**CLOUD COMPUTING**

**UNIT-4**

UNIT IV: Cloud Resource Management and Scheduling: Policies and Mechanisms for Resource Management, Applications of Control Theory to Task Scheduling on a Cloud, Stability of a Two Level Resource Allocation Architecture, Feedback Control Based on Dynamic Thresholds. Coordination of Specialized Autonomic Performance Managers, Resource Bundling, Scheduling Algorithms for Computing Clouds-Fair Queuing, Start Time Fair Queuing.

**Policies and Mechanisms for Resource Management**

Resource management is a core function of any man-made system. It affects the three basic criteria for the evaluation of a system: performance, functionality, and cost. An inefficient resource management has a direct negative effect on performance and cost and an indirect effect on the functionality of a system.

Cloud resource management requires complex policies and decisions for multi-objective optimization. Cloud resource management is extremely challenging because of the complexity of the system, which makes it impossible to have accurate global state information, and because of the unpredictable interactions with the environment. The strategies for resource management associated with the three cloud delivery models, IaaS, PaaS, and SaaS, differ from one another.

POLICIES AND MECHANISMS FOR RESOURCE MANAGEMENT A policy typically refers to the principal guiding decisions, whereas mechanisms represent the means to implement policies. Separation of policies from mechanisms is a guiding principle in computer science. Cloud resource management policies can be loosely grouped into five classes:

1. Admission control.

2. Capacity allocation.

3. Load balancing.

4. Energy optimization.

5. Quality-of-service (QoS) guarantees The explicit goal of anadmissioncontrolpolicy is to prevent the system from accepting workloads in violation of high-level system policies.

Capacityallocationmeans to allocate resources for individual instances; an instance is an activation of a service.

Loadbalancingandenergyoptimizationcan be done locally, but global load-balancing and energy optimization policies encounter the same difficulties. Load balancing and energy optimization are correlated and affect the cost of providing the services Qualityof serviceis that aspect of resource management that is probably the most difficult to address and, at the same time, possibly the most critical to the future of cloud computing.

Control theory application to cloud resource management (CRM)

The resource management policies are defined as addressing multiple concerns. The resource management policies are divided into five categories such as load balancing, capacity allocation, admission control, energy optimization, and QoS guarantees. The main goal is to prevent the system from accepting workload.

It is the management that generally managed the resources and within the policies. It is known as resource management.

**Control Theory**

The feedback mechanism is generally used by control theory and it is also used to predict the nature of the system. The control theory uses feedback but it is only used to predict local behavior. The global behavior is second prior and it also uses Kalman filters for simple models. This is known as the control theory.

**Benefits**

The benefits of control theory are given below:

• The main benefit of control theory is to optimize system use, analyze data storage, protection from deadlock, and internet connections.

• The main use of control theory is scheduling, late scheduling in some resources, and techniques for allocating resources.

• The control theory concept is also used to design self-managing systems. It does not need outer help from other techniques.

• It also uses Pontryagin’s principle to know the better possible control and it changes the dynamic system from one to another.

**Problems**

The problems of control theory are given below:

• The main problem of control theory is to first predict local behavior and at last, it predicts global behavior.

• The control theory is too complex and it is a solid foundation. It does not also scale well as compared to other mechanisms.

• It does not use the control theory for optimizing other non-behavior resources which are apart from the system.

• The problem is to handle the speed and technique during the prediction of local behavior and global behavior.

Stability of a Two Level Resource Allocation Architecture:

**Resource Allocation Methods in Cloud Computing**

The allocation of resources and services from a cloud provider to a customer is known as resource provisioning in cloud computing, sometimes called cloud provisioning. Resource provisioning is the process of choosing, deploying, and managing software (like load balancers and database server management systems) and hardware resources (including CPU, storage, and networks) to assure application performance.

To effectively utilize the resources without going against SLA and achieving the QoS requirements, Static Provisioning/Dynamic Provisioning and Static/Dynamic Allocation of resources must be established based on the application needs. Resource over and under-provisioning must be prevented. Power usage is another significant restriction. Care should be taken to reduce power consumption, dissipation, and VM placement. There should be techniques to avoid excess power consumption.

Therefore, the ultimate objective of a cloud user is to rent resources at the lowest possible cost, while the objective of a cloud service provider is to maximize profit by effectively distributing resources.

### Importance of Cloud Provisioning:

* **Scalability:**Being able to actively scale up and down with flux in demand for resources is one of the major points of cloud computing
* **Speed:**Users can quickly spin up multiple machines as per their usage without the need for an IT Administrator
* **Savings:**Pay as you go model allows for enormous cost savings for users, it is facilitated by provisioning or removing resources according to the demand

### Challenges of Cloud Provisioning:

* **Complex management:**Cloud providers have to use various different tools and techniques to actively monitor the usage of resources
* **Policy enforcement:** Organisations have to ensure that users are not able to access the resources they shouldn’t.
* **Cost:**Due to automated provisioning costs may go very high if attention isn’t paid to placing proper checks in place. Alerts about reaching the cost threshold are required.

### Tools for Cloud Provisioning:

* Google Cloud Deployment Manager
* IBM Cloud Orchestrator
* AWS CloudFormation
* Microsoft Azure Resource Manager

### Types of Cloud Provisioning:

* **Static Provisioning or Advance Provisioning:**Static provisioning can be used successfully for applications with known and typically constant demands or workloads. In this instance, the cloud provider allows the customer with a set number of resources. The client can thereafter utilize these resources as required. The client is in charge of making sure the resources aren’t overutilized. This is an excellent choice for applications with stable and predictable needs or workloads. For instance, a customer might want to use a database server with a set quantity of CPU, RAM, and storage.  
  When a consumer contracts with a service provider for services, the supplier makes the necessary preparations before the service can begin. Either a one-time cost or a monthly fee is applied to the client.  
  Resources are pre-allocated to customers by cloud service providers. This means that before consuming resources, a cloud user must select how much capacity they need in a static sense. Static provisioning may result in issues with over or under-provisioning.
* **Dynamic provisioning or On-demand provisioning:** With dynamic provisioning, the provider adds resources as needed and subtracts them as they are no longer required. It follows a pay-per-use model, i.e. the clients are billed only for the exact resources they use. Consumers must pay for each use of the resources that the cloud service provider allots to them as needed and when necessary. The pay-as-you-go model is another name for this. “Dynamic provisioning” techniques allow VMs to be moved on-the-fly to new computing nodes within the cloud, in situations where demand by applications may change or vary. This is a suitable choice for programs with erratic and shifting demands or workloads. For instance, a customer might want to use a web server with a configurable quantity of CPU, memory, and storage. In this scenario, the client can utilize the resources as required and only pay for what is really used. The client is in charge of ensuring that the resources are not oversubscribed; otherwise, fees can skyrocket.
* **Self-service provisioning or user self-provisioning:** In user self-provisioning, sometimes referred to as cloud self-service, the customer uses a web form to acquire resources from the cloud provider, sets up a customer account, and pays with a credit card. Shortly after, resources are made accessible for consumer use.

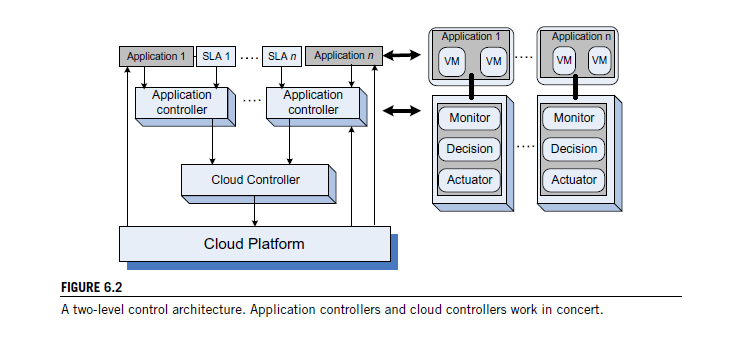
**Stability of a two-level resource allocation architecture:**

The main components of a control system are the inputs, the control system components, and the outputs. The inputs in such models are the offered workload and the policies for admission control, the capacity allocation, the load balancing, the energy optimization, and the QoS guarantees in the cloud.

The system components are *sensors* used to estimate relevant measures of performance and *controllers* that implement various policies; the output is the resource allocations to the individual applications .

The controllers use the feedback provided by sensors to stabilize the system; stability is related to the change of the output. If the change is too large, the system may become unstable. In our context the system could experience thrashing, the amount of useful time dedicated to the execution of applications

becomes increasingly small and most of the system resources are occupied by management functions.

****

There are three main sources of instability in any control system:

**1.** The delay in getting the system reaction after a control action.

**2.** The granularity of the control, the fact that a small change enacted by the controllers leads to very large changes of the output.

**3.** Oscillations, which occur when the changes of the input are too large and the control is too weak, such that the changes of the input propagate directly to the output.

Two types of policies are used in autonomic systems:

* 1. threshold-based policies and
  2. (ii) sequential

decision policies based on Markovian decision models. In the first case, upper and lower bounds on performance trigger adaptation through resource reallocation. Such policies are simple and intuitive but require setting per-application thresholds.

A first observation is that the actions of the control system should be carried out in a rhythm that does not lead to instability. Adjustments should be carried out only after the performance of the system has stabilized. The controller should measure the time for an application to stabilize and adapt to the manner in which the controlled system reacts.

If upper and lower thresholds are set, instability occurs when they are too close to one another if the variations of the workload are large enough and the time required to adapt does not allow the system to stabilize. The actions consist of allocation/deallocation of one or more virtual machines; sometimes allocation/deallocation of a single VM required by one of the thresholds may cause crossing of the other threshold and this may represent, another source of instability.

**Feedback control based on dynamic thresholds:**

The elements involved in a control system are sensors, monitors, and actuators. The *sensors* measure the parameter(s) of interest, then transmit the measured values to a *monitor*, which determines whether the system behavior must be changed, and, if so, it requests that the *actuators* carry out the necessary actions.

Often the parameter used for admission control policy is the current system load; when a threshold, e.g., 80%, is reached, the cloud stops accepting additional load.

In practice, the implementation of such a policy is challenging or outright infeasible.

First, due to the very large number of servers and to the fact that the load changes rapidly in time, the estimation of the current system load is likely to be inaccurate. Second, the ratio of average to maximal resource requirements of individual users specified in a service-level agreement is typically very high. Once an agreement is in place, user demands must be satisfied; user requests for additional resources within the SLA limits cannot be denied.

**Thresholds.**

A *threshold* is the value of a parameter related to the state of a system that triggers a change in the system behavior. Thresholds are used in control theory to keep critical parameters of a system in a predefined range. The threshold could be *static*, defined once and for all, or it could be *dynamic*. A dynamic threshold could be based on an average of measurements carried out over a time

interval, a so-called *integral control*. The dynamic threshold could also be a function of the values of multiple parameters at a given time or a mix of the two.

To maintain the system parameters in a given range, a *high* and a *low* threshold are often defined.

The two thresholds determine different actions; for example, a high threshold could force the system to limit its activities and a low threshold could encourage additional activities. *Control granularity* refers to the level of detail of the information used to control the system. *Fine control* means that very detailed

information about the parameters controlling the system state is used, whereas *coarse control* means that the accuracy of these parameters is traded for the efficiency of implementation.

**Proportional Thresholding.** Application of these ideas to cloud computing, in particular to the *IaaS* delivery model, and a strategy for resource management called *proportional thresholding* are discussedin.

The questions addressed are:

• Is it beneficial to have two types of controllers, (1) *application controllers* that determine whether additional resources are needed and (2) *cloud controllers* that arbitrate requests for resources and allocate the physical resources?

• Is it feasible to consider *fine control*? Is *course control* more adequate in a cloud computing environment?

• Are dynamic thresholds based on time averages better than static ones?

• Is it better to have a high and a low threshold, or it is sufficient to define only a high threshold?

The essence of the proportional thresholding is captured by the following algorithm:

**1.** Compute the integral value of the high and the low thresholds as averages of the maximum and, respectively, the minimum of the processor utilization over the process history.

**2.** Request additional VMs when the average value of the CPU utilization over the current time slice exceeds the high threshold.

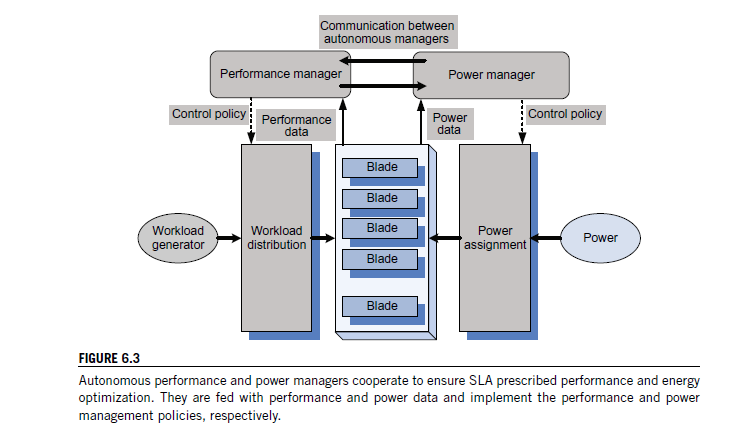
**3.** Release a VM when the average value of the CPU utilization over the current time slice falls below the low threshold.

**Coordination of specialized autonomic performance managers:**

Virtually all modern processors support dynamic voltage scaling (DVS) as a mechanism for energy saving. Indeed, the energy dissipation scales quadratically with the supply voltage.

The power management controls theCPUfrequency and, thus, the rate of instruction execution. For some compute-intensive workloads the performance decreases linearly with the CPU clock frequency, whereas for others the effect of lower clock frequency is less noticeable or nonexistent. The clock frequency of individual blades/servers is controlled by a power manager, typically implemented in the firmware; it adjusts the clock frequency several times a second.

The approach to coordinating power and performance management in is based on several ideas:



• Use a joint utility function for power and performance. The joint performance-power utility function,

*Upp(R, P)*, is a function of the response time, *R*, and the power, *P*, and it can be of the form

*Upp(R, P)* = *U(R)* − × *P* or *Upp(R, P)* = *U(R)*

*P*

*,* with*U(R)* the utility function based on response time only and a parameter to weight the influence

of the two factors, response time and power.

Identify a minimal set of parameters to be exchanged between the two managers.

• Set up a power cap for individual systems based on the utility-optimized power management policy.

• Use a standard performance manager modified only to accept input from the power manager regarding the frequency determined according to the power management policy.

The power manager consists of Tcl (Tool Command Language) and C programs to compute the per-server (per-blade) power caps and send them via IPMI5 to the firmware controlling the blade power.

The power manager and the performance manager interact, but no negotiation between the two agents is involved.

Use standard software systems. For example, use the WebSphere Extended Deployment (WXD), middleware that supports setting performance targets for individual Web applications and for the monitor response time, and periodically recompute the resource allocation parameters to meet the targets set. Use the Wide-Spectrum Stress Tool from the IBM Web Services Toolkit as a workload

generator.

Three types of experiments were conducted: (i) with the power management turned off; (ii) when the dependence of the power consumption and the response time were determined through a set of exhaustive experiments; and (iii) when the dependency of the powercap *pκ* on *nc* was derived via reinforcement-learning models.

The second type of experiment led to the conclusion that both the response time and the power consumed are nonlinear functions of the powercap, *pκ* , and the number of clients, *nc*; more specifically,

the conclusions of these experiments are:

• At a low load the response time is well below the target of 1,000 msec.

• At medium and high loads the response time decreases rapidly when *pk* increases from 80 to 110 watts.

• For a given value of the powercap, the consumed power increases rapidly as the load increases.

**Resource bundling: Combinatorial auctions for cloud resources:**

Resources in a cloud are allocated in *bundles*, allowing users get maximum benefit from a specific combination of resources. Indeed, along with CPU cycles, an application needs specific amounts of main memory, disk space, network bandwidth, and so on. Resource bundling complicates traditional resource allocation models and has generated interest in economic models and, in particular, auction algorithms.

In the context of cloud computing, an auction is the allocation of resources to the highest bidder.

**Combinatorial Auctions.** Auctions in which participants can bid on combinations of items, or *packages*, are called *combinatorial auctions*. Such auctions provide a relatively simple, scalable, and tractable solution to cloud resource allocation. Two recent combinatorial auction algorithms are the *simultaneous clock auction*  and the *clock proxy auction* . The algorithm discussed in this chapter and introduced in is called the *ascending clock auction (ASCA)*. In all these algorithms the current price for each resource is represented by a “clock” seen by all participants at the auction.

We consider a strategy in which prices and allocation are set as a result of an auction. In this auction, users provide bids for desirable bundles and the price they are willing to pay. We assume a population of *U* users, *u* = {1*,* 2*, . . . ,U*}, and *R* resources, *r* = {1*,* 2*, . . . , R*}.

The bid of user *u* is *Bu* = {*Qu, πu*} with*Qi* = *(q*1 *u , q*2 *u , q*3 *u , . . . )* an *R*-component vector; each element of this vector, *qiu* , represents a bundle of resources user *u* would accept and, in return, pay the total price *πu*.

The bidding process aims to optimize an *objective function f (x, p)*. This function could be tailored to measure the net value of all resources traded, or it can measure the *total surplus* – the difference between the maximum amount users are willing to pay minus the amount they pay. Other optimization functions could be considered for a specific system, e.g., the minimization of energy consumption or of security risks.

**Pricing and Allocation Algorithms.** A pricing and allocation algorithm partitions the set of users into two disjoint sets, winners and losers, denoted as *W* and *L*, respectively. The algorithm should:

**1.** Be computationally tractable. Traditional combinatorial auction algorithms such as Vickey-Clarke- Groves (VLG) fail this criteria, because they are not computationally tractable.

**2.** Scale well. Given the scale of the system and the number of requests for service, scalability is a necessary condition.

**3.** Be objective. Partitioning in winners and losers should only be based on the price *πu* of a user’s bid.

If the price exceeds the threshold, the user is a winner; otherwise the user is a loser.

**4.** Be fair.Make sure that the prices are *uniform*. All winners within a given resource pool pay the same price.

**5.** Indicate clearly at the end of the auction the unit prices for each resource pool.

**6.** Indicate clearly to all participants the relationship between the supply and the demand in the system.