Jin-Soo Kim
(*jinsoo.kim@snu.ac.kr*)

Systems Software &
Architecture Lab.

Seoul National University

Fall 2021

Lottery Scheduling

(Carl Waldspurger et al., OSDI '94)

Priority-based Scheduling Schemes

- The notion of priority does not provide the encapsulation and modularity properties
- The assignment of priorities and dynamic priority adjustment schemes are ad-hoc
  - Adjusting scheduling parameters is at best a black art
- Poorly understood
- Schedulers are complex and difficult to control
- Priority inversion problem
- Fair share schedulers are implemented by adjusting priorities with a feedback loop (relatively coarse control over long-running applications)

Goals

- Flexible and responsive control over the relative execution rates of computations
- Proportional sharing
- Support for modular resource management
- Simple and efficient implementation

Lottery Scheduling

- A randomized resource allocation mechanism based on tickets and lotteries
- Tickets
  - Encapsulate abstract, relative, and uniform resource rights
  - Abstract, Relative, and Uniform
  - Similar to the properties of money
- Lotteries
  - Scheduler picks the winning ticket randomly, and gives the owner the resource
  - Probabilistically fair
  - The scheduling algorithm is randomized

Performance Characteristics

- The number of lotteries won by a client:
  - Binomial distribution
  - The winning probability *p* (total *T* tickets): $\boldsymbol{p = t/T}$
  - The expected number of wins *w* after *n* lotteries:

$$\boldsymbol{E[w] = np} \qquad \boldsymbol{\sigma_w^2 = np(1-p)} \qquad \boldsymbol{\sigma_w / E[w] = \sqrt{(1-p)/np}}$$

  - A client's throughput is proportional to its ticket allocation
- The number of lotteries required for a client's first win:
  - Geometric distribution
  - The expected number of lotteries *n* that a client must wait before its first win:

$$\boldsymbol{E[n] = 1/p} \qquad \boldsymbol{\sigma_n^2 = (1-p)/p^2}$$

  - The client's average response time is inversely proportional to its ticket allocation

Performance Characteristics (cont'd)

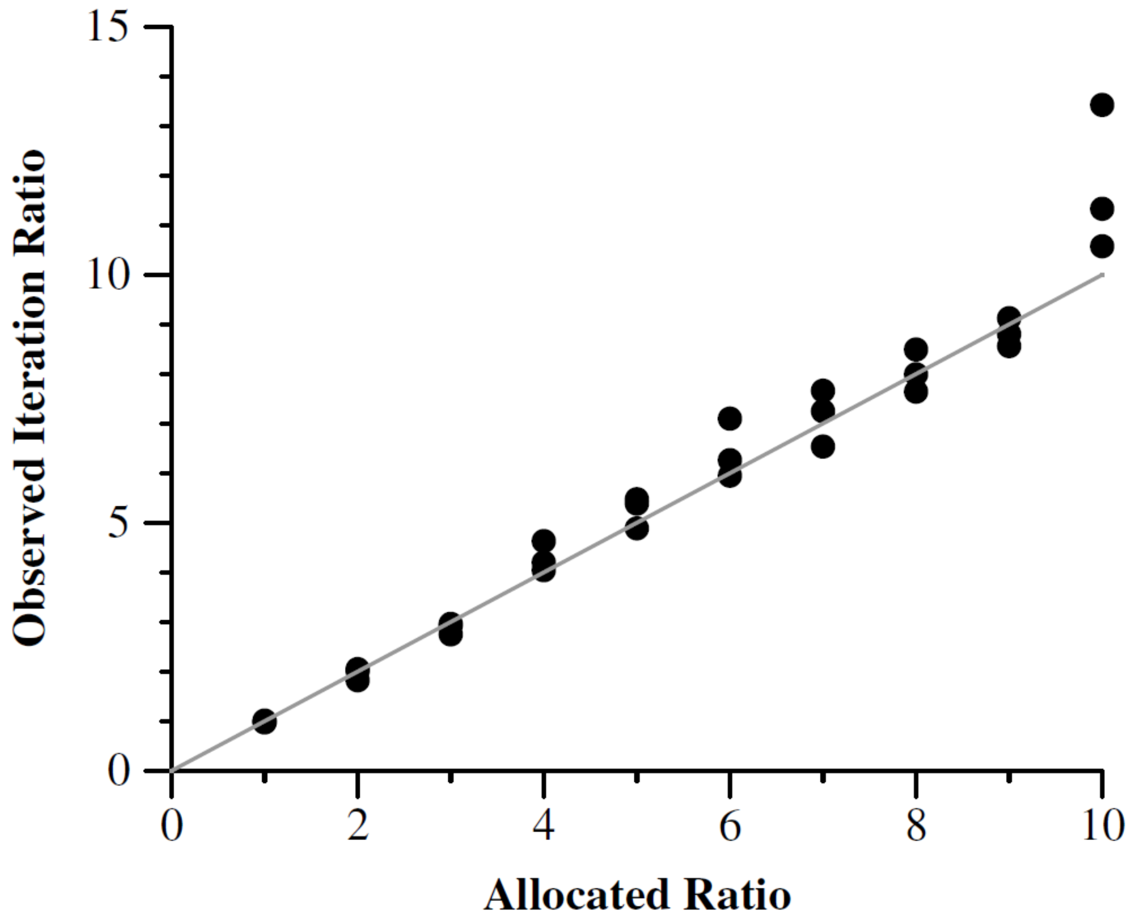
- The accuracy improves with $\sqrt{n}$
  - Need frequent lotteries
  - Mostly accurate, but short-term inaccuracies are possible
- No starvation
  - Any client with a non-zero number of tickets will eventually win a lottery
- Responsive
  - Any changes to relative ticket allocations are immediately reflected in the next lottery

Example: Fairness

- Two Dhrystone (CPU-intensive) benchmark tasks for 60 sec.

- The variance is greater for larger ratios:
  - 13.42 : 1 (for 10 : 1)

- Even larger ratios converge over longer time intervals:
  - 19.08 : 1 (for 20 : 1, for 3 min.)

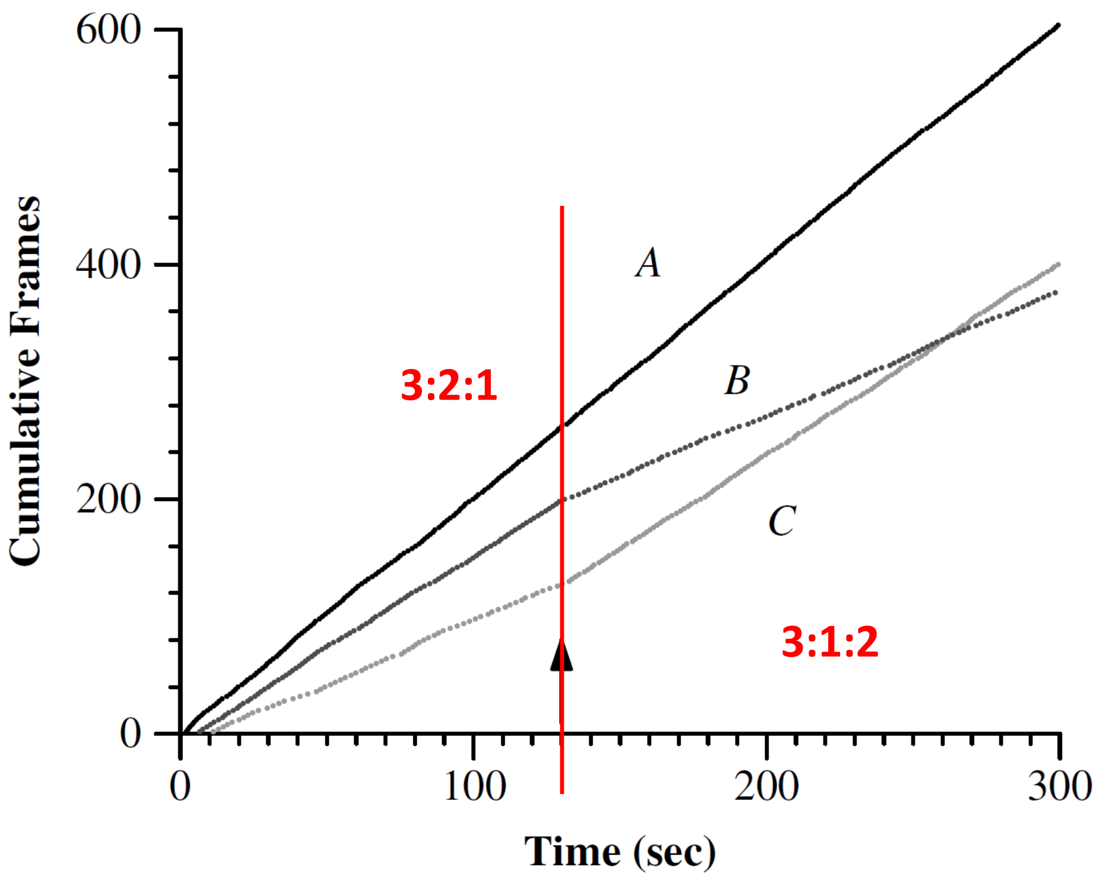


Example: Multimedia Applications

- Three mpeg_play video viewers

- Not exact
  - 1.92 : 1.50 : 1 (3 : 2 : 1)
  - 1.92 : 1 : 1.53 (3 : 1 : 2)

- Due to the round-robin processing of client requests by the single-threaded X11R5 server
  - 3.06 : 2.04 : 1 with `-no-display` option



Compensation Tickets

- What happens if a thread is I/O-bound and blocks before its quantum expires?

- If a thread consumes only a fraction f of the quantum, its tickets are inflated by $1/f$ until the next time you win
  - If A on average uses 1/5 of a quantum, its tickets will be inflated 5x and it will win 5 times as often and get its correct share overall

Ticket Transfer

- If you are blocked on someone else, give them your tickets
- Useful for client-server system
  - Server has no tickets of its own
  - Clients give their tickets to server threads during RPC
  - Server's priority is the sum of the priorities of all of its active clients
  - Server can use lottery scheduling to give preferential service to high-priority clients
  - Clients also have the ability to divide ticket transfers across multiple servers on which they may be waiting
- Avoid priority inversion problem

Ticket Inflation

- Make up your own tickets (print your own money)
- Only works among mutually trusting clients
  - *Why?*
- Presumably works best if inflation is temporary
- Allows clients to adjust their resource allocations without explicit communication
- Examples
  - Monte-Carlo algorithm: dynamically adjust the number of tickets as a function of its current relative error
  - Graphics-intensive programs: a large share to display a crude outline initially, and then a smaller share to compute details

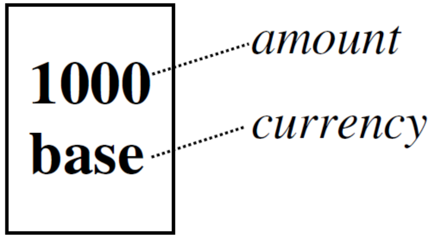
Ticket Currencies

- Express resource rights in units that are local to each group of mutually trusting clients

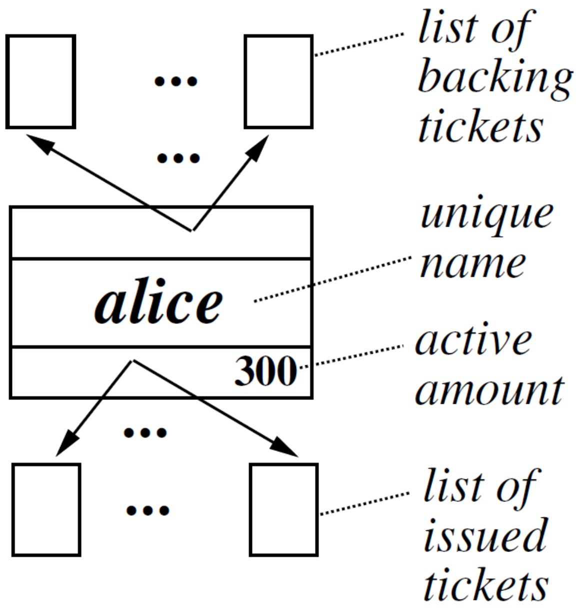
- A unique currency is used to denominate tickets within each trust boundary
  - Each currency is backed, or funded, by tickets that are denominated in more primitive currencies
  - The effects of inflation can be locally contained by maintaining an exchange rate

- Useful for flexible naming, sharing, and protecting resource rights

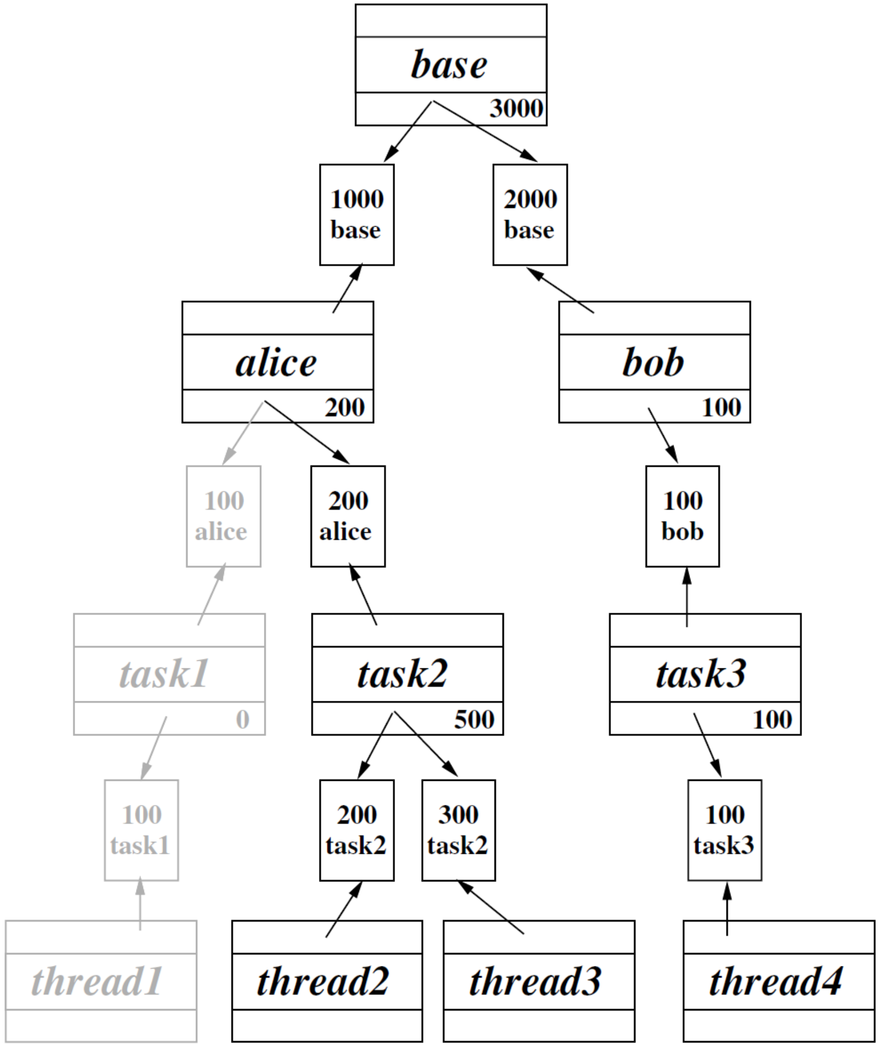
Ticket Currencies: Example



ticket

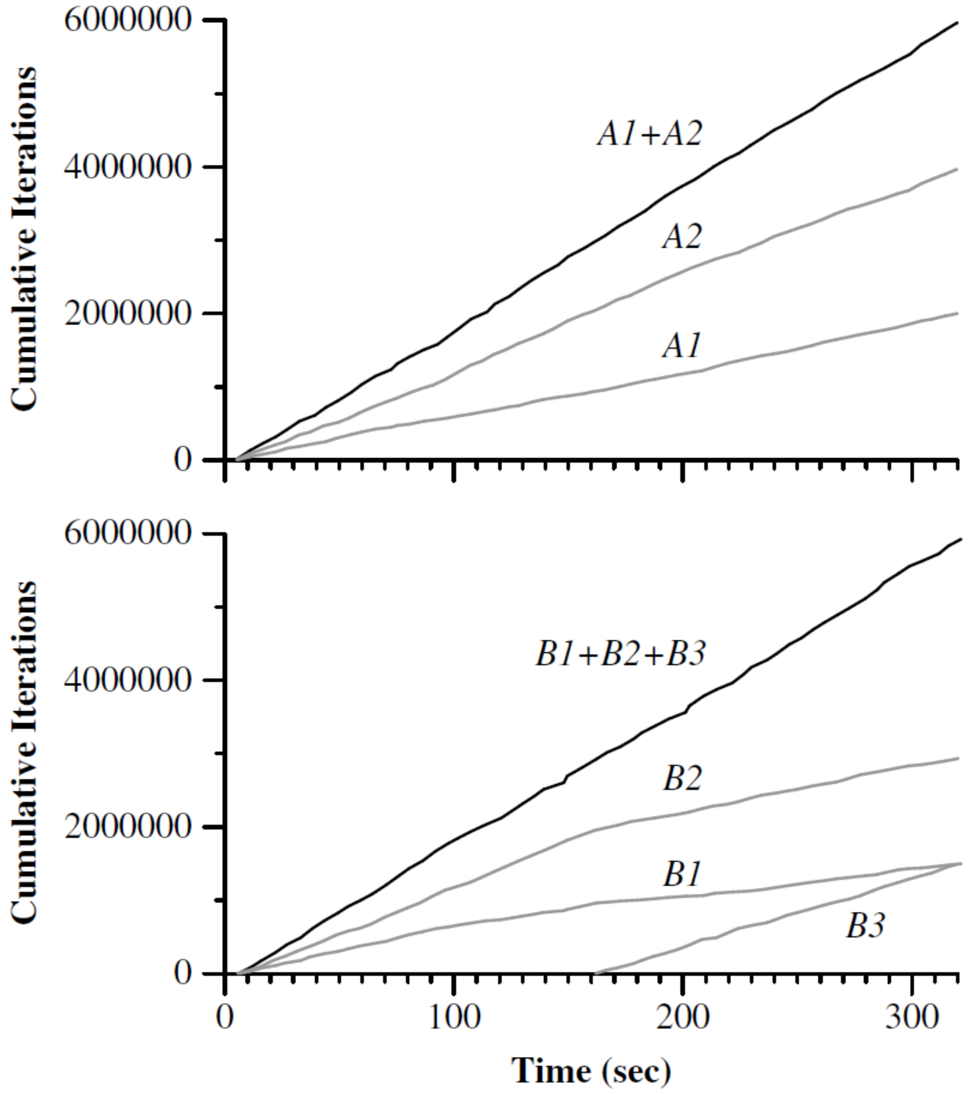


currency



Ticket Currencies: Load Insulation

- 5 Dhrystone tasks
- Two currencies A and B
  - Funded equally
- Task group A
  - A1 with 100.A
  - A2 with 200.A
- Task group B
  - B1 with 100.B
  - B2 with 200.B
  - Later, added B3 with 300.B

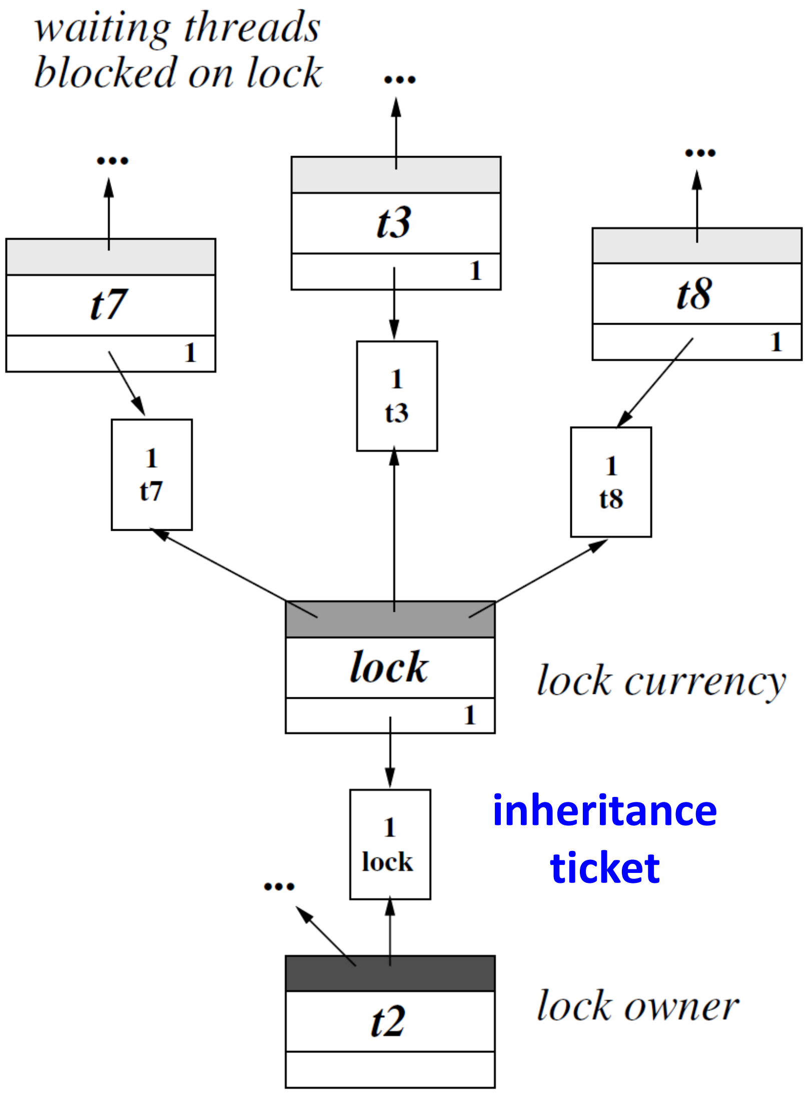


Ticket Currencies: Lock Funding

- A lottery-scheduled mutex has
  - Mutex currency
  - Inheritance ticket

- Waiting threads fund the mutex currency

- When done, mutex holder conducts a lottery to determine the next holder



Discussion

- ***What's good?***

- ***What's bad?***