
NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE
CE/CZ4052
Cloud Computing
Cloud CPU Scheduling
Dr. Tan, Chee Wei
Email: cheewei.tan@ntu.edu.sg
Office: N4-02c-104

Outline

- Why need CPU scheduling?
- Some definitions
- Examples of scheduling algorithms in Xen

Motivation

- When multiple VMs send instructions to the host, which instructions should the host CPU execute first?
  - Resource contention
- On a host with 1GHz CPU, how to emulate a 400MHz CPU?
- CPU scheduling is important in delivering performance guarantees and resource isolation.

CPU Scheduling

- Implemented in the hypervisor

Fairness

- Idea: use weights
- Each VM gets a share of CPU in proportion to the weight
- Weighted max-min; Proportional fairness, Priority scheduling

Utilization

- What if a VM is just idle, while the other has lots of things to do? (suppose they have equal weights)
- Should the host CPU allocate all the available cycles to the busy VM?

Utilization

- Work-conserving: The host CPU is idle if and only if there is no runnable VM. The weights act as guarantees. Utilization is good.

- Non work-conserving: Weights are caps. Each VM owns its fraction of CPU.

A Tradeoff

- Tradeoff between fairness and utilization
  - work-conserving schedulers are more efficient, but less fair
  - non work-conserving schedulers are less efficient, but more fair

Simple earliest deadline first

- SEDF. EDF is a widely used scheduling algorithm in OS.
- Each VM specifies its CPU requirement with a tuple (s_i, p_i, x_i)
- VM i wants to receive at least s_i units of time in each period of length p_i.
- x_i: whether to receive extra CPU time (work conserving)

SEDF

- For each VM, the scheduler keeps two variables:

  d_i: time at which VM *i*'s current period ends, i.e. *deadline*

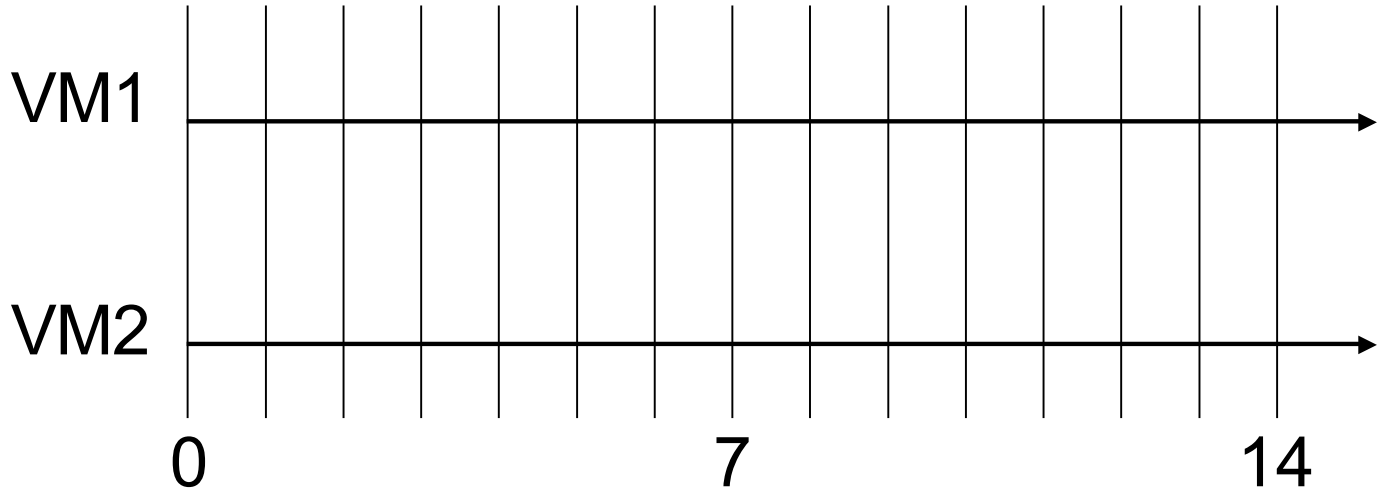
  r_i: remaining CPU time in the current period

- At each time slot, among the VMs whose r_i is positive, schedule the one with the earliest deadline to run. Ties are broken arbitrarily.

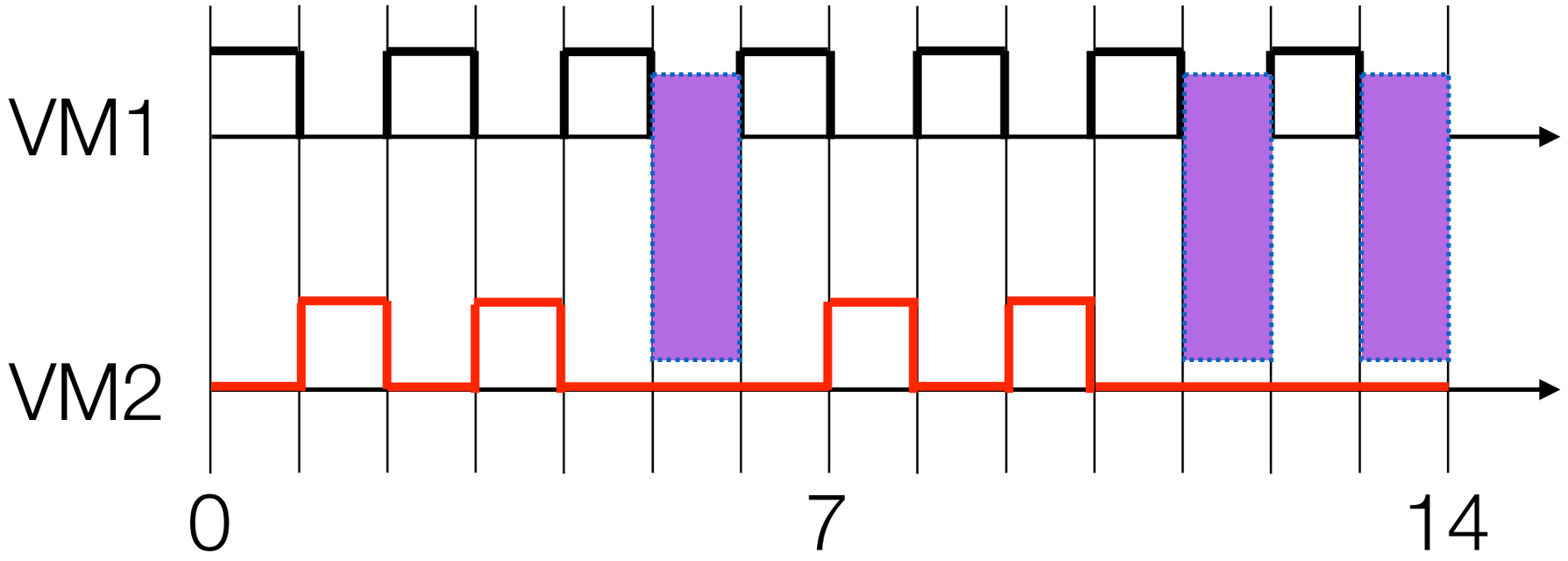
SEDF example

- When x_i=0, non work conserving, the VM is made runnable periodically, i.e. r_i is reset to s_i at the start of each period.

- Example: VM 1: (1,2,0), VM 2: (2,7,0)



SEDF example



- The least common multiple of the periods is 14. So the scheduling pattern will repeat every 14 slots.

- What's the CPU utilization? It suffices to look at the first 14 slots.

SEDF – schedulability

- Any VM set is schedulable by SEDF if and only if (iff)

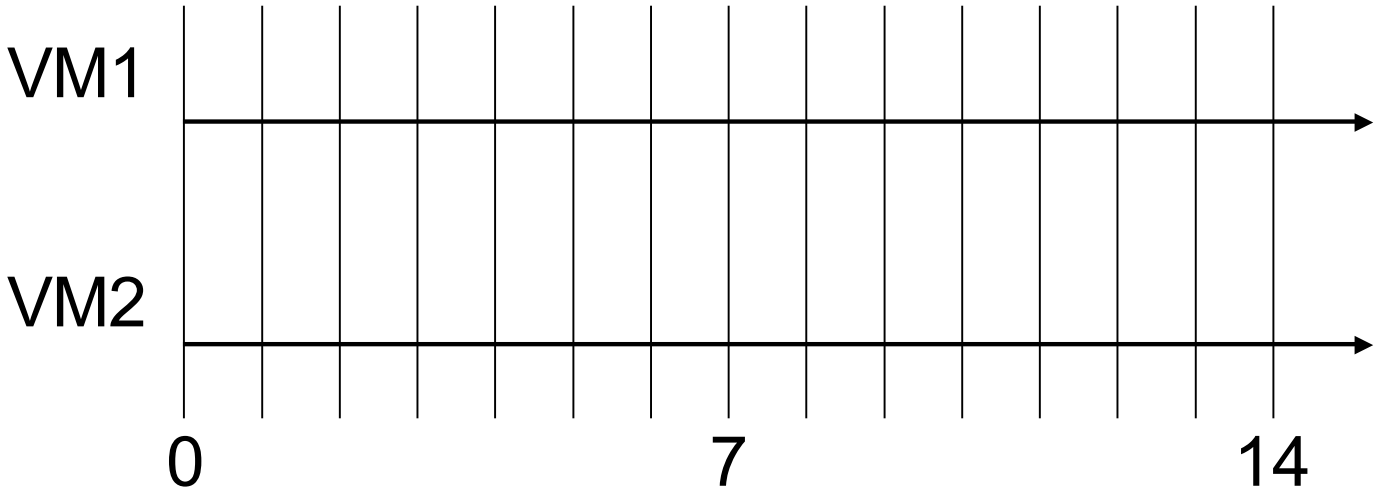
$$\sum_i \frac{s_i}{p_i} \leq 1$$

- Run the scheduability test first before admitting a new VM

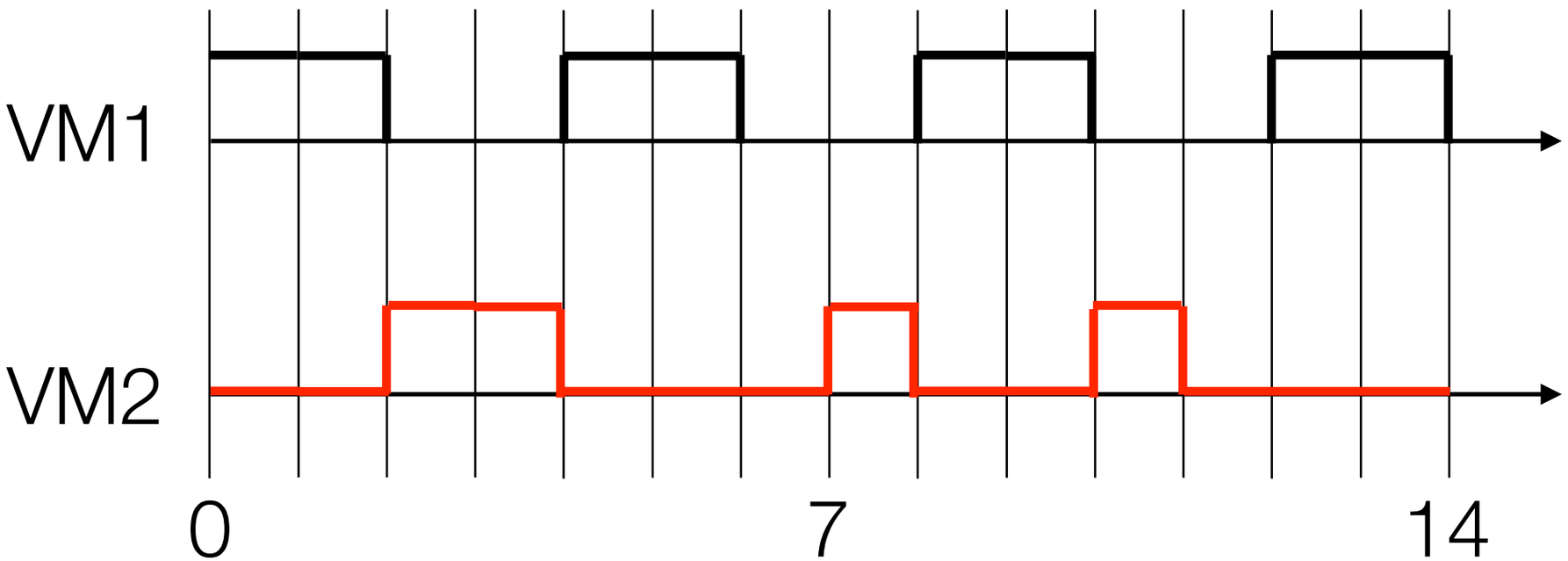
SEDF – time granularity

- The time granularity of the period impacts scheduler fairness.

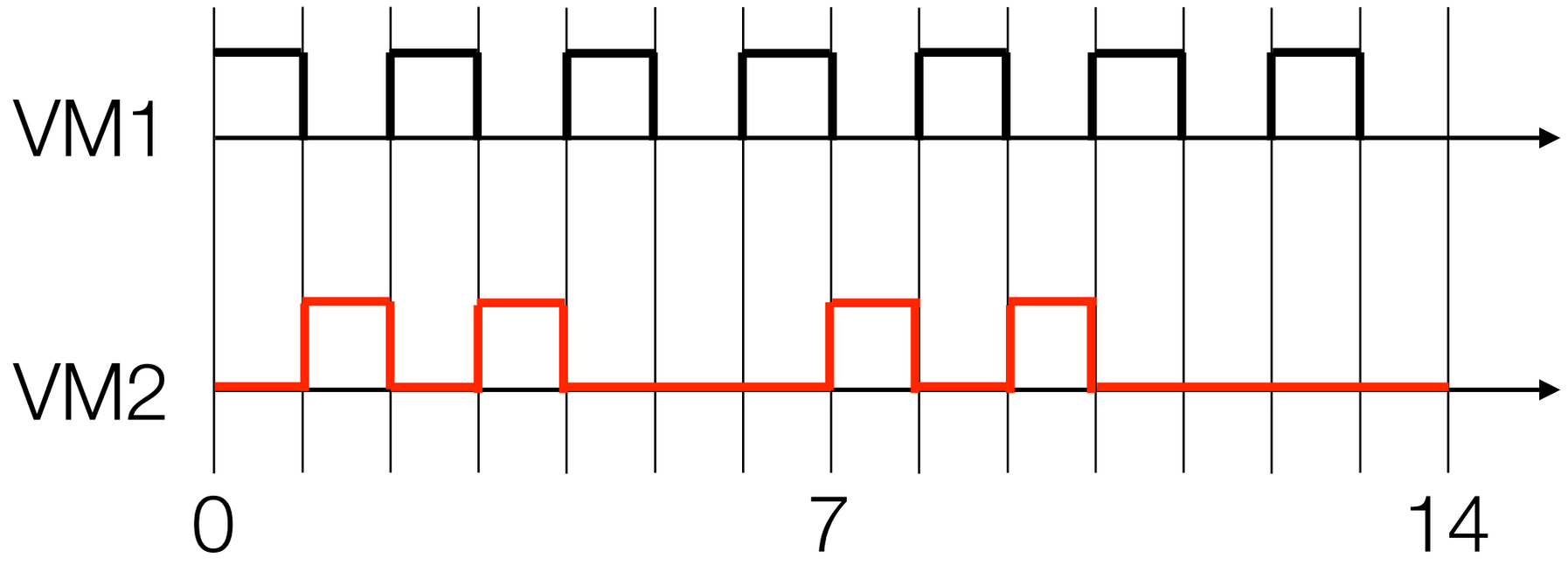
- Example: VM 1: (2,4,0), VM 2: (2,7,0)



VM 1: $(2, 4, 0)$, VM 2: $(2, 7, 0)$



VM 1: $(1, 2, 0)$, VM 2: $(2, 7, 0)$

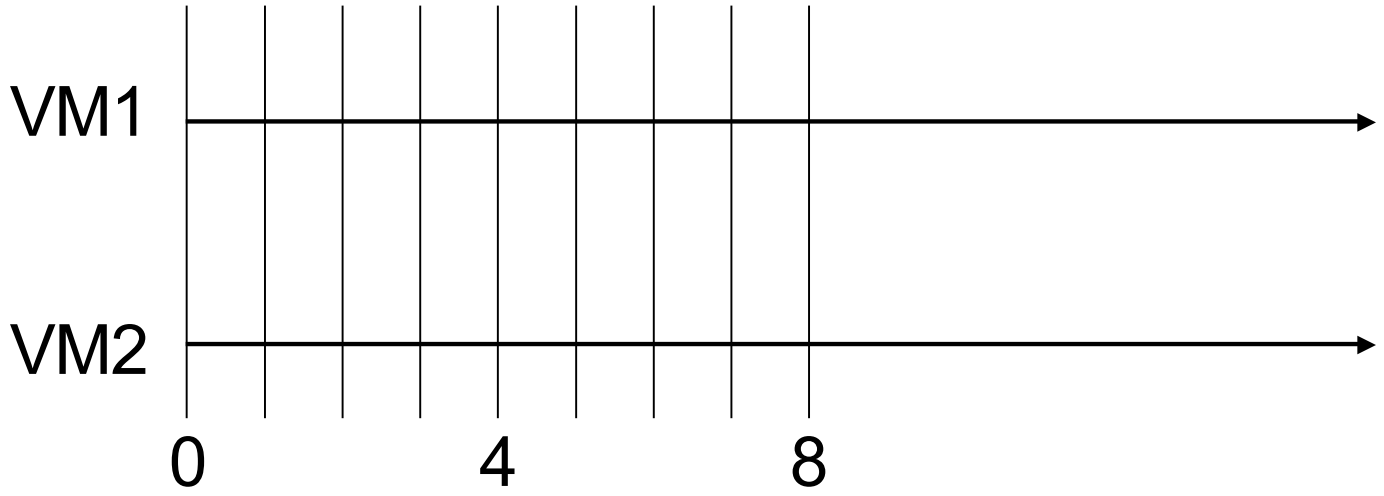


SEDF – work-conserving

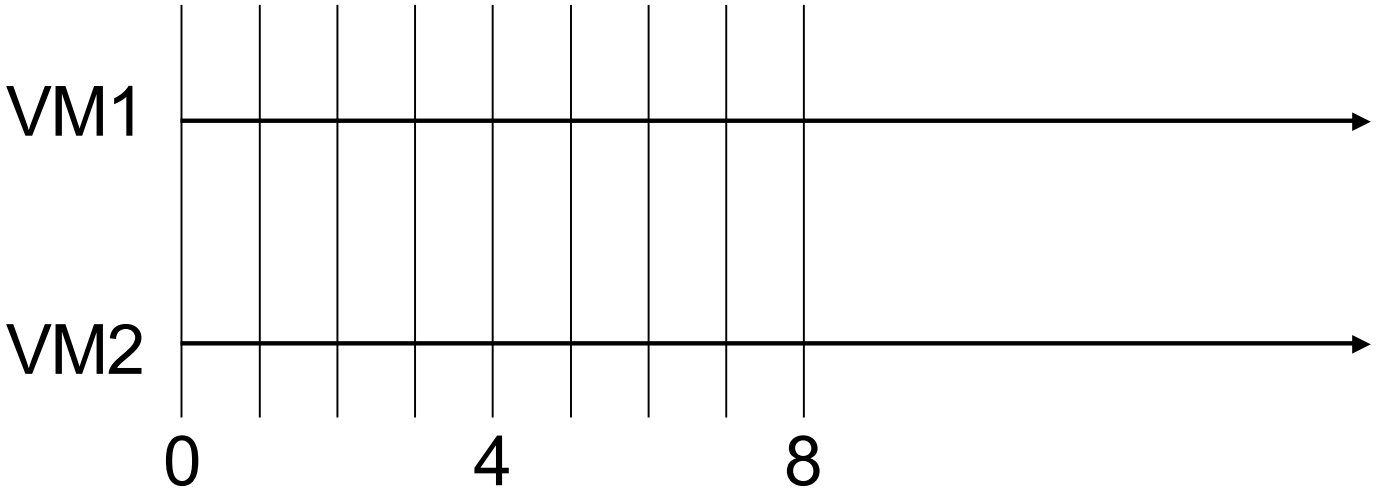
- In WC mode ($x_i = 1$), a VM is always runnable (given it has work to do). r_i is reset to s_i immediately after it reduces to 0, and d_i is incremented by p_i at the same time.

- The slack CPU time is allocated to VMs (whose $x_i=1$) in a weighted fair sharing manner.

VM 1: (1, 2, 0), VM 2: (1, 4, 0)



VM 1: (1, 2, 1), VM 2: (1, 4, 1)



SEDF – work-conserving

- Q: what about the following case?

$$\text{VM 1: } (1, 2, 1),\ \text{VM 2: } (1, 4, 0)$$

- Q: or this one?

$$\text{VM 1: } (1, 2, 0),\ \text{VM 2: } (1, 4, 1)$$

SEDF – more examples

- Exercise for tutorial Try to schedule the following cases.

VM 1: (2, 6, 0), VM 2: (1, 3, 0), VM 3: (1, 6, 0)

VM 1: (1, 8, 0), VM 2: (2, 5, 0), VM 3: (4, 6, 10)

- Optional and experimental: Try to set x_i=1 for some of the VMs…

SEDF – summary

- Features:
  - Fairness depends on the value of the period
  - WC and NWC modes
- Disadvantage:
  - Implements per-CPU scheduling. Lacks global load balancing on multiprocessors

Credit scheduler

- Xen's latest PS (proportional share) scheduler featuring automatic load balancing across physical CPUs.
- Default scheduler in Xen.
- Each VM is assigned a weight and a cap.

Credit scheduler

- Idea based on Stride Scheduling

  - Jobs have a "stride" value
    - A stride value describes the counter pace when the job should give up the CPU
    - Stride value is **inverse in proportion** to the job's number of tickets (more tickets = smaller stride)
  - Total system tickets = 10,000
    - Job A has 100 tickets → A_{stride} = 10000/100 = 100 stride
    - Job B has 50 tickets → B_{stride} = 10000/50 = 200 stride
    - Job C has 250 tickets → C_{stride} = 10000/250 = 40 stride
  - Stride scheduler tracks "pass" values for each job (A, B, C)

Stride scheduler

- Basic algorithm:
  1. Stride scheduler picks job with the **lowest** pass value
  2. Scheduler increments job's pass value by its stride and starts running the job for the current time slice
  3. Stride scheduler increments a system counter
  4. After scheduling quantum, scheduler returns to #1

- Stride scheduler always runs job(s) with the lowest pass value(s)

- KEY: Jobs having low "PASS" values are scheduled more often because their pass values increase more slowly than others…

Stride scheduler

- Stride values
  - Tickets = priority to select job
  - Stride is inverse to tickets
  - Lower stride = more chances to run (higher priority)

Priority

C stride = 40

A stride = 100

B stride = 200

Stride scheduler

- Three-way tie: randomly pick job A (all pass values=0)
- Set A's pass value to A's stride = 100
- Increment sys counter by A's stride. counter → 100
- Pick a new job: two-way tie

Tickets
C = 250
A = 100
B = 50

Pass values

Pass(A) (stride=100)	Pass(B) (stride=200)	Pass(C) (stride=40)	Who Runs?
0	0	0	A
100	0	0	B
100	200	0	C
100	200	40	C
100	200	80	C
100	200	120	A
200	200	120	C
200	200	160	C
200	200	200	...

Initial job selection is random. All @ 0

C has the most tickets and receives a lot of opportunities to run…

Stride scheduler

- We set A's counter (pass value) to A's stride = 100
- Next scheduling decision between B (pass=0) and C (pass=0)
  - Randomly choose B
- C has the lowest counter for next 3 rounds

Pass(A) (stride=100)	Pass(B) (stride=200)	Pass(C) (stride=40)	Who Runs?
0	0	0	A
100	0	0	B
100	200	0	C
100	200	40	C
100	200	80	C
100	200	120	A
200	200	120	C
200	200	160	C
200	200	200	...

Tickets
C = 250
A = 100
B = 50

C has the most tickets and is selected to run more often …

Stride scheduler

- Job counters support determining which job to run next
- Over time jobs are scheduled to run based on their priority represented as their **share of tickets…**
- **Tickets are analogous to job priority**

Tickets
C = 250
A = 100
B = 50

Pass(A) (stride=100)	Pass(B) (stride=200)	Pass(C) (stride=40)	Who Runs?
0	0	0	A
100	0	0	B
100	200	0	C
100	200	40	C
100	200	80	C
100	200	120	A
200	200	120	C
200	200	160	C
200	200	200	…

Stride scheduler

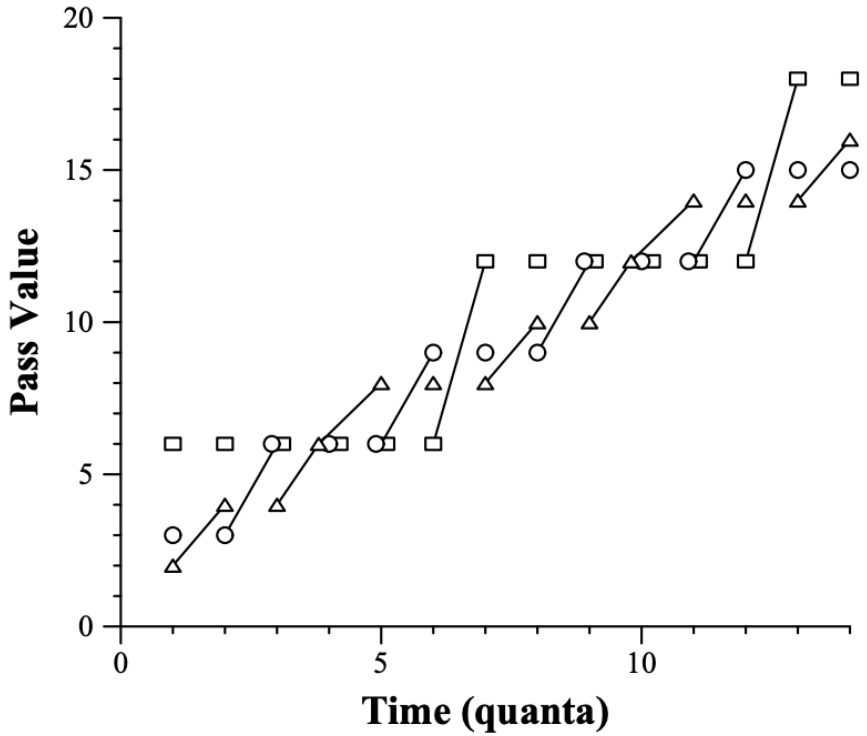


Figure 3-12: **Stride Scheduling Example.** Clients A (triangles), B (circles), and C (squares) have a 3 : 2 : 1 ticket ratio. In this example, $stride_1 = 6$, yielding respective strides of 2, 3, and 6. For each quantum, the client with the minimum pass value is selected, and its pass is advanced by its stride.

C. Waldspurger's PhD Thesis at MIT: http://www.waldspurger.org/carl/papers/phd-mit-tr667.pdf

Credit scheduler

- Weight: default value is 256. Range: [1,65535]
- Cap: optionally fix the CPU this VM can get, even if the CPU has idle cycles (NWC)
  - default is 0, WC
  - set to percentages. For example 30, meaning this VM can get 30% of the CPU at most.

Credit scheduler – basics

- Time unit/slot: 30ms
- A VM has priority, which can be one of two values: “over” or “under” representing wether this VM has or hasn't yet exceeded its fair share of CPU resource in the ongoing accounting period.
- Each CPU manages a FIFO queue of runnable VMs sorted by priority (under precedes over).
- At each slot, the VM off the head of the queue gets to run

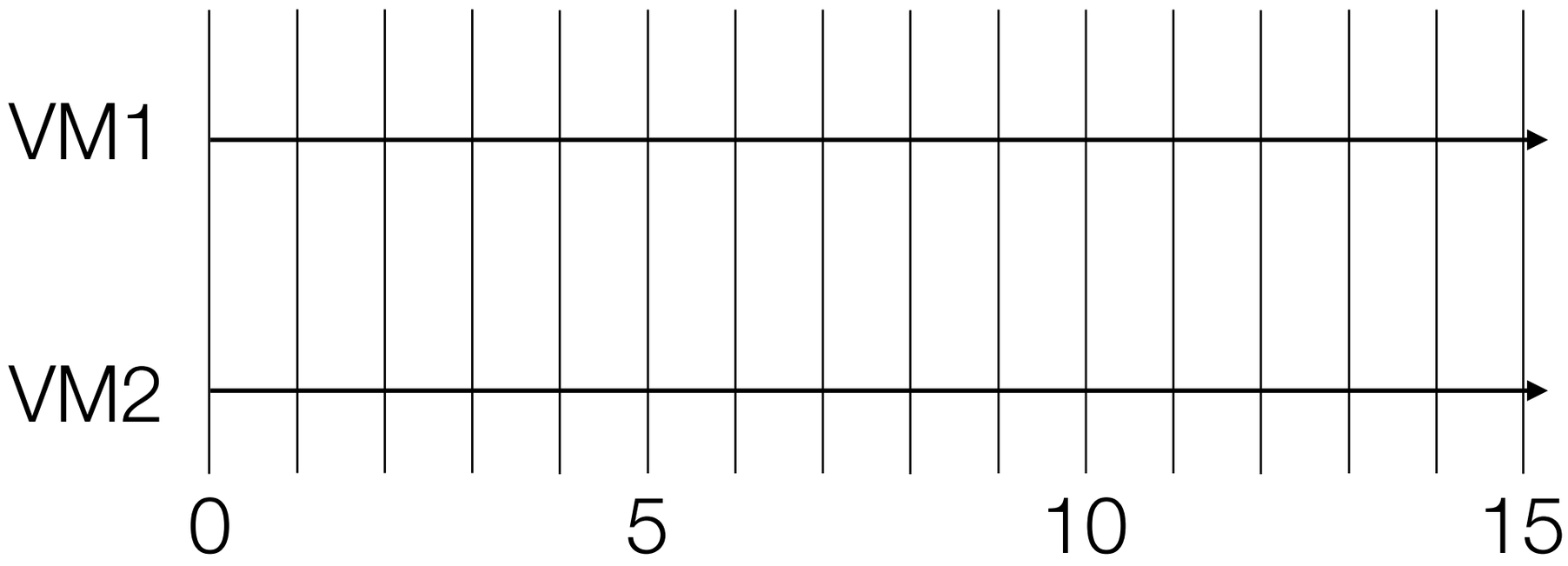
Credit scheduler – accounting

- As a VM runs, it consumes credits. Every 30ms, a system-wide accounting thread recomputes the credits for each active VM.
- Every so often, the system bumps credit to each VM. The credit is allocated in a PS fashion (weighted fair sharing).

Credit scheduler – example

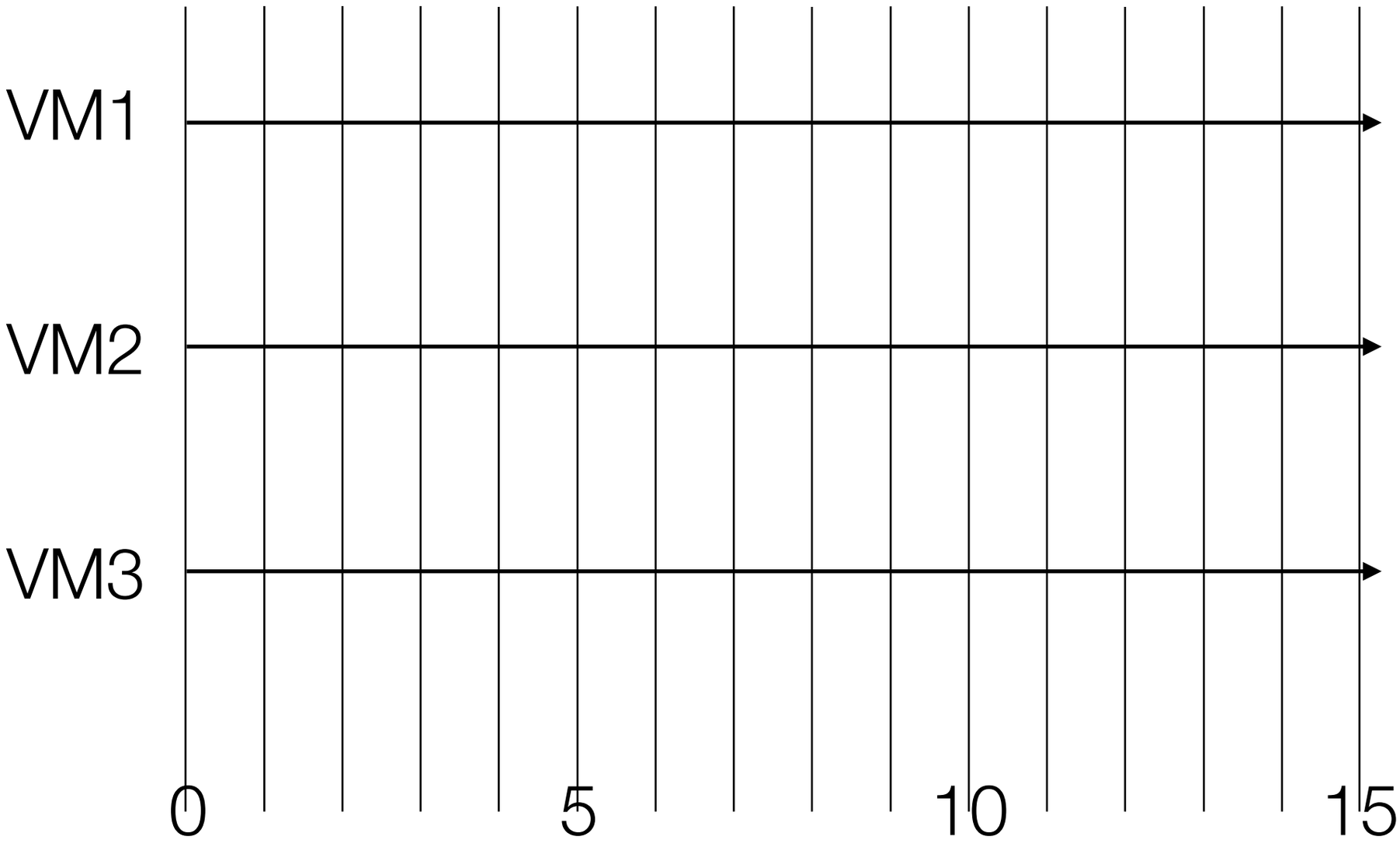
- Assume 1 credit = 10ms. The system allocates 15 credits every 150ms, i.e. 5 time slots.

- VM1 weight: 256, VM2 weight: 512



Credit scheduler – example

- Under the same setting, consider the case with three VMs, with weights 256, 512, and 512.



Credit scheduler – summary

- Features:
  - Global load balancing
  - WC and NWC modes
  - Easy to implement