Cloud Resource Management

Contents

Resource management and scheduling

- Critical function of any man-made system.
- It affects the three basic criteria for the evaluation of a system:
  - Functionality.
  - Performance.
  - Cost.
- Scheduling in a computing system → deciding how to allocate resources of a system, such as CPU cycles, memory, secondary storage space, I/O and network bandwidth, between users and tasks.
- Policies and mechanisms for resource allocation.
  - Policy → principles guiding decisions.
  - Mechanisms → the means to implement policies.

Motivation

- Cloud resource management .
  - Requires complex policies and decisions for multi-objective optimization.
  - It is challenging - the complexity of the system makes it impossible to have accurate global state information.
  - Affected by unpredictable interactions with the environment, e.g., system failures, attacks.
  - Cloud service providers are faced with large fluctuating loads which challenge the claim of cloud elasticity.

- The strategies for resource management for IaaS, PaaS, and SaaS are different.

Cloud resource management (CRM) policies

1. Admission control → prevent the system from accepting workload in violation of high-level system policies.

2. Capacity allocation → allocate resources for individual activations of a service.

3. Load balancing → distribute the workload evenly among the servers.

4. Energy optimization → minimization of energy consumption.

5. Quality of service (QoS) guarantees → ability to satisfy timing or other conditions specified by a Service Level Agreement.

Mechanisms for the implementation of resource management policies

- Control theory → uses the feedback to guarantee system stability and predict transient behavior.
- Machine learning → does not need a performance model of the system.
- Utility-based → require a performance model and a mechanism to correlate user-level performance with cost.
- Market-oriented/economic → do not require a model of the system, e.g., combinatorial auctions for bundles of resources.

6.2 Applications of control theory to task scheduling on a cloud

- Control theory has been used to design adaptive resource management for many classes of applications, including power management , task scheduling , QoS adaptation in Web servers , and load balancing.
- The classical feedback control methods are used in all these cases to regulate the key operating parameters of the system based on measurement of the system output; the feedback control in these methods assumes a linear time-invariant system model and a closed-loop controller.
- This controller is based on an open-loop system transfer function that satisfies stability and sensitivity constraints.
- The following discussion considers a single processor serving a stream of input requests. We attempt to minimize a cost function that reflects the response time and the power consumption. Our goal is to illustrate the methodology for optimal resource management based on control theory concepts. The analysis is intricate and cannot be easily extended to a collection of servers.

Control Theory Principles.

- Optimal control generates a sequence of control inputs over a look-ahead horizon while estimating changes in operating conditions.
- A convex cost function has arguments
  - x (k) -> the state at step k, and
  - u(k)-> the control vector;

  this cost function is minimized, subject to the constraints imposed by the system dynamics.
- The discrete-time optimal control problem is to determine the sequence of control variables u(i), u(i + 1), . . . , u(n − 1) to minimize the expression

$$J(i) = \Phi(n, x(n)) + \sum_{k=i}^{n-1} L^k(x(k), u(k)),$$

- where
  - (n, x (n)) is the cost function of the final step, n,
  - L k (x (k), u(k)) is a timevarying cost function at the intermediate step k over the horizon [i , n].
- The minimization is subject to the constraints

$$x(k+1) = f^k(x(k), u(k)),$$

  - where x (k + 1) -> the system state at time k + 1, is a function of x (k), the state at time k, and of u(k), the input at time k; in general, the function f ^{k} is time-varying; thus, its superscript.

- One of the techniques to solve this problem is based on the Lagrange multiplier method of finding the extremes (minima or maxima) of a function subject to constrains.
- More precisely, if we want to maximize the function g(x , y) subject to the constraint h(x , y) = k, we introduce a Lagrange multiplier λ. Then we study the function

$$\Lambda(x, y, \lambda) = g(x, y) + \lambda \times [h(x, y) - k].$$

- A necessary condition for the optimality is that (x , y, λ) is a stationary point for (x , y, λ). In other words,

$$\nabla_{x,y,\lambda}\Lambda(x, y, \lambda) = 0 \text{ or } \left(\frac{\partial\Lambda(x, y, \lambda)}{\partial x}, \frac{\partial\Lambda(x, y, \lambda)}{\partial y}, \frac{\partial\Lambda(x, y, \lambda)}{\partial \lambda}\right) = 0. \qquad (6.4)$$

- The Lagrange multiplier at time step k is λk and we solve Eq.(6.4) as an unconstrained optimization problem. We define an adjoint cost function that includes the original state constraints as the Hamiltonian function H , then we construct the adjoint system consisting of the original state equation and the costate equation governing the Lagrange multiplier. Thus, we define a two-point boundary problem3 ; the state xk develops forward in time whereas the costate occurs backward in time.

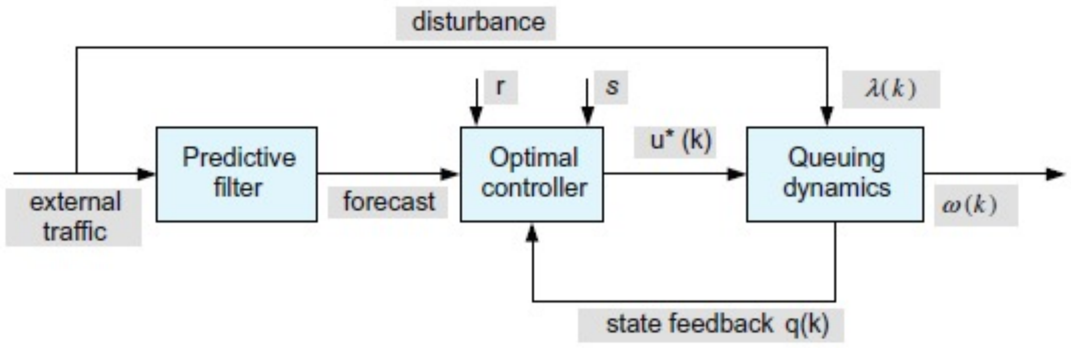


FIGURE 6.1

The structure of an optimal controller described in [369]. The controller uses the feedback regarding the current state as well as the estimation of the future disturbance due to environment to compute the optimal inputs over a finite horizon. The two parameters *r* and *s* are the weighting factors of the performance index.

A Model Capturing Both QoS and Energy Consumption for a Single-Server System.

- To compute the optimal inputs over a finite horizon, the controller in Figure 6.1 uses feedback regarding the current state, as well as an estimation of the future disturbance due to the environment.
- The control task is solved as a state regulation problem updating the initial and final states of the control horizon.
- We use a simple queuing model to estimate the response time.
- Requests for service at processor P are processed on a first-come, first-served (FCFS) basis.
- We do not assume a priori distributions of the arrival process and of the service process; instead, we use the estimate ^Λ(k)of the arrival rate(k)at time k.

- We also assume that the processor can operate at frequencies $u(k)$ in the range $u(k) \in [u_{min}, u_{max}]$ and
- call $\hat{c}(k)$ the time to process a request at time k when the processor operates at the highest frequency in the range, u_{max}.
- Then we define the scaling factor
  - $\alpha(k) = u(k)/u_{max}$ and
- we express an estimate of the processing rate $N(k)$ as $\alpha(k)/\hat{c}(k)$.

- The behavior of a single processor is modeled as a nonlinear, time-varying, discretetime state equation.
- If T_s is the sampling period, defined as the time difference between two consecutive observations of the system, e.g., the one at time (k + 1) and the one at time k, then the size of the queue at time (k + 1) is

$$q(k+1) = \max\left\{\left[q(k) + \left(\hat{\Lambda}(k) - \frac{u(k)}{\hat{c}(k) \times u_{max}}\right) \times T_s\right], 0\right\}.$$

- The first term, q(k), is the size of the input queue at time k, and the second one is the difference between the number of requests arriving during the sampling period, T_s , and those processed during the same interval.

- The response time $\omega(k)$ is the sum of the waiting time and the processing time of the requests

  $\omega(k) = (1 + q(k)) \times \hat{c}(k)$.
- Indeed, the total number of requests in the system is $(1 + q(k))$ and the departure rate is $1/\hat{c}(k)$.
- We want to capture both the QoS and the energy consumption, since both affect the cost of providing the service.
- A utility function, such as the one depicted in Figure 6.4, captures the rewards as well as the penalties specified by the service-level agreement for the response time.
- In our queuing model the utility is a function of the size of the queue; it can be expressed as a quadratic function of the response time
  - $S(q(k)) = 1/2(s \times (\omega(k) - \omega_0)^2)$,
    - with $\omega0$, the response time set point and $q(0) = q0$, the initial value of the queue length.
- The energy consumption is a quadratic function of the frequency $R(u(k)) = 1/2(r \times u(k)^2)$

- The two parameters s and r are weights for the two components of the cost, the one derived from the utility function and the second from the energy consumption. We have to pay a penalty for the requests left in the queue at the end of the control horizon, a quadratic function of the queue length
  - $(q(N)) = 1/2(v \times q(n)^2)$.
- The performance measure of interest is a cost expressed as

$$J = \Phi(q(N)) + \sum_{k=1}^{N-1} [S(q(k)) + R(u(k))].$$

- The problem is to find the optimal control u∗ and the finite time horizon [0, N] such that the trajectory of the system subject to optimal control is q ∗ , and the cost J in the above Equation is minimized subject to the following constraints

$$q(k+1) = \left[q(k) + \left(\hat{\Lambda}(k) - \frac{u(k)}{\hat{c}(k) \times u_{max}} \right) \times T_s \right], \; q(k) \geqslant 0, \text{ and } u_{min} \leqslant u(k) \leqslant u_{max}.$$

- When the state trajectory q(·) corresponding to the control u(·) satisfies the constraints

$$\Gamma 1 : q(k) > 0,$$

$$\Gamma 2 : u(k) \geqslant u_{min},$$

$$\Gamma 3 : u(k) \leqslant u_{max},$$

- then the pair q(·), u(·) is called a feasible state. If the pair minimizes, then the pair is optimal.
- The Hamiltonian H in our example is

$$H = S(q(k)) + R(u(k)) + \lambda(k+1) \times \left[q(k) + \left(\Lambda(k) - \frac{u(k)}{c \times u_{max}} \right) T_s \right]$$
$$+ \mu_1(k) \times (-q(k)) + \mu_2(k) \times (-u(k) + u_{min}) + \mu_3(k) \times (u(k) - u_{max}).$$

- According to Pontryagin´s minimum principle,4 pairs to be optimal pairs is the existence of a
  - μ = [μ1(k), μ2(k), μ3(k)]
- such that the necessary condition for a sequence of feasible sequence of costates λ and a Lagrange multiplier
  - H (k, q∗ , u∗ , λ∗ , μ∗)H (k, q, u∗ , λ∗ , μ∗), ∀q0 (6.14)
- where the Lagrange multipliers, μ1(k), μ2(k), μ3(k), reflect the sensitivity of the cost function to the queue length at time k and the boundary constraints and satisfy several conditions
  - μ 1 (k)0, μ1 (k)(−q(k)) = 0,
  - μ2 (k)0, μ2(k)(−u(k) + umi n) = 0,
  - μ3 (k)0, μ3 (k)(u(k) − umax) = 0.

6.3 Stability of a two-level resource allocation architecture

- In this section we discuss a two-level resource allocation architecture based on control theory concepts for the entire cloud.
- The automatic resource management is based on two levels of controllers, one for the service provider and one for the application, see Figure 6.2.

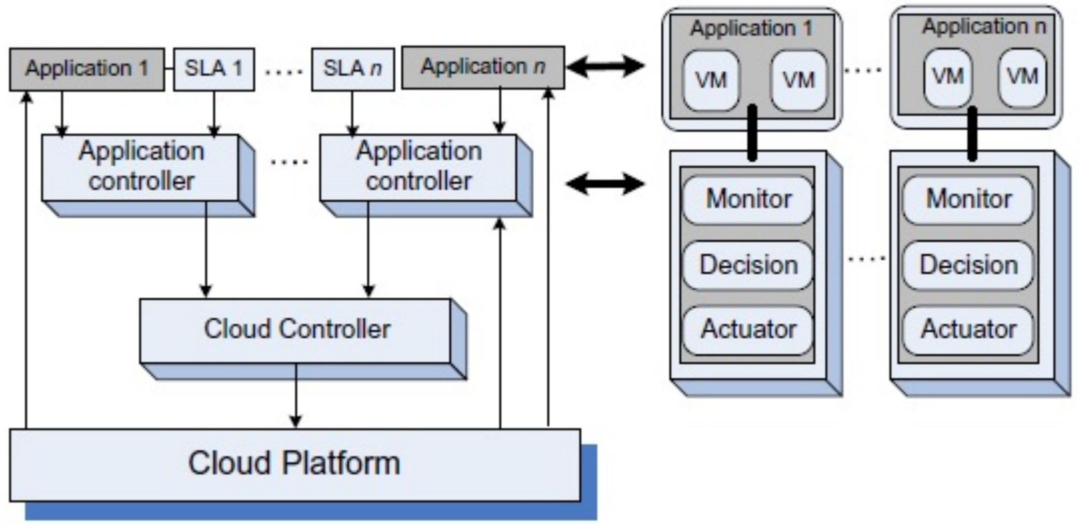


FIGURE 6.2

A two-level control architecture. Application controllers and cloud controllers work in concert.

- The main components of a control system are
  - the inputs - the capacity allocation, the load balancing, the energy optimization, and the QoS guarantees in the cloud.
  - the control system components –
    - sensors used to estimate relevant measures of performance and
    - controllers that implement various policies
  - the outputs - 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:
  - (i) threshold-based policies - upper and lower bounds on performance trigger adaptation through resource reallocation
  - (ii) sequential decision policies based on Markovian decision models.

Lessons learned from the experiments with two levels of controllers

- The actions of the control system should be carried out in a rhythm that does not lead to instability.
- Adjustments should only be carried out after the performance of the system has stabilized.
- If upper and a lower thresholds are set, then 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/dealocation of a single VM required by one of the threshold may cause crossing of the other, another source of instability.

6.4 Feedback control based on dynamic thresholds

- The elements involved in a control system are
  - sensors,
  - monitors,
  - 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.

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,
  - 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,
  - Coarse control means that the accuracy of these parameters is traded for the efficiency of implementation.

Proportional Thresholding

- Used in IaaS delievery model
- 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.

6.5 Coordination of specialized autonomic performance managers

Components are

1. Performance Manager
2. Power Manager
3. Workload distribution
4. Power assignment

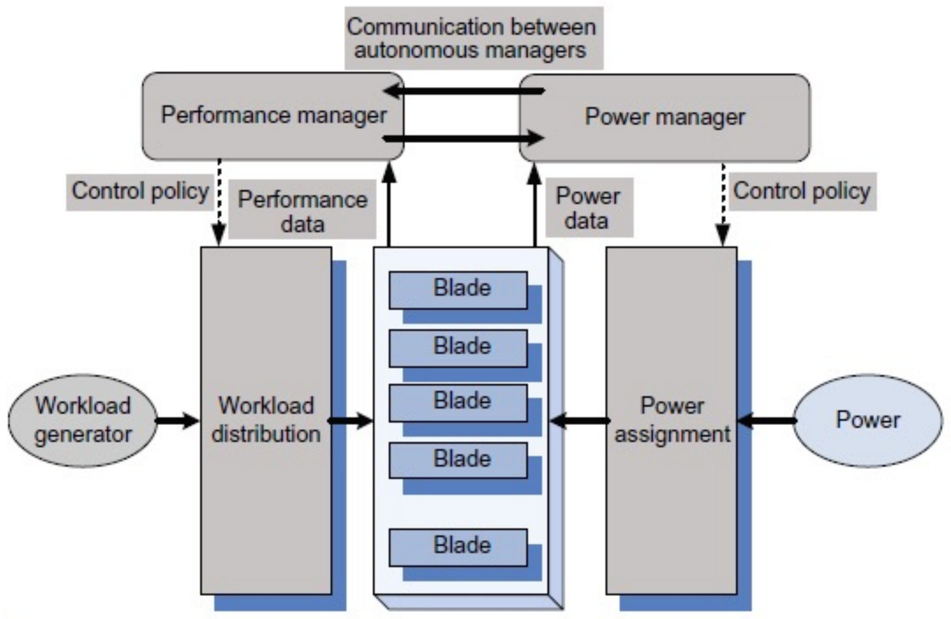


FIGURE 6.3

Autonomous performance and power managers cooperate to ensure SLA prescribed performance and energy optimization. They are fed with performance and power data and implement the performance and power management policies, respectively.

- 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 the CPU frequency 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 is based on several ideas:
- Use a joint utility function for power and performance. The joint performance power utility function, $U_{pp}(R, P)$, is a function of the response time, R, and the power, P , and it can be of the form

$$U_{pp}(R,P) = U(R) - \epsilon \times P \quad \text{or} \quad U_{pp}(R,P) = \frac{U(R)}{P},$$

  - 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 IPMI to the firmware controlling the blade power. The power manager and the performance manager interact, but no negotiation between the two agents is involved.

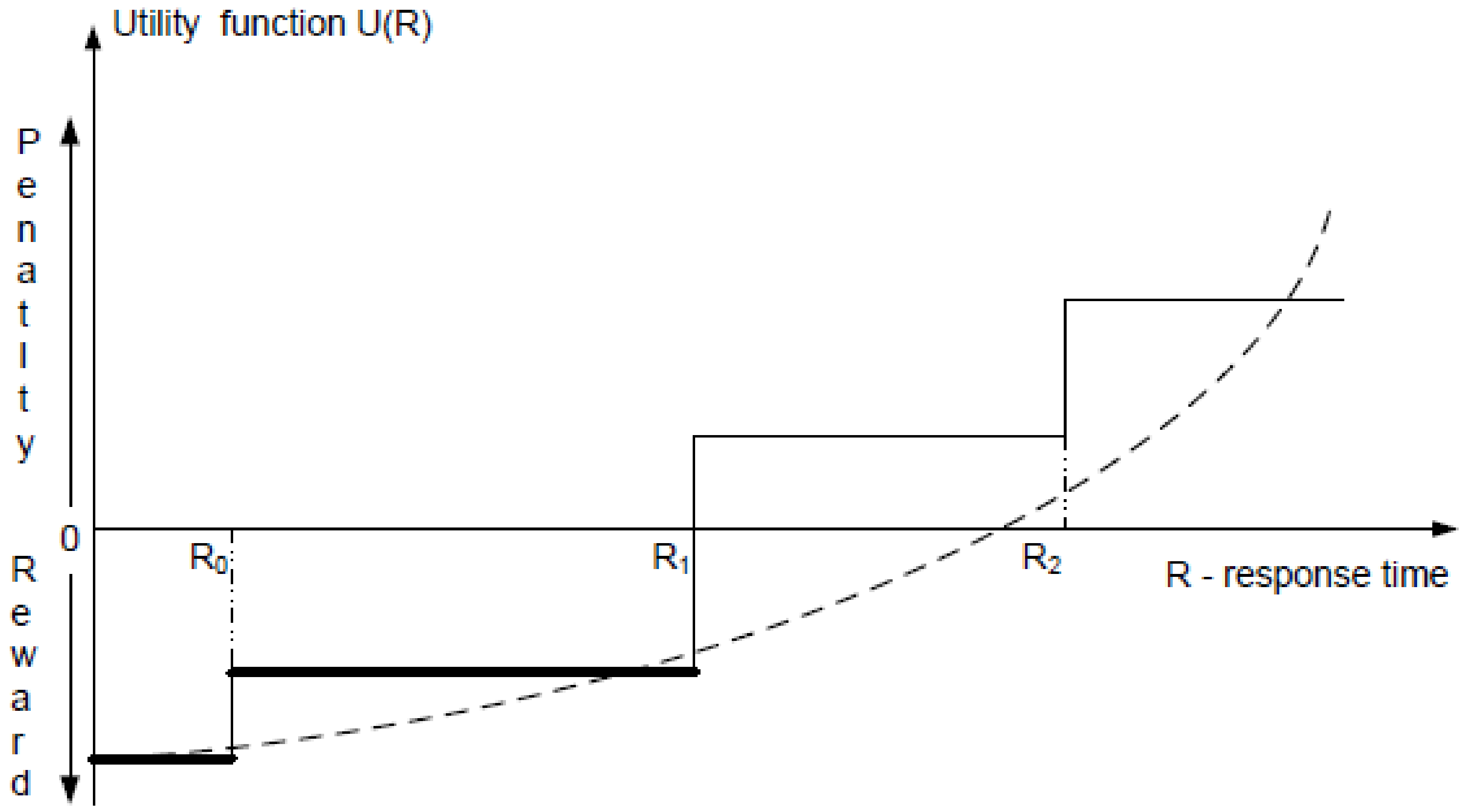
- Intelligent Platform Management Interface (IPMI) is a standardized computer system interface developed by Intel and used by system administrators to manage a computer system and monitor its operation.
- Use standard software systems. For example, use the WebSphere Extended Deployment (WXD), middleware 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 WideSpectrum Stress Tool from the IBM Web Services Toolkit as a workload generator. For practical reasons the utility function was expressed in terms of nc , the number of clients, and pκ , the powercap, as in
  - U (pκ , nc) = U pp (R(pκ , nc), P (pκ , nc)).
- The optimal powercap pκ opt is a function of the workload intensity expressed by the number of clients, nc ,

$$p_{\kappa}^{opt}(n_c) = \arg\max U'(p_{\kappa}, n_c).$$

- 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;
  - (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.
- The machine learning algorithm used for the third type of experiment was based on the Hybrid Reinforcement Learning algorithm.
- In the experiments using the machine learning model, the powercap required to achieve a response time lower than 1,000 msec for a given number of clients was the lowest when = 0.05 and the first utility function given by Eq. (6.18) was used. For example, when nc = 50, then pκ = 109 Watts when = 0.05, whereas pκ = 120 when = 0.01.

A utility-based model for cloud-based Web services

- A *utility function* relates the “benefits” of an activity or service with the “cost” to provide the service.
- For example, the benefit could be revenue and the cost could be the power consumption.
- service-level agreement (SLA) often specifies the rewards as well as the penalties associated with specific performance metrics.
- Sometimes the quality of services translates into average response time; this is the case of cloud-based Web services when the SLA often explicitly specifies this requirement
- For example, Figure 6.4 shows the case when the performance metrics is R, the response time.
- The largest reward can be obtained when $R <= R0$; a slightly lower reward corresponds to $R0 < R <= R1$.
- When $R1 < R <=R2$, instead of gaining a reward, the provider of service pays a small penalty; the penalty increases when $R > R2$. A utility function, *U(R)*, which captures this behavior, is a sequence of step functions.
- A utility function, *U(R)*, which captures this behavior, is a sequence of step functions. The utility function is sometimes approximated by a quadratic curve



The utility function U(R) is a series of step functions with jumps corresponding to the response time, $R=R_0 \mid R_1 \mid R_2$, when the reward and the penalty levels change according to the SLA. The dotted line shows a quadratic approximation of the utility function.

- The goal is to maximize the total profit computed as the difference between the revenue guaranteed by an SLA and the total cost to provide the services.
- We assume a cloud providing $|K|$ different classes of service, each class *k* involving *Nk* applications.
- For each class $k \in K$ call *vk* the revenue (or the penalty) associated with a response time *rk* and assume a linear dependency for this utility function of the form
- Call $$v_k = v_k^{max}\left(1 - r_k / r_k^{max}\right)$$
   - *k* the slope of the utility function.
- The system is modeled as a network of queues with multiqueues for each server and with a delay center that models the think time of the user after the completion of service at one server and the start of processing at the next server

$$m_k = -v_k^{max} / r_k^{max}$$

- Upon completion, a class *k* request either completes with probability $(1 - \sum_{k' \in K} \pi_{k,k'})$
- or returns to the system as a class *k* request with transition probability *πk,k'*
- Call *λk* the external arrival rate of class *k* requests and *k* the aggregate rate for class *k*, where

$$\Lambda_k = \lambda_k + \sum_{k' \in K} \Lambda_{k'} \pi_{k,k'}.$$

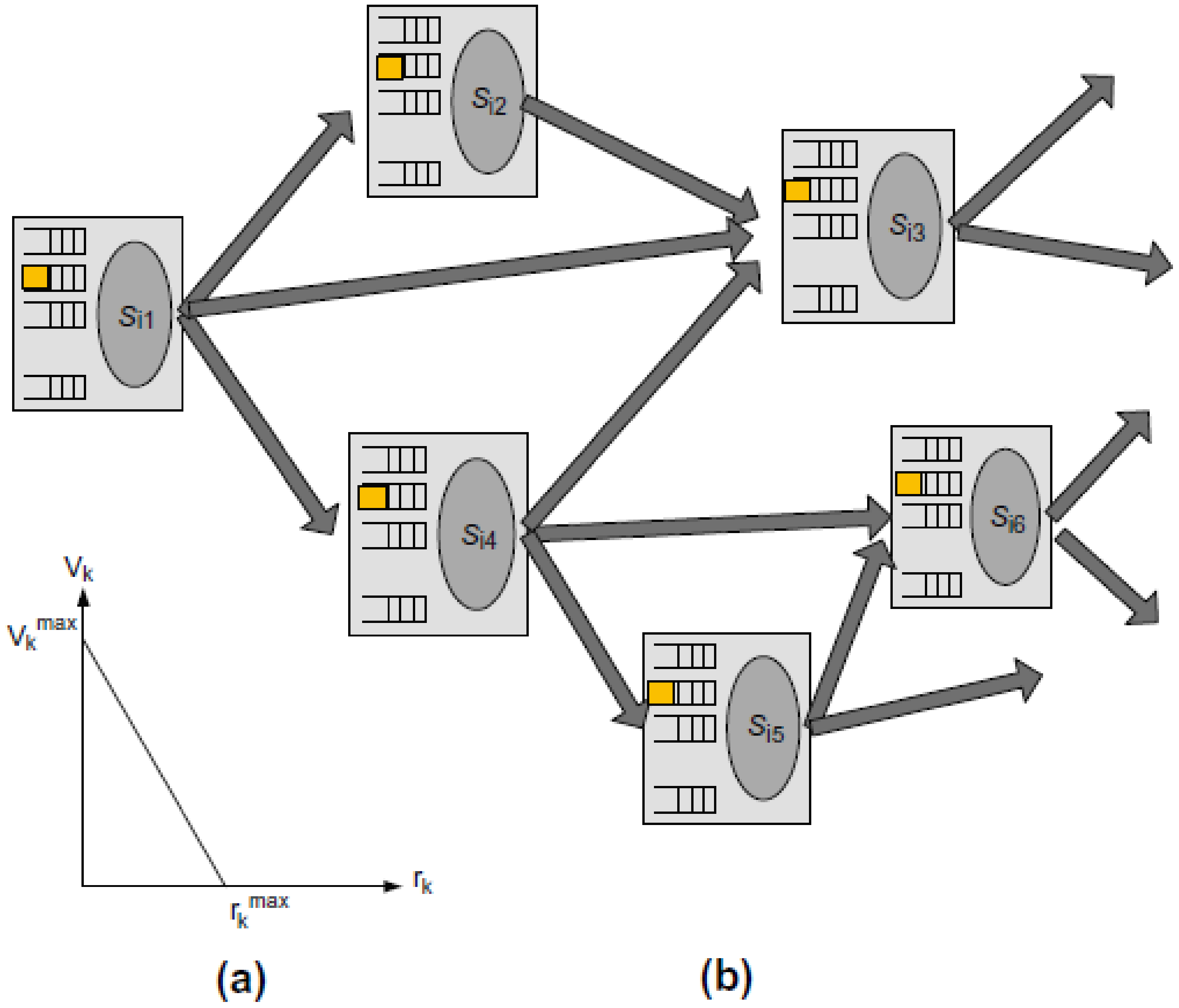


FIGURE 6.5

(a) The utility function, v_k the revenue (or the penalty) associated with a response time r_k for a request of class $k \in K$. The slope of the utility function is $m_k = -v_k^{max}/r_k^{max}$. (b) A network of multiqueues. At each server S_i there are $|K|$ queues for each one of the $k \in K$ classes of requests. A tier consists of all requests of class $k \in K$ at all servers $S_{ij} \in I$, $1 \leqslant j \leqslant 6$.

- Typically, CPU and memory are considered representative for resource allocation; for simplicity we assume a single CPU that runs at a discrete set of clock frequencies and a discrete set of supply voltages according to a Dynamic Voltage and Frequency Scaling (DVFS) model.
- The power consumption of a server is a function of the clock frequency.
- The scheduling of a server is work-conserving and is modeled as a Generalized Processor Sharing (GPS) scheduling
- The optimization problem formulated involves five terms: A and B reflect revenues; C the cost of servers in a low-power, stand-by mode; D the cost of active servers, given their operating frequency; E, the cost of switching servers from low-power, stand-by mode to active state, and F, the cost of migrating VMs from one server to another.
- There are nine constraints $1, 2, \ldots, 9$ for this mixed integer, nonlinear programming problem. The decision variables for this optimization problem are listed in Table 6.2 and the parameters used are shown in Table 6.3.

Table 6.2 Decision variables for the optimization problem.

Name	Description
x_i	$x_i = 1$ if server $i \in I$ is running, $x_i = 0$ otherwise
$y_{i,h}$	$y_{i,h} = 1$ if server i is running at frequency h, $y_{i,h} = 0$ otherwise
$z_{i,k,j}$	$z_{i,k,j} = 1$ if application tier j of a class k request runs on server i, $z_{i,k,j} = 0$ otherwise
$w_{i,k}$	$w_{i,k} = 1$ if at least one class k request is assigned to server i, $w_{i,k} = 0$ otherwise
$\lambda_{i,k,j}$	Rate of execution of applications tier j of class k requests on server i
$\phi_{i,k,j}$	Fraction of capacity of server i assigned to tier j of class k requests

Table 6.3 The parameters used for the A, B, C, D, E, and F terms and the constraints Γ_i of the optimization problem.

Name	Description
I	The set of servers
K	The set of classes
Λ_k	The aggregate rate for class $k \in K$, $\Lambda_k = \lambda_k + \sum_{k' \in K} \Lambda_{k'} \pi_{k,k'}$
a_i	The availability of server $i \in I$
A_k	Minimum level of availability for request class $k \in K$ specified by the SLA
m_k	The slope of the utility function for a class $k \in K$ application
N_k	Number of applications in class $k \in K$
H_i	The range of frequencies of server $i \in I$
$C_{i,h}$	Capacity of server $i \in I$ running at frequency $h \in H_i$
$c_{i,h}$	Cost for server $i \in I$ running at frequency $h \in H_i$
$\bar{c}_i$	Average cost of running server i
$\mu_{k,j}$	Maximum service rate for a unit capacity server for tier j of a class k request
cm	The cost of moving a virtual machine from one server to another
cs_i	The cost for switching server i from the stand-by mode to an active state
$RAM_{k,j}$	The amount of main memory for tier j of class k request
$\overline{RAM}_i$	The amount of memory available on server i

The expression to be maximized is:

$$(A + B) - (C + D + E + F)$$

with

$$A = \max \sum_{k \in K} \left(-m_k \sum_{i \in I, j \in N_k} \frac{\lambda_{i,k,j}}{\sum_{h \in H_i} (C_{i,h} \times y_{i,h})\, \mu_{k,j} \times \phi_{i,k,j} - \lambda_{i,k,j}} \right),$$

$$B = \sum_{k \in K} u_k \times \Lambda_k,$$

$$C = \sum_{i \in I} \bar{c}_i, \quad D = \sum_{i \in I, h \in H_i} c_{i,h} \times y_{i,h}, \quad E = \sum_{i \in I} cs_i \max(0, x_i - \bar{x}_i),$$

and

$$F = \sum_{i \in I, k \in K, j \in N_j} cm \max(0, z_{i,j,k} - \bar{z}_{i,j,k}).$$

The nine constraints are:

(Γ_1) $\sum_{i \in I} \lambda_{i,k,j} = \Lambda_k, \forall k \in K, j \in N_k \Rightarrow$ the traffic assigned to all servers for class k requests equals the predicted load for the class.

(Γ_2) $\sum_{k \in K, j \in N_k} \phi_{i,k,j} \leqslant 1 \, \forall i \in I \Rightarrow$ server i cannot be allocated an workload more than its capacity.

(Γ_3) $\sum_{h \in H_i} y_{i,h} = x_i, \forall i \in I \Rightarrow$ if server $i \in I$ is active it runs at one frequency in the set H_i, and only one $y_{i,h}$ is nonzero.

(Γ_4) $z_{i,k,j} \leqslant x_i, \forall i \in I, k \in K, j \in N_k \Rightarrow$ requests can only be assigned to active servers.

(Γ_5) $\lambda_{i,k,j} \leqslant \Lambda_k \times z_{i,k,j}, \forall i \in I, k \in K, j \in N_k \Rightarrow$ requests may run on server $i \in I$ only if the corresponding application tier has been assigned to server i.

(Γ_6) $\lambda_{i,k,j} \leqslant \left(\sum_{h \in H_i} C_{i,h} \times y_{i,h}\right) \mu_{k,j} \times \phi_{i,k,j}, \forall i \in I, k \in K, j \in N_k \Rightarrow$ resources cannot be saturated.

(Γ_7) $RAM_{k,j} \times z_{i,k,j} \leqslant \overline{RAM}_i, \forall i \in I, k \in K \Rightarrow$ the memory on server i is sufficient to support all applications running on it.

(Γ_8) $\Pi_{j=1}^{N_k} \left(1 - \Pi_{i=1}^{M}(1 - a_i^{w_{i,k}}\right) \geqslant A_k, \forall k \in K \Rightarrow$ the availability of all servers assigned to class k request should be at least equal to the minimum required by the SLA.

(Γ_9) $\sum_{j=1}^{N_k} z_{i,k,j} \geqslant N_k \times w_{i,k}, \forall i \in I, k \in K$

$\lambda_{i,j,k}, \phi_{i,j,k} \geqslant 0, \forall i \in I, k \in K, j \in N_k$

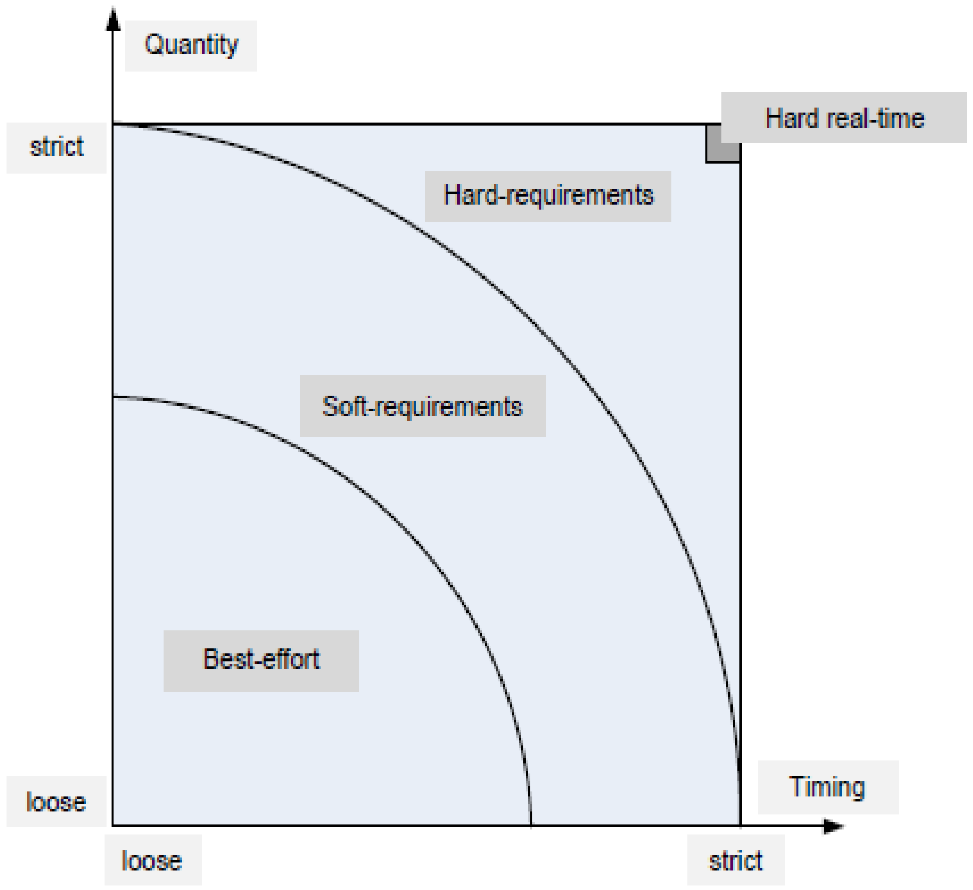
$x_i, y_{i,h}, z_{i,k,j}, w_{i,k} \in \{0, 1\}, \forall i \in I, k \in K, j \in N_k \Rightarrow$ constraints and relations among decision variables.

Cloud scheduling algorithms (1/2)

- Scheduling → responsible for resource sharing at several levels:
  - A server can be shared among several virtual machines.
  - A virtual machine could support several applications.
  - An application may consist of multiple threads.
- A scheduling algorithm should be efficient, fair, and starvation-free.
- The objectives of a scheduler:
  - Batch system → maximize throughput and minimize turnaround time.
  - Real-time system → meet the deadlines and be predictable.
- Best-effort: batch applications and analytics.
- Common algorithms for best effort applications:
  - Round-robin.
  - First-Come-First-Serve (FCFS).
  - Shortest-Job-First (SJF).
  - Priority algorithms.

Cloud scheduling algorithms (2/2)

- Multimedia applications (e.g., audio and video streaming)
  - Have soft real-time constraints.
  - Require statistically guaranteed maximum delay and throughput.
- Real-time applications have hard real-time constraints.
- Scheduling algorithms for real-time applications:
  - Earliest Deadline First (EDF).
  - Rate Monotonic Algorithms (RMA).
- Algorithms for integrated scheduling of several classes of applications (best-effort, multimedia, real-time):
  - Resource Allocation/Dispatching (RAD) .
  - Rate-Based Earliest Deadline (RBED).



Best-effort policies → do not impose requirements regarding either the amount of resources allocated to an application, or the timing when an application is scheduled.

Soft-requirements policies → require statistically guaranteed amounts and timing constraints

Hard-requirements policies → demand strict timing and precise amounts of resources.

max-min fairness criterion

- Consider a resource with bandwidth B shared among n users who have equal rights.
- Each user requests an amount bi and receives Bi .
- Then, according to themax-min criterion, the following conditions must be satisfied by a fair allocation:
  - C1. The amount received by any user is not larger than the amount requested, Bi <= bi .
  - C2. If the minimum allocation of any user is Bmin no allocation satisfying condition C1 has a higher Bmin than the current allocation.
  - C3. When we remove the user receiving the minimum allocation Bmin and then reduce the total amount of the resource available from B to (B − Bmin), the condition C2 remains recursively true.
- A fairness criterion for CPU scheduling [142] requires that the amount of work in the time interval from *t*1 to *t*2 of two runnable threads a and b, Ω*a(t1,t2)*and Ω*b(t1,t2)* respectively, minimize the expression

$$\left|\frac{\Omega_a(t_1,t_2)}{w_a}-\frac{\Omega_b(t_1,t_2)}{w_b}\right|,$$

  - where *wa* and *wb* are the weights of the threads *a* and *b*, respectively

Few points to be considered before algo

- we discuss can be used for scheduling packet transmission as well as threads.
- Interconnection networks allow cloud servers to communicate with one another and with users.
- These networks consist of communication links of limited bandwidth and switches/routers/gateways of limited capacity.
- When the load exceeds its capacity, a switch starts dropping packets because it has limited input buffers for the switching fabric and for the outgoing links, as well as limited CPU cycles.
- A switch must handle multiple flows and pairs of source-destination endpoints of the traffic.
- Thus, a scheduling algorithm has to manage several quantities at the same time: the *bandwidth*, the amount of data each flow is allowed to transport; the *timing* when the packets of individual flows are transmitted; and the *buffer space* allocated to each flow

FCFS

- A first strategy to avoid network congestion is to use a FCFS scheduling algorithm. The advantage of the FCFS algorithm is a simple management of the three quantities:
  - bandwidth,
  - timing, and
  - buffer space.
- Nevertheless, the FCFS algorithm does not guarantee fairness; greedy flow sources can transmit at a higher rate and benefit from a larger share of the bandwidth.

a fair queuing algorithm

- a fair queuing algorithm requires that separate queues, One per flow, be maintained by a switch and that the queues be serviced in a round-robin manner.
- This algorithm
  - guarantees the fairness of buffer space management,
  - but does not guarantee fairness of bandwidth allocation.
  - Indeed, a flow transporting large packets will benefit from a larger bandwidth
- The *fair queuing (FQ)* algorithm introduces a *bit-by-bit round-robin (BR)* strategy; as the name implies, in this rather impractical scheme a single bit from each queue is transmitted and the queues are visited in a round-robin fashion.
- Let *R(t)* be the number of rounds of the BR algorithm up to time *t* and
- *Nactive(t)* be the number of active flows through the switch.
- Call t^a_i - the time when the packet *i* of flow *a*, of size P^a_i bits arrives, and
- call S^a_i and F^a_i the values of *R(t)* when the first and the last bit, respectively, of the packet *i* of flow *a* are transmitted. Then

$$F_i^a = S_i^a + P_i^a \quad \text{and} \quad S_i^a = \max\left[F_{i-1}^a, R(t_i^a)\right].$$

- The quantities *R(t)*, *Nactive(t)*, S^a_i , and F^a_i depend only on the arrival time of the packets, t^a_i , and not on their transmission time, provided that a flow *a* is active as long as $R(t) \leqslant F_i^a \quad \text{when} \quad i = \max\left(j \mid t_i^a \leqslant t\right).$
- The use for packet-by-packet transmission time the following nonpreemptive scheduling rule, which emulates the BR strategy:
  - *The next packet to be transmitted is the one with the smallest* F^a_i
- A preemptive version of the algorithm requires that the transmission of the current packet be interrupted as soon as one with a shorter finishing time, F^a_i arrives.

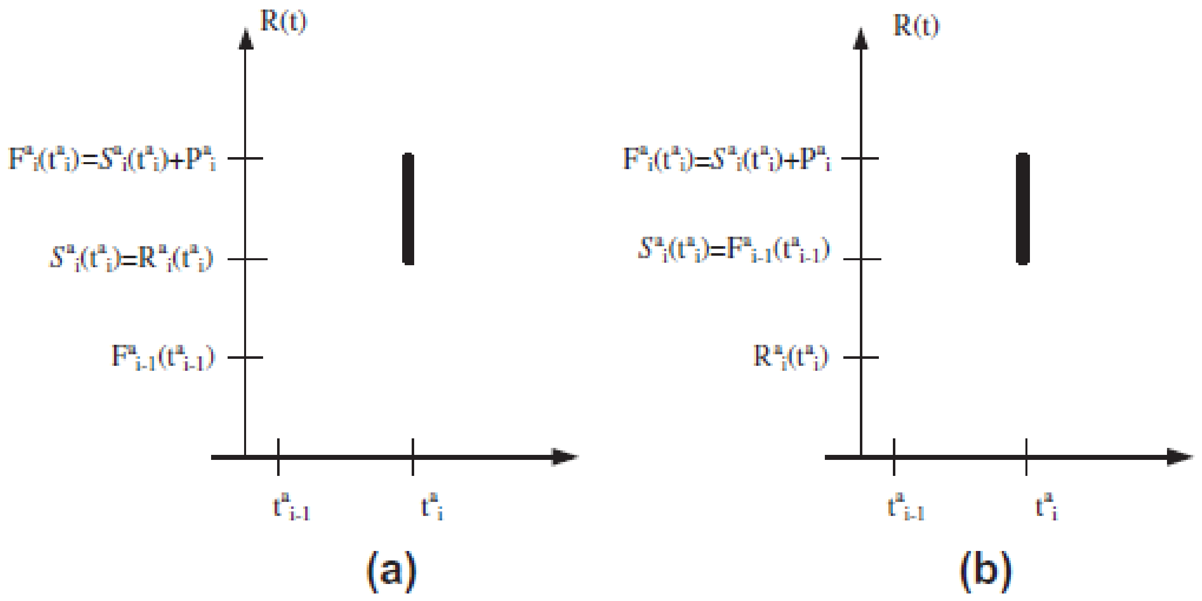

R(t)
$F^a_i(t^a_i)=S^a_i(t^a_i)+P^a_i$
$S^a_i(t^a_i)=R^a_i(t^a_i)$
$F^a_{i-1}(t^a_{i-1})$
t^a_{i-1}
t^a_i
(a)
R(t)
$F^a_i(t^a_i)=S^a_i(t^a_i)+P^a_i$
$S^a_i(t^a_i)=F^a_{i-1}(t^a_{i-1})$
$R^a_i(t^a_i)$
t^a_{i-1}
t^a_i
(b)

FIGURE 6.8

Transmission of a packet i of flow a arriving at time t_i^a of size P_i^a bits. The transmission starts at time $S_i^a = \max[F_{i-1}^a, R(t_i^a)]$ and ends at time $F_i^a = S_i^a + P_i^a$ with $R(t)$ the number of rounds of the algorithm. (a) The case $F_{i-1}^a < R(t_i^a)$. (b) The case $F_{i-1}^a \geqslant R(t_i^a)$.

- A fair allocation of the bandwidth does not have an effect on the timing of the transmission.
- A possible strategy is to allow less delay for the flows using less than their fair share of the bandwidth.
- We propose the introduction of a quantity called the *bid*, B^a_i , and scheduling the packet transmission based on its value.
- The bid is defined as

$$B_i^a = P_i^a + \max\left[F_{i-1}^a, \left(R\left(t_i^a\right) - \delta\right)\right],$$

  - with δ a nonnegative parameter.

Start-time fair queuing

- The basic idea of the *start-time fair queuing (SFQ)* algorithm is to organize the consumers of the CPU bandwidth in a tree structure;
- the root node is the processor and the leaves of this tree are the threads of each application.
- A scheduler acts at each level of the hierarchy.
- The fraction of the processor bandwidth, *B*, allocated to the intermediate node *i* is

$$\frac{B_i}{B} = \frac{w_i}{\sum_{j=1}^{n} w_j}$$

  - with *wj* , <=1 <= *j* *n*, the weight of the *n* children of node *i* ;

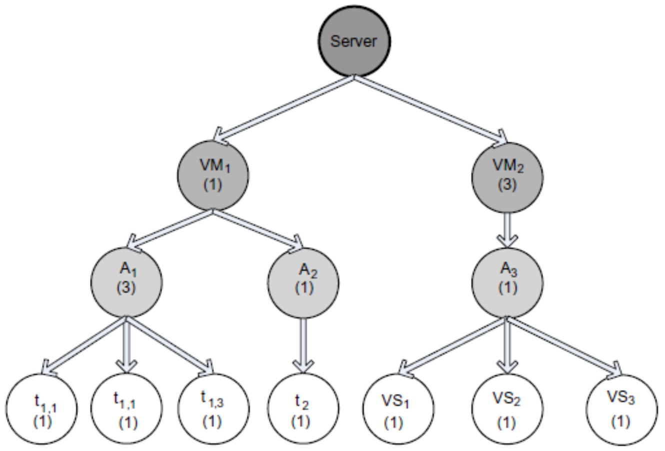


FIGURE 6.9

The SFQ tree for scheduling when two virtual machines, VM_1 and VM_2, run on a powerful server. VM_1 runs two best-effort applications A_1, with three threads $t_{1,1}$, $t_{1,2}$, and $t_{1,3}$, and A_2 with a single thread, t_2. VM_2 runs a video-streaming application, A_3, with three threads vs_1, vs_2, and vs_3. The weights of virtual machines, applications, and individual threads are shown in parenthesis.

- When a virtual machine is not active, its bandwidth is reallocated to the other VMs active at the time.
- When one of the applications of a virtual machine is not active, its allocation is transferred to the other applications running on the same VM.
- Similarly, if one of the threads of an application is not runnable, its allocation is transferred to the other threads of the applications
- Call *va(t)* and *vb(t)* the virtual time of threads *a* and *b*, respectively, at real time *t*.
- The virtual time of the scheduler at time *t* is denoted by *v(t)*.
- Call *q* the time quantum of the scheduler in milliseconds.
- The threads *a* and *b* have their time quanta, *qa* and *qb*, weighted by *wa* and *wb*, respectively; thus, in our example, the time quanta of the two threads are *q/wa* and *q/wb*, respectively.
- The *i* -th activation of thread *a* will start at the virtual time S_i^a and will finish at virtual time F_i^a
- . We call τ *j* the real time of the *j* -th invocation of the scheduler.

An SFQ scheduler follows several rules:

- R1. The threads are serviced in the order of their virtual start-up time; ties are broken arbitrarily.
- R2. The virtual startup time of the *i* -th activation of thread *x* is

$$S_x^i(t) = \max\left[v\left(\tau^j\right), F_x^{(i-1)}(t)\right] \quad \text{and} \quad S_x^0 = 0.$$

  - The condition for thread *i* to be started is that thread *(i − 1)* has finished and that the scheduler is active.
- R3. The virtual finish time of the *i* -th activation of thread *x* is

$$F_x^i(t) = S_x^i(t) + \frac{q}{w_x}.$$

- R4. The virtual time of all threads is initially zero, $v_0^x = 0$. The virtual time *v(t)* at real time *t* is computed as follows:

$$v(t) = \begin{cases} \text{Virtual start time of the thread in service at time } t, & \text{if CPU is busy} \\ \text{Maximum finish virtual time of any thread,} & \text{if CPU is idle.} \end{cases}$$