Instructor Notes

Waldspurger, C.A. and Weihl, W.E.
Lottery Scheduling: Flexible Proportional-Share Resource Management
Proceedings of the First OSDI, Monterey CA, November 1994, pp. 1-11.

1. What is the motivation for needing a new scheduler? What are the goals of a lottery scheduler?

- Existing priority-based schedulers give poor control over relative computation rates
  - instead optimize performance of interactive vs. cpu-bound jobs (approx SJF w/ multi-level feedback Q)
  - difficult to adjust priorities to get fairness or fixed rates

Goals?

- give proportional-share resource management
- modularity (hierarchical)

2. What are the advantages of using lottery tickets to represent resource rights?

- abstract (no machine details)

- relative (fraction of resources → more if lightly loaded)

- uniform (can be used for diff resources – cpu, mem, disk)

3. How is a scheduling decision made? What is the expectation for which process will be scheduled? Does lottery scheduling need to do anything special to guard against starvation?

- Hold a lottery - winner gets scheduled

- probabilistically fair

p, prob of winning : $\frac{t}{T}$

Starvation?

- MLFQ - mechanism to prevent starvation (cludgy)

- if have ticket, eventually win, so nothing extra needed

4. How does one **implement** a lottery? (See Figure 1.) What are ways to optimize the search for the winner?

- Fast random number generator

  0 ... n-1 (n is # active tickets → jobs on ready queue)

- List clients, traverse, calc. ticket sum

  O(c) → # active clients

Opt? Sort by tickets

Put in tree O(lg c)

5. **Ticket currencies** enable mutually trusting clients to redistribute tickets in a modular fashion. In Figure 3, how many base tickets does thread2 have? Thread3? Thread4? If thread1 became active, how many base tickets would each thread have?

t2: $\frac{200}{500} \cdot \frac{200}{200} \cdot 1000 = 400$

t3: $\frac{300}{500} \cdot \frac{200}{200} \cdot 1000 = 600$

t4: $\frac{100}{100} \cdot \frac{100}{100} \cdot 2000 = 2000$

T1 active → 300 alice issued fix

no change to t4 base or relative

t1: $\frac{100}{100} \cdot \frac{100}{300} \cdot 1000 = 333$

t2: $\frac{200}{500} \cdot \frac{200}{300} \cdot 1000 = 267$

t3: $\frac{300}{500} \cdot \frac{200}{300} \cdot 1000 = 400$

1000 tickets

6. **Ticket inflation** involves a client escalating its resource rights by creating more lottery tickets. When is this useful?

- Only within trusting/cooperative clients (e.g. within a currency)

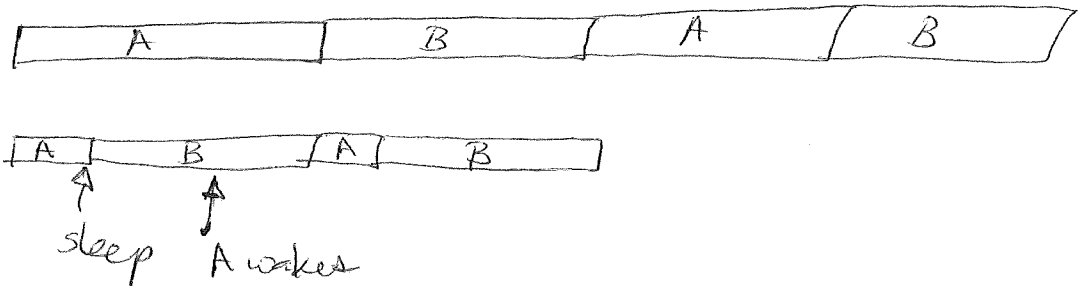
7. **Ticket transfers** are the explicit transfers of tickets from one client to another. Can you think of two examples where ticket transfers could be useful?

- Another process doing work on your behalf (server - RPC)

- Waiting for another process (holding a lock)

8. **Compensation tickets** are used to temporarily inflate tickets by 1/f when a process only uses a fraction f of its quantum. When does this not work as desired?

each has 50 tix



if A uses $\frac{1}{10}$ of quatum, give it 10·50 tix for next lottery

- more likely to win next lottery, but doesn't preempt B...

-Problem: Can't win lotteries not active for!

9. What do Figures 4 and 5 show? Is randomness a good quality for a scheduler?

- Probabilistically fair w/ variance

- Make deterministic scheduler (stride)

- No variance, but not conceptually intriguing!

10. What do Figures 6 and 7 and 9 show?

-Works well, can dynamically adjust tickets
7: Give tix to server on ~~your~~ client behalf
-9: Currencies insulate loads

11. What problem does Figure 8 reveal?

3:2:1 → 1.9 : 1.5 : 1

Other components don't use proportional share (RR in display)

12. Conclusions?

- Revived interest in scheduling
- Great match for hierarchies; extensible systems
- Good match for proportional shares in shared services
- Simple, cute w/ randomness

(but why not deterministic version?)

- Doesn't handle I/O well

(lots harder because of state - disk head)