Lottery Scheduling

Flexible Proportional-Share Resource Management

Carl A. Waldspurger

William E. Weihl

Parallel Software Group
MIT Laboratory for Computer Science

November 15, 1994

Overview

- Scheduling Issues
- Lottery Scheduling
- Implementation
- Experiments
- Related Work
- Conclusions

Scheduling Issues

- **Context**
  - multiplex scarce resources
  - concurrently executing clients
  - service requests of varying importance
- **Quality of Service**
- **Software Engineering**

Conventional Scheduling

- **Priority Scheduling**
  - absolute control (but crude)
  - decay-usage scheduling
- **Problems**
  - often ad hoc
  - resource rights don’t vary smoothly
  - unable to control service rates
  - no modular abstraction

Solution: Lottery Scheduling

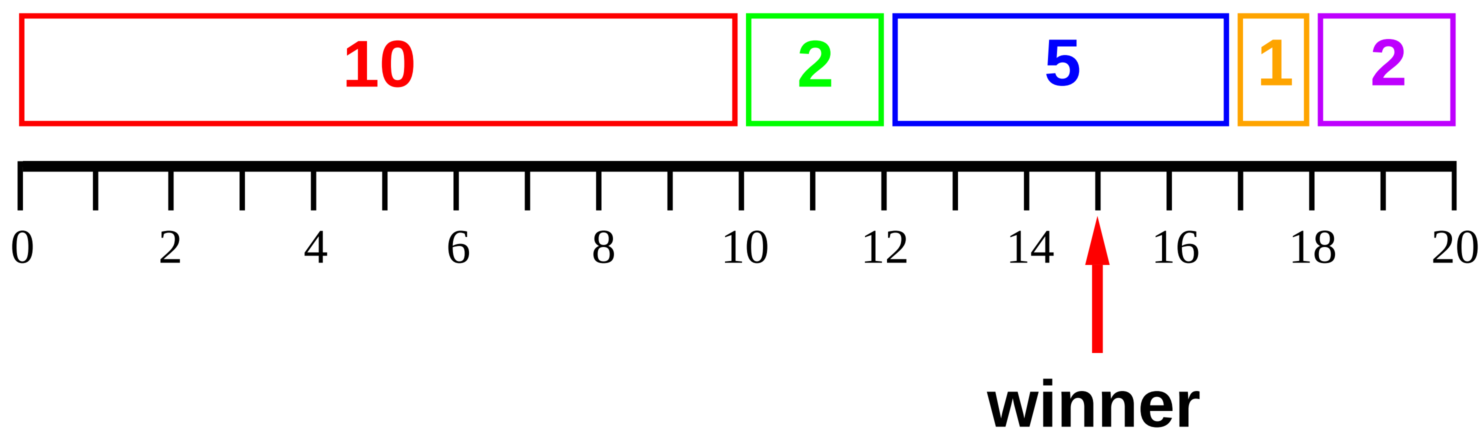
- Easily Understood Behavior
- Resource Rights Vary Smoothly
- Flexible Control Over Service Rates
- Modular Abstraction

Lottery Scheduling Basics

- **Randomized Mechanism**
- **Lottery Tickets**
  - encapsulate resource rights
  - issued in different amounts
  - first-class objects
- **Lotteries**
  - randomly select winning ticket
  - grant resource to client holding winning ticket

Example Lottery

total = 20
random [1 .. 20] = 15



Lottery Scheduling Advantages

- **Probabilistic Guarantees**
  - throughput proportional to ticket allocation
  - response time inversely proportional to ticket allocation
- **Proportional-Share Fairness**
  - direct control over service rates
  - easily understood behavior
- **Supports Dynamic Environments**
  - immediately adapts to changes
  - fair chance to win each allocation

Managing Diverse Resources

- **Processor Time**
- **Lock Access**
- **I/O Bandwidth**
  - disk bandwidth
  - network bandwidth
- **Space-Shared Resources**
  - resident VM pages
  - disk buffer cache

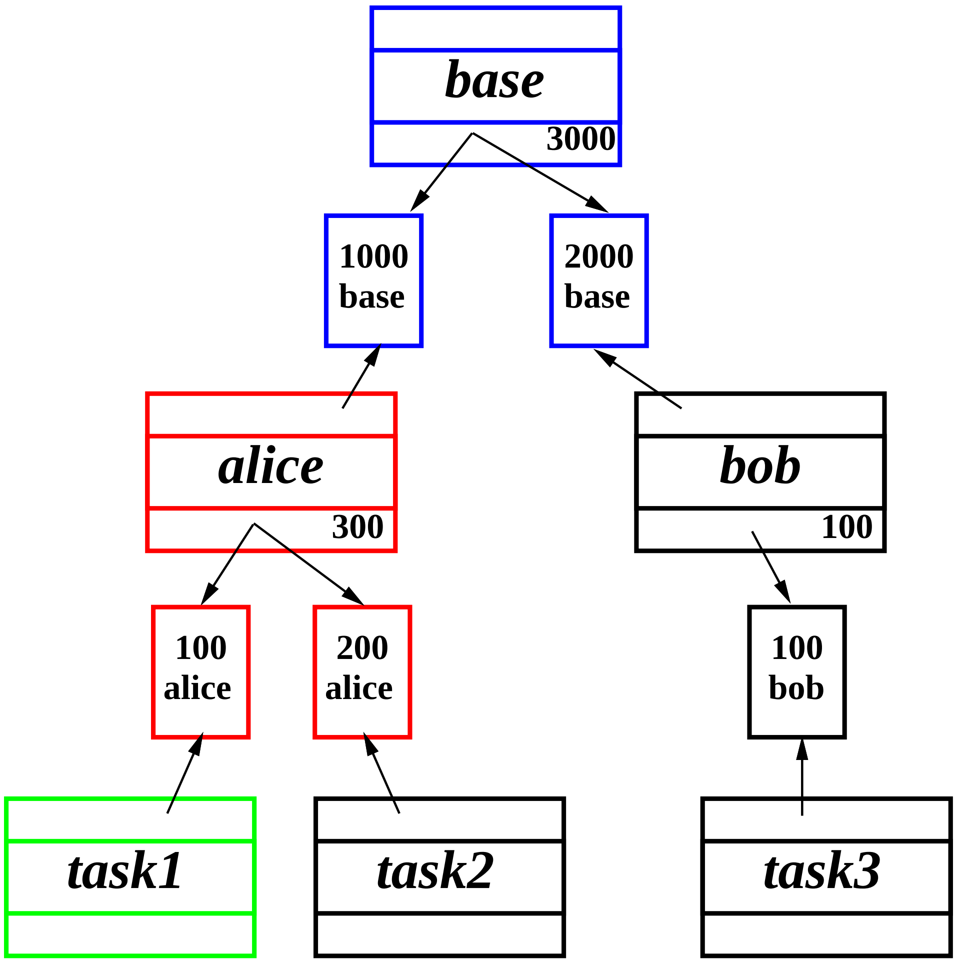
Flexible Resource Management

- **Ticket Transfers**
  - explicit transfer between clients
  - useful when client blocks while waiting
- **Ticket Inflation**
  - client creates more tickets
  - violates modularity and load insulation
  - convenient among mutually trusting clients

Ticket Currencies

- **Tickets 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

Currency Implementation



- **Computing Values**
  - currency:
    sum value of
    backing tickets
  - ticket:
    compute share of
    currency value
- **Example**
  - task1 funding in
    base units?
  - $\frac{100}{300} \times 1000$
  - 333 base units

Kernel Implementation

- **Objects: Ticket, Currency**
- **Operations**
  - create/destroy ticket, currency
  - fund/unfund currency
  - compute value of ticket, currency
- **Algorithms**
  - straightforward list-based lottery
  - simple currency conversion scheme

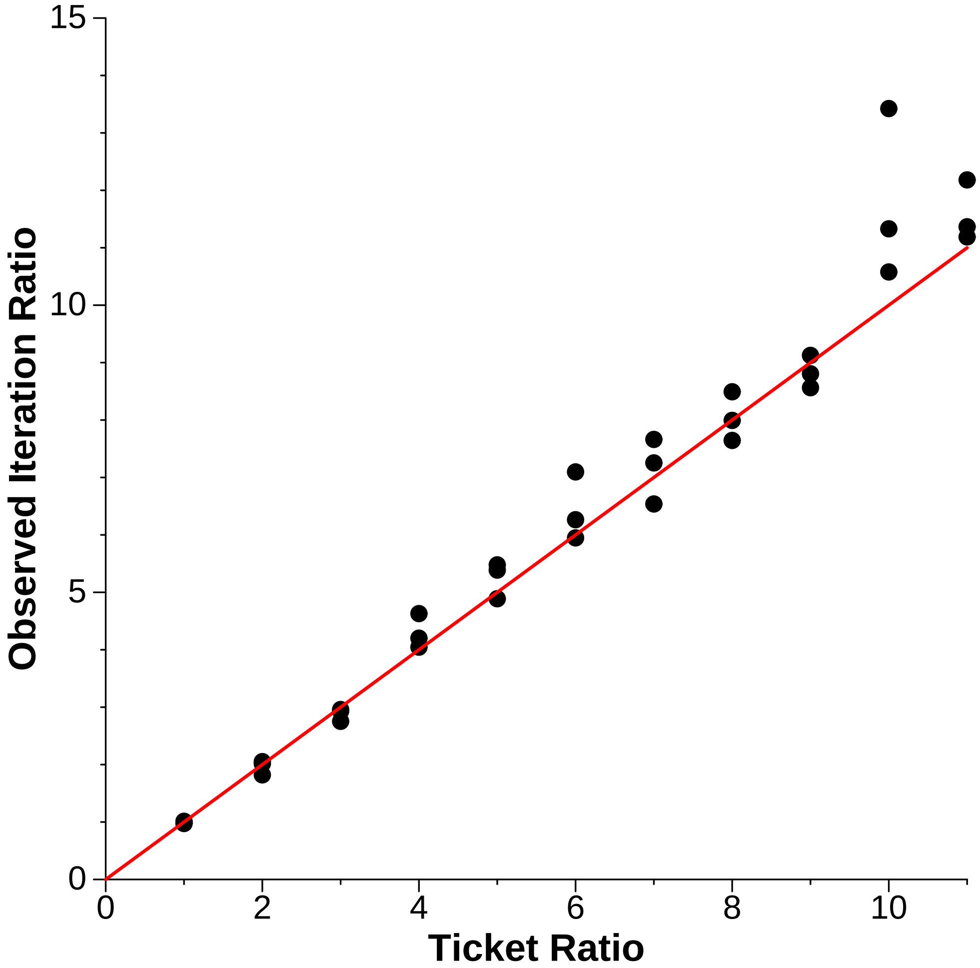
Prototype

- **Platform**
  - modified Mach 3.0 microkernel (MK82)
  - 25 MHz DECStation 5000/125
  - 100 millisecond quantum
- **System Overhead**
  - overhead comparable to standard scheduler
  - lightweight core mechanism
  - unoptimized prototype

Experiments

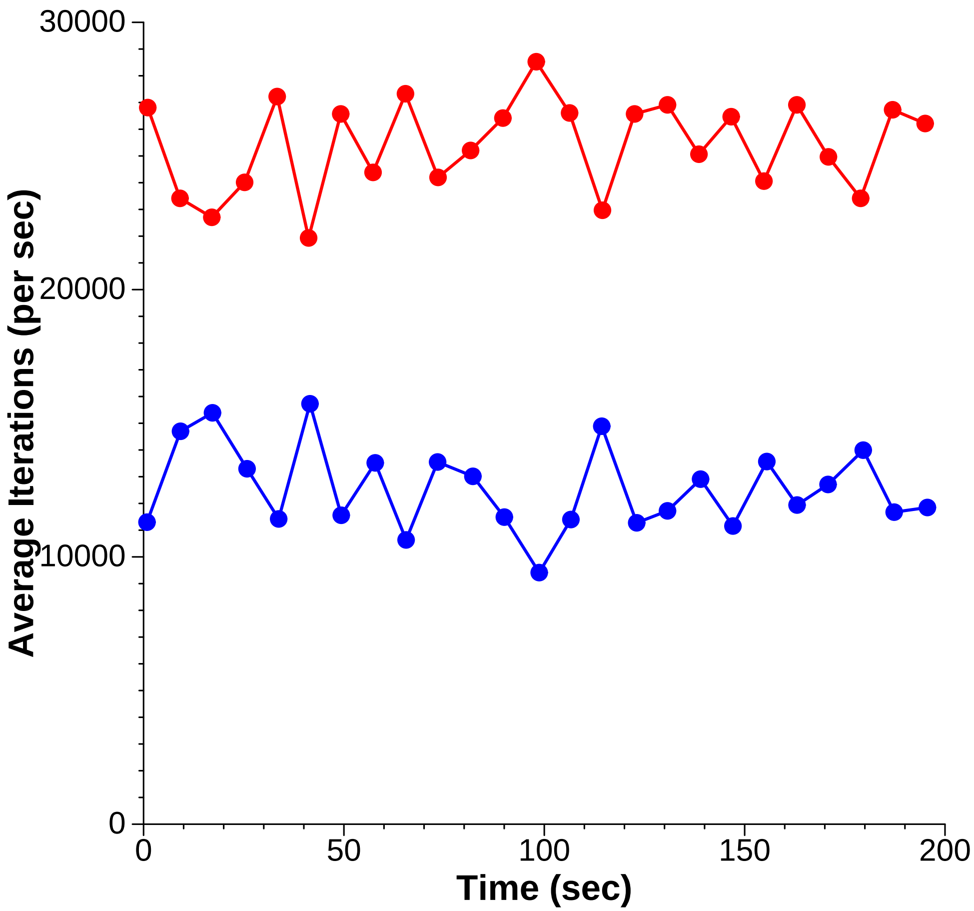
- Proportional-Share Service Rates
- Dynamic Ticket Inflation
- Client-Server Ticket Transfers
- Currency Load Insulation
- Lock Waiting Times

Relative Rates



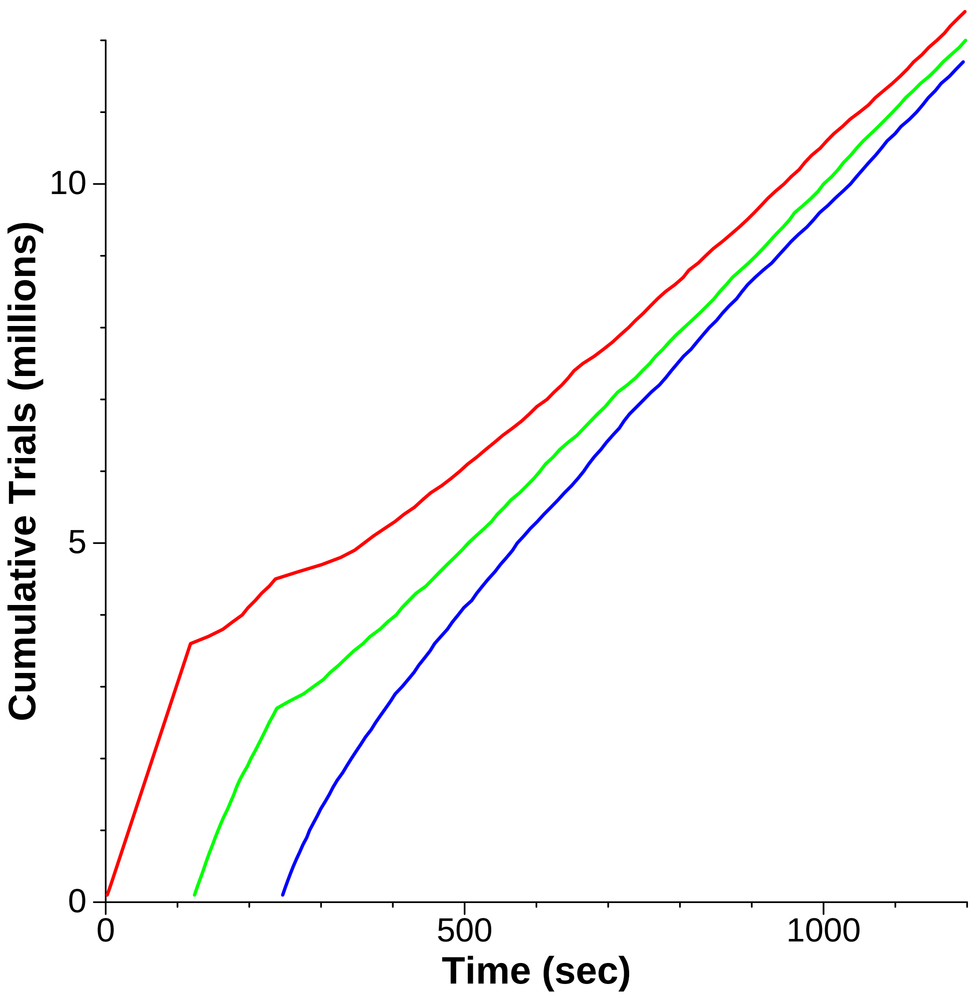
- **Dhrystone benchmark**
- **two tasks**
- **three 60-second runs for each ratio**

Fairness Over Time



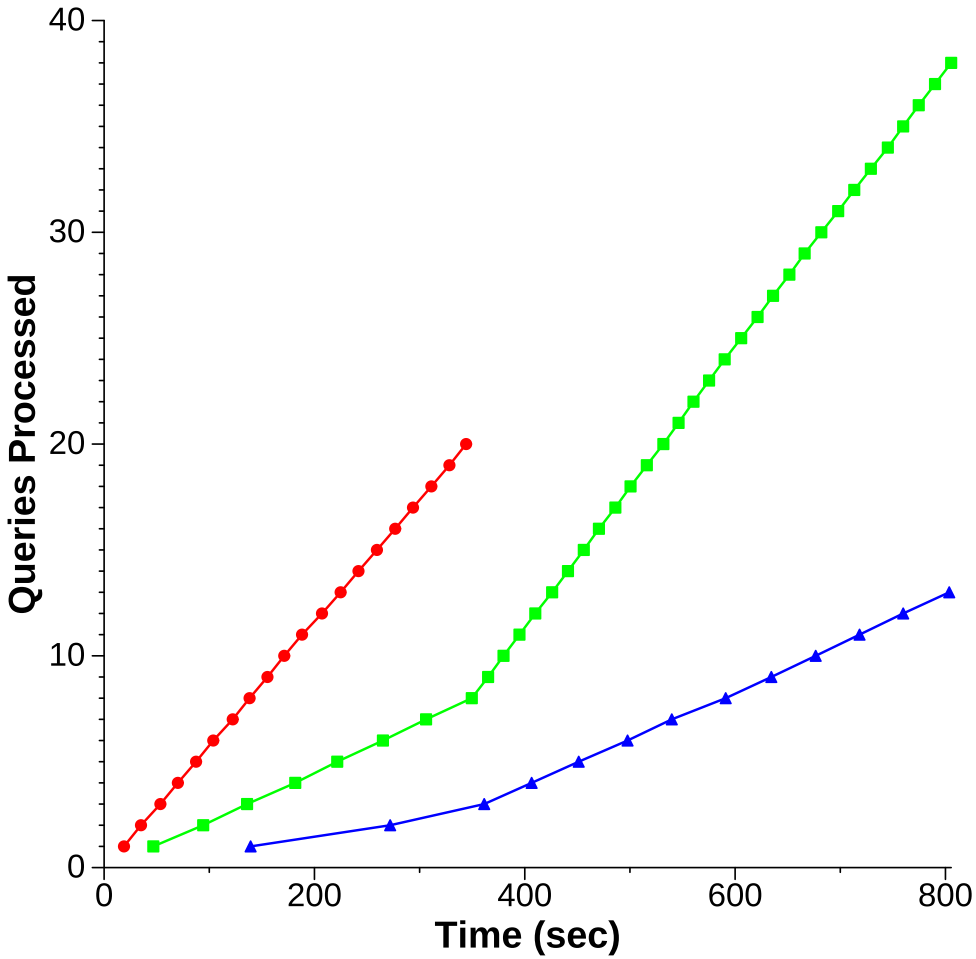
- Dhrystone benchmark
- two tasks
- 2 : 1 allocation
- 8-second averages

Monte-Carlo Rates



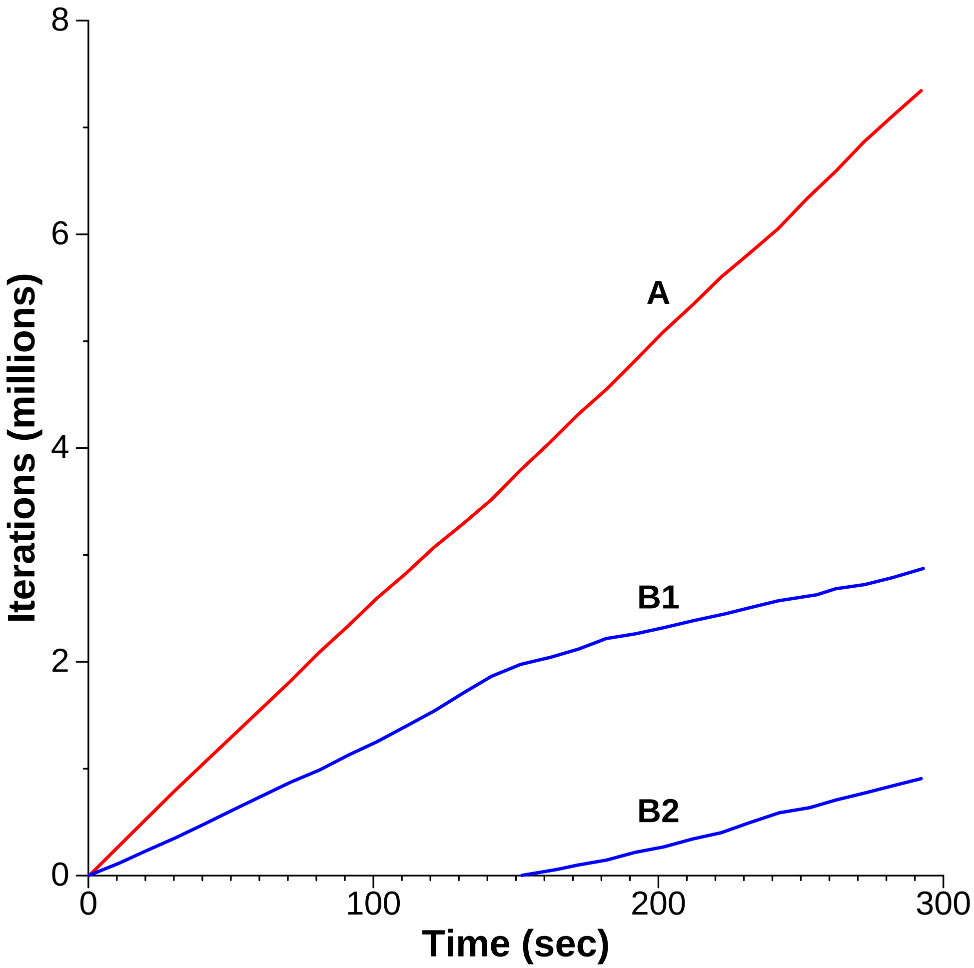
- **many trials for accurate results**
- **three tasks**
- **ticket inflation**
- **funding based on relative error**

Query Processing Rates



- multithreaded "database" server
- three clients
- 8 : 3 : 1 allocation
- ticket transfers

Currencies Insulate Loads



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

Lottery-Scheduled Locks

- **Waiting to Acquire**
  - waiters transfer funding to lock owner
  - lock owner inherits aggregate funding to acquire CPU
- **Release**
  - return funding to waiters
  - hold lottery among waiters
  - new winner inherits funding
- **Avoids Priority Inversion**

Lock Experiment

- **Groups A, B with 2 : 1 Allocation**
- **Acquire, Hold 50ms, Release, Compute 50ms**
- **Average Waiting Time**
  - A waits 450ms, B waits 948ms
  - 1 : 2.11 response time ratio
- **Lock Acquisitions**
  - A completes 763, B completes 423
  - 1.80 : 1 throughput ratio

Related Work

- **Priority Schedulers**
- **Fair-Share Schedulers**
  - dynamically manipulate priorities
  - [Hen84,Kay88,Hel93]
- **Microeconomic Schedulers**
  - auctions, bidding for resources
  - [Dre88,Fer88,Wal92]
- **AN2 Network Switch Scheduler**
  - statistical matching technique
  - [And93]

Conclusions

- Novel Randomized Scheduling Mechanism
- Easily Understood Behavior
- Precise Control Over Service Rates
- Modular Resource Management
- Simple, Efficient Implementation
- Generalizes to Diverse Resources