Lottery and Stride Scheduling

Flexible Proportional-Share Resource Management

Carl A. Waldspurger

Parallel Software Group
MIT Laboratory for Computer Science

August 25, 1995

Overview

- Context
- Framework
- Mechanisms
- Prototypes
- Diverse Resources
- Conclusions

Problem

- **Environment**
  - multiplex scarce resources
  - concurrently executing clients
  - service requests of varying importance
- **Goals**
  - manage computation rates dynamically
  - enable flexible application-level policies
  - promote software engineering principles

Related Work

- **Priority-Based Scheduling**
  - operating systems
  - real-time systems
- **Share-Based Scheduling**
  - fair-share
  - proportional-share
  - microeconomic
- **Rate-Based Network Flow Control**
  - virtual clock, WFQ
  - AN2 switch

Contributions

- **New Framework**
  - simple, powerful abstractions
  - modular resource management
- **Novel Mechanisms**
  - randomized and deterministic algorithms
  - precise control over service rates
- **Resource-Specific Techniques**
  - proportional-share control
  - locks, memory, disk I/O

Resource Management Framework

- **Simple**
  - direct control over service rates
  - resource rights aggregate and vary smoothly
- **Modular**
  - powerful abstraction mechanism
  - insulate concurrent modules
- **Flexible**
  - can express sophisticated policies
  - adapts to dynamic changes
  - general-purpose, scalable

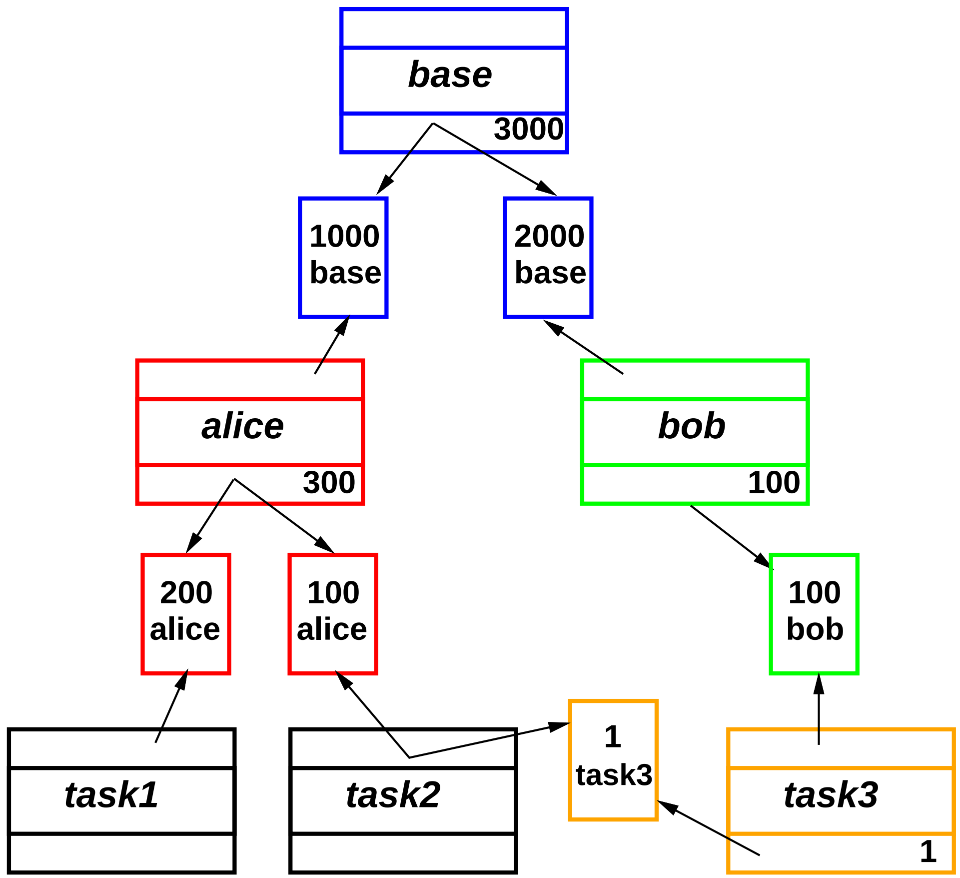
Framework Abstractions

- **Tickets**
  - first-class objects
  - encapsulate resource rights
  - proportional throughput
  - inversely proportional response time
- **Currencies**
  - modular abstraction mechanism
  - name, share, protect sets of tickets
  - flexibly group or isolate sets of clients

Dynamic Management Techniques

- **Ticket Transfers**
  - explicit transfer between clients
  - useful when client blocks while waiting
  - *example:* synchronous IPC
- **Ticket Inflation and Deflation**
  - clients create and destroy tickets
  - effects locally contained by currencies
  - *example:* progress-based allocation

Example Currency Graph



- **Computing Values**
  - currency:
    sum value of
    backing tickets
  - ticket:
    compute share of
    currency value
- **Example**
  - task2 funding in
    base units?
  - $\frac{100}{300}1000 + \frac{1}{1}\frac{100}{100}2000$
  - 2333 base units

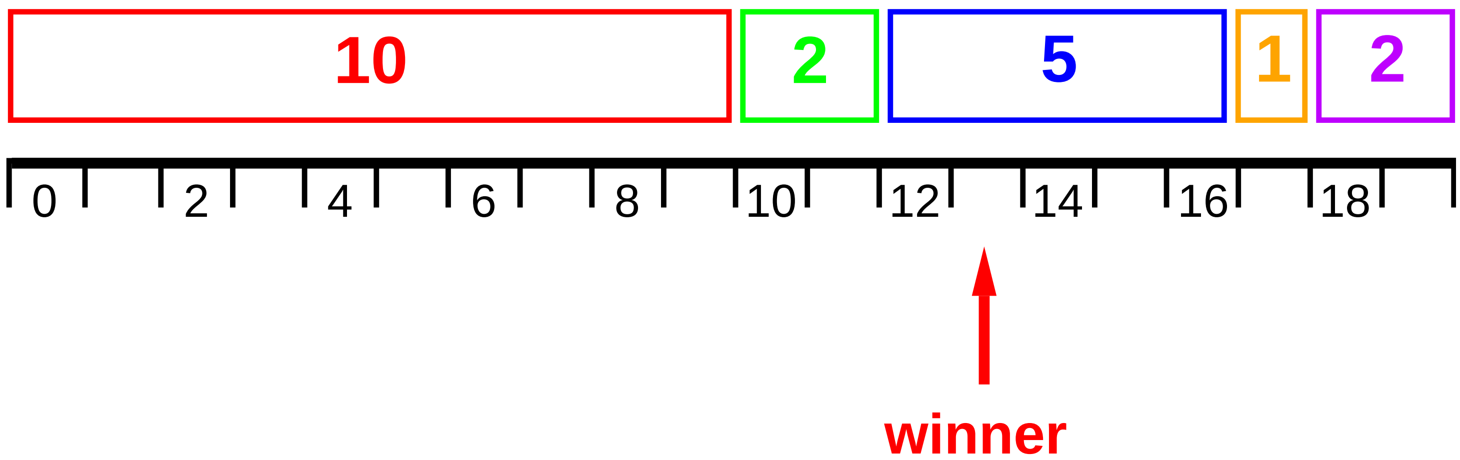
Proportional-Share Mechanisms

- **Randomized**
  - lottery
  - multi-winner lottery
- **Deterministic**
  - stride
  - hierarchical stride
- **Evaluation Criteria**
  - throughput accuracy
  - response-time variability
  - algorithmic complexity

Lottery Scheduling Example

total = 20

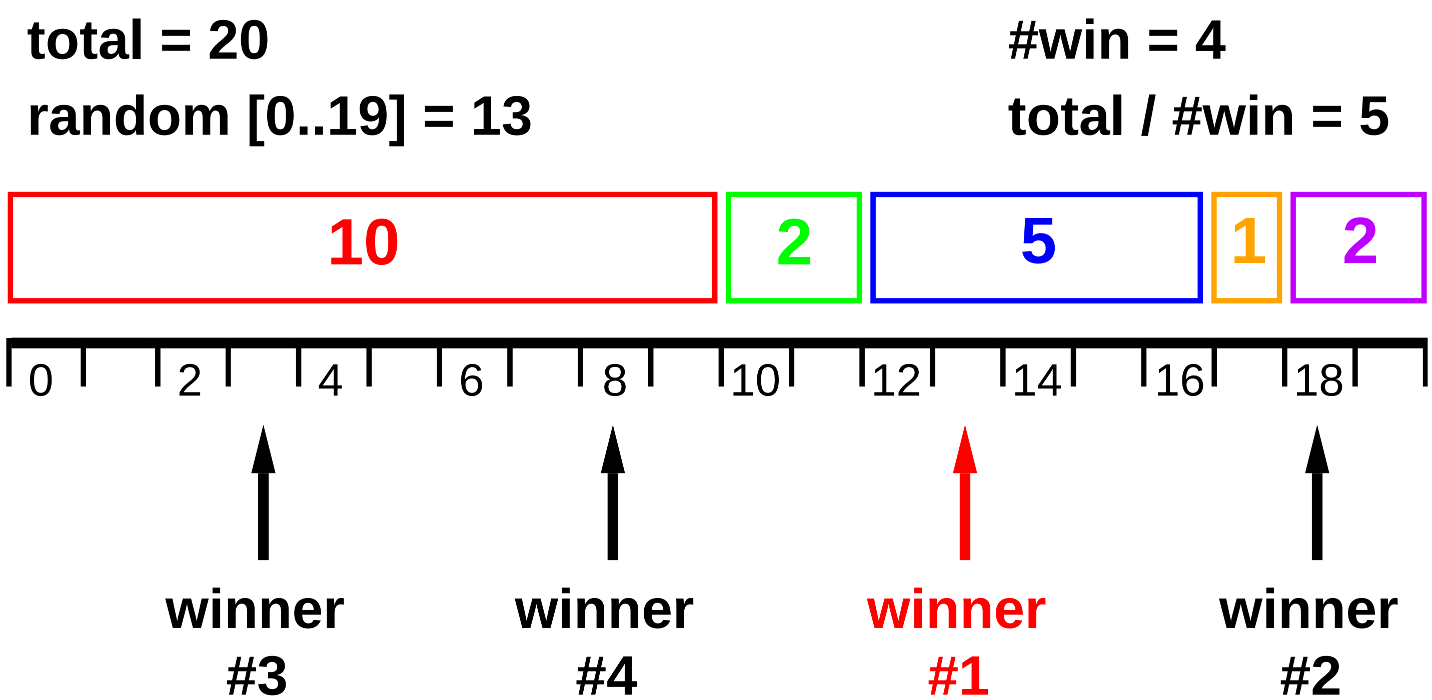
random [0..19] = 13



Lottery Scheduling Analysis

- **Strengths**
  - simple, stateless algorithm
  - supports dynamic operations
  - randomization prevents cheating
- **Weaknesses**
  - guarantees are probabilistic
  - poor short-term accuracy: $O(\sqrt{n_a})$ absolute error
  - high response-time variability: σ/μ = $\sqrt{1-p}$

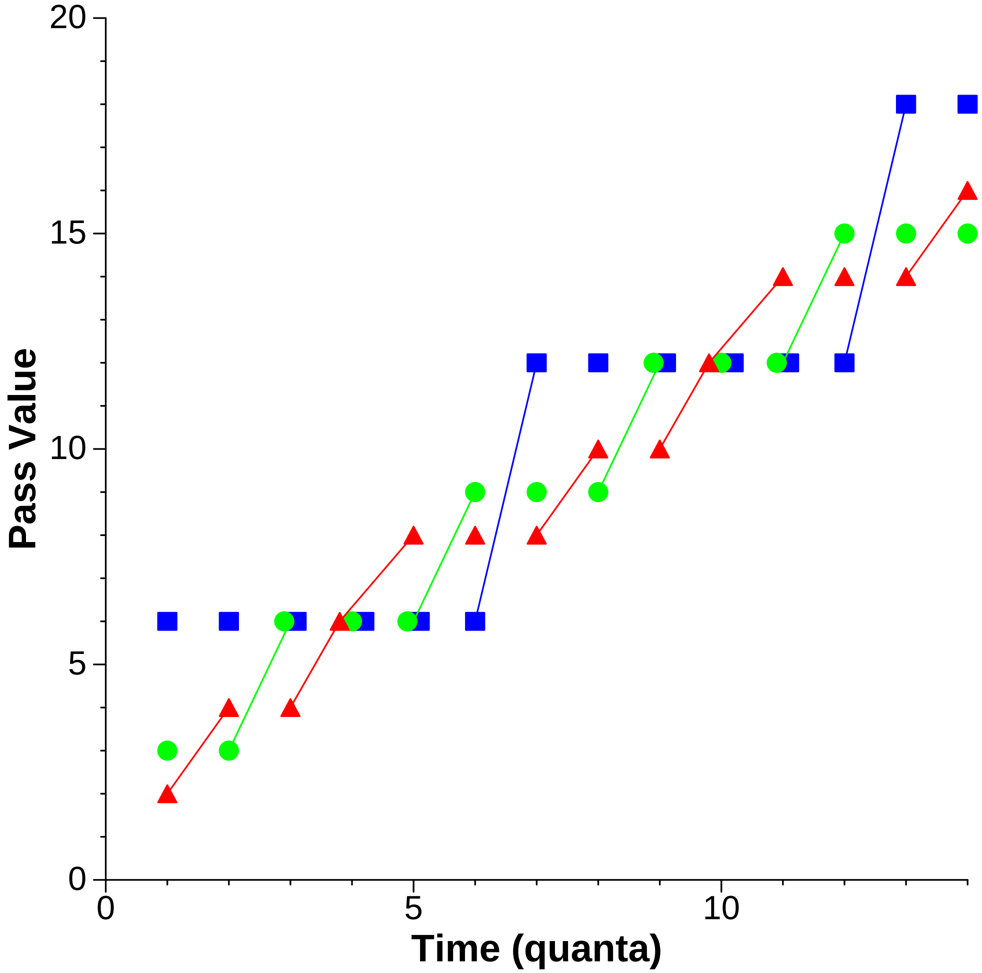
Multi-Winner Lottery Example



Multi-Winner Lottery Analysis

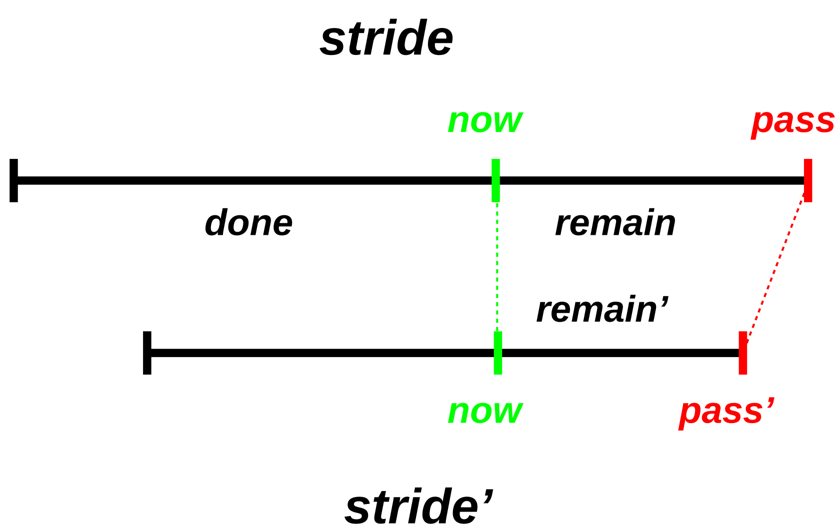
- **Strengths**
  - improves accuracy for large clients
  - guarantees $\lfloor n_w \frac{t}{T} \rfloor$ quanta per superquantum
  - bounds worst-case response time
  - improves list-based efficiency
- **Weaknesses**
  - probabilistic guarantees for small clients
  - dynamic operations terminate superquantum

Stride Scheduling Example



- **3 : 2 : 1 allocation**
- **Initialization**
  - stride = $\frac{\text{stride}_1}{\text{tickets}}$
  - pass = stride
  - stride_1 = 6
    strides: 2, 3, 6
- **Allocation**
  - choose client C
    with minimum pass
  - C.pass += C.stride

Dynamic Stride Allocation Change

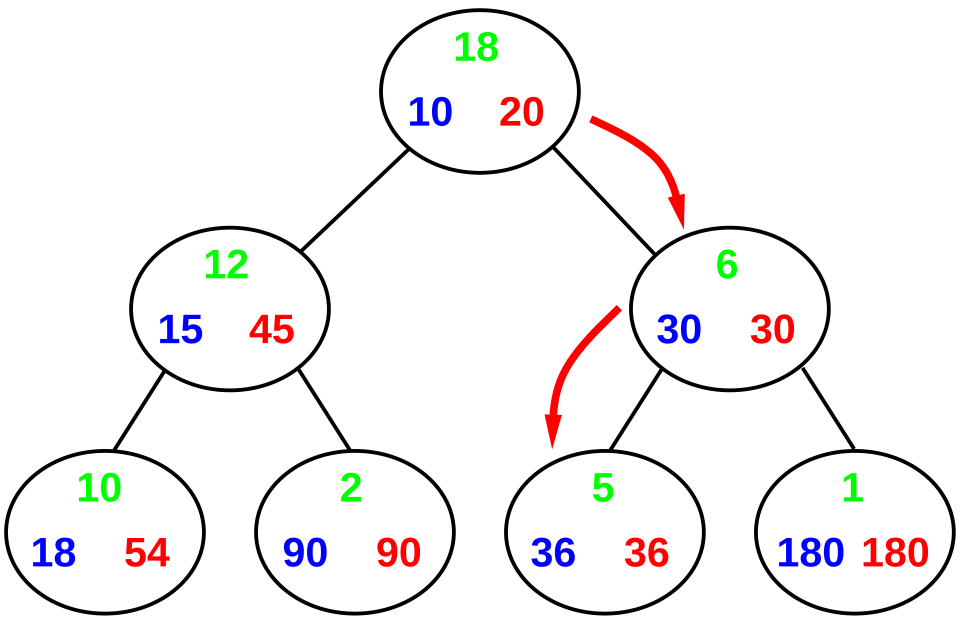


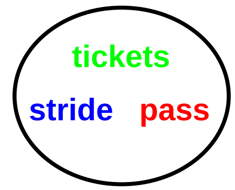
- **Allocation Change**
  - tickets $\rightarrow$ tickets$'$
  - stride$'$ = $\frac{\text{stride}_1}{\text{tickets}'}$
  - remain$'$ = $\frac{\text{stride}'}{\text{stride}}$ remain
  - pass$'$ = now + remain$'$
- **no updates needed for other clients**

Stride Scheduling Analysis

- **Strengths**
  - strong deterministic guarantees
  - throughput error independent of n_a
  - maximum relative error is one quantum
- **Weaknesses**
  - $O(n_c)$ absolute error
  - poor behavior for skewed ticket allocations

Hierarchical Stride Example



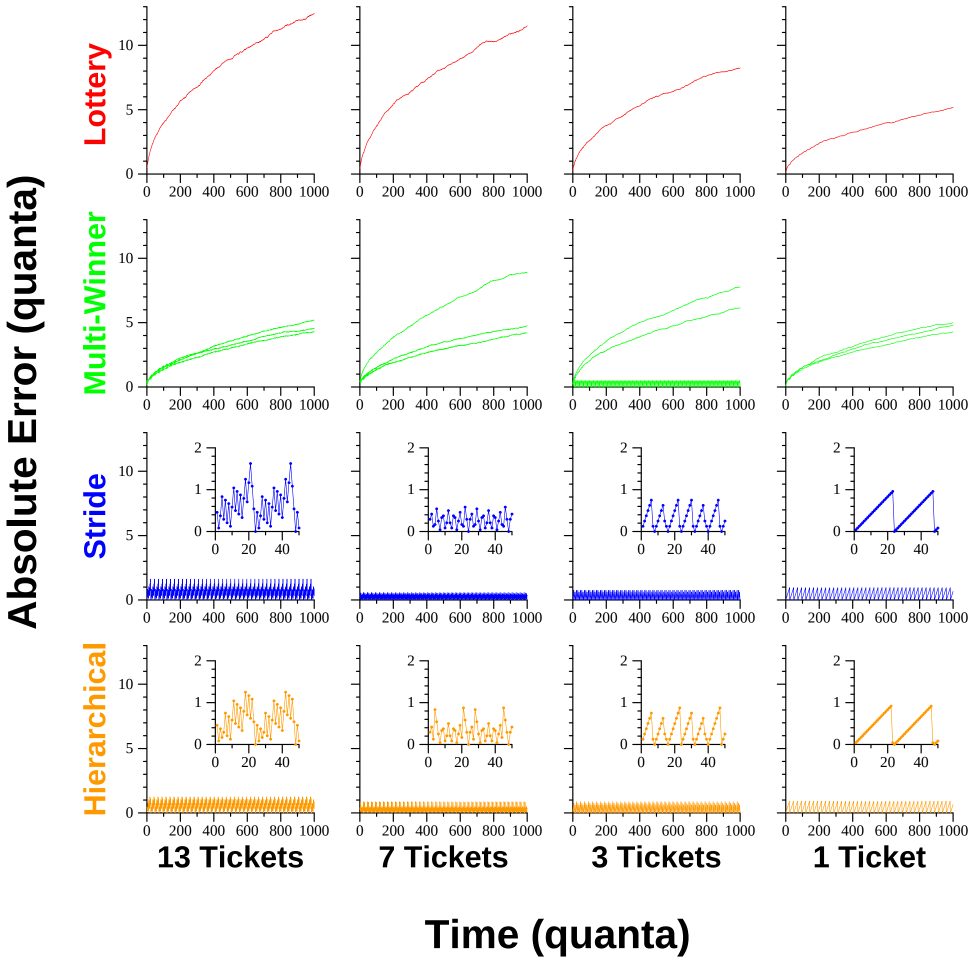
- 10 : 2 : 5 : 1 Ratio
- 

Node
- **Initialization**
  - $\text{stride} = \frac{\text{stride}_1}{\text{tickets}}$
  - pass = stride
  - $\text{stride}_1 = 180$
- **Allocation**
  - follow child C with smaller pass value
  - C.pass += C.stride

Hierarchical Stride Analysis

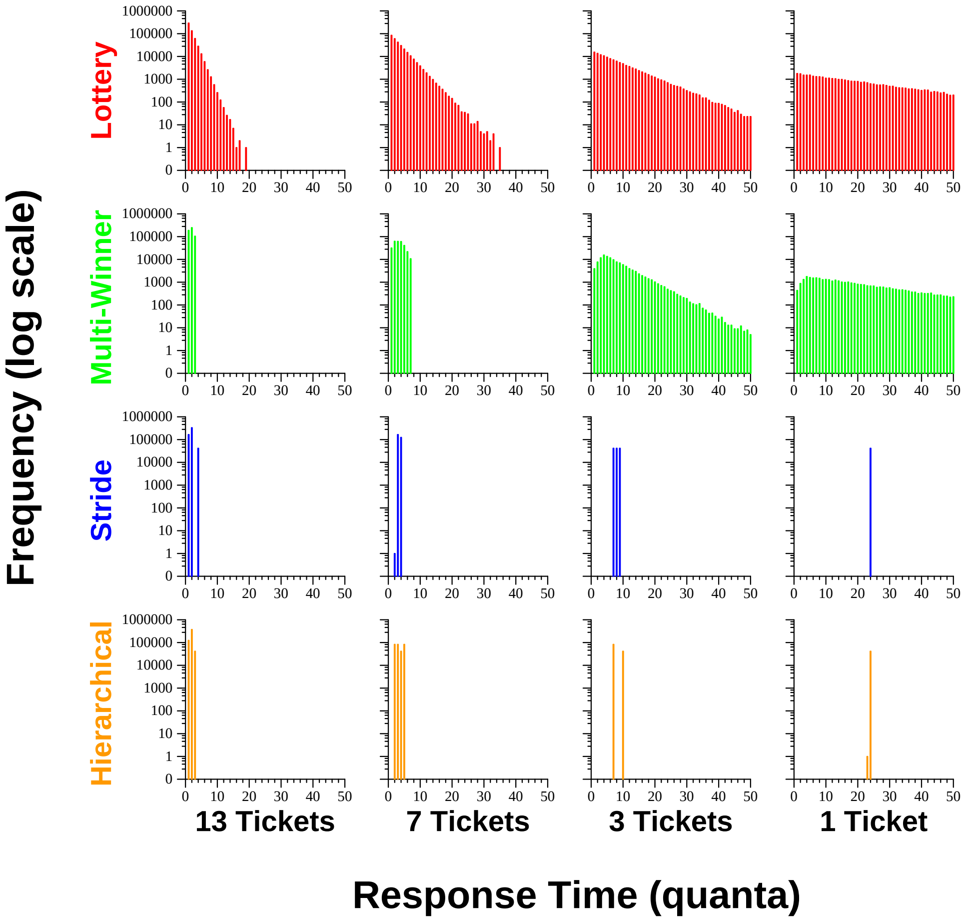
- **Strengths**
  - $O(\lg n_c)$ absolute error
  - reduces worst-case response-time variability
  - avoids worst-case stride scheduling behavior
- **Weaknesses**
  - can increase response-time variability
  - actual error can exceed stride scheduling error
  - complex dynamic operations

Throughput Accuracy Comparison



- Static Allocation
- 13 : 7 : 3 : 1 Ratio
- Mechanisms
  - lottery
  - multi-winner (2,4,8)
  - stride
  - hierarchical

Response-Time Comparison

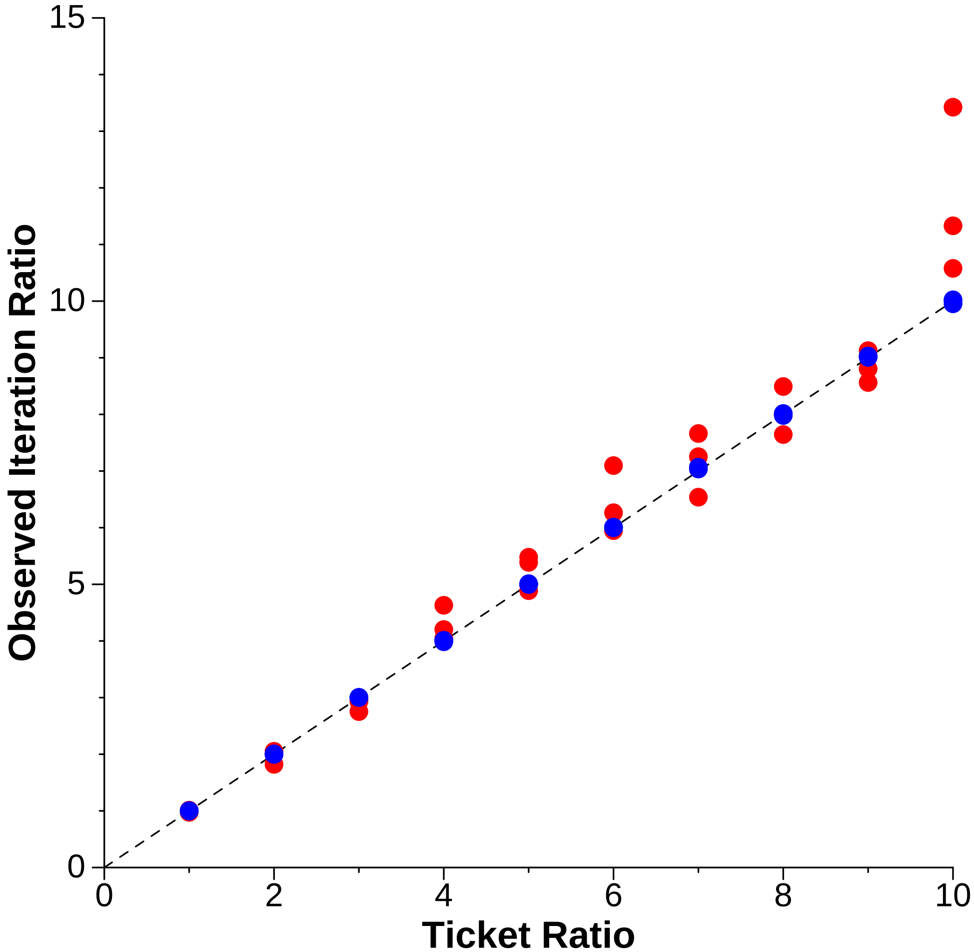


- **Static Allocation**
- **13 : 7 : 3 : 1 Ratio**
- **Mechanisms**
  - lottery
  - multi-winner (4)
  - stride
  - hierarchical

Prototype Process Schedulers

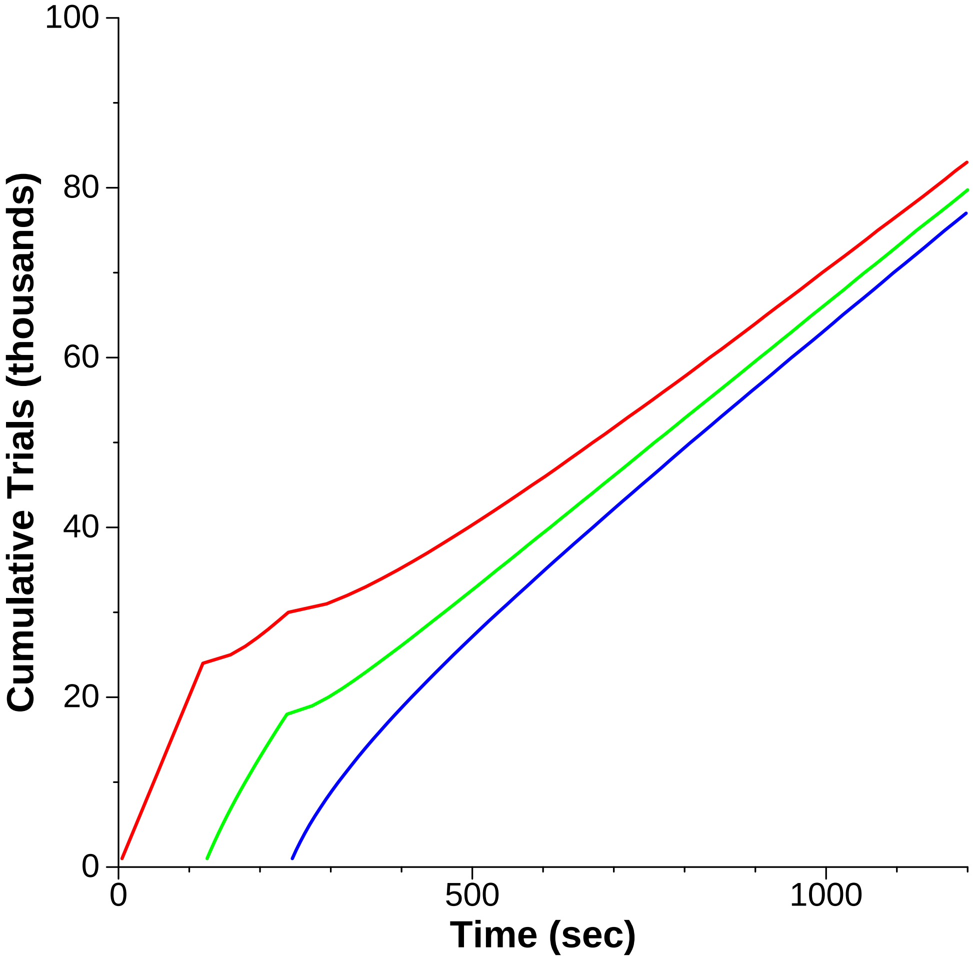
- **Lottery Scheduler**
  - modified Mach microkernel
  - DECStation 5000/125
  - complete framework implementation
- **Stride Scheduler**
  - modified Linux kernel
  - IBM Thinkpad 350C
  - no ticket transfers or currencies
- **Low System Overhead**

Relative Rate Accuracy



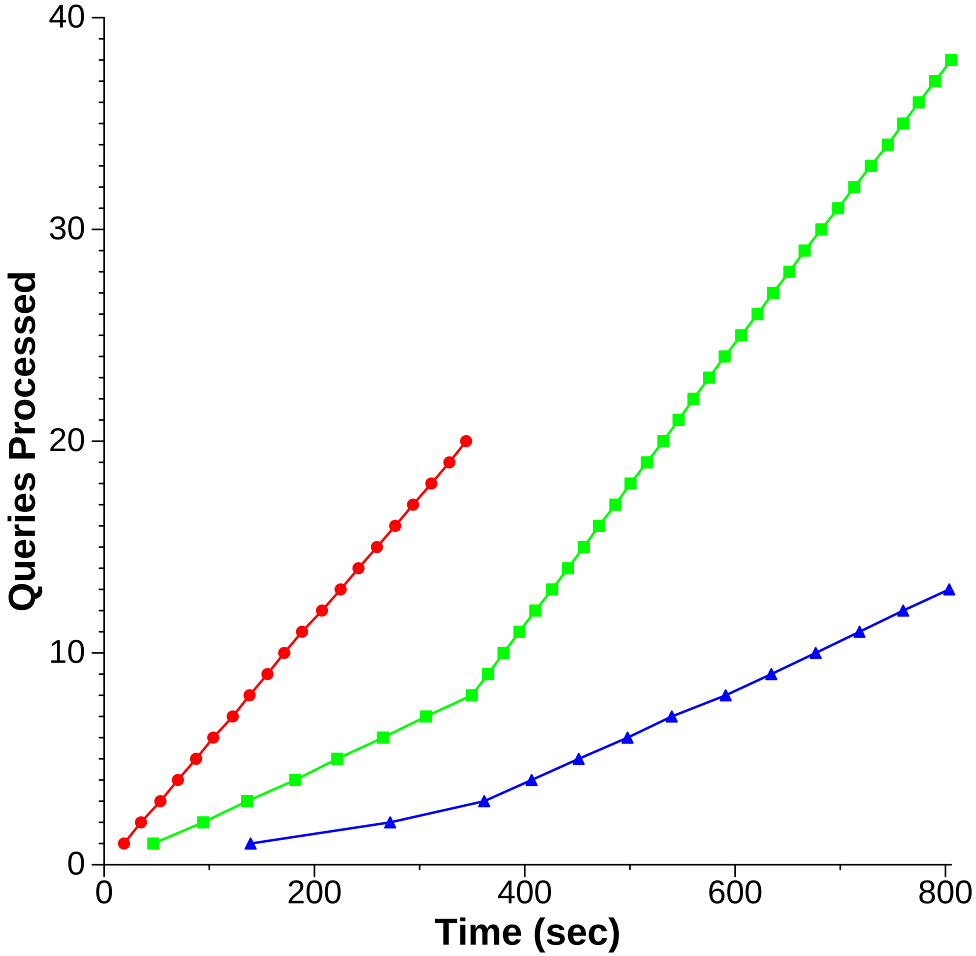
- **Lottery Scheduler**
  - Dhrystone benchmark
  - two tasks
  - three 60-second runs for each ratio
- **Stride Scheduler**
  - arith benchmark
  - two tasks
  - three 30-second runs for each ratio

Dynamic Ticket Deflation



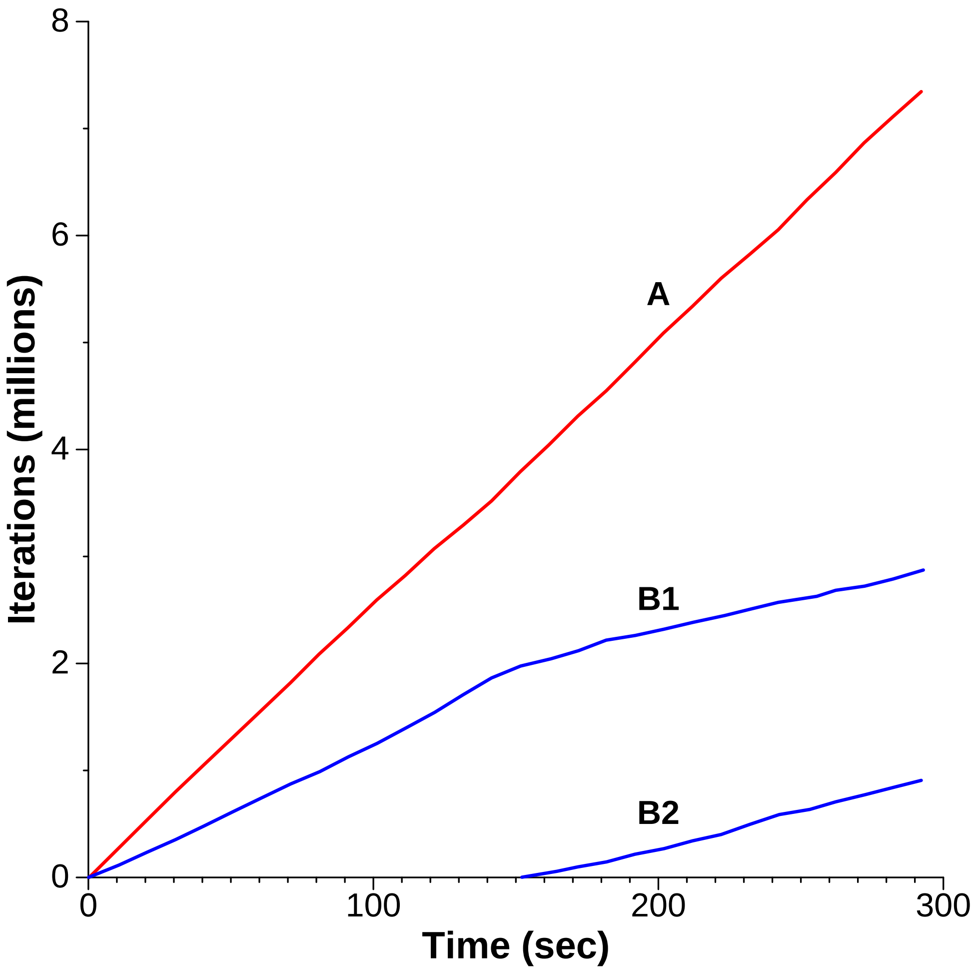
- **stride scheduler**
- **Monte-Carlo simulations**
- **many trials for accurate results**
- **three tasks**
- **funding based on relative error**

Dynamic Ticket Transfers



- lottery scheduler
- query processing
- multithreaded "database" server
- three clients
- 8 : 3 : 1 allocation

Modular Load Insulation



- lottery scheduler
- currencies A, B
  2 : 1 funding
- task A
  funding 100.A
- task B1
  funding 100.B
- task B2 joins with
  funding 100.B

Managing Diverse Resources

- **Synchronization Resources**
  - locks, condition variables
  - ticket inheritance, repayment
- **Space-Shared Resources**
  - inverse lotteries
  - minimum-funding revocation
- **Disk I/O Bandwidth**
- **Multiple Resources**

Conclusions

- **General Framework**
  - direct application-level control
  - simple, modular, flexible
  - widely applicable
- **Proportional-Share Algorithms**
  - lottery and stride scheduling
  - efficient $O(\lg n_c)$ operations
  - techniques for locks, memory, disk

Future Directions

- **Multiple Resources**
  - manage *all* critical resources
  - develop tools for adaptive software
  - microeconomic vs. proportional-share
- **Human-Computer Interaction**
  - improve application responsiveness
  - GUI elements for resource management