Chapter 6 – Cloud Resource Management and Scheduling

Contents

- Resource management and scheduling.
- Policies and mechanisms.
- Applications of control theory to cloud resource allocation.
- Stability of a two-level resource allocation architecture.
- Proportional thresholding.
- Coordinating power and performance management.
- A utility-based model for cloud-based Web services.
- Resource bundling and combinatorial auctions.
- Scheduling algorithms.
- Fair queuing.
- Start-up fair queuing.
- Borrowed virtual time.
- Cloud scheduling subject to deadlines.

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.

Tradeoffs

- To reduce cost and save energy we may need to concentrate the load on fewer servers rather than balance the load among them.
- We may also need to operate at a lower clock rate; the performance decreases at a lower rate than does the energy.

CPU speed (GHz)	Normalized energy (%)	Normalized performance (%)
0.6	0.44	0.61
0.8	0.48	0.70
1.0	0.52	0.79
1.2	0.58	0.81
1.4	0.62	0.88
1.6	0.70	0.90
1.8	0.82	0.95
2.0	0.90	0.99
2.2	1.00	1.00

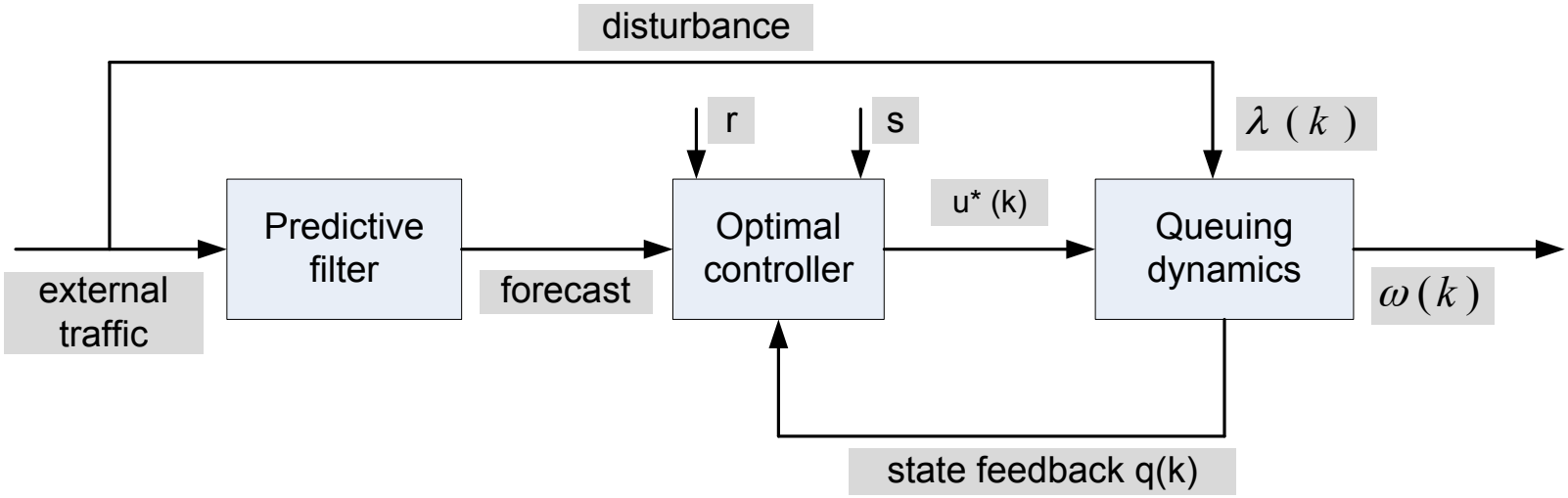
Control theory application to cloud resource management (CRM)

- The main components of a control system:
  - The inputs → 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 control system components → *sensors* used to estimate relevant measures of performance and *controllers* which implement various policies.
  - The outputs → the resource allocations to the individual applications.

Feedback and Stability

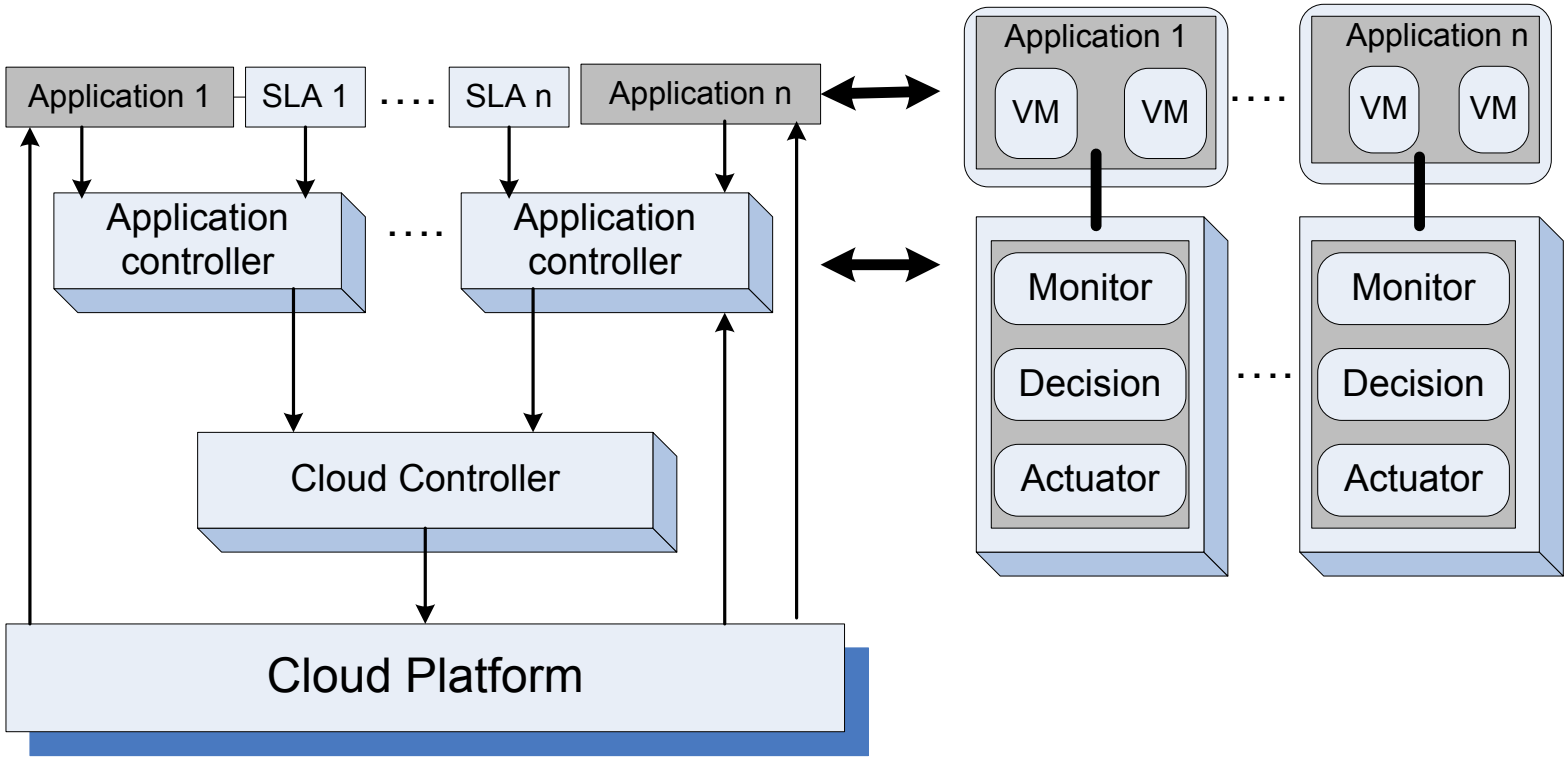
- Control granularity → the level of detail of the information used to control the system.
  - Fine control → very detailed information about the parameters controlling the system state is used.
  - Coarse control → the accuracy of these parameters is traded for the efficiency of implementation.
- The controllers use the feedback provided by sensors to stabilize the system. Stability is related to the change of the output.
- Sources of instability in any control system:
  - The delay in getting the system reaction after a control action.
  - The granularity of the control, the fact that a small change enacted by the controllers leads to very large changes of the output.
  - Oscillations, 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.

The structure of a cloud controller



The controller uses the feedback regarding the current state and the estimation of the future disturbance due to environment to compute the optimal inputs over a finite horizon. ***r*** and ***s*** are the weighting factors of the performance index.

Two-level cloud controller



Lessons from the two-level experiment

- 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.

Control theory application to CRM

- Regulate the key operating parameters of the system based on measurement of the system output.
- The feedback control assumes a linear time-invariant system model, and a closed-loop controller.
- The system transfer function satisfies stability and sensitivity constraints.
- A threshold → the value of a parameter related to the state of a system that triggers a change in the system behavior.
- Thresholds → used to keep critical parameters of a system in a predefined range.
- Two types of policies:
  1. threshold-based → upper and lower bounds on performance trigger adaptation through resource reallocation; such policies are simple and intuitive but require setting per-application thresholds.
  2. sequential decision → based on Markovian decision models.

Design decisions

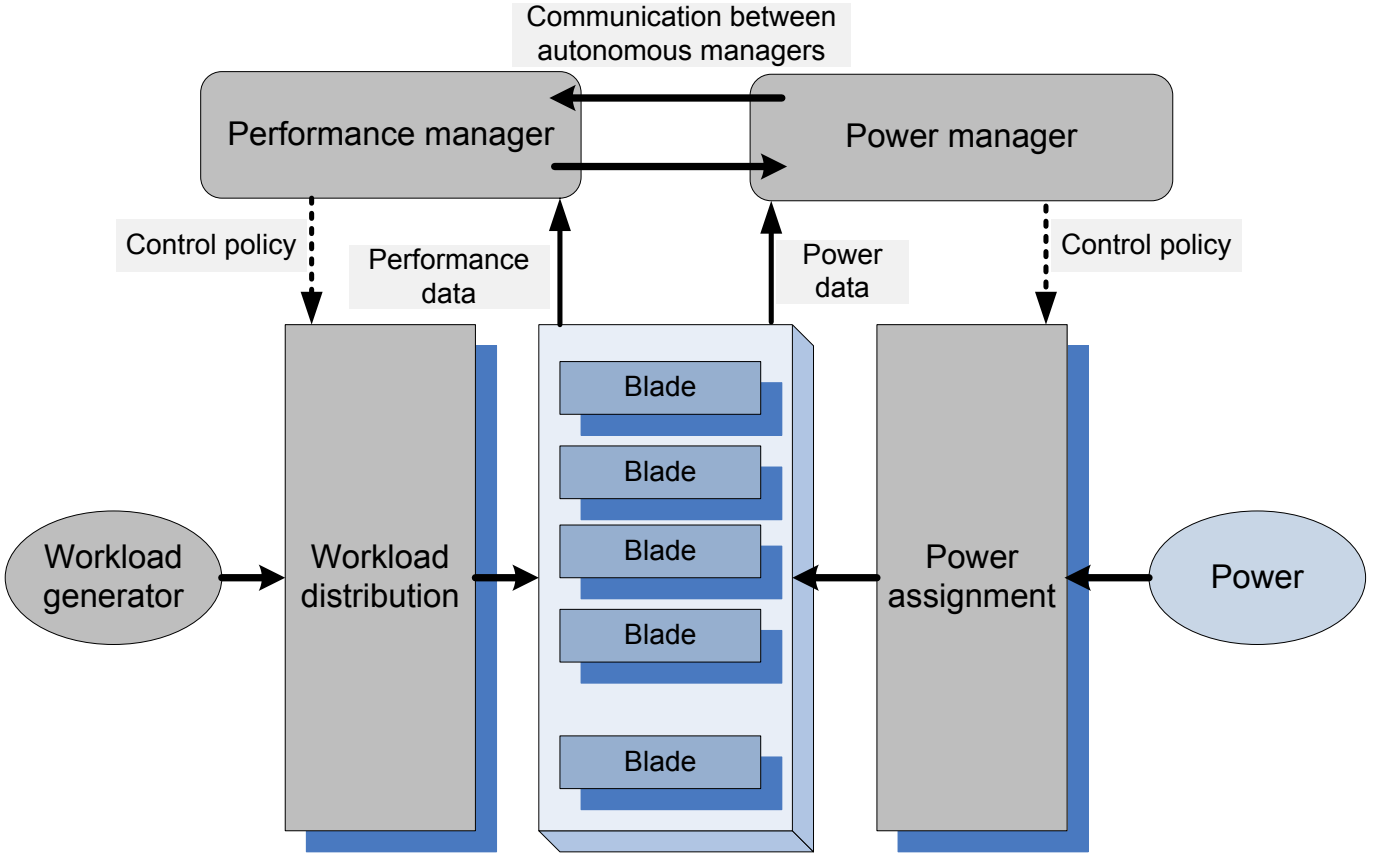
- Is it beneficial to have two types of controllers:
  - application controllers → determine if additional resources are needed.
  - cloud controllers → arbitrate requests for resources and allocates the physical resources.
- Choose fine versus coarse control.
- Dynamic thresholds based on time averages better versus static ones.
- Use a high and a low threshold versus a high threshold only.

Proportional thresholding

- Algorithm
  - Compute the integral value of the high and the low threshold as averages of the maximum and, respectively, the minimum of the processor utilization over the process history.
  - Request additional VMs when the average value of the CPU utilization over the current time slice exceeds the high threshold.
  - Release a VM when the average value of the CPU utilization over the current time slice falls below the low threshold.
- Conclusions
  - Dynamic thresholds perform better than the static ones.
  - Two thresholds are better than one.

Coordinating power and performance management

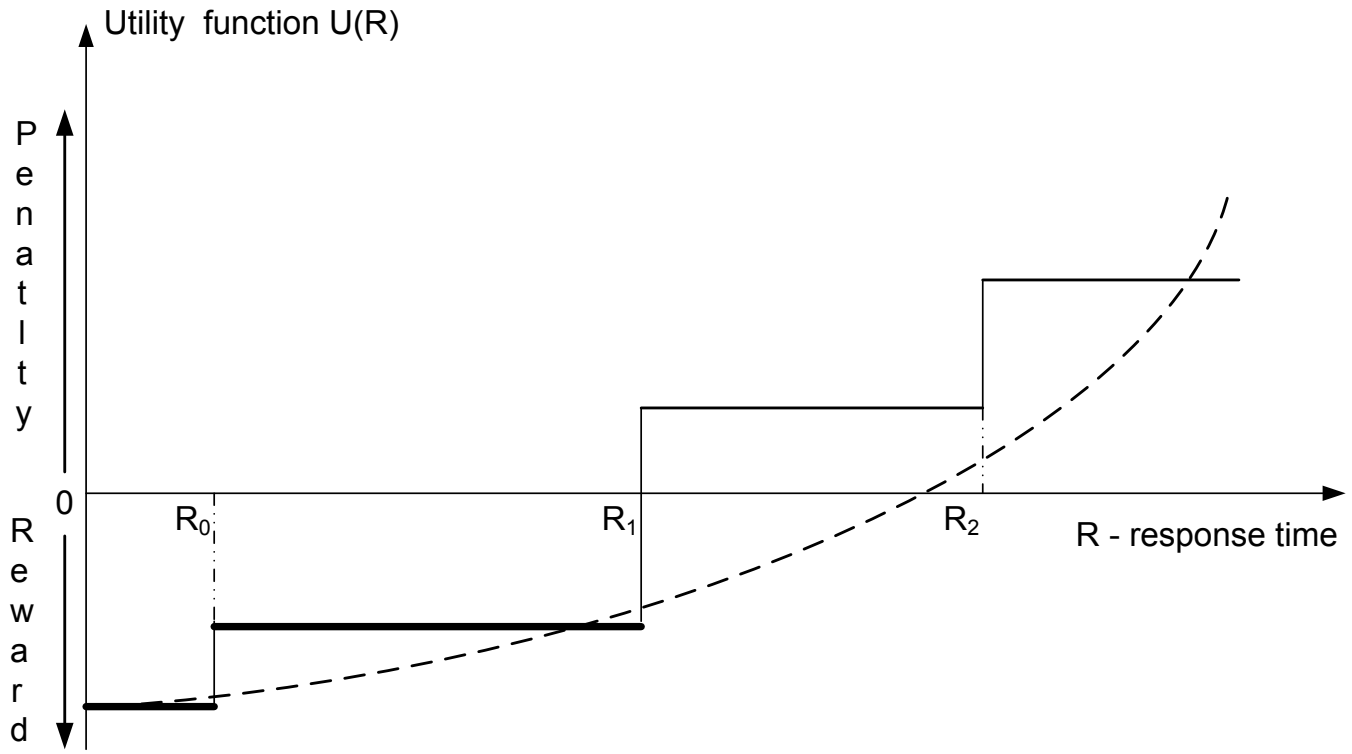
- Use separate controllers/managers for the two objectives.
- Identify a minimal set of parameters to be exchanged between the two managers.
- Use a joint utility function for power and performance.
- Set up a power cap for individual systems based on the utility -optimized power management policy.
- Use a standard performance manager modified only to accept input from the power manager regarding the frequency determined according to the power management policy.
- Use standard software systems.



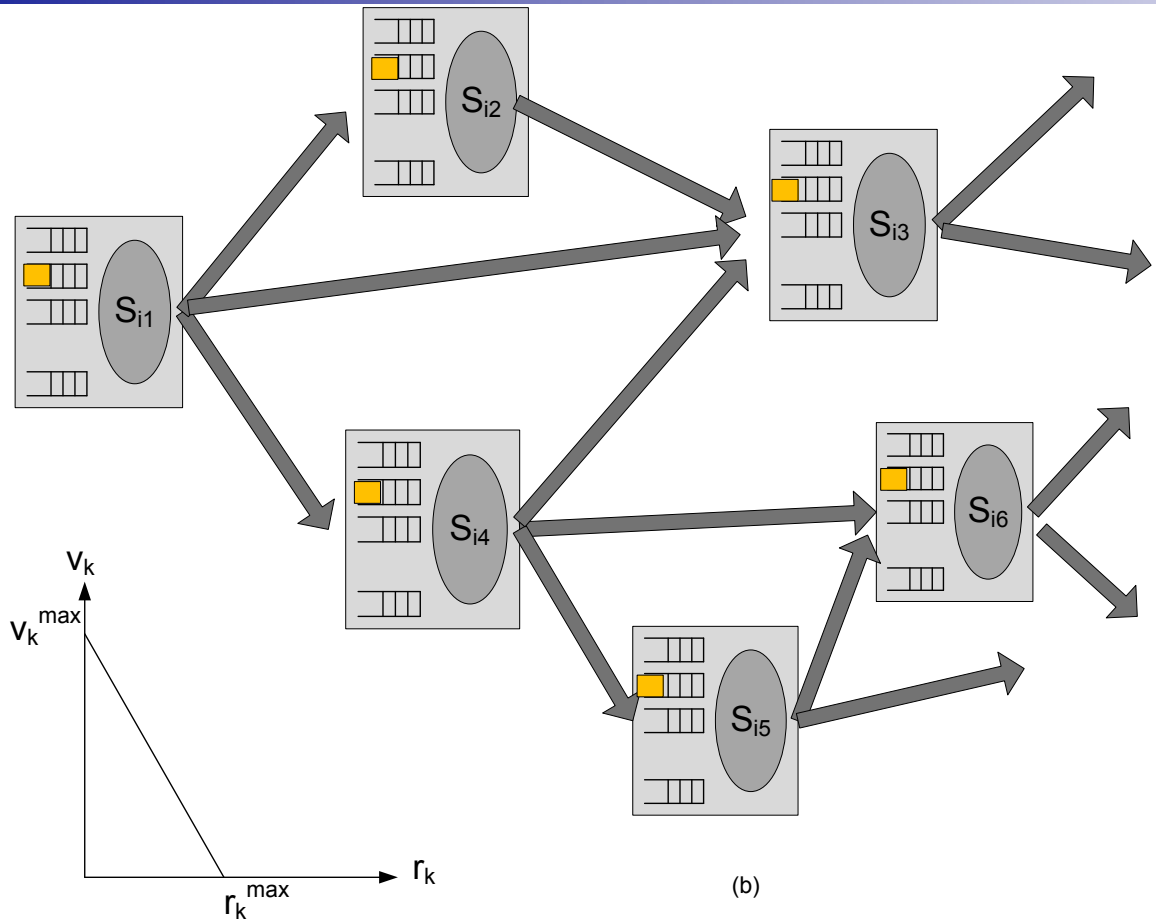
Autonomous performance and power managers cooperate to ensure prescribed performance and energy optimization; they are fed with performance and power data and implement the performance and power management policies

A utility-based model for cloud-based web services

- A service level agreement (SLA) → specifies the rewards as well as penalties associated with specific performance metrics.
- The SLA for cloud-based web services uses the average response time to reflect the Quality of Service.
- We assume a cloud providing K different classes of service, each class k involving N_k applications.
- The system is modeled as a network of queues with multi-queues for each server.
- A delay center models the think time of the user after the completion of service at one server and the start of processing at the next server.



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.



(a) The utility function: v_k the revenue (or the penalty) function of the response time r_k for a request of class k.

(b) A network of multiqueues.

The model requires a large number of parameters

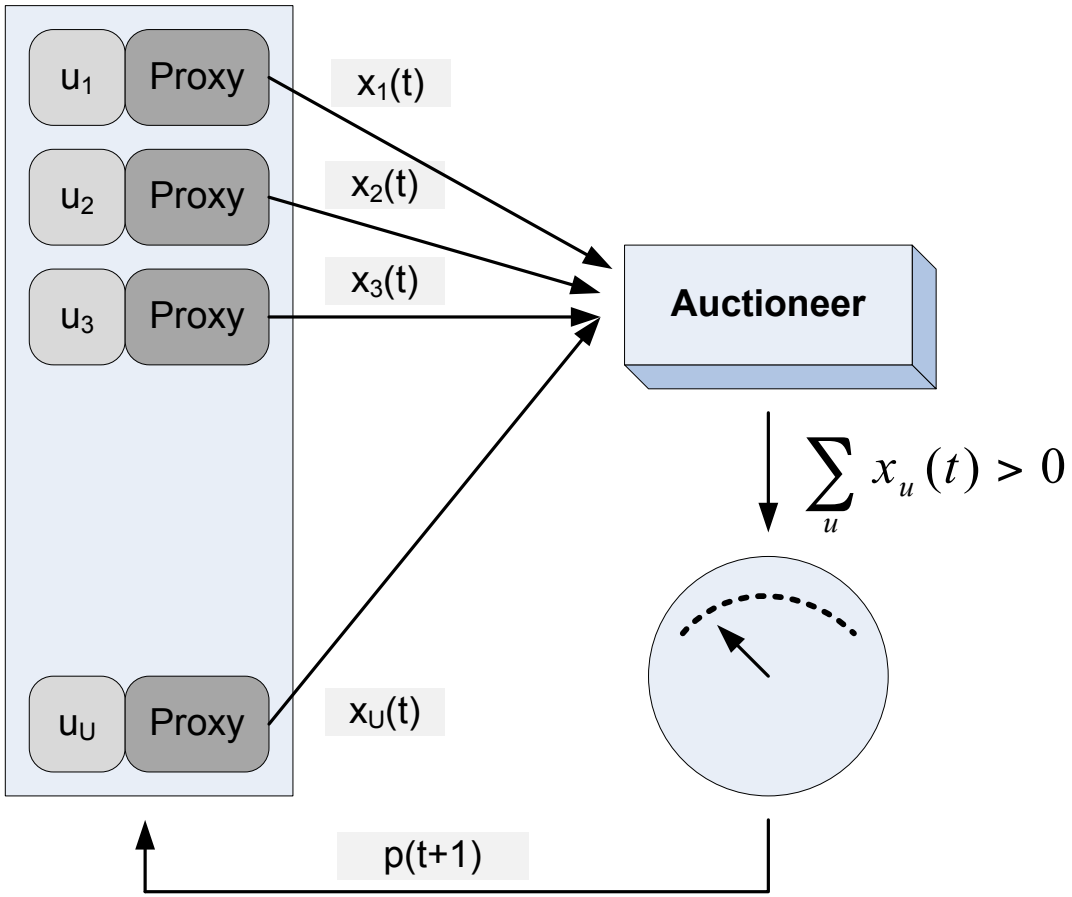
Name	Description
I	the set of servers
K	the set of classes
Λ_k	the aggregate rate for class $k \in K$, $\Lambda_k = \lambda_k + \sum_{k' \in K} \Lambda_{k'} \pi_{k,k'}$
a_i	the availability of server $i \in I$
A_k	minimum level of availability for request class $k \in K$ specified by the SLA
m_k	the 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

Resource bundling

- Resources in a cloud are allocated in bundles.
- Users get maximum benefit from a specific combination of resources: CPU cycles, main memory, disk space, network bandwidth, and so on.
- Resource bundling complicates traditional resource allocation models and has generated an interest in economic models and, in particular, in auction algorithms.
- The bidding process aims to optimize an objective function *f(x,p)*.
- In the context of cloud computing, an auction is the allocation of resources to the highest bidder.

Combinatorial auctions for cloud resources

- Simultaneous clock auction, clock proxy auction, and ascending clock auction (ASCA) are 3 known combinatorial auctions.
- Users provide bids for desirable bundles and the price they are willing to pay.
- Prices and allocation are set as a result of an auction.
- Ascending Clock Auction, (ASCA) → the current price for each resource is represented by a "clock" seen by all participants at the auction.
- The algorithm involves user bidding in multiple rounds; to address this problem the user proxies automatically adjust their demands on behalf of the actual bidders.



The schematics of the ASCA algorithm; to allow for a single round auction users are represented by proxies which place the bids $x_u(t)$. The auctioneer determines if there is an excess demand and, in that case, it raises the price of resources for which the demand exceeds the supply and requests new bids.

Pricing and allocation algorithms

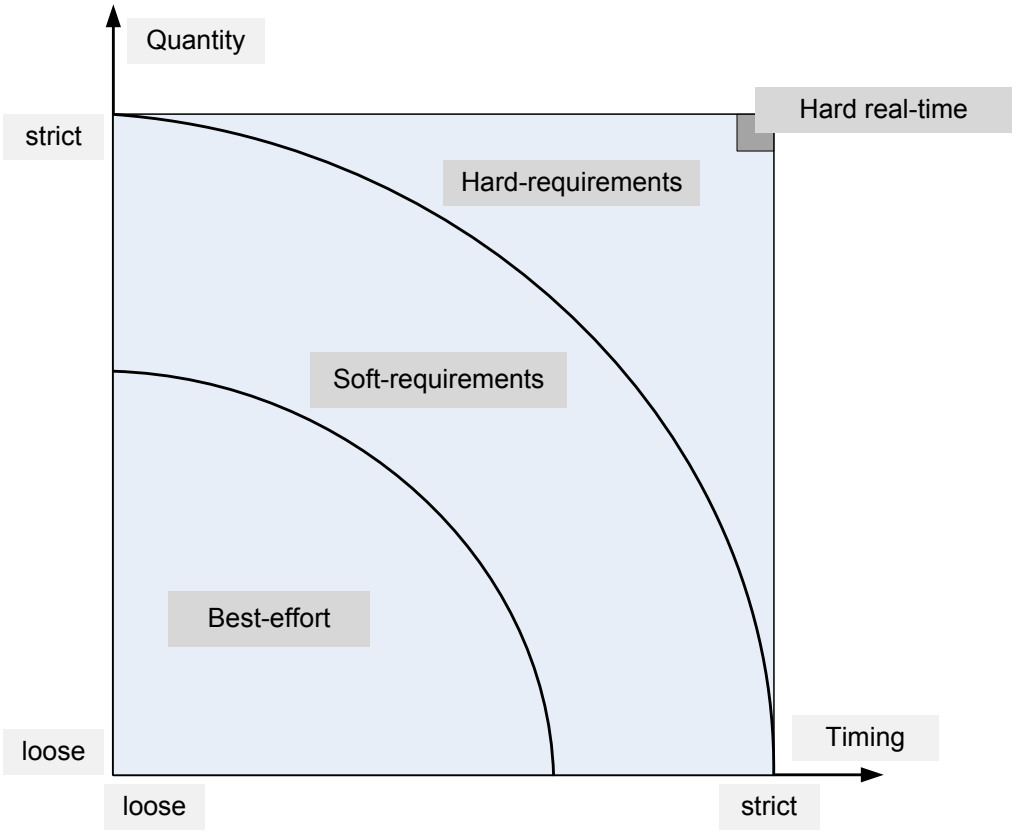
- A pricing and allocation algorithm partitions the set of users in two disjoint sets, winners and losers.
- Desirable properties of a pricing algorithm:
  - Be computationally tractable; traditional combinatorial auction algorithms e.g., Vickey-Clarke-Groves (VLG) are not computationally tractable.
  - Scale well - given the scale of the system and the number of requests for service, scalability is a necessary condition.
  - Be objective - partitioning in winners and losers should only be based on the price of a user's bid; if the price exceeds the threshold then the user is a winner, otherwise the user is a loser.
  - Be fair - make sure that the prices are uniform, all winners within a given resource pool pay the same price.
  - Indicate clearly at the end of the auction the unit prices for each resource pool.
  - Indicate clearly to all participants the relationship between the supply and the demand in the system.

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).

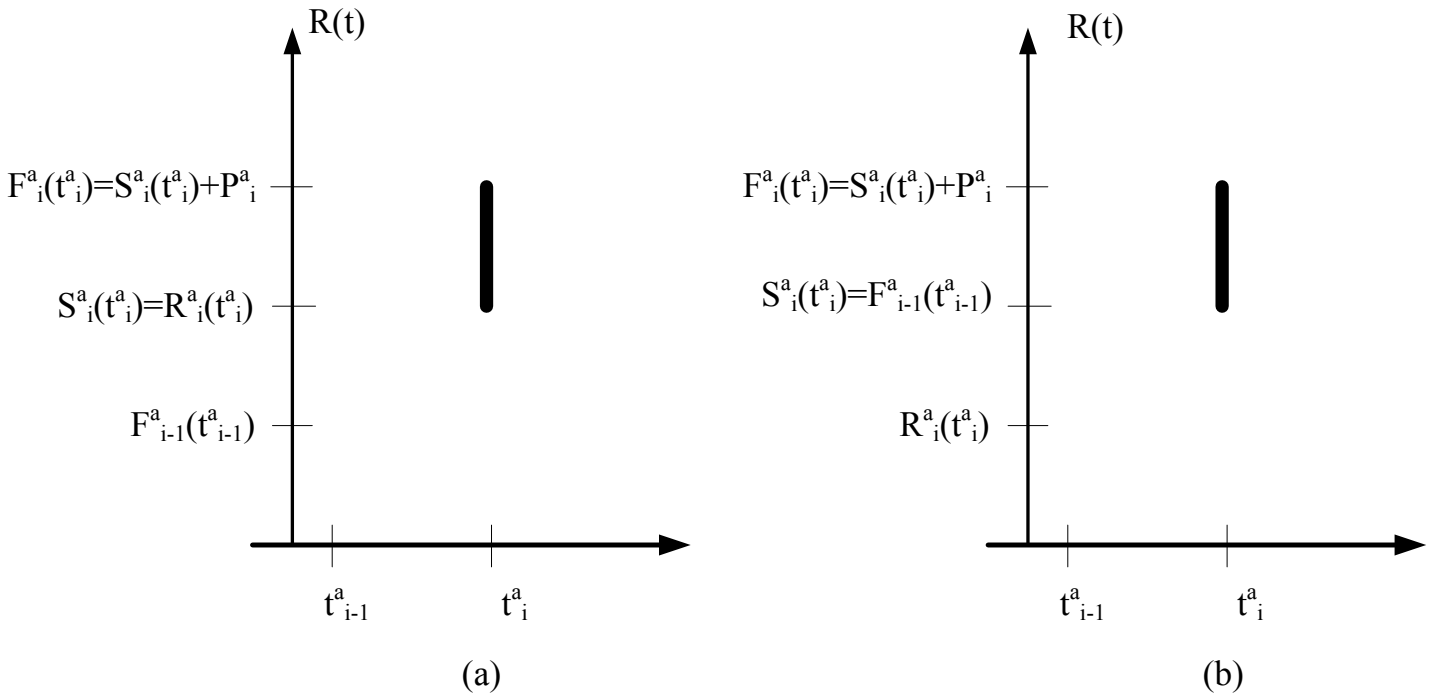


Best-effort policies → do not impose requirements regarding either the amount of resources allocated to an application, or the timing when an application is scheduled.

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

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

Fair queuing - schedule multiple flows through a switch



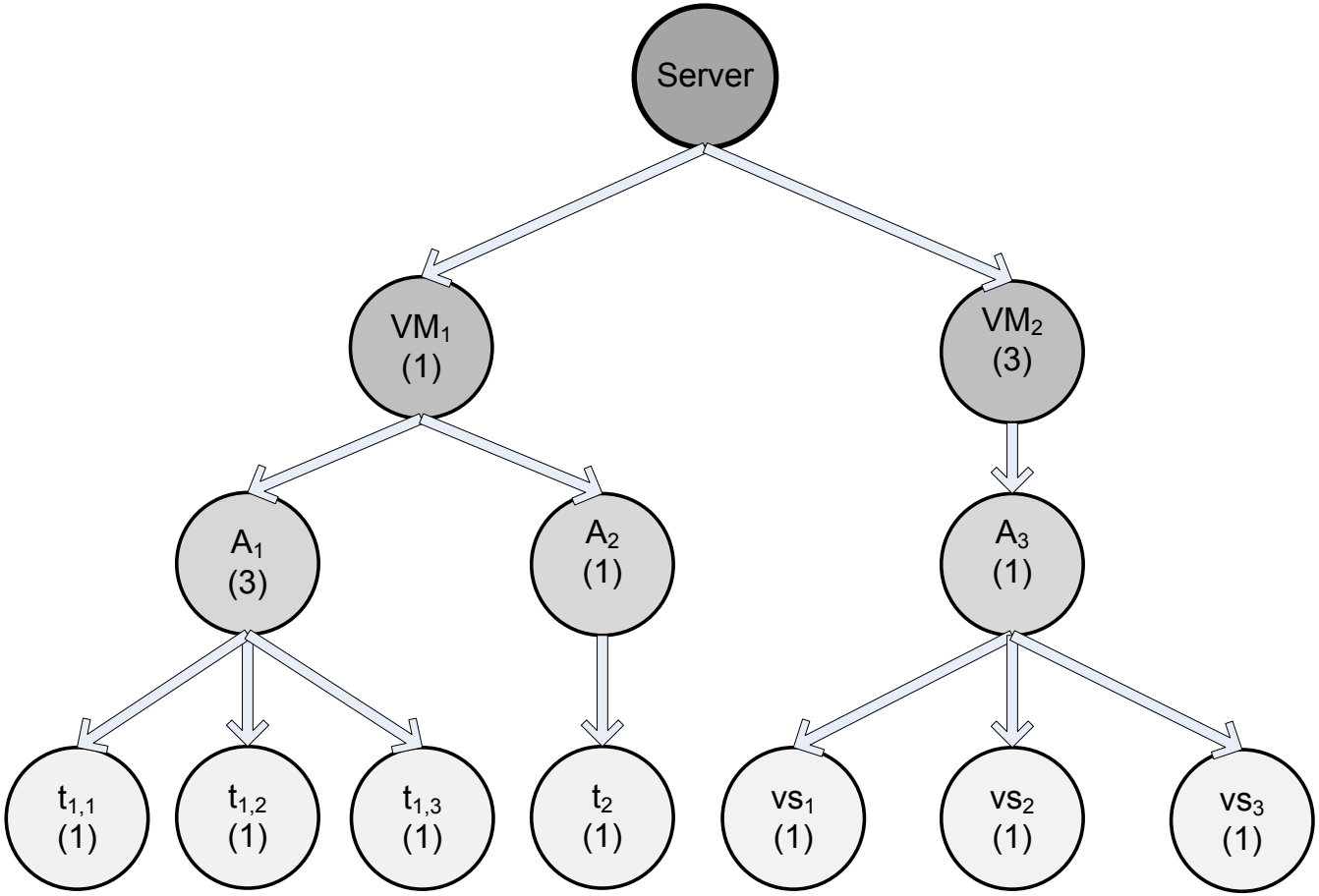
The transmission of packet ***i*** of a flow can only start after the packet is available and the transmission of the previous packet has finished.

(a) The new packet arrives after the previous has finished.

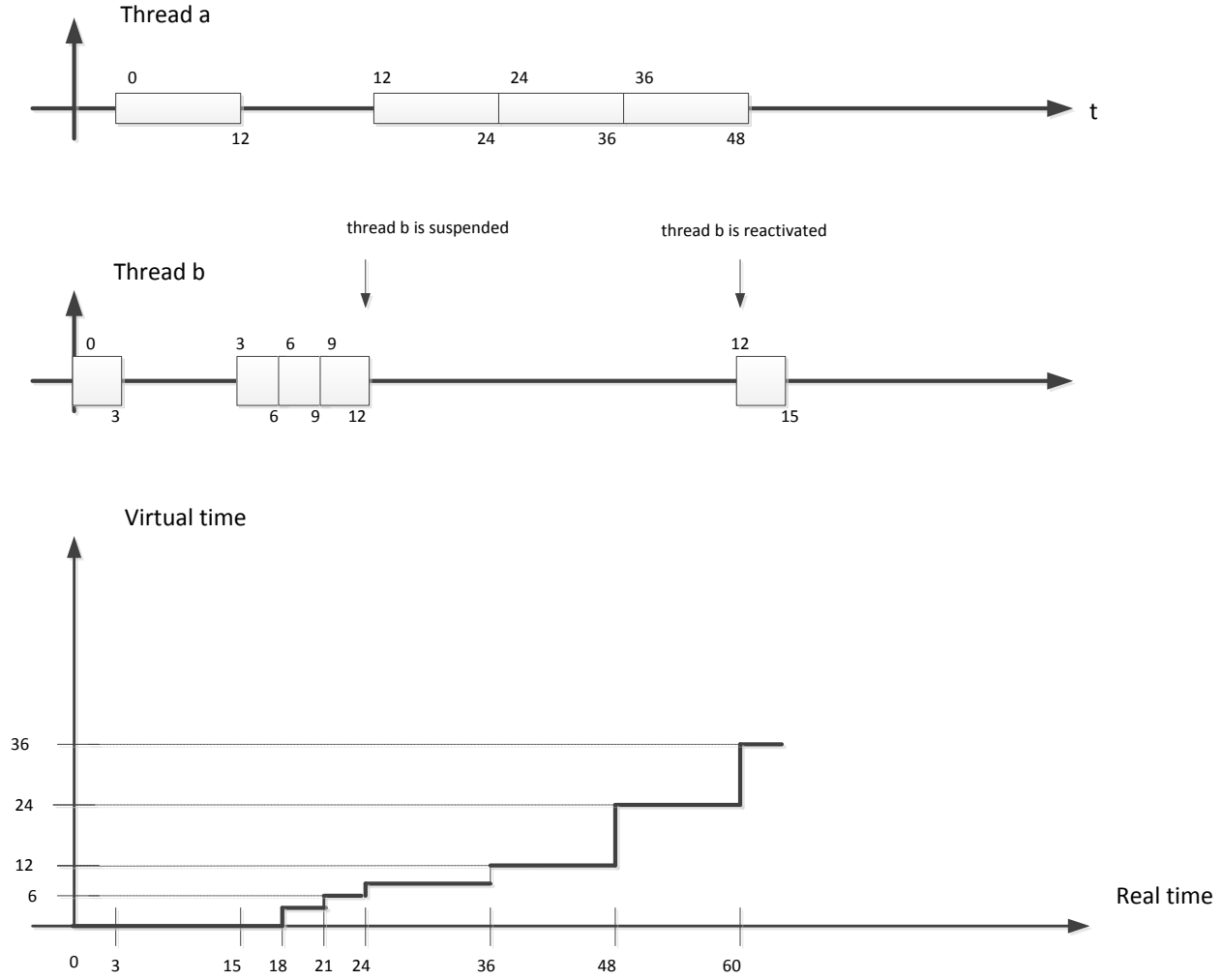
(b) The new packet arrives before the previous one was finished.

Start-time fair queuing

- 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.
  - 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.
  - If one of the threads of an application is not runnable then its allocation is transferred to the other threads of the applications.



The SFQ tree for scheduling when two virtual machines VM_1 and VM_2 run on a powerful server

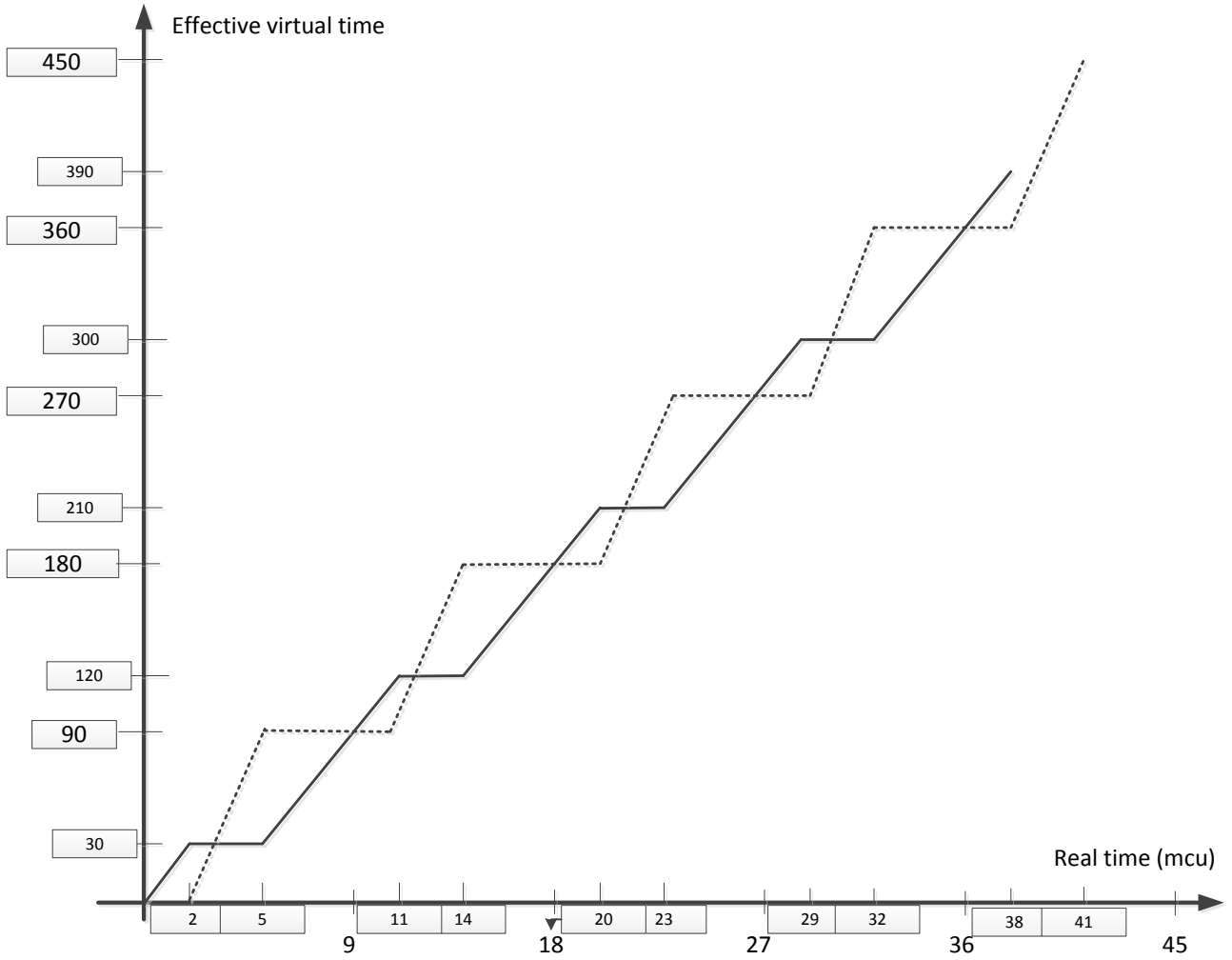


Top → the virtual startup time and the virtual finish time and function of the real time t for each activation of threads ***a*** and ***b***.

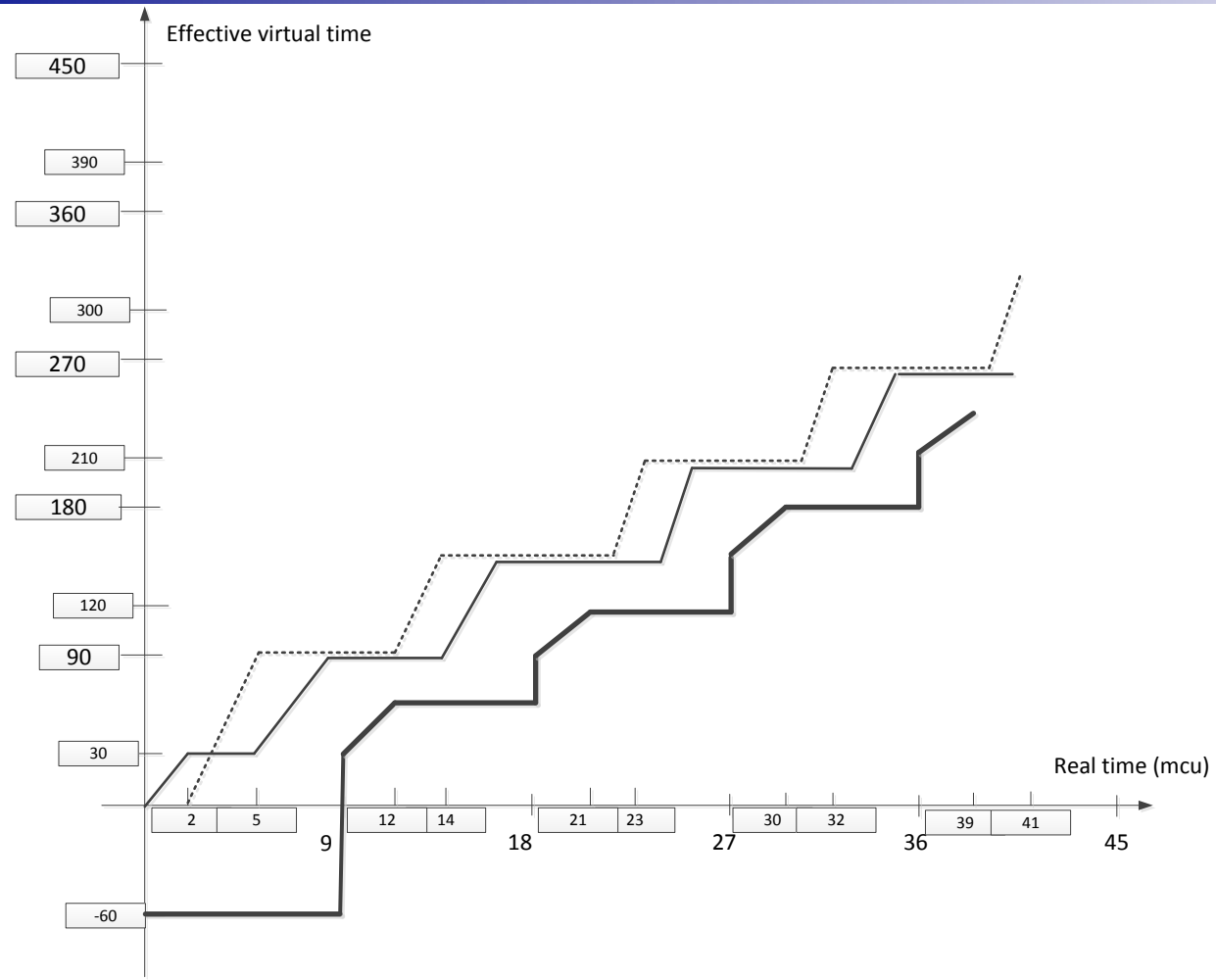
Bottom → the virtual time of the scheduler ***v(t)*** function of the real time

Borrowed virtual time (BVT)

- Objective - support low-latency dispatching of real-time applications, and weighted sharing of CPU among several classes of applications.
- A thread ***i*** has
  - an effective virtual time, E_i.
  - an actual virtual time, A_i.
  - a virtual time warp, W_i.
- The scheduler thread maintains its own scheduler virtual time (SVT) defined as the minimum actual virtual time of any thread.
- The threads are dispatched in the order of their effective virtual time, policy called the Earliest Virtual Time (EVT).
- Context switches are triggered by events such as:
  - the running thread is blocked waiting for an event to occur.
  - the time quantum expires.
  - an interrupt occurs.
  - when a thread becomes runnable after sleeping.



The effective virtual time and the real time of the threads ***a*** (solid line) and ***b*** (dotted line) with weights $w_a = 2\ w_b$ when the actual virtual time is incremented in steps of 90 mcu.



The effective virtual time and the real time of the threads ***a*** (solid line), ***b*** (dotted line), and the ***c*** with real-time constraints (thick solid line). Thread ***c*** wakes up periodically at times t=9, 18, 27, 36,…, is active for 3 units of time and has a time warp of 60 mcu.

Cloud scheduling subject to deadlines

- Hard deadlines → if the task is not completed by the deadline, other tasks which depend on it may be affected and there are penalties; a hard deadline is strict and expressed precisely as milliseconds, or possibly seconds.

- Soft deadlines → more of a guideline and, in general, there are no penalties; soft deadlines can be missed by fractions of the units used to express them, e.g., minutes if the deadline is expressed in hours, or hours if the deadlines is expressed in days. (cloud schedules are usually in this category)

- We consider only aperiodic tasks with arbitrarily divisible workloads.

System Model

- Aperiodic tasks with arbitrarily divisible workloads only.
- The application runs on a partition of a cloud, a virtual cloud with a head node and n worker nodes.
- The system is homogeneous, all workers are identical, and the communication time from the head node to any worker node is the same.
- The problems to be resolved are:
  1. The order of execution of the tasks.
  2. The workload partitioning and task mapping to the worker nodes.
- The most common scheduling policies to determine the order of execution of the tasks are:
  1. FIFO
  2. Earliest deadline first (EDF)
  3. Maximum workload derivative first (MWF)

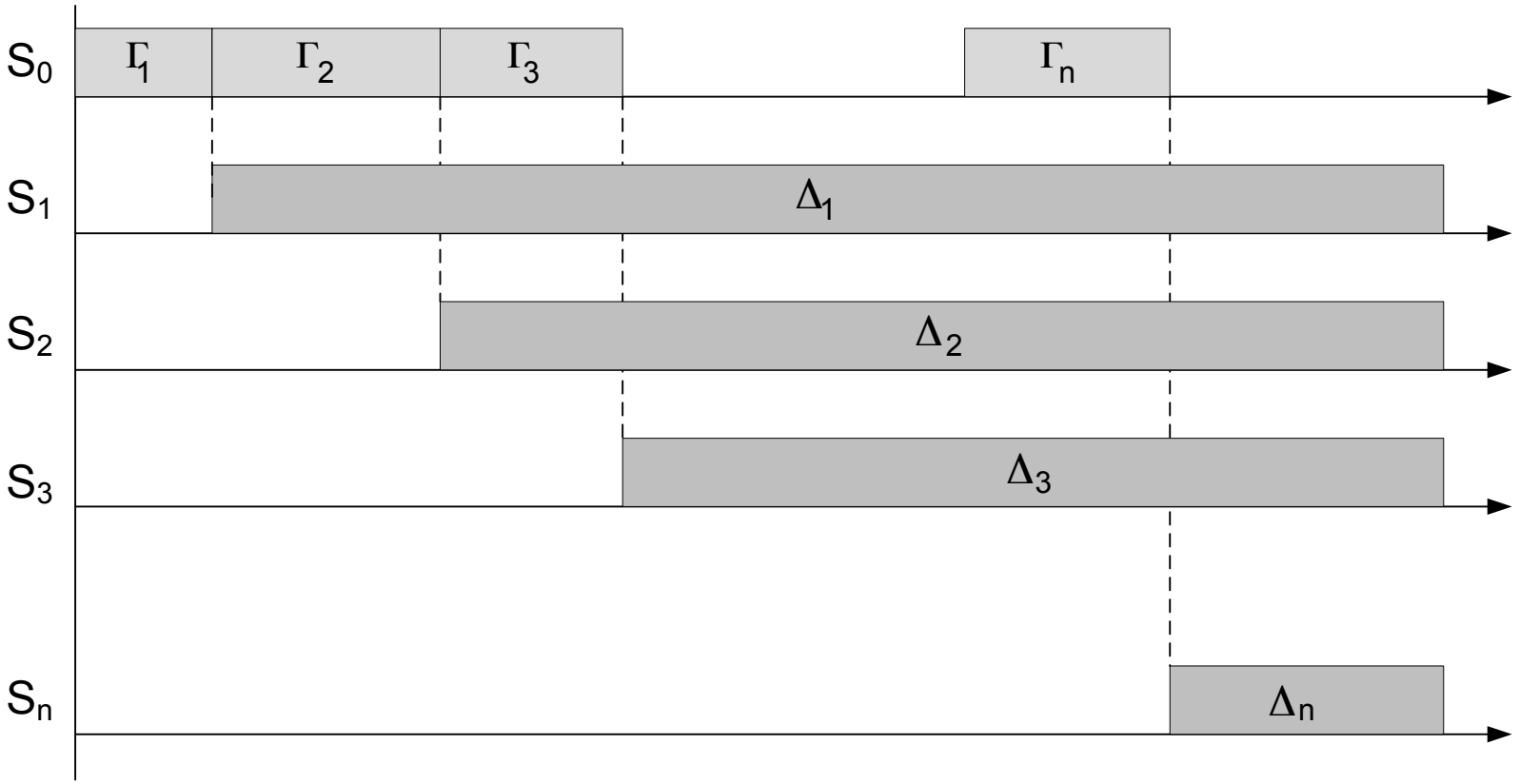
Scheduling Policies

- First in, First out (FIFO) → The tasks are scheduled for execution in the order of their arrival.
- Earliest deadline first (EDF) → The task with the earliest deadline is scheduled first.
- Maximum workload derivative first (MWF) → The tasks are scheduled in the order of their derivatives, the one with the highest derivative first. The number n of nodes assigned to the application is kept to a minimum.

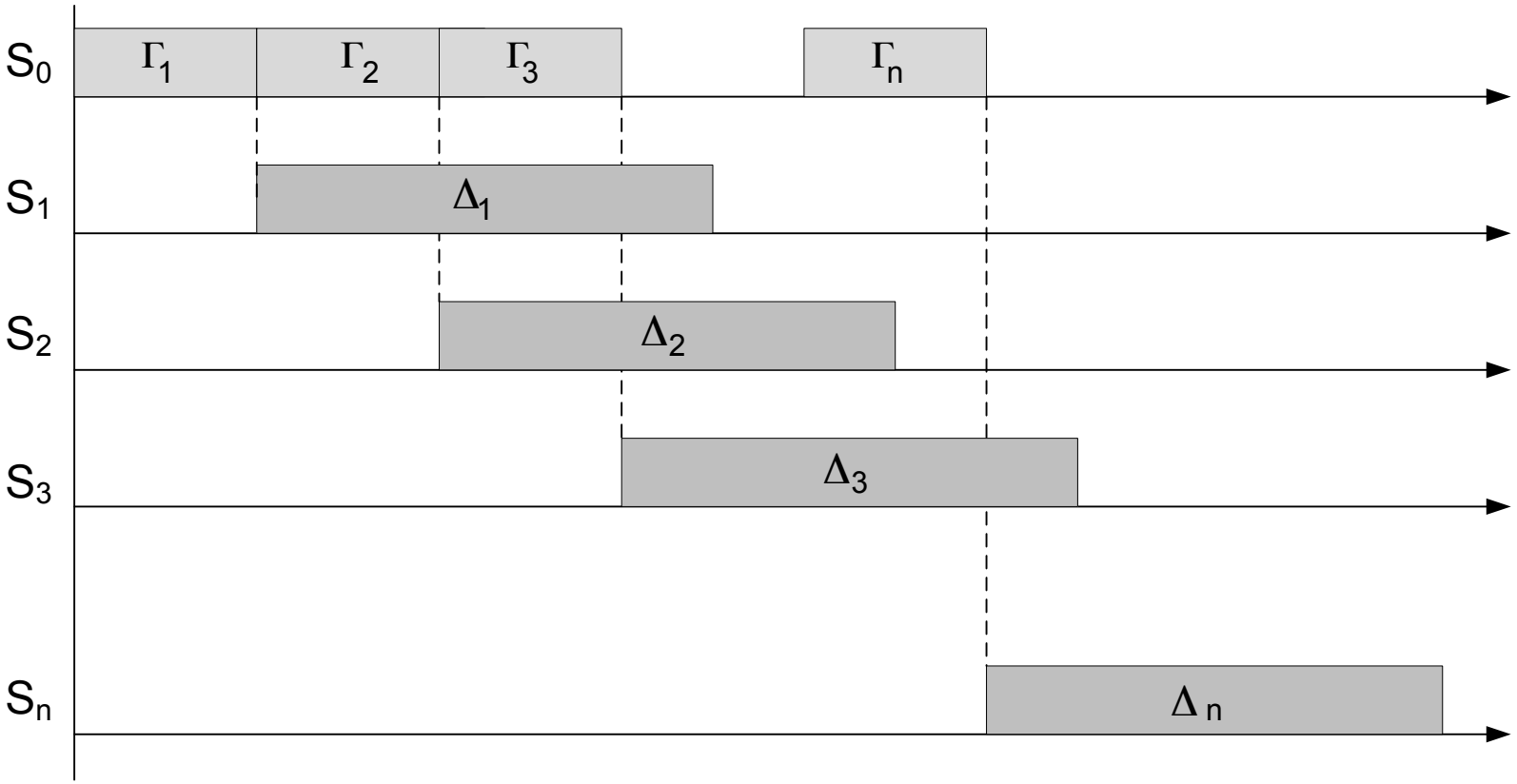
Workload Partitioning Rules

- Optimal Partitioning Rule (OPR) → the workload is partitioned to ensure the earliest possible completion time and all tasks are required to complete at the same time.
  - The head node distributes sequentially the data to individual worker nodes.
  - Worker nodes start processing the data as soon as the transfer is complete.

- Equal Partitioning Rule (EPR) → assigns an equal workload to individual worker nodes.
  - The head node distributes sequentially the data to individual worker nodes.
  - Worker nodes start processing the data as soon as the transfer is complete.
  - The workload is partitioned in equal segments.



The timing diagram for the Optimal Partitioning Rule; the algorithm requires worker nodes to complete execution at the same time. The head node, S_0, distributes sequentially the data to individual worker nodes.



The timing diagram for the Equal Partitioning Rule; the algorithm assigns an equal workload to individual worker nodes.

Scheduling MapReduce Applications

subject to deadlines

Four commonly referenced scheduling are:

- The default FIFO schedule
- The Fair Scheduler
- The Capacity Scheduler
- The Dynamic Proportional Scheduler

They are all based upon the assumptions:

1. The system is homogeneous
2. Load equipartition