, or SaaS). We present these systems by taking into account the elements that are fundamental for realizing computing infrastructures that support MOCC. These criteria govern the logical organization of these systems—rather than their physical layout and hardware characteristics—and provide guidance for designing architectures that are market oriented.

There are four major components of the architecture:

• Users and brokers. They originate the workload that is managed in the cloud datacenter. Users either require virtual machine instances to which to deploy their systems (IaaS scenario) or deploy applications in the virtual environment made available to them by the provider (PaaS scenario). These service requests are issued by service brokers that act on behalf of users and look for the best deal for them.

• SLA resource allocator. The allocator represents the interface between the datacenter and the cloud service provider and the external world. Its main responsibility is ensuring that service requests are satisfied according to the SLA agreed to with the user. Several components coordinate allocator activities in order to realize this goal:

• Service Request Examiner and Admission Control Module. This module operates in the front-end and filters user and broker requests in order to accept those that are feasible given Service the current status of the system and the workload that is already processing. Accepted requests are allocated and scheduled for execution. IaaS service providers allocate one or more virtual

machine instances and make them available to users. PaaS providers identify a suitable collection of computing nodes to which to deploy the users' applications.

• Pricing Module. This module is responsible for charging users according to the SLA they signed. Different parameters can be considered in charging users; for instance, the most common case for IaaS providers is to charge according to the characteristics of the virtual machines requested in terms of memory, disk size, computing capacity, and the time they are used. It is very common to calculate the usage in time blocks of one hour, but several other pricing schemes exist. PaaS providers can charge users based on the number of requests served by their application or the usage of internal services made available by the development platform to the application while running.

• Accounting Module. This module maintains the actual information on usage of resources and stores the billing information for each user. These data are made available to the Service Request Examiner and Admission Control module when assessing users' requests. In addition, they constitute a rich source of information that can be mined to identify usage trends and improve the vendor's service offering.

• Dispatcher. This component is responsible for the low-level operations that are required to realize admitted service requests. In an IaaS scenario, this module instructs the infrastructure to deploy as many virtual machines as are needed to satisfy a user's request. In a PaaS scenario, this module activates and deploys the user's application on a selected set of nodes; deployment can happen either within a virtual machine instance or within an appropriate sandboxed environment.

• Resource Monitor. This component monitors the status of the computing resources, either physical or virtual. IaaS providers mostly focus on keeping track of the availability of VMs and their resource entitlements. PaaS providers monitor the status of the distributed middleware, enabling the elastic execution of applications and loading of each node.

• Service Request Monitor. This component keeps track of the execution progress of service requests. The information collected through the Service Request Monitor is helpful for analyzing system performance and for providing quality feedback about the provider's capability to satisfy requests. For instance, elements of interest are the number of requests satisfied versus the number of incoming requests, the average processing time of a request, or its time to execution. These data are important sources of information for tuning the system. The SLA allocator executes the main logic that governs the operations of a single datacenter or a collection of datacenters. Features such as failure management are most likely to be addressed by other software modules, which can either be a separate layer or can be integrated within the SLA resource allocator.

• Virtual machines (VMs). Virtual machines constitute the basic building blocks of a cloud computing infrastructure, especially for IaaS providers. VMs represent the unit of deployment for addressing users' requests. Infrastructure management software is in charge of keeping operational the computing infrastructure backing the provider's commercial service offering. VMs play a fundamental role in providing an appropriate hosting environment for users' applications and, at the same time, isolate application execution from the infrastructure, thus preventing applications from harming the hosting environment. Moreover, VMs are among the most important components influencing the QoS with which a user request is served. VMs can be tuned in terms of their emulated hardware characteristics so that the amount of computing resource of the physical hardware allocated to a user can be finely controlled. PaaS providers do not directly expose VMs to the final user, but they may internally leverage virtualization technology in order to fully and securely utilize their own infrastructure. PaaS providers often leverage given middleware for executing user applications and might use different QoS parameters to charge application execution rather than the emulated hardware profile.

• Physical machines. At the lowest level of the reference architecture resides the physical infrastructure that can comprise one or more datacenters. This is the layer that provides the resources to meet service demands.

Energy-Efficient and Green Cloud Computing Architecture [18]

A high-level architecture for supporting energy-efficient resource allocation in a green cloud computing infrastructure consists of four main components:

• Consumers/brokers. Cloud consumers or their brokers submit service requests from anywhere in the world to the cloud. It is important to note that there can be a difference between cloud consumers and users of deployed services. For instance, a consumer can be a company deploying a Web application, which presents varying workloads according to the number of "users" accessing it.

• Green Resource Allocator. Acts as the interface between the cloud infrastructure and consumers. It requires the interaction of the following components to support energy-efficient resource management:

• Green Negotiator. Negotiates with the consumers/brokers to finalize the SLAs with specified prices and penalties (for violations of SLAs) between the cloud provider and the consumer, depending on the consumer's QoS requirements and energy-saving schemes. In Web applications, for instance, the QoS metric can be 95% of requests being served in less than 3 seconds.

• Service Analyzer. Interprets and analyzes the service requirements of a submitted request before deciding whether to accept or reject it. Hence, it needs the latest load and energy information from VM Manager and Energy Monitor, respectively.

• Consumer Profiler. Gathers specific characteristics of consumers so that important consumers can be granted special privileges and prioritized over other consumers.

• Pricing. Decides how service requests are charged to manage the supply and demand of computing resources and facilitate prioritizing service allocations effectively.

• Energy Monitor. Observes and determines which physical machines to power on or off.

• Service Scheduler. Assigns requests to VMs and determines resource entitlements for allocated VMs. It also decides when VMs are to be added or removed to meet demand.

• VM Manager.Keeps track of the availability of VMs and their resource entitlements. It is also in charge of migrating VMs across physical machines.

• Accounting. Maintains the actual usage of resources by requests to compute usage costs. Historical usage information can also be used to improve service allocation decisions.

• VMs. Multiple VMs can be dynamically started and stopped on a single physical machine to meet accepted requests, hence providing maximum flexibility to configure various partitions of resources on the same physical machine to different specific requirements of service requests. Multiple VMs can also run concurrently applications based on different operating system environments on a single physical machine. In addition, by dynamically migrating VMs across physical machines, workloads can be consolidated and unused resources can be put on a low power state, turned off, or configured to operate at low performance levels (e.g., using Dynamic Voltage and Frequency Scaling, or DVFS) to save energy.

• Physical machines. The underlying physical computing servers provide hardware infrastructure for creating virtualized resources to meet service demands.

Energy-aware dynamic resource allocation

Recent developments in virtualization have resulted in its use across datacenters. Virtualization enables dynamic migration of VMs across physical nodes according to QoS requirements. Unused VMs can be logically resized and consolidated on a minimal number of physical nodes, while idle nodes can be turned off (or hibernated). Through consolidation of VMs, large numbers of users can share a single physical server, which increases utilization and in turn reduces the total number of servers required. Moreover, VM consolidation can be applied dynamically by capturing the workload variability and adapting the VM placement at runtime using migration.

Currently, resource allocation in a cloud datacenter aims at providing high performance while meeting SLAs, with limited or no consideration for energy consumption during VM allocations. However, to explore both performance and energy efficiency, two crucial issues must be addressed. First, turning off resources in a dynamic environment puts QoS at risk; aggressive consolidation may cause some VMs to obtain insufficient resources to serve a spike in load. Second, agreed SLAs bring challenges to application performance management in virtualized environments. These issues require effective consolidation policies that can minimize energy use without compromising user QoS requirements. The current approaches to dynamic VM consolidation are weak in terms of providing performance guarantees. One of the ways to prove performance bounds is to divide the problem of energy-efficient dynamic VM consolidation into a few sub problems that can be analyzed individually. It is important to analytically model the problem and derive optimal and near optimal approximation algorithms that provide provable efficiency. To achieve this goal, clouds need novel analytical models and QoS-based resource

allocation algorithms that optimize VM placements with the objective of minimizing energy consumption under performance constraints.

Inter Clouds and integrated allocation of resources

Cloud providers have been deploying datacenters in multiple locations throughout the globe. For example, Amazon EC2 Cloud services are available via Amazon datacenters located in the United States, Europe, and Singapore. This disbursement is leading to the emergence of a notion, called the Inter Cloud, supporting scalable delivery of application services by harnessing multiple datacenters from one or more providers. In addition to enhancing performance and reliability, these Inter Clouds provide a powerful means of reducing energy-related costs. One reason is that the local demand for electricity varies with time of day and weather. This causes time-varying differences in the price of electricity at each location. Moreover, each site has a different source of energy (such as coal, hydroelectric, or wind), with different environmental costs. This gives scope to adjust the load sent to each location, and the number of servers powered on at each location, to improve efficiency.

In such environments, algorithms that make routing decisions by considering the location of the user, the energy-efficiency of the hardware at each site, the energy mix, and the number of servers currently on at each location are needed. A particularly promising approach is to use this routing to make work “follow the renewables.” A major problem with renewable energy is that most sources are intermittent and uncontrollable. Dynamically routing requests to locations with available renewable energy can greatly reduce the nonrenewable energy used and facilitate the widespread use of clean energy. Sending loads to remote datacenters incurs both delay costs and energy costs due to the increased amounts of data that are transferred over the Internet. Improvements in energy-efficient transport technology should lead to significant reductions in the power consumption of the cloud software services.

Resource Allocation, Leases [20-21]

Resource allocation: I am allocating say 20 computers for your application.

Job scheduling: Your application will have many jobs. They needs to be allocate to different resources deepening on application structure and job dependencies and how execution is progressing.

An advance reservation lease (A_i) is a kind of lease that needs to start and end at specific times. We present the AR lease in the form of 4-tuple as follows:

$$A_i = <S, E, N, T>$$

where S is the start time of the lease A_i, E is the execution time of the lease A_i, N is the number of nodes required to complete the lease A_i and T is the execution type of the lease A_i. The end time of the lease A_i (ET) is the sum of start time (S) and execution time (E). For example, a lease A_1 = <8:00, 1:00, 3, 0> indicates that A_1 starts at 8:00 AM and requires a non-preemptive 3 nodes and 1 hour period which is shown in Table 1. Note that T is a Boolean variable defined as follows:

$$T = \begin{cases} 0 \text{ Non-Preemptive} \\ 1 \text{ Preemptive} \end{cases}$$

Table 1. AR lease

Node 1	A_1
Node 2	
Node 3	
Node 4	
Node 5	

8 AM 9 AM

In general, AR lease is non-preemptive in nature and it has higher priority over all the types of leases.

Unlike AR, a BE lease (B_i) do not require to start at a specific time. Indeed, the customer is ready to wait until the resources become available. We present the BE lease in the form of 4-tuple as follows:

$$B_i = <A, E, N, T>$$

where A = Arrival time of the lease B_i . For example, a lease B_1 = <6:00, 1:00, 4, 1> indicates that B_1needs a preemptive, 4-nodes and 1 hour period as shown in Table 2. However, there is no

specific start time. We assume that at 9 AM, the resources become available. Hence, the lease B_1 is scheduled from 9 to 10 AM.

Table 2. BE lease

Node 1	B_1
Node 2	
Node 3	
Node 4	
Node 5	

9 AM 10 AM

BE lease is preemptive in nature and it has lower priority than AR lease. When a user request BE lease, the request is placed in a queue and executed in first come first serve basis.

Let us assume that at 9:10 AM, an AR lease A_2 = <9:30, 1:00, 5, 0> is arrived. As a result, it preempts the currently executing BE lease B_1 and resumes B_1 after completion of the AR lease A_2which is clearly shown in Table 3.

Table 3. Preemptive BE lease

Node 1	B_1	A_2	B_1
Node 2			
Node 3			
Node 4			
Node 5			

9 AM 9:30 AM 10:30 AM 11 AM

A non-preemptive best effort lease (NB_i) is a lease that does not allow to preempt the BE lease intermittently on arrival of higher priority lease such as AR. It is desired as preemption is the major overhead of the BE lease. The only difference between the BE and NBE lease is the last attribute of 4-tuple (i.e., T) that is 1 and 0 respectively. Table 3 is regenerated as Table 4 with the non-preemptive BE lease NB_1= <6:00, 1:00, 4, 0>.

Table 4. Non-Preemptive BE lease

Node 1	NB_1	A_2
Node 2		
Node 3		
Node 4		
Node 5		

9 AM 10 AM 11 AM

An immediate lease (I_i) is a lease that needs resources on its arrival [6]. The lease may be accepted (or rejected) based on the availability of resources. It is non-preemptive in nature. We represent the IM lease in the form of 4-tuple as follows:

$$I_i = <S, E, N, T>$$

For example, a lease I_1 = <10:00, 2:00, 4, 0> indicates that I_1 starts at 10 AM and requires a non-preemptive, 4nodes and 2 hour period which is shown in Table 5.

Table 5. Immediate lease

Node 1	I_1
Node 2	
Node 3	
Node 4	
Node 5	

10 AM 12 Noon

The only difference between the IM lease and other leases is that the arrival time of the IM lease is same as the start time whereas it need not be true for other leases.

A deadline lease (D_i) is a lease that needs resources before the given deadline. It is preemptive in nature but there is restriction to their preempt ability, which says that the lease is preempted if and only if the lease manager assures that it will be completed before the deadline. We represent the DS lease in the form of 5-tuple as follows:

$$D_i = <S, E, N, T, D>$$

where D = Deadline of the lease D_i. For example, a lease D_1 = <9:00, 1:00, 5, 1, 12:00> indicates that D_1 starts at 9 AM and requires a 5-nodes and 1 hour period on or before 12:00 noon. The two possible Gantt charts are shown in Tables 6-7.

Table 6. Deadline Sensitive lease

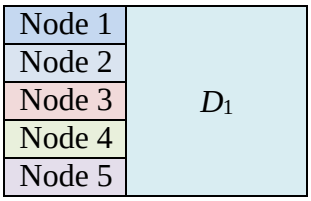
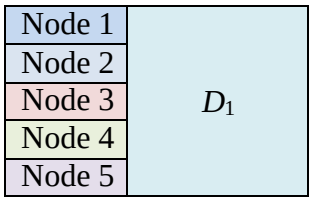


Table 7. Deadline Sensitive lease



A best effort with deadline lease (DB_i) is a lease that needs resources on or before the given deadline. It is preemptive in nature. It is desirable as the continuous arrival of AR leases will lead the BE leases to starvation. Hence, deadline is enforced to avoid the starvation among the BE leases. Sometimes, BED lease is referred as DS lease [2]. However, we assume that a BED lease may not be complete before the deadline whereas DS does. We present the BED lease in the form of 5-tuple as follows:

$$DB_i = <A, E, N, T, D>$$

For example, a lease DB_1 = <6:00, 1:00, 4, 1, 12:00> indicates that DB_1 requires a 4 nodes and 1 hour period on or before 12:00 noon. The two possible Gantt charts are shown in Tables 8-9.

Table 8. Best Effort with Deadline lease

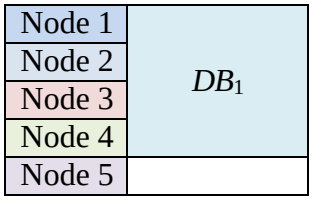
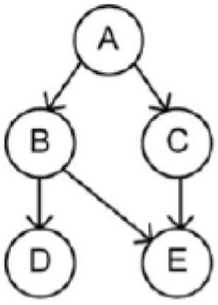


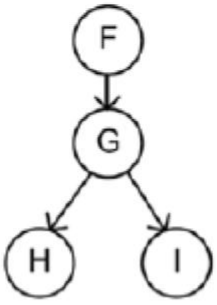
Table 9. Best Effort with Deadline lease

A negotiated lease (NE_i) is a lease that offers resources with respect to the time. It is essential if and only if there is no sufficient resources and need of the customers are flexible. It is preemptive in nature. For example, a lease NE_i requires 2-nodes and 1 hour slot before the given deadline 12 noon. Then the lease resource manager (say, Haizea) may offer the customer as follows. 1) The manager will offer a 4-nodes and 1 hour slot to the customer if he/she extends the deadline to 4 PM. 2) A 4-nodes, 2 hours slot can be offered if the deadline is extended to 6 PM. 3) A 50% discount may be offered to extend the deadline to 11 PM.

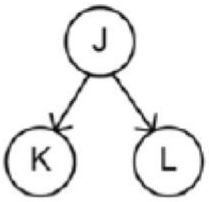
Task Scheduling: RR, CLS and CMMS [22]



Application 1,
Arrival time: 0



Application 2,
Arrival time: 3



Application 3,
Arrival time: 9

	A	B	C	D	E	F	G	H	I	J	K	L
Cloud 1	2	6	5	7	5	4	8	2	4	8	9	2
Cloud 2	3	8	3	10	9	2	8	3	5	4	3	3
Cloud 3	5	4	8	5	2	3	4	6	7	6	7	4

	A	B	C	D	E	F	G	H	I	J	K	L
RR	1	2	3	1	2	1	2	3	1	1	2	3
Sch	1	3	2	3	3	2	3	1	2	2	2	1

---

time	0~2	3~7	9~15	15~19	19~26	26~28
Cloud 1	A	F	J	I	D	J
time		2~7	7~15	15~18	18~27	28~31
Cloud 2		B	G	B	E	K
time		2~10		15~21	28~32	
Cloud 3		C		H	L	

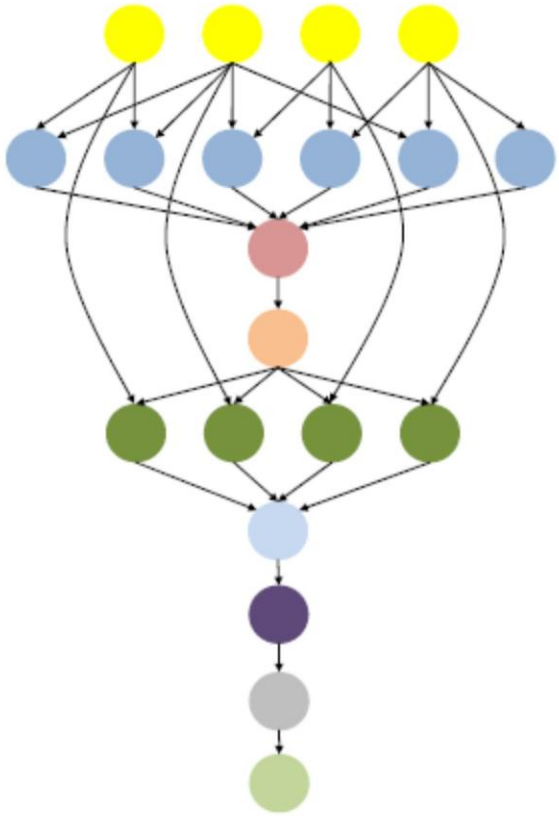
time	0~2				9~11			18~20
Cloud 1	A				H			L
time		2~3	3~5	5~7	9~14	14~18		18~21
Cloud 2		C	F	C	I	J		K
time		2~5		5~9	9~10	10~15	15~17	
Cloud 3		B		G	B	D	E	

As shown in the ‘‘RR’’ row of Figure, the tasks are assigned to the clouds evenly, regardless of the heterogeneous performance across different clouds. The execution orders of three clouds are presented in next Figure. In this schedule, task G preempts task B at time 7, since task G is an AR task. And task J is scheduled as soon as possible, starting at time 9, pausing at time 15, and resuming right after previously assigned tasks, i.e., tasks I and D. The total execution time is 32. We assume the execution time of a given application starts from the time when the application is submitted to the time when the application is done. With this scheduling, the average of three application execution time is 22.67 time unit. By using our CLS algorithm, we generate a schedule with the consideration of the heterogeneous performance in the cloud system. The tasks assignment is shown in the ‘‘Sch’’ row of Figure. And the execution order of three clouds are shown in Figure. In this schedule, tasks are likely assigned to the cloud that can execute them in the shortest time. Task F and G preempt task C and B, respectively. The total execution time is only 21 time unit, which is 34% faster than the round-robin schedule. And the average execution time is 13.33, 41% faster than the round-robin schedule.

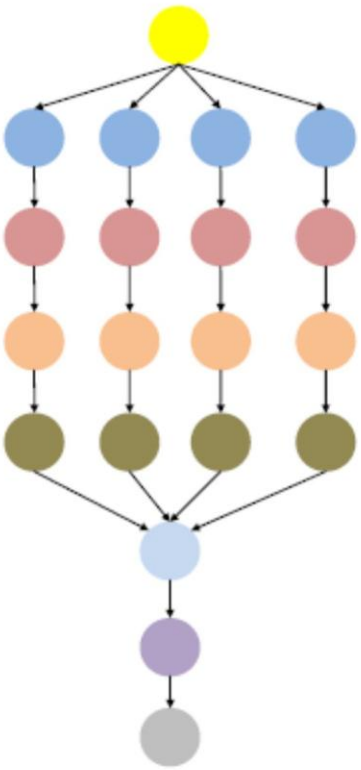
Workflow Scheduling, Montage, Epigenomics, SIPHT, LIGO, CyberShake [23]

The structure of five realistic workflows from diverse scientific applications, which are:

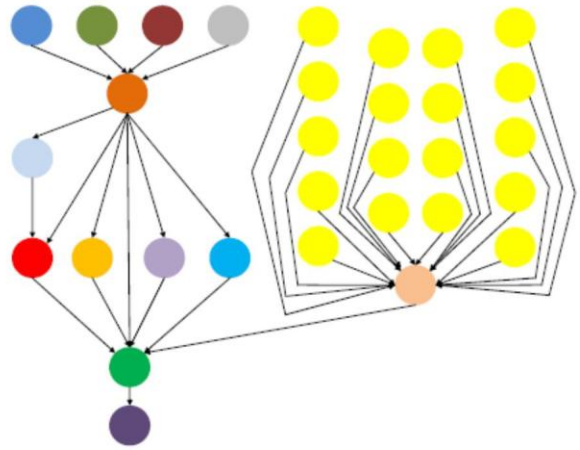
1. Montage: astronomy
2. CyberShake: earthquake science
3. Epigenomics: biology
4. LIGO: gravitational physics
5. SIPHT: biology.



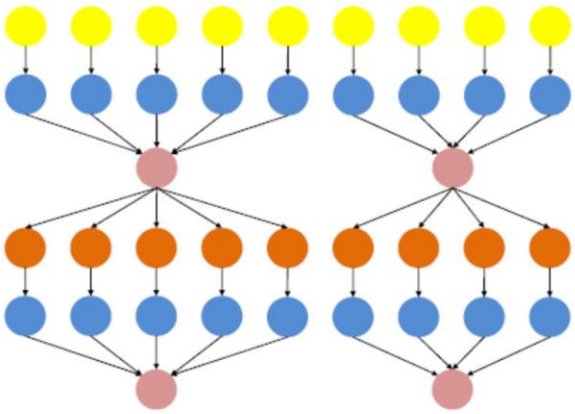
(a) Montage.



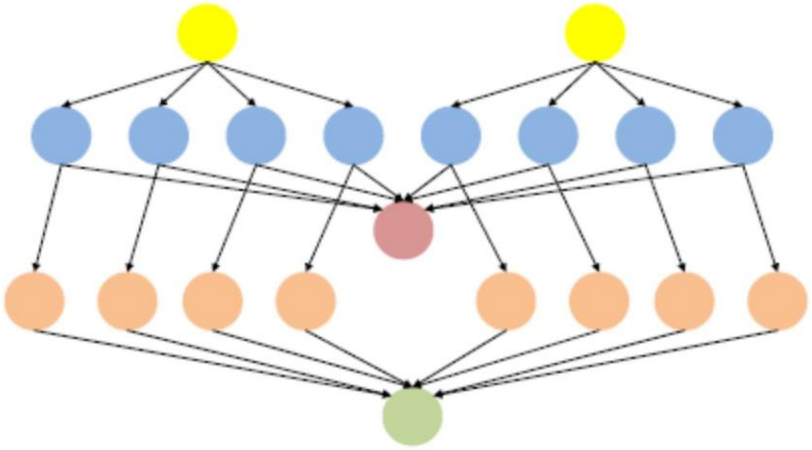
(b) Epigenomics.



(c) SIPHT.



(d) LIGO.



(e) CyberShake.

Task Consolidation [24]

Task consolidation is a process of maximizing resource utilization in a cloud system. However, maximum usage of resources does not necessarily imply that there will be proper use of energy as some resources which are sitting idle, also consume considerable amount of energy. Recent studies show that energy consumption due to idle resources is approximately 1 to 20%. So, the idle resources are assigned with some tasks to utilize the idle period, which in turn reduces the overall energy consumption of the resources. Note that higher resource utilization merely leads to high energy consumption. So, the tasks are likely to be assigned to all the resources for the proper use of energy.

Assume a typical example in which approximate power consumption of VM with respect to the utilization is shown in Table I. This is the same table as used in [25]. The utilization value is divided into four levels based on the power consumption in contrast to six levels as addressed by [12]. The utilization values from 1 to 20 are called idle level, from 21 to 50 is level 1, from 51 to 70 is level 2 and 71 to 100 is level 3 [26]. Here, we assume that an idle resource draws 20% utilization. Therefore, the idle resource consumes 78.5 watts as per Table I. Note that the idle resource consumes 78.5 watts even if the resource does not execute a single task. In contrary, a resource with 50% utilization consumes 334.5 watts (i.e., a total of 78.5 + 83 + 85 + 88 watts). Now suppose, the basic information of a set of tasks with their identity (*ID*), arrival time (*AT*), start time (*ST*), period (*P*), finish time (*FT*) and utilization (*U*) is given as shown in Table II. The power consumption of the tasks with respect to utilization is given in Table I, e.g., T_1 has power consumption of 83 watts as it has 25% utilization. There are three different tasks with arrival time 0, 4 and 10 respectively. These tasks are needed 25% utilization for 50 time units, 35% for 40 time units and 30% utilization for 54 time units respectively. If we dispatch task T_1 to an idle VMs, then it consumes 6000 watts (i.e., $(78.5 + (8.3 \times 5)) \times 50$ as per Table I) as task T_1 requires a utilization of 25%. A single VM can accommodate these tasks as sum of the utilization value (i.e., 90%) does not exceed 100%. But, the energy consumption is drastically increased. One of the solutions is to start another idle VM that can share some workloads. So, the energy consumption is reduced up to some extent. This is the main motivation behind the proposed work.

Power Consumption of VMs with respect to Utilization [25]

Level	Utilization	Power Consumption (in Watts (approx.)) (p_i)
Idle	1~20	p_{20} = 78.5
1	21~30	p_{30} = 83
	31~40	p_{40} = 85
	41~50	p_{50} = 88
2	51~60	p_{60} = 93
	61~70	p_{70} = 102
3	71~80	p_{80} = 109
	81~90	p_{90} = 122
	91~100	p_{100} =136

A Set of Tasks with Their 6-Tuple

ID	*AT*	*ST*	*P*	*FT*	*U*
T_1	0	2	50	52	25%
T_2	4	7	40	47	35%
T_3	10	15	54	69	30%

A VM may have following states: execute, sleep/awake and idle. However, we use only two states i.e., execute and idle for simplicity. When a VM is computing tasks then that VM is present in the execute state. In contrary, when a VM is not computing any task or waiting for a task to arrive, the VM will be in the idle state. On the other hand, when a VM receives a task, it makes a transition from idle-to-execute state. It returns back to idle state by making execute-to-idle state transition, once the task is successfully executed.

Introduction to CloudSim, Cloudlet, Virtual Machine and its Provisioning, Time and Spaceshared Provisioning [27]

CloudSim offers the following novel features:

(i) support for modeling and simulation of large scale Cloud computing environments, including data centers, on a single physical computing node;

(ii) a self-contained platform for modeling Clouds, service brokers, provisioning, and allocations policies;

(iii) support for simulation of network connections among the simulated system elements; and

(iv) facility for simulation of federated Cloud environment that inter-networks resources from both private and public domains, a feature critical for research studies related to Cloud-Bursts and automatic application scaling.

Some of the unique features of CloudSim are:

(i) availability of a virtualization engine that aids in creation and management of multiple, independent, and co-hosted virtualized services on a data center node; and

(ii) flexibility to switch between space-shared and time-shared allocation of processing cores to virtualized services. These compelling features of CloudSim would speed up the development of new application provisioning algorithms for Cloud computing.

Physical Cloud resources along with core middleware capabilities form the basis for delivering IaaS and PaaS. The user-level middleware aims at providing SaaS capabilities. The top layer focuses on application services (SaaS) by making use of services provided by the lower layer services. PaaS/SaaS services are often developed and provided by 3rd party service providers, who are different from the IaaS providers.

Cloud Applications: This layer includes applications that are directly available to end-users. We define end-users as the active entity that utilizes the SaaS applications over the Internet. These applications may be supplied by the Cloud provider (SaaS providers) and accessed by end-users either via a subscription model or a pay-per-use basis. Alternatively, in this layer, users deploy their own applications. In the former case, there are applications such as Salesforce.com that supply business process models on clouds (namely, customer relationship management software) and social networks. In the latter, there are e-Science and e-Research applications, and Content-Delivery Networks.

User-Level Middleware: This layer includes the software frameworks such as Web 2.0 Interfaces (Ajax, IBM Workplace) that help developers in creating rich, cost-effective user-

interfaces for browser-based applications. The layer also provides those programming environments and composition tools that ease the creation, deployment, and execution of applications in clouds. Finally, in this layer several frameworks that support multi-layer applications development, such as Spring and Hibernate, can be deployed to support applications running in the upper level.

Core Middleware: This layer implements the platform level services that provide runtime environment for hosting and managing User-Level application services. Core services at this layer include Dynamic SLA Management, Accounting, Billing, Execution monitoring and management, and Pricing (are all the services to be capitalized?). The well-known examples of services operating at this layer are Amazon EC2, Google App Engine, and Aneka. The functionalities exposed by this layer are accessed by both SaaS and IaaS services. Critical functionalities that need to be realized at this layer include messaging, service discovery, and load balancing. These functionalities are usually implemented by Cloud providers and offered to application developers at an additional premium. For instance, Amazon offers a load balancer and a monitoring service (Cloudwatch) for the Amazon EC2 developers/consumers. Similarly, developers building applications on Microsoft Azure clouds can use the .NET Service Bus for implementing message passing mechanism.

System Level: The computing power in Cloud environments is supplied by a collection of data centers that are typically installed with hundreds to thousands of hosts. At the System Level layer there exist massive physical resources (storage servers and application servers) that power the data centers. These servers are transparently managed by the higher level virtualization services and toolkits that allow sharing of their capacity among virtual instances of servers. These VMs are isolated from each other, thereby making fault tolerant behavior and isolated security context possible.

Initial releases of CloudSim used SimJava as discrete event simulation engine that supports several core functionalities, such as queuing and processing of events, creation of Cloud system entities (services, host, data center, broker, virtual machines), communication between components, and management of the simulation clock. However in the current release, SimJava layer has been removed in order to allow some advanced operations that are not supported by it.

The CloudSim simulation layer provides support for modeling and simulation of virtualized Cloud-based data center environments including dedicated management interfaces for virtual machines (VMs), memory, storage, and bandwidth. The fundamental issues such as provisioning of hosts to VMs, managing application execution, and monitoring dynamic system state are handled by this layer. A Cloud provider, who wants to study the efficiency of different policies in allocating its hosts to VMs (VM provisioning), would need to implement their strategies at this layer. Such implementation can be done by programmatically extending the core VM provisioning functionality. There is a clear distinction at this layer related to provisioning of

hosts to VMs. A Cloud host can be concurrently allocated to a set of VMs that execute applications based on SaaS provider's defined QoS levels. This layer also exposes functionalities that a Cloud application developer can extend to perform complex workload profiling and application performance study. The top-most layer in the CloudSim stack is the User Code that exposes basic entities for hosts (number of machines, their specification and so on), applications (number of tasks and their requirements), VMs, number of users and their application types, and broker scheduling policies. By extending the basic entities given at this layer, a Cloud application developer can perform following activities: (i) generate a mix of workload request distributions, application configurations; (ii) model Cloud availability scenarios and perform robust tests based on the custom configurations; and (iii) implement custom application provisioning techniques for clouds and their federation.

As Cloud computing is still an emerging paradigm for distributed computing, there is a lack of defined standards, tools and methods that can efficiently tackle the infrastructure and application level complexities. Hence, in the near future there would be a number of research efforts both in academia and industry towards defining core algorithms, policies, and application benchmarking based on execution contexts. By extending the basic functionalities already exposed with CloudSim, researchers will be able to perform tests based on specific scenarios and configurations, thereby allowing the development of best practices in all the critical aspects related to Cloud Computing.

The infrastructure-level services (IaaS) related to the clouds can be simulated by extending the Datacenter entity of CloudSim. The Data Center entity manages a number of host entities. The hosts are assigned to one or more VMs based on a VM allocation policy that should be defined by the Cloud service provider. Here, the VM policy stands for the operations control policies related to VM life cycle such as: provisioning of a host to a VM, VM creation, VM destruction, and VM migration. Similarly, one or more application services can be provisioned within a single VM instance, referred to as application provisioning in the context of Cloud computing. In the context of CloudSim, an entity is an instance of a component. A CloudSim component can be a class (abstract or complete), or set of classes that represent one CloudSim model (data center, host).

A Datacenter can manage several hosts that in turn manage VMs during their life cycles. Host is a CloudSim component that represents a physical computing server in a Cloud: it is assigned a pre-configured processing capability (expressed in millions of instructions per second – MIPS), memory, storage, and a provisioning policy for allocating processing cores to virtual machines. The Host component implements interfaces that support modeling and simulation of both single-core and multi-core nodes.

VM allocation (provisioning) is the process of creating VM instances on hosts that match the critical characteristics (storage, memory), configurations (software environment), and

requirements (availability zone) of the SaaS provider. CloudSim supports the development of custom application service models that can be deployed within a VM instance and its users are required to extend the core Cloudlet object for implementing their application services. Furthermore, CloudSim does not enforce any limitation on the service models or provisioning techniques that developers want to implement and perform tests with. Once an application service is defined and modeled, it is assigned to one or more pre-instantiated VMs through a service specific allocation policy. Allocation of application-specific VMs to Hosts in a Cloud-based data center is the responsibility of a Virtual Machine Allocation controller component (called VmAllocationPolicy). This component exposes a number of custom methods for researchers and developers that aid in implementation of new policies based on optimization goals (user centric, system centric or both). By default, VmAllocationPolicy implements a straightforward policy that allocates VMs to the Host in First-Come-First-Serve (FCFS) basis. Hardware requirements such as the number of processing cores, memory and storage form the basis for such provisioning. Other policies, including the ones likely to be expressed by Cloud providers, can also be easily simulated and modeled in CloudSim. However, policies used by public Cloud providers (Amazon EC2, Microsoft Azure) are not publicly available, and thus a pre-implemented version of these algorithms is not provided with CloudSim.

For each Host component, the allocation of processing cores to VMs is done based on a host allocation policy. This policy takes into account several hardware characteristics such as number of CPU cores, CPU share, and amount of memory (physical and secondary) that are allocated to a given VM instance. Hence, CloudSim supports several simulation scenarios that assign specific CPU cores to specific VMs (a space-shared policy) or dynamically distribute the capacity of a core among VMs (time-shared policy); and assign cores to VMs on demand.

Each Host component also instantiates a VM scheduler component, which can either implement the space-shared or the time-shared policy for allocating cores to VMs. Cloud system/application developers and researchers, can further extend the VM scheduler component for experimenting with custom allocation policies. In the next section, the finer level details related to the time-shared and space-shared policies are described. Fundamental software and hardware configuration parameters related to VMs are defined in the VM class. Currently, it supports modeling of several VM configurations offered by Cloud providers such as the Amazon EC2.

References

1. R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg and I. Brandic, Cloud Computing and Emerging IT Platforms: Vision, Hype and Reality for Delivering Computing as the 5th Utility, Future Generation Computer Systems, Elsevier, Vol. 25, pp. 599-616, 2009.
2. G. F. Pfister, In Search of Clusters, 2nd Edition, Prentice Hall, 1998.
3. R. Buyya (Ed.), High Performance Cluster Computing: Architectures and Systems, Vol. 1, Prentice Hall, 1999.
4. Top 500 List, http://top500.org/lists/2014/11/, Accessed on 29th January 2015.
5. K. Hwang and Z. Xu, Scalable Parallel Computing: Technology, Architecture, Programming, McGraw-Hill, 1998.
6. Granularity, http://en.wikipedia.org/wiki/Granularity, Accessed on 30th January 2015.
7. Distributed Shared Memory, http://en.wikipedia.org/wiki/Distributed_shared_memory, Accessed on 30th January 2015.
8. Kernel (Operating System), http://en.wikipedia.org/wiki/Kernel_(operating_system), Accessed on 30th January 2015.
9. N. Boden, Myrinet – A Gigabit-per-Second Local-Area Network. IEEE Micro, 1995.
10. MPI, https://computing.llnl.gov/tutorials/mpi/, Accessed on 19th February 2015.
11. PVM, http://www.csm.ornl.gov/pvm/, Accessed on 19th February 2015.
12. Oracle Grid Engine, http://en.wikipedia.org/wiki/Oracle_Grid_Engine, Accessed on 19th February 2015.
13. CODINE, http://www.angelfire.com/mi/codine1/codine.html, Accessed on 19th February 2015.
14. I. P. Egwutuoha, D. Levy, B. Selic and S. Chen, "A Survey of Fault Tolerance Mechanisms and Checkpoint/Restart Implementations for High Performance Computing Systems", The Journal of Supercomputing, Vol. 65, pp. 1302-1326, 2013.
15. Computer Architecture Links, http://www.cs.wisc.edu/~arch/www/.
16. Heterogeneous Cluster Computing, http://www.nec-labs.com/research/heterogeneous-cluster-computing/80, Accessed on 19th February 2015.
17. I. Foster, C. Kesselman and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", International Journal of Supercomputer Applications, 2001.
18. R. Buyya, C. Vecchiola and S. T. Selvi, "Mastering Cloud Computing: Foundations and Applications Programming", Morgan Kaufmann, Elsevier, 2013.
19. B. Sosinsky, "Cloud Computing Bible". Wiley Publishing, 2011.
20. Haizea, http://haizea.cs.uchicago.edu/whatis.html, Accessed on 28th December 2014.
21. S. K. Panda and P. K. Jana, "Novel Leases for IaaS Cloud". (Yet to Appear)
22. J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin and Z. Gu, "Online Optimization for Scheduling Preemptable Tasks on IaaS Cloud System", Journal of Parallel Distributed Computing, Elsevier, Vol. 72, pp. 666-677, 2012.

23. S. Abrishami, M. Naghibzadeh and D. H. J. Epema, "Deadline-Constrained Workflow Scheduling Algorithms for Infrastructure as a Service Clouds", Future Generation Computer Systems, Vol. 29, pp. 158-169, 2013.
24. S. K. Panda and P. K. Jana, "An Efficient Energy Saving Task Consolidation Algorithm for Cloud Computing", 3rd IEEE International Conference on Parallel, Distributed and Grid Computing (PDGC), IEEE, pp. 262-267, 2014.
25. Lien, C., Liu, M. F., Bai, Y., Lin, C. H. and Lin, M. 2006. Measurement by the Software Design for the Power Consumption of Streaming Media Servers. Instrumentation and Measurement Technology Conference, IEEE. 1597-1602.
26. Hsu, C., Slagter, K. D., Chen, S. and Chung, Y. Optimizing Energy Consumption with Task Consolidation in Clouds. 2014. Information Sciences, Elsevier. 258, 452-462.
27. R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose and R. Buyya, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms", pp. 1-24.