Jae-Hoon Shim
Min-Wook Kim

Lottery Scheduling:
Flexible Proportional-Share
Resource Management

Spring 2019

4190.568: Advanced Operating Systems

Scheduling Issues

- Context
  - Multiplex scarce resources
  - Concurrently executing clients
  - Service request of varying importance

- Quality of Service
  - Long-running computation
  - Interactive computation

Conventional Solutions

- Priority-based
  - Absolute priority
  - Dynamic priority adjustment

- Fair share & Microeconomic
  - Client간 자원 사용량의 공정한 share 보장
  - 시간에 대한 사용량 추적 필요
  - 여러 client간 재화를 주고받으며 경매를 통해 자원 할당

Problems

- Priority-based
  - Starvation problem
  - Dynamic priority adjustment are ad-hoc
  - Resource rights don’t vary smoothly
- Fair share & Microeconomic
  - Assumptions and overheads limits them to coarse control
  - Cannot support interactive systems

Simplicity

Modularity

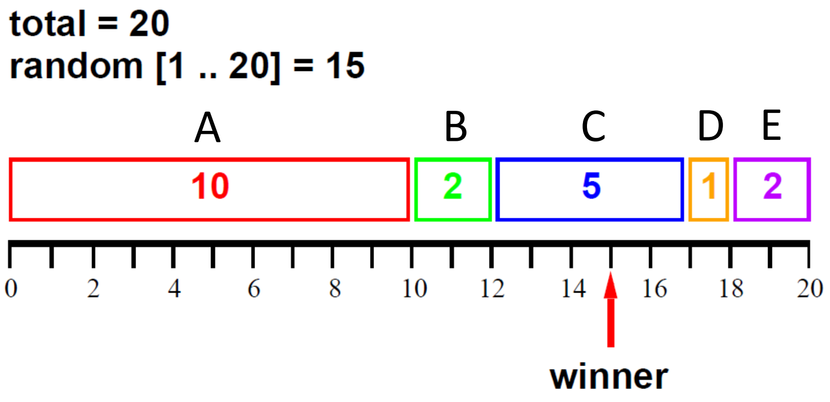
Flexibility

Idea: Abstraction & Randomization

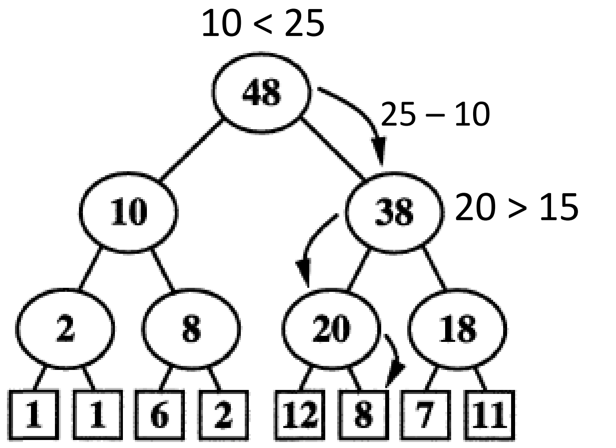
- Lottery tickets == Resource rights
- Randomization
- Simplicity

Lottery Ticket

- Encapsulates resource rights
- Only quantity of tickets matters
- Ticket schemes are homogeneous
  - Quantify resource rights independently of machine details



List-based lottery (winner = 15)



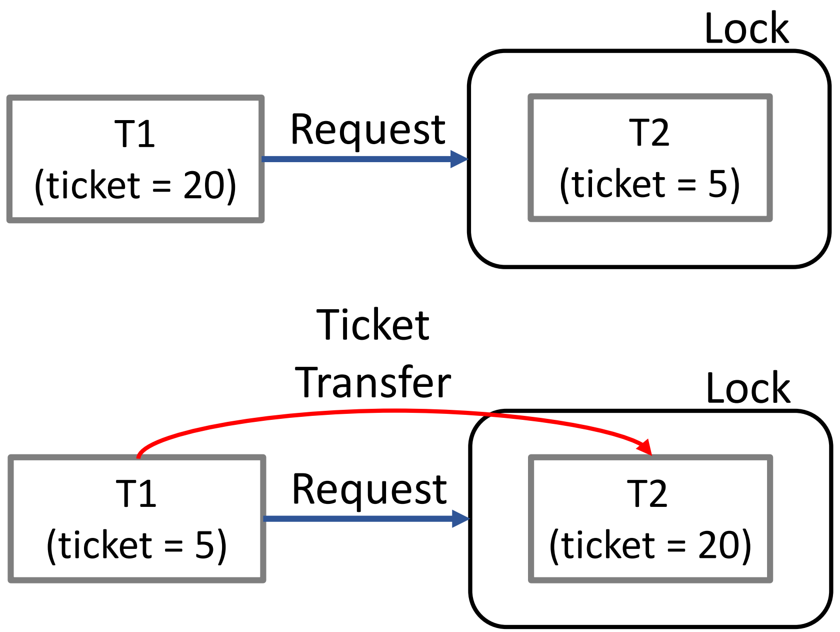
Tree-based lottery (winner = 25)

Lottery Scheduling Advantages

- Proportional-Share Fairness
  - Throughput proportional to ticket allocation
  - Response time inversely proportional to ticket allocation
- Modularity & Simplicity
  - Supports dynamic environments
  - Easily understood behavior
- Flexibility
  - Immediately adapts to changes
  - Direct control over service rates

Ticket Transfer

- Transfer tickets to another client
- Useful when client blocks due to some dependency
- Prevent priority inversion



Ticket Inflation

- Alternative way to escalate resource right
  - Client creates more tickets
  - No explicit communication

- Single client can easily monopolize a resource

- Convenient among mutually trusting clients

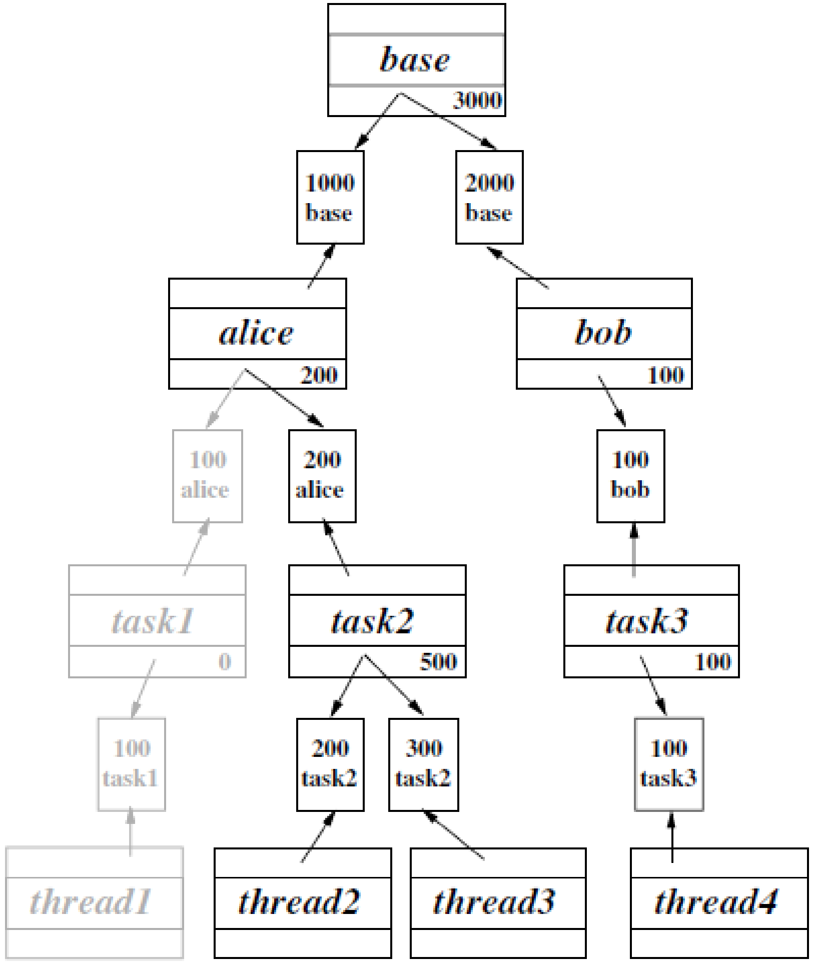
Ticket Currency

- Tickets are Denominated in Currencies

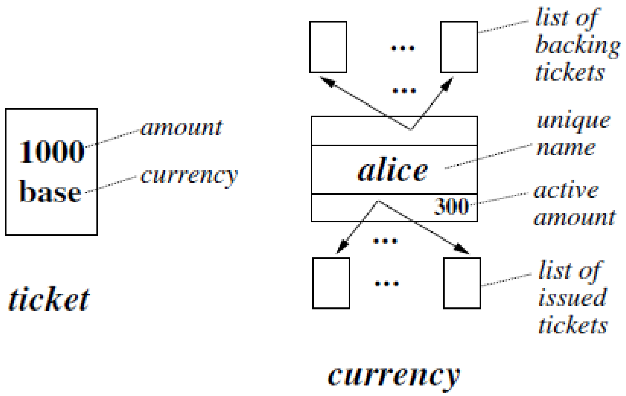
- Modular Resource Management
  - Locally contain effects of inflation
  - Isolate loads across logical trust boundaries

- Powerful Abstraction
  - Name, share, and protect resource rights
  - Flexibly group or isolate users and tasks

Ticket Currency Implementation



Example Currency Graph



Initial

Thread1 : 333 base units

Thread2 : 266 base units

Thread3 : 401 base units

Thread4 : 2000 base units

Total : 3000 base units

T1 removed from run Q →

Final

Thread1 : 0 base units

Thread2 : 400 base units

Thread3 : 600 base units

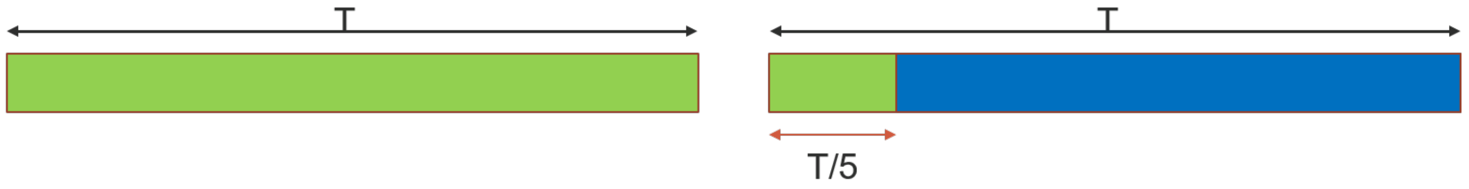
Thread4 : 2000 base units

Total : 3000 base units

Compensation Tickets

- Granted to client which consumes less fraction *f* of its allocated time quantum
- Inflates its value by 1/f until client starts its next quantum
- Consistent with proportional sharing
  - ex) Permits I/O-bound tasks to start quickly

CPU Bound Process | I/O Bound Process



CPU Bound Process

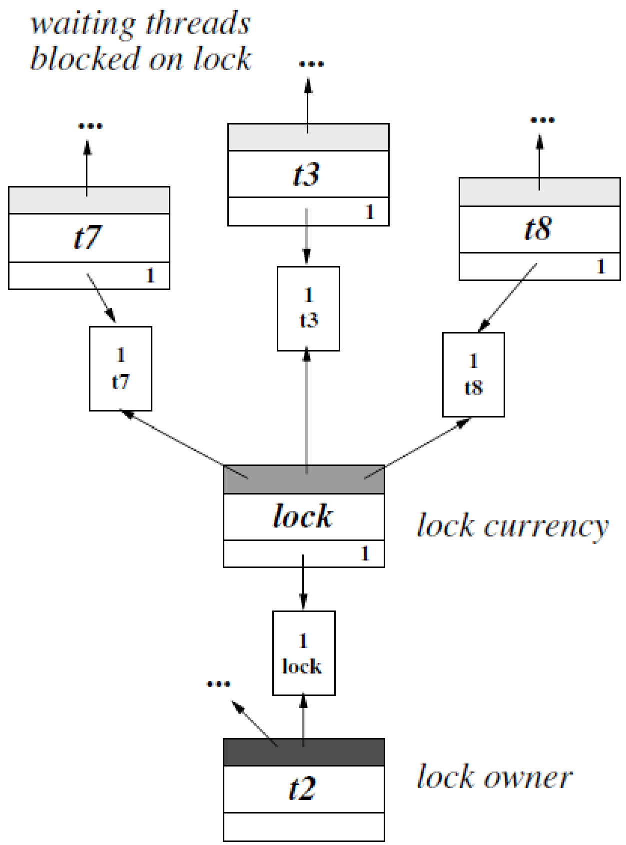
- Use whole time slice
- Amount of ticket
  10 → 10

I/O Bound Process

- Use a part of time slice
- f = 1/5
- Amount of ticket
  10 → 50 (= 10 * 1/f)

Managing Diverse Resources

- Synchronization resources
  - Can be used to control threads competing for lock access
  - Avoids Priority Inversion
- Waiting to acquire
  - Waiters transfer funding to lock currency
- Release
  - Hold lottery among waiters
  - New winner inherits the ticket



Benefits Gained

- No starvation & probabilistically fair

- Simple concept & easy to implement

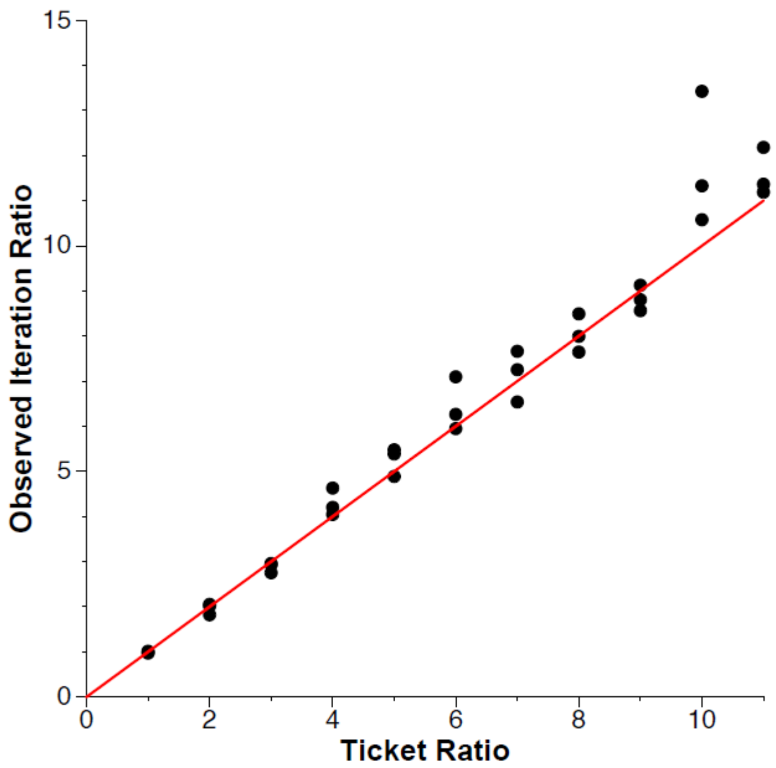
- Flexible control

- Provides support for modular resource management
  - Can be generalized to manage many diverse resources

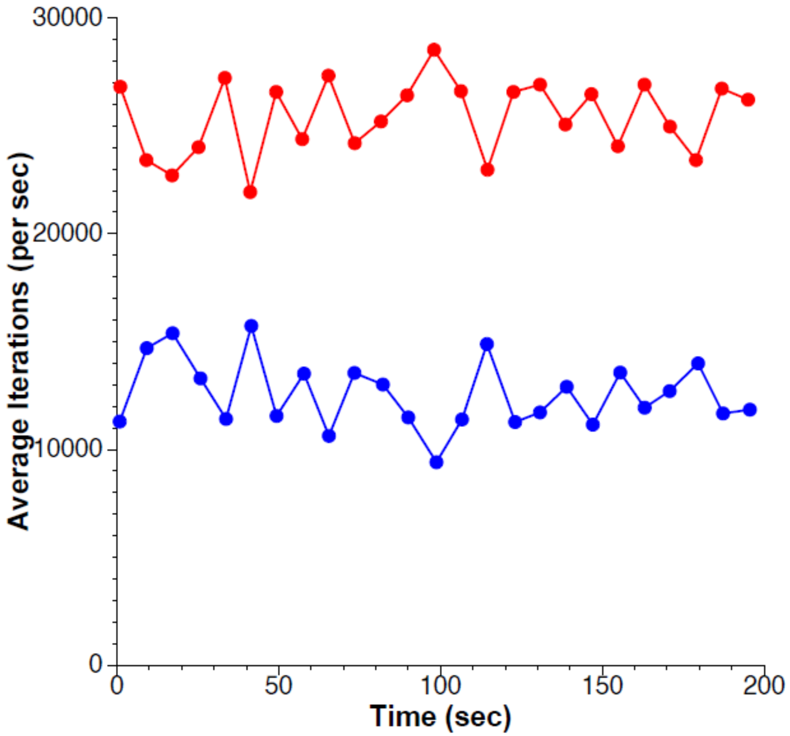
Kernel Implementation

- Modified Mach 3.0 microkernel
  - 25 MHz DECStation 5000/125
  - 100 millisecond quantum
- Support ticket transfers, inflation, currencies, and compensation tickets
- List-based lottery

Experiment (Fairness)



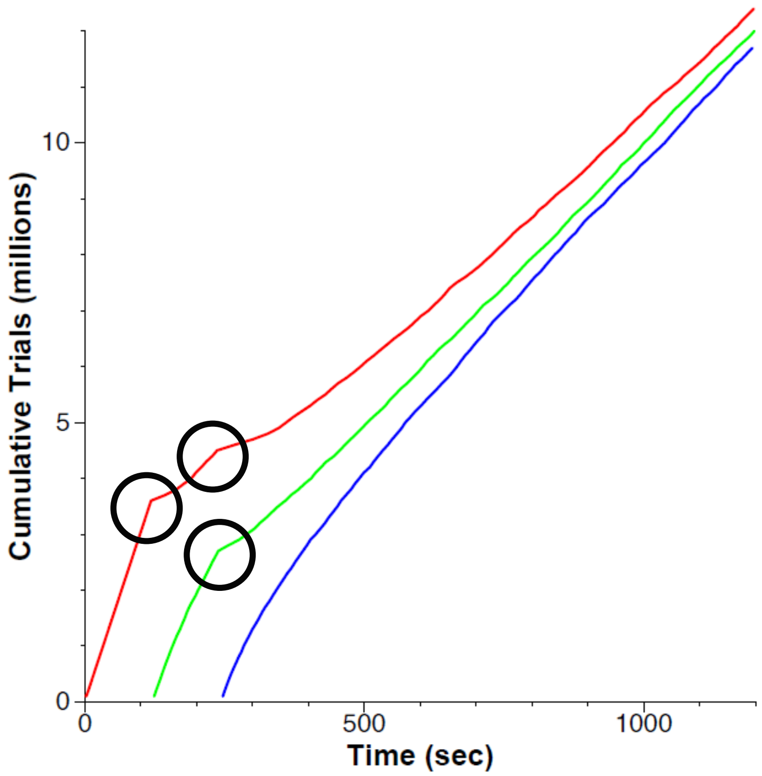
Relative Rate Accuracy
(3 runs / 60 seconds)



Fairness Over Time

Experiment (Flexible Control)

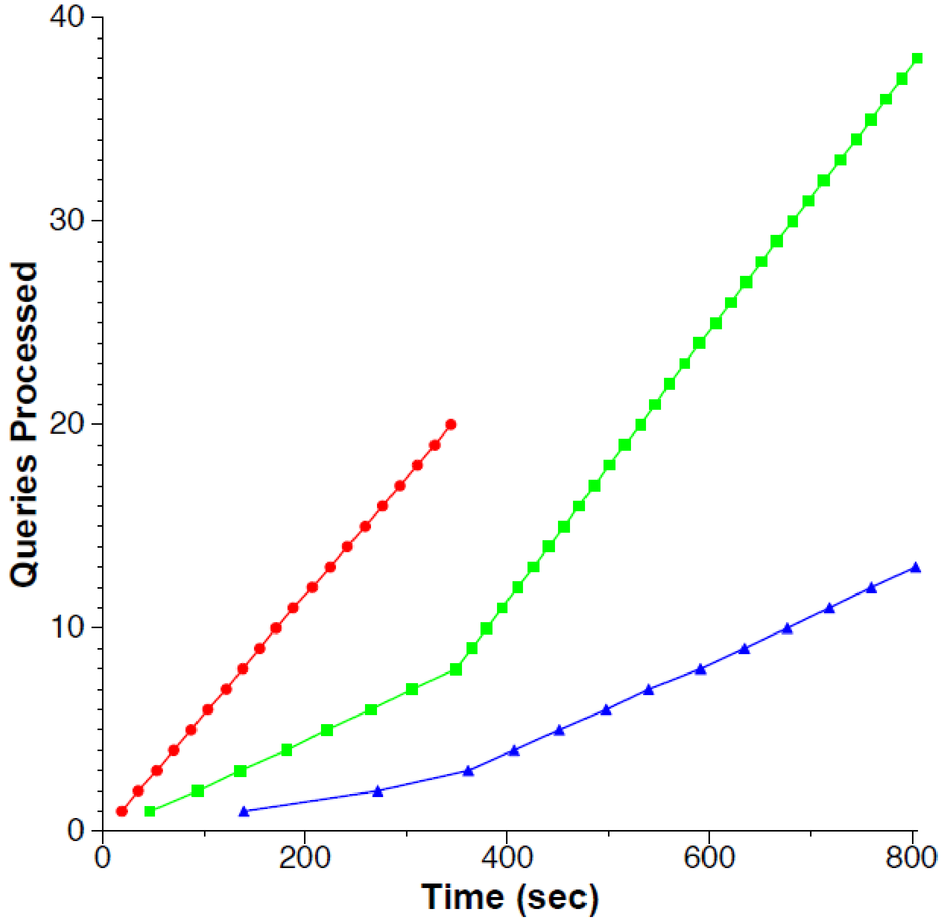
- Three Monte-Carlo tasks
  - Ticket inflation
  - Funding based on relative error



Monte-Carlo Execution Rates

Experiment (Client-Server Computation)

- Multithreaded DB Server
  - server has no tickets of its own
- 8 : 3 : 1 allocation
- Ticket transfers

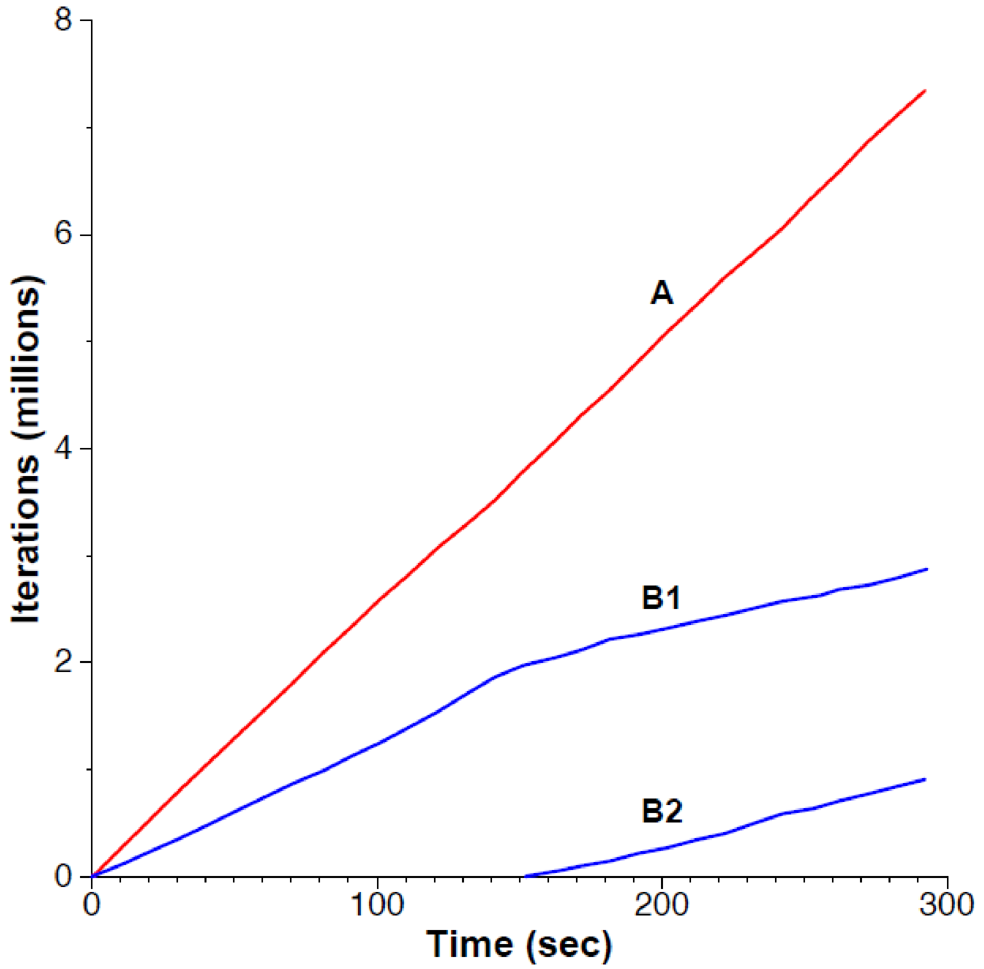


Query Processing Rates

Experiment (Load Insulation)

- Currencies A, B = 2 : 1

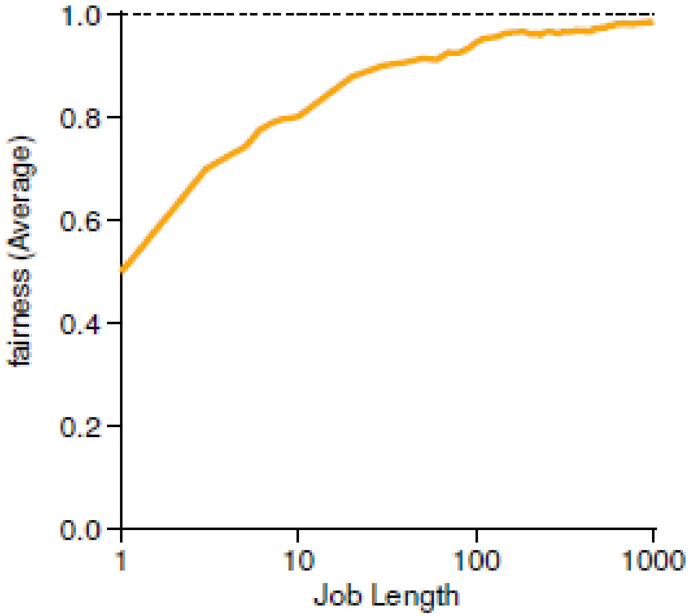
- Task A : funding 100.A
- Task B1 : funding 100.B
- Task B2 : funding 100.B



Currencies Insulate Loads

Limitation (1)

- Short time interval 에서 적절한 제어 불확실
  - 『Charge-Based Proportional Scheduling』, U.Maheshwari
    - "randomization does not afford sufficient control on the execution rates over short periods of time such as 10 timeslices."



Lottery Fairness Study

- Ticket-assignment Problem
  - How many tickets should you assign to each application?

Limitation (2)

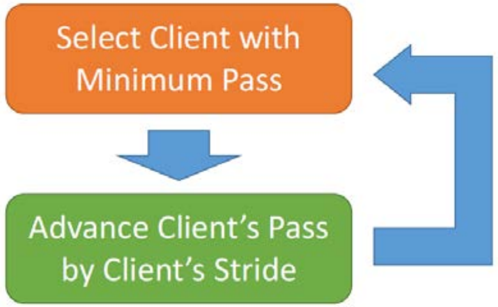
- Not suitable for interactive workloads
  - Succeeds at scheduling processes which never voluntarily relinquish the CPU in proportion to the number of tickets that they hold

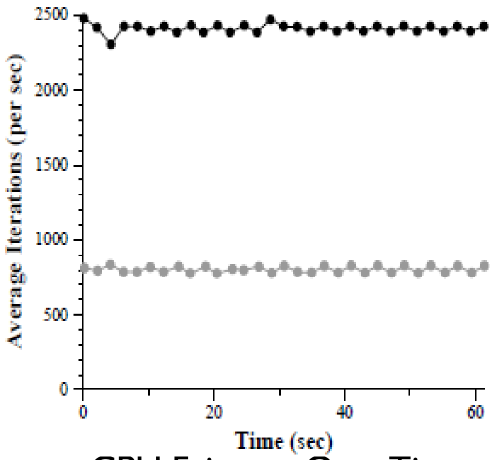
- Interactive jobs
  - Spend most of their time idle
  - Concerned with how responsive they are to user input

- Lottery scheduling does not distinguish between CPU-bound and interactive jobs
  - Often fails to schedule interactive jobs first

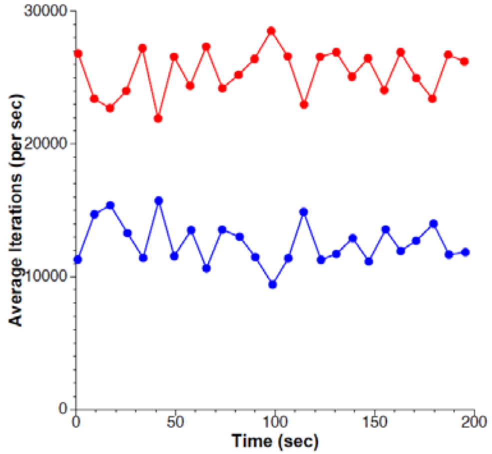
Subsequent Work (1)

- C. A. Waldspurger and W. E. Weihl. “Stride Scheduling: Deterministic Proportional-Share Resource Management”, 1999.
  - Why not deterministic?
  - Deterministic fair-share scheduler
  - Stride (stride1/tickets) : interval between selection
  - Pass (pass += stride) : Virtual index of next selection



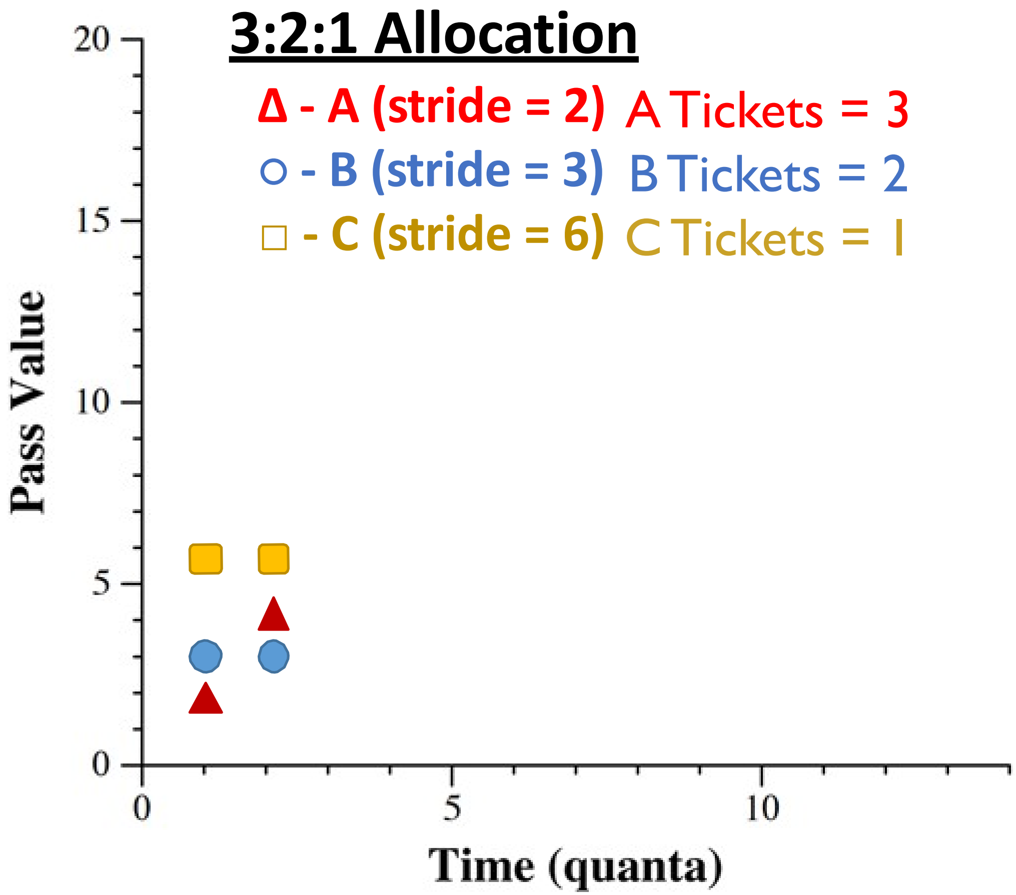


CPU Fairness Over Time



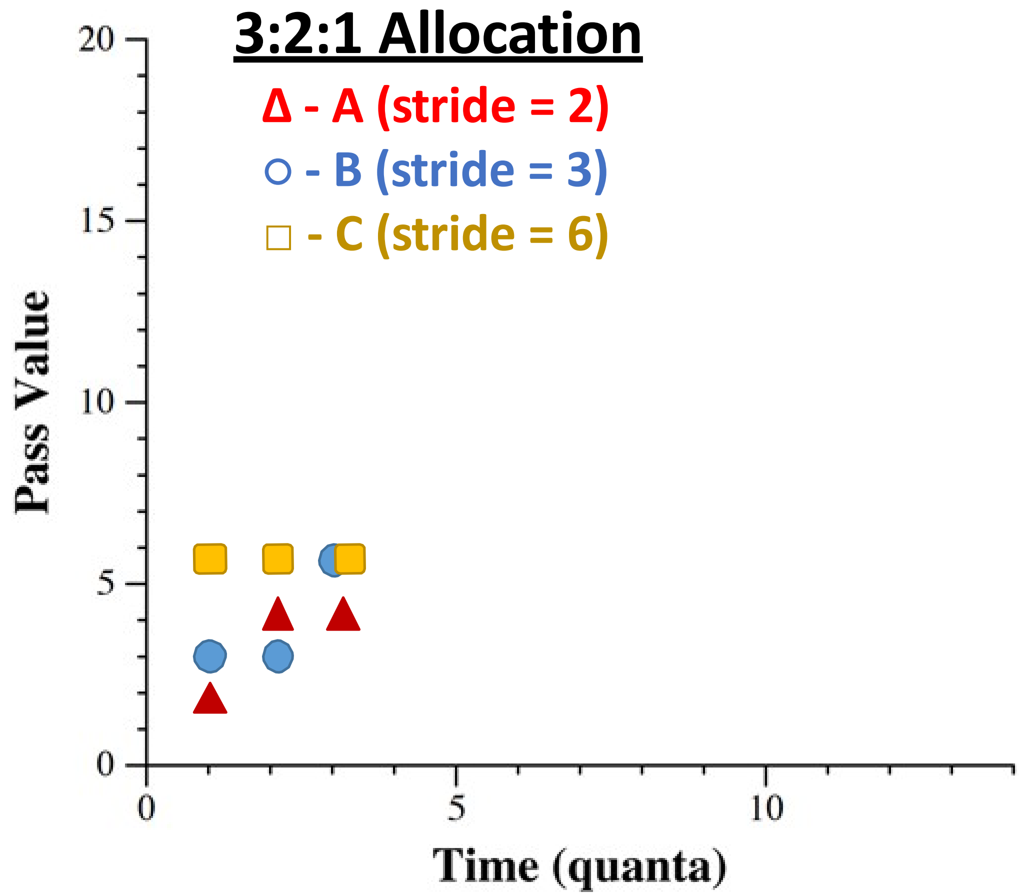
Fairness Over Time

Stride Scheduling - Example



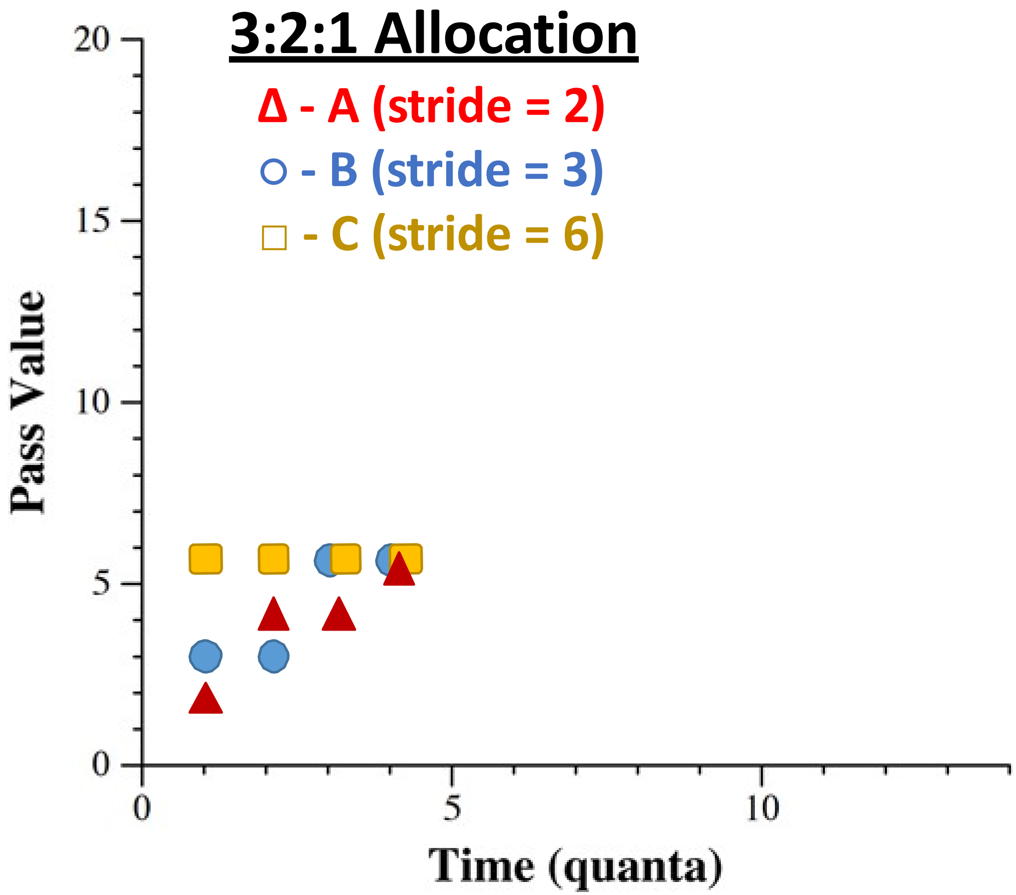
	▲	●	■
Time 1:	2	3	6
	+2		
Time 2:	4	3	6

Stride Scheduling - Example



	▲	●	■
Time 1:	2	3	6
	+2		
Time 2:	4	3	6
		+3	
Time 3:	4	6	6

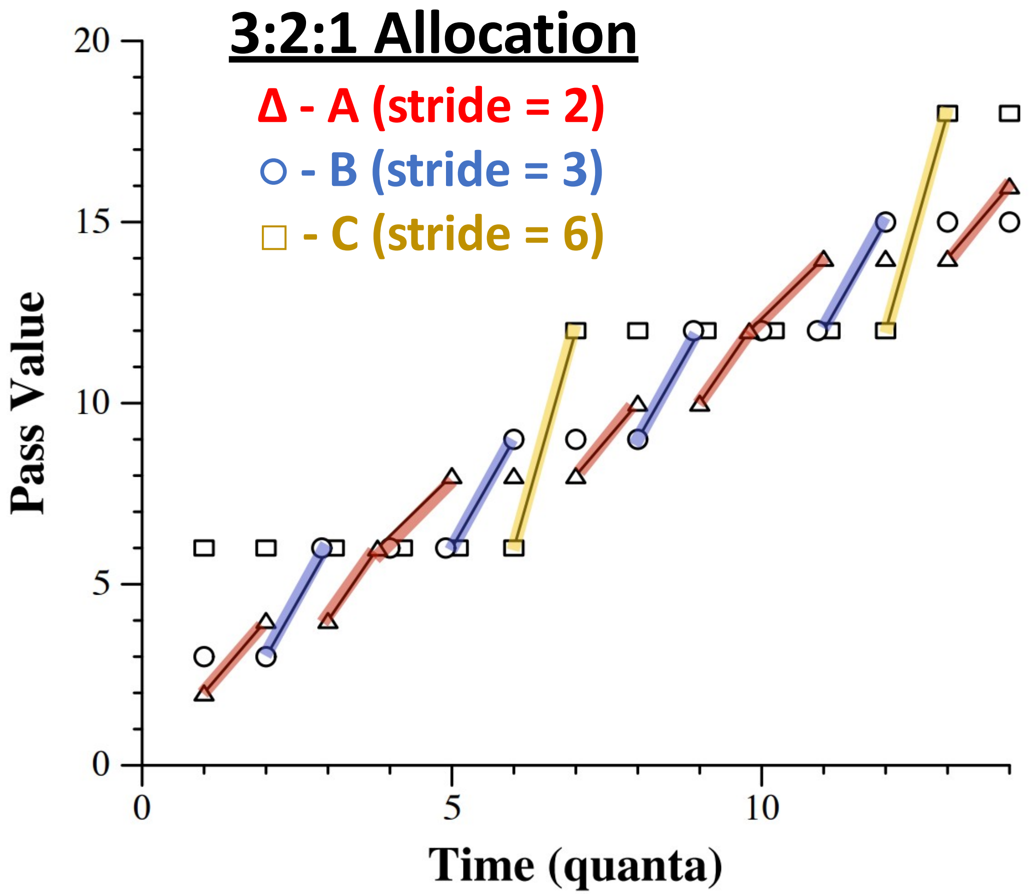
Stride Scheduling - Example



	▲	●	■
Time 1:	2	3	6
	+2		
Time 2:	4	3	6
		+3	
Time 3:	4	6	6
	+2		
Time 4:	6	6	6

Stride Scheduling - Example

Real Allocation = 7 : 4 : 2



	▲	●	■
Time 1:	2	3	6
	+2		
Time 2:	4	3	6
		+3	
Time 3:	4	6	6
	+2		
Time 4:	6	6	6

⋮

Subsequent Work (2)

- D. Petrou et al. "Implementing Lottery Scheduling: Matching the Specializations in Traditional Schedulers", 1999.
  - Extends lottery scheduling to provide the performance assurances present in traditional non-real time process schedulers
- C. A. Waldspurger. "Memory Resource Management in VMWare ESX Server", 2002.
  - Works well in environments with well-defined allocation of resource & fairness is important
  - Allocation algorithms extended from proportional-share allocation of space-shared resources
- A. Fox et al. "Cluster-based scalable network services", 1997.
  - Load balancing manager
  - use lottery scheduling to select a distiller for each request