Optimizing Resource Sharing Systems

VARUN GUPTA
Carnegie Mellon University

Based on papers co-authored with:

JIM DAI
Georgia Tech

MOR HARCHOL-BALTER
Carnegie Mellon

KARL SIGMAN, WARD WHITT
Columbia University

BERT ZWART
CWI, Netherlands
Resource sharing systems are everywhere...

Benefits of resource sharing?

I/O+CPU+Bandwidth by Web servers

CPU cycles by OS task scheduler

Wireless channel by WAPs

...and you!
Why resource sharing: A queueing theory primer

**Processor Sharing (PS)**

- Arrivals
- $n$ jobs $\Rightarrow$ each job gets $1/n$ capacity
- $3+3+6=12$

**First-Come-First-Served (FCFS)**

- Arrivals
- Earliest job to arrive is served until completed
- $4+5+6=15$

Which has smaller mean response time?

- **✓ PS**
- **✗ FCFS**
Why resource sharing: A queueing theory primer

Processor Sharing (PS)

arrivals

n jobs ⇒ each job gets 1/n capacity

n jobs = 6+6+6 = 18

First-Come-First-Served (FCFS)

arrivals

Earliest job to arrive is served until completed

2+4+6 = 12

Now which has smaller mean response time?

☐ PS

☑ FCFS
Why resource sharing: A queueing theory primer

**Processor Sharing (PS)**
- arrivals
- \( n \) jobs \( \Rightarrow \) each job gets \( 1/n \) capacity
- ✔ Good for **high** job-size variability

**First-Come-First-Served (FCFS)**
- arrivals
- Earliest job to arrive is served until completed
- ✔ Good for **low** job-size variability
Why resource sharing: A queueing theory primer

**Processor Sharing (M/G/1/PS)**

- Poisson arrivals
- Job sizes i.i.d. $X$
- $n$ jobs $\Rightarrow$ each job gets $1/n$ capacity

- ✓ Good for high job-size variability

\[
E[T^{PS}] = \frac{E[X]}{1-\rho}
\]

\[
\rho = \text{arrival rate} \cdot E[X]
\]

measure of system utilization

UNIX process lifetimes: $C^2 > 40$
Files transferred over Internet: $C^2 > 25$

**First-Come-First-Served (M/G/1/FCFS)**

- Poisson arrivals
- Job sizes i.i.d. $X$
- Earliest job to arrive is served until completed

- ✓ Good for low job-size variability

\[
E[T^{FCFS}] = E[T^{PS}] \left(1 + \rho \cdot \frac{C^2-1}{2}\right)
\]

\[
C^2 = \frac{\text{var}(X)}{E[X]^2}
\]

measure of job size variability

Variability matters!
Real world ≠ Ideal theoretical policies

Reality check 1: Context-switch overheads
① Quantum-based Round-Robin
③ How to choose the optimal quantum size?

Reality check 2: Thrashing
① Impose a Multi-Programming-Limit (MPL)
③ How to choose the optimal MPL?

Reality check 3: Load balancing in server farms
③ How do load-balancing algorithms interact with servers?
③ What are good load-balancing algorithms?
Quantum-based Round-Robin (RR)

- External arrivals
- Incomplete jobs
- Completed jobs
- Jobs served for $q$ units at a time
- $h$ units of context-switch overhead after every quantum
Quantum-based Round-Robin (RR)

External arrivals → Incomplete jobs → Jobs served for $q$ units at a time → Completed jobs

$h$ units of context-switch overhead after every quantum
$E[T^{PS}] = \frac{E[X]}{1-\rho}$

- Context-switches cause overhead

$E[T^{FCFS}] = E[T^{PS}] \left(1 + \rho \frac{C^2 - 1}{2}\right)$

- Variable job sizes cause long delays

$q \to 0 \quad h = 0$

$q = \infty \quad h = 0$

small $q$  large $q$
A hammer for most occasions, 
...the $H^*$ job-size distribution

$H^* \sim \begin{cases} 
0 & \text{w.p. } p \\
\text{Exp}(\gamma) & \text{w.p. } 1 - p 
\end{cases}$

- 2 degrees of freedom
- Can match any $E[X]$ and $C^2 \geq 1$

- $\text{Exp}(\gamma) \equiv$ Exponential distribution
  - easy to analyze $\iff$ Markov chains

- $H^*$ captures the key phenomenon of (frequent) small vs. (rare) big jobs

For many systems (all cases in this talk), $H^*$ provides a good approximation for mean response time.
Step 1: M/G/1/RR with no overheads

\[ E[T^{RR}] \approx E[T^{PS}] \left[ 1 + \frac{C^2 - 1}{C^2 + 1} \cdot \frac{\rho}{q} + \frac{2}{C^2 + 1} \right] \]

For high \( C^2 \):

\[ E[T^{RR}] \approx E[T^{PS}](1 + \frac{\rho q}{E[X]}) \]
Step 2: Optimizing $q$

1. System with context-switch overhead $h \rightarrow$ a system with no overheads
   - New quantum size = $q+h$
   - Stretch job sizes by a factor $(1+h/q)$

2. OPT quantum \[ q^* = \arg\min_q E[T^{RR}] \]

   **Common case:** $h \ll E[X]$
   \[ q^* \approx \alpha(\rho) \sqrt{hE[X]} \]

   **EXAMPLE:** Linux context switch time $\approx 5$ microseconds

   Assume: mean job size = 5 sec, 80% utilization
   \[ q^* \approx 15 \text{ msec} \]

   **Actual Linux quantum size = between 10 and 200 msec**
$E[X] = 1$, $C^2 = 19$, $\rho = 0.8$
Mean response time

service quantum \((q)\)

\[ \mathbb{E}[X] = 1, \quad C^2 = 19, \quad \rho = 0.8 \]
Mean response time vs. service quantum ($q$)

Approximation $q^*$ and optimum $q$

$E[X] = 1$, $C^2 = 19$, $\rho = 0.8$
1. Effect of context-switch overheads can be significant - performance quite far from ideal PS
2. Choosing too small a $q$ is very bad, OK to err towards larger $q$
3. Performance of $q^*$ close to OPT
Real world ≠ Ideal theoretical policies

Reality check 1: Context-switch overheads
⚠️ Quantum-based Round-Robin
❓ How to choose the optimal quantum size?

Reality check 2: Thrashing
⚠️ Impose a Multi-Programming-Limit (MPL)
❓ How to choose the optimal MPL?

Reality check 3: Load balancing in server farms
❓ How do load-balancing algorithms interact with servers?
❓ What are good load-balancing algorithms?
Real world ≠ Ideal theoretical policies

Reality check 1: Context-switch overheads
⚠ Quantum-based Round-Robin
❓ How to choose the optimal quantum size?

Reality check 2: Thrashing
⚠ Impose a Multi-Programming-Limit (MPL)
❓ How to choose the optimal MPL?

Reality check 3: Load balancing in server farms
❓ How do load-balancing algorithms interact with servers?
❓ What are good load-balancing algorithms?
Tale of a typical server

Efficiency
(useful work per second)

K*
# concurrent threads/txns

Q: Max number of tasks allowed to share server?
Common solution: K*

Tasks not-yet-started

Active tasks

Server

Admission Control
A Queueing-theoretic model

PS server

Speed $\mu(n)$

# jobs at server (n)
A Queueing-theoretic model

The M/G/PS-MPL model

• Poisson(λ) arrival process
• Job sizes i.i.d. ~ X

\[ C^2 = \frac{\text{var}(X)}{E[X]^2} \]

• Sizes unknown, distribution of X known

GOAL: Find MPL (i.e. K) to minimize mean response time
Optimal MPL = K* ?

**Example**

Poisson(0.8) arrival process

- Exponential job sizes ($C^2 = 1$)
- Mean response time
  - $K^* = 5$
  - OPT MPL
  - Mean response time
- Weibull job sizes ($C^2 = 19$)
  - Mean response time
  - 45% improvement
  - OPT MPL
Optimal MPL = $K^*$?

**Example**

Poisson(0.8) arrival process

**Mean response time**

**Exponential job sizes ($C^2 = 1$)**

**Weibull job sizes ($C^2 = 19$)**

45% improvement

DEPENDS!
Intuition for the effect of MPL

High job-size variability
\((C^2)\)

High arrival rate
\((\lambda)\)

Increasing MPL

Processor Sharing (PS)

<Diagram>

- FCFS
- Introducing MPL
- Server

- High job-size variability
- High arrival rate

- Speed
- \(\mu(n)\)
- \(K^*\)

- # jobs at server
- 0 to 20
- 0.25 to 1.25
Intuition for the effect of MPL

High job-size variability $(C^2)$

High arrival rate $(\lambda)$

$C^2 \uparrow \implies$ Optimal MPL $\uparrow$

Arrival rate $\uparrow \implies$ Optimal MPL $\rightarrow K^*$
Step 1: M/G/PS-MPL approximation

Approximation assumption:
Job size distribution $\sim H^*$

$$ E[T_X] = E[T_X^Q(K)] + E[T_X^S(K)] $$

$$ E[T_X] \approx \frac{C^2 + 1}{2} E[T_{Exp}^Q] + E[T_{Exp}^S] $$
Step 2: Optimizing MPL

Set $MPL = MPL^*$, where:

$$MPL^* = \arg\min_K \left\{ \frac{C^2+1}{2} E \left[ T_{Exp}^Q(K) \right] + E \left[ T_{Exp}^S(K) \right] \right\}$$
Our approx. accurately predicts the behavior of the curve, and hence the correct MPL.

- Higher arrival rate $\implies$ MPL* decreases

K* gives 25% worse performance than MPL*. 

K* gives 45% worse performance than MPL*. 

Our approx. increases MPL.
Going even further...

Straw man proposal 1: Choose a “robust” static MPL
- Must choose MPL=K*: but suboptimal in light/moderate traffic

Straw man proposal 2: Learn the arrival rate
- Can’t adapt to changes on small scale/correlations

We Demonstrate: A Dynamic MPL control policy which is
1. Traffic-oblivious: self-adapts to variations in the arrival process
2. Light-weight: makes decisions based only on current queue length, Q(t), and current MPL, K(t)
Structure of our dynamic policy

- obtained by combining policy iteration with some new tricks (happy to discuss offline)
- robust to unknown and non-Poisson arrival processes
  - 20% performance loss in the worst case (compared to the optimal traffic-aware MPL)
  - MPL=K* becomes worse under non-Poisson arrivals
What we’ve learnt...

- Running the system at maximum efficiency is not optimal for mean response time
  - At moderate arrival rate: \( \text{MPL} > K^* \) can result in more than 45% smaller mean response time
- If don’t know arrival process: a dynamic policy can self-adapt while only knowing current queue length and MPL
Real world ≠ Ideal theoretical policies

Reality check 1: Context-switch overheads

⚠️ Quantum-based Round-Robin
❓ How to choose the optimal quantum size?

Reality check 2: Thrashing

⚠️ Impose a Multi-Programming-Limit (MPL)
❓ How to choose the optimal MPL?

Reality check 3: Load balancing in server farms

❓ How do load-balancing algorithms interact with servers?
❓ What are good load-balancing algorithms?
Real world ≠ Ideal theoretical policies

Reality check 1: Context-switch overheads

⚠️ Quantum-based Round-Robin
❓ How to choose the optimal quantum size?

Reality check 2: Thrashing

⚠️ Impose a Multi-Programming-Limit (MPL)
❓ How to choose the optimal MPL?

Reality check 3: Load balancing in server farms

❓ How do load-balancing algorithms interact with servers?
❓ What are good load-balancing algorithms?
A typical Web server farm

Load Balancer
(Immediate Dispatch)

Commodity servers

Timeshare service among current requests
Model: PS server farm

Load Balancer (Immediate Dispatch)

Commodity servers

Timeshare service among current requests
Model: PS server farm

- Load Balancer (Immediate Dispatch)

- $K$ homogeneous, PS servers
Model: PS server farm

- K homogeneous, PS servers
- Poisson arrivals
- Job sizes i.i.d. \( \sim X \)
Model: PS server farm

GOAL
Good Load balancing algorithms for PS server farms
Which is a good FCFS load balancer? (Hint: your local supermarket)

- Random
- Round-Robin
- Least-Work-Left
- Shortest Queue
PS server farms vs. FCFS server farms

Which is a good PS load balancer?
- Random
- Round-Robin
- Least-Work-Left
- Size-based-splitting
- Shortest Queue

Which is a good FCFS load balancer?
- Random
- Round-Robin
- Least-Work-Left
- Size-based-splitting
- Shortest Queue

Why?

- greedy!
- reduces $C^2$
E[T] under SQ/PS is “nearly insensitive” to the variability of job size distribution.
E[T] under SQ/PS is “nearly insensitive” to the variability of job size distribution.
E[T] under SQ/PS is "nearly insensitive" to the variability of job size distribution.

CONJECTURE: SQ load balancer is "nearly optimal" for PS servers.
What we’ve learnt...

- Good load balancers for FCFS and PS servers are different!
  - Least-Work-Left and Size-based-splitting are bad for PS!

- Shortest Queue (SQ) load balancing is ‘near-optimal’ for PS servers
  - Independent of job size distribution

- Shortest Queue (SQ) load balancing ‘preserves’ insensitivity of PS to job-size variability
Bridging the gap between practice and theory

1: Quantum-based Round-Robin

- Overheads matter – Ideal PS a bad model
- Right quantum size is important
- We give expression for OPT quantum

2: Systems with thrashing

- Running system at max efficiency not always optimal
- We find OPT MPL
- Dynamic policies can self-adapt to unknown arrival processes

3: Load balancing for PS server farms

- Scheduling policy of backend servers is integral for choosing load balancer
- Shortest Queue (SQ) is near optimal for PS servers – independent of job size distribution