CS 202: Advanced Operating Systems

Lottery Scheduling

Problems with Traditional schedulers

- Priority systems are ad hoc: highest priority always wins
- Try to support fair share by adjusting priorities with a feedback loop
  - Works over long term
  - highest priority still wins all the time, but now the Unix priorities are always changing
- Priority inversion: high-priority jobs can be blocked behind low-priority jobs
- Schedulers are complex and difficult to control

Lottery scheduling

- Elegant way to implement proportional share scheduling
- Priority determined by the number of tickets each thread has:
  - Priority is the relative percentage of all of the tickets whose owners compete for the resource
- Scheduler picks winning ticket randomly, gives owner the resource
- Tickets can be used for a variety of resources

Example

- Three threads
  - A has 5 tickets
  - B has 3 tickets
  - C has 2 tickets
- If all compete for the resource
  - B has 30% chance of being selected
- If only B and C compete
  - B has 60% chance of being selected

It's fair

- Lottery scheduling is ***probabilistically fair***
- If a thread has a t tickets out of T
  - Its probability of winning a lottery is $p = t/T$
  - Its expected number of wins over n drawings is np
    - Binomial distribution
    - Variance $\sigma^2 = np(1 - p)$

Fairness (II)

- Coefficient of variation of number of wins
  $\boldsymbol{\sigma/np = \sqrt{(1-p)/np}}$
  - Decreases with $\sqrt{\boldsymbol{n}}$
- Number of tries before winning the lottery follows a ***geometric distribution***
- As time passes, each thread ends receiving its share of the resource

Ticket transfers

- How to deal with dependencies?
  - Explicit transfers of tickets from one client to another

- Transfers can be used whenever a client blocks due to some dependency
  - When a client waits for a reply from a server, it can temporarily transfer its tickets to the server
    - Server has no tickets of its own
  - Server priority is sum of priorities of its active clients
    - Can use lottery scheduling to give service to the clients

- Similar to priority inheritance
  - Can solve priority inversion

Ticket inflation

- Let users create new tickets
  - Like printing their own money
  - Counterpart is ***ticket deflation***
  - Let mutually trusting clients adjust their priorities dynamically without explicit communication

- Currencies: set up an exchange rate
  - Enables inflation within a group
  - Simplifies mini-lotteries (e.g., for mutexes)

Example (I)

- A process manages three threads
  - A has 5 tickets
  - B has 3 tickets
  - C has 2 tickets

- It creates 10 extra tickets and assigns them to thread C
  - Why?
  - Process now has 20 tickets

Example (II)

- These 20 tickets are in a new currency whose exchange rate with the base currency is 10/20

- The total value of the process' tickets expressed in the base currency is still equal to 10

Compensation tickets (I)

- I/O-bound threads likely get less than their fair share of the CPU because they often block before their CPU quantum expires

- Compensation tickets address this imbalance

Compensation tickets (II)

- A client that consumes only a fraction *f* of its CPU quantum ***can*** be granted a ***compensation ticket***
  - Ticket inflates the value by 1/f until the client starts gets the CPU

Example

- CPU quantum is 100 ms
- Client A releases the CPU after 20ms
  - ***f* = 0.2 or 1/5**
- Value of ***all*** tickets owned by A will be multiplied by 5 until A gets the CPU

Compensation tickets (III)

- Compensation tickets
  - Favor I/O-bound—and interactive—threads
  - Helps them getting their fair share of the CPU

Implementation

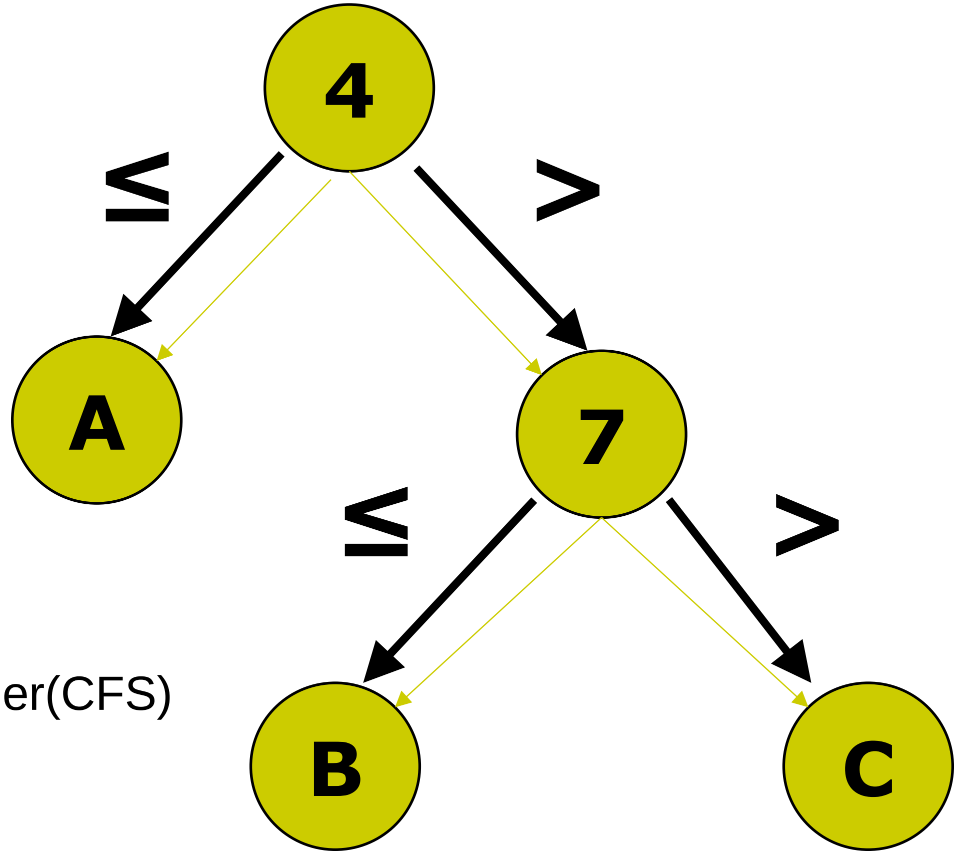
- On a MIPS-based DEC station running Mach 3 microkernel
  - Time slice is 100ms
    - *Fairly large as scheme does not allow preemption*
- Requires
  - A fast RNG
  - A fast way to pick lottery winner

Example

- Three threads
  - A has 5 tickets
  - B has 3 tickets
  - C has 2 tickets
- List contains
  - A (0-4)
  - B (5-7)
  - C (8-9)

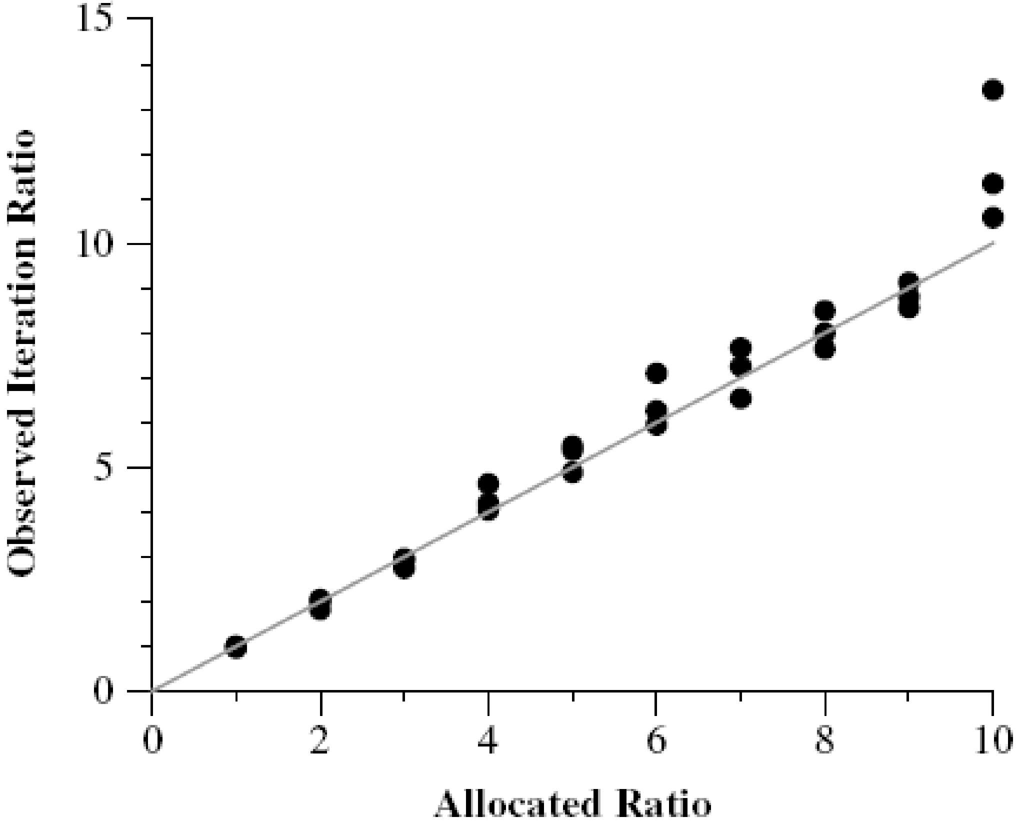
Search time is O(n)
where n is list length

Optimization – use tree

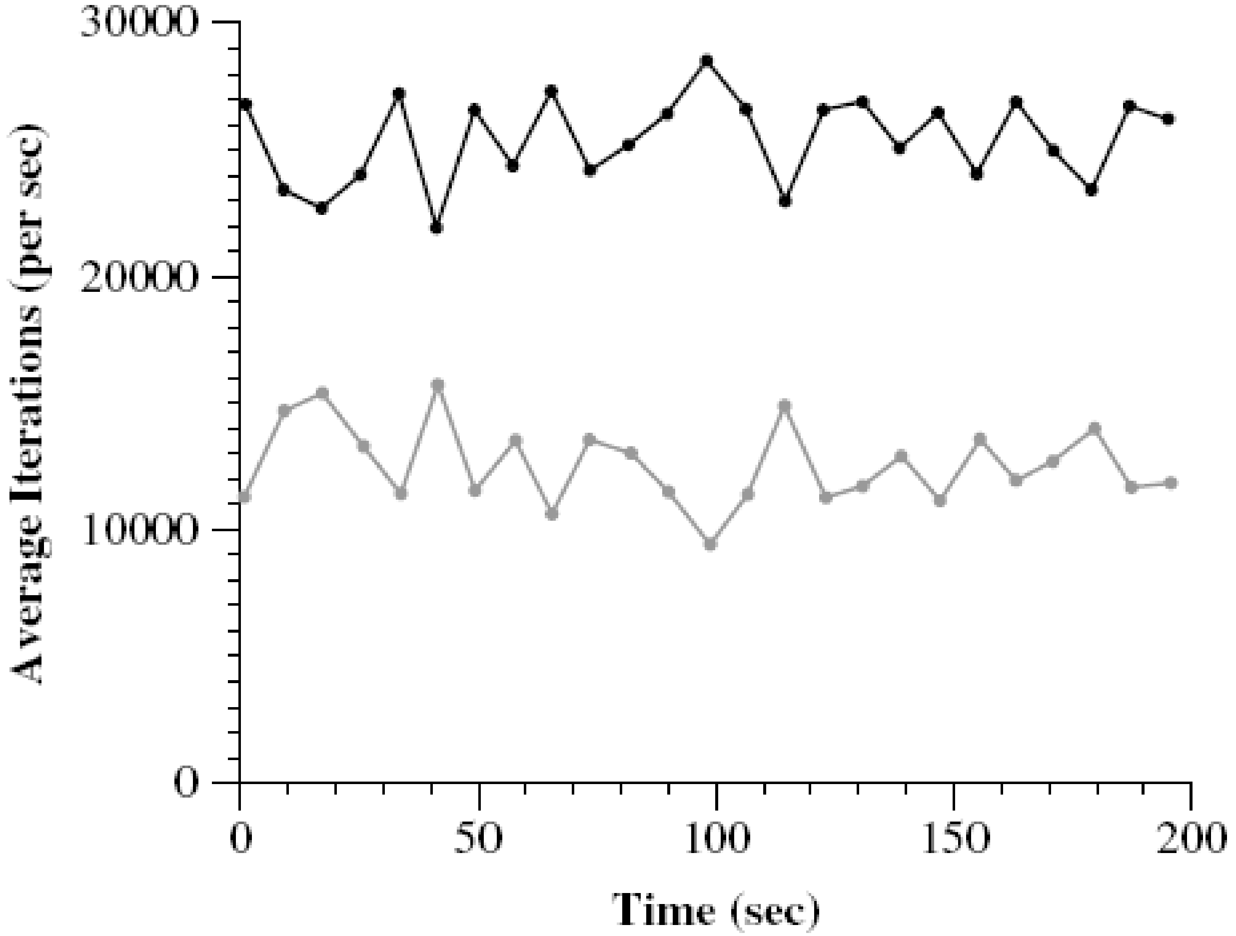


RB Tree used in Linux
Completely fair scheduler(CFS)
--not lottery based

Long-term fairness (I)



Short term fluctuations



For 2:1 ticket alloc. ratio