An Analytical Model for Computer Systems with Non-Exponential Service Times and Memory Thrashing Overhead

Feng Zhang and Lester Lipsky
Computer Science and Engineering
Univ. of Connecticut
June 26, 2007
Motivation

- Highly varying job demands:
  \[ C_v^2 = \frac{\sigma^2}{\bar{x}^2} > 1 \]

- No preemption and no discretion on individual jobs: e.g. FCFS
  - Mean system time (or response time) \( \bar{T} \) is proportional to \( C_v^2 \)
    - Pollaczek-Khinchin formula (M/G/1 queue):
      \[
      \bar{T} = \frac{\bar{x} + \bar{x}\rho \cdot C_v^2 - 1}{1 - \rho} \quad \frac{1 - \rho}{2}
      \]
      \[ \rho = \lambda \bar{x} < 1 \] (Processor utilization)

To choose an effective strategy, we need to know the characteristics of jobs service times. In the last 15 years, it has been clear that job service times are often highly varied and the coefficient of variation \( C_v^2 \) is greater than 1. Here, \( \bar{x} \) and sigma represent the mean and standard deviation of service times. Recall that the value of \( C_v^2 \) is one for exponential.

As the well-known PK formula shows, the mean system time of a single server queue using FCFS is proportional to \( C_v^2 \). Since \( C_v^2 \) could be much larger than 1 for highly varying job demands, the performance of FCFS is apparently poor. **The reason is that FCFS is not preemptive.** So by serving a long job, many short jobs may be delayed greatly.
Effective Handling with Preemption and/or Discretion

- **SRPT**: shortest remaining processing time
  - Assumption: The exact service times of individual jobs are known
  - Optimal
- **SRT (SERPT)**: shortest residual time
  - Assumption: The job service time distribution is known
- **FB**: Foreground-background (also called least attained time)
  - Keeping track of the attained times of individual jobs
- **PS**: processor sharing
  - $\bar{T}$ is the same as that of $M/M/1$ queue ($C_v^2 = 1$) [BCMP75]

\[
\bar{T} = \frac{\bar{x}}{1 - \rho}
\]

On the other hand, most strategies that are effective for handling highly varying job demands are preemptive. For example, the strategies, such as SRPT, SRT, FB, and processor sharing, are all preemptive. (They all prevent long jobs from starving short jobs.)

It has been shown that by using processor sharing in a single server $M/G/1$ queue, the mean system time is the same as that of $M/M/1$ queue. So this is a remarkable improvement in comparison to FCFS.

In this work, we consider processor sharing and address its limitations.
The major limitation of processor sharing (as well as other preemptive strategies) is due to its unrealistic requirement. In particular, under heavy load conditions, processor sharing requires a computer system to handle a large number of jobs at the same time. However, this requirement can hardly be satisfied because of finite cache and main-memory size.

In other words, with many jobs inside the system, significant memory thrashing is inevitable.

Hence, system performance will get worse beyond some point.

Generally, if degraded performance is detected, it indicates occurrence of memory thrashing.
To bound the overhead and maximize performance, two population size constraints need to be imposed on the system. First, only a limited number of jobs should be allowed to share the processors at any time. This is essentially restricted processor sharing.

Further, the total number of jobs inside the system at any time has to be restricted as well.

With these population size constraints imposed on a system with non-exponential service times, an important question is how the system perform.

While simulations or measurements of real systems could be carried out to measure the performance. They are often infeasible if more than a few configurations need to be studied.

On the other hand, an analytical model if available often takes less time and effort, and can provide more insight.

In the view that the existing models do not apply here, our work aims at developing new analytical models for such systems.
In the following, I will define the problem first.
Problem Definition

- \( n-1 \): number of I/Os
- \( n \): number of tasks (or CPU service rounds)

- \( p_d = \frac{n-1}{n} \)
- \( x = n \)
- \( x_d = (n-1) x_{dr} \)

We assume that a job consists of \( n \) sequential tasks on average. All tasks but the last one are assumed to require data disk access at the end of their execution. Notice that the number of tasks is actually a random variable. Alternatively, we say that a job moves to access the data disk with probability \( p_d \). The relation between \( p_d \) and \( n \) is as the formula shows.

The paging disk is needed when we consider main-memory thrashing.

We assume that the mean and the distributions of task service times are known. We don’t assume that the service times of individual tasks are known.

Further, we only assume CPU time distribution is non-exponential. In other words, we assume exponential disk access times. However, our work can be extended to the case of non-exponential disk access times.

Three components: CPU subsystem; data disk for I/O activities; paging disk for paging activities. By replacement, we mean that a job is viewed as a new one after I/O access. However, the total CPU time requirement is the summation of all the rounds of CPU service. By resume, we mean that a job needs to resume where it left off due to a page fault.
We use this notation to denote systems with the population size constraints. There are P processors. At most K jobs are allowed to be in the CPU subsystem and C jobs are allowed to be inside the system.

The jobs in the CPU subsystem share the cache, while all the jobs inside the system share the main memory.

There are several queues in the system. If there are C jobs in the system, newly arrived jobs have to wait in the outside queue. If there are K jobs in the CPU subsystem, the internal processor queue holds extra jobs waiting for CPU service. If K>P, the round-robin queue is needed to handle the jobs inside the CPU subsystem. Similarly, we have paging and data disk queues to hold jobs waiting for data and paging disk access.
This work faces the following challenges. First, the systems we consider are quite complicated. The analytical solutions of Jackson networks do not apply because of non-exponential service times and population size constraints.

Second, even in single family M/G/P queues, restricted process sharing has not been studied much. Third, in more complicated systems, the interplay among the parameters is not well understood at all.
Outline

- Motivation
- Problem definition
- Analytical model
- Numerical results
- Cross-validation with simulation results
- Conclusions and future work
**Analytical model**

- **Assumptions:**
  - CPU time distribution $G$: represented by an $m$-dimensional matrix
    - E.g., Erlangian-2 / hyperexponential-2 has 2 exponential phases
  - Data/paging disk access time distribution: exponential

Note that $G$ is assumed to be represented by an $m$-dimensional matrix. In other words, $G$ consists of $m$ exponential phases. For example, E2 has 2 phases.
To deal with non-exponential CPU service times, we developed the model by applying LAQT.

Such information is captured in several key matrices. Due to the time limit, I will not describe how the matrices are constructed.

To account for cache thrashing overhead, we slow down the CPU service rates.
Deriving maximum throughput and mean system time

- After computing $\pi_c$, maximum throughput $\Omega$ can be obtained.
- To derive mean system time $\bar{T}$
  - $U_c$ is calculated first: an iterative approach [Neuts 1981, Lipsky 1992]
  - $U_n(n \in \{1, C-1\})$ are computed backwards
  - Then, $x_n(n \in \{1, C\})$ are computed

\[
\begin{align*}
\Omega &= \frac{\pi_c V_C e_G'}{\pi_c Y_C R_C} \\
\pi_C &= \pi_C Y_C R_C \\
\bar{T} &= \frac{\rho_0}{\lambda} \left[ \sum_{n=1}^{C-1} nx_n e_n' + \sum_{n=C}^{\infty} nU_{C-n} e_G' \right] \\
\frac{1}{\rho_0} &= 1 + \sum_{n=1}^{C-1} x_n e_n' + x_C (I_C - U_C)^{-1} e_G'
\end{align*}
\]

\[
\begin{align*}
x_1 &= \rho U_1 \\
x_n &= x_{n-1} R_n U_n, \ n \in \{2, C\} \\
U_C &= \lambda [I_C + B_C - U_C M_C Q_C R_C]^{-1} \\
U_n &= \lambda [I_n + B_n - R_{n+1} U_{n+1} M_{n+1} Q_{n+1}]^{-1}, \ 1 \leq n < C
\end{align*}
\]

After constructing the matrices, we can compute maximum throughput and mean system time using the following formulas. To solve the equations, we take an iterative approach.

We have developed an Matlab program to construct the matrices and do the computation.

Notice that the equations are the same when we consider data disk and paging disk as well. The only difference is that the matrices will be more complicated and much larger as we consider the disks.

Initially $U_C=\lambda V_C$
With Paging

- States need to contain information of phases with page faults
  - Jobs resume where they left off
- Page fault rate: number of page faults per unit time of CPU service
  - $\gamma(c)$: Generally an increasing function of $c (\leq C)$, the number of jobs in the system
  - Assumption: exponential inter page fault time
- $p_g, p_d, p_e$ are a function of $\gamma(c)$

It has to be pointed out that it is not a simple addition of another disk. Recall that after data disk access, the next task is handled. **This is like a replacement.** If main-memory thrashing happens, jobs need to resume at the phases where page faults occurred. So the handling of main-memory thrashing is more complicated. In the new model, we extend the state representation to contain information of phases with page faults. The page fault rate is used to denote the severity of main-memory thrashing.

Before we move on, I want to emphasize that if we assume exponential CPU service times, as in Jackson networks, resume and replacement are the same. Only when we consider non-exponential service times, do they behave differently. And the handling of resume is much more complicated.
A Single Job Paging-Resume Sub-Model

- Key idea: using \( m \) additional states to distinguish which phase has page fault
- \( \gamma \): page fault rate; \( \mu' = \mu + \gamma \)

Because of the time limit, I will not delve into details of the model. I just want to sketch the key idea of handling paging-resume of a single job. Specifically, we can imagine that there is a corresponding paging access phase for each CPU phase. In other words, we use \( m \) additional states.

Based on the single job paging-resume scheme, we established the full model. There are some existing work on studying resume and replacement within other contexts, like job failure. However, their results were generally obtained using some transformation methods. A drawback of such approaches is that it is not easy to do a parametric study. In this work, we handle resume and replacement using LAQT, which allows us to study the interplay among the parameters.
Space Complexity and Time Complexity

- Reduced product space: \( < \left( \frac{C+3m}{C} \right) \)

- Time complexity: a polynomial of largest matrix size

- Drawback: high complexity

<table>
<thead>
<tr>
<th>CPU Dist</th>
<th>P</th>
<th>K</th>
<th>C</th>
<th>No paging and I/O Max matrix size</th>
<th>With I/O Max matrix size</th>
<th>With paging and I/O Max matrix size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exponential</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1x1</td>
<td>2x2</td>
<td>3x3</td>
</tr>
<tr>
<td>Exponential</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1x1</td>
<td>4x4</td>
<td>10x10</td>
</tr>
<tr>
<td>Exponential</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1x1</td>
<td>4x4</td>
<td>10x10</td>
</tr>
<tr>
<td>Exponential</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1x1</td>
<td>5x5</td>
<td>15x15</td>
</tr>
<tr>
<td>Exponential</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>1x1</td>
<td>6x6</td>
<td>21x21</td>
</tr>
<tr>
<td>Hyper-exponential</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2x2</td>
<td>3x3</td>
<td>5x5</td>
</tr>
<tr>
<td>Hyper-exponential</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2x2</td>
<td>7x7</td>
<td>40x40</td>
</tr>
<tr>
<td>Hyper-exponential</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4x4</td>
<td>10x10</td>
<td>35x35</td>
</tr>
<tr>
<td>Hyper-exponential</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5x5</td>
<td>15x15</td>
<td>70x70</td>
</tr>
<tr>
<td>Hyper-exponential</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>2x2</td>
<td>9x9</td>
<td>95x95</td>
</tr>
<tr>
<td>Hyper-exponential</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>2x2</td>
<td>11x11</td>
<td>161x161</td>
</tr>
<tr>
<td>5-phase TPT</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5x5</td>
<td>6x6</td>
<td>11x11</td>
</tr>
<tr>
<td>5-phase TPT</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>5x5</td>
<td>16x16</td>
<td>386x386</td>
</tr>
<tr>
<td>5-phase TPT</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>35x35</td>
<td>56x56</td>
<td>286x286</td>
</tr>
<tr>
<td>5-phase TPT</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>70x70</td>
<td>126x126</td>
<td>1001x1001</td>
</tr>
<tr>
<td>5-phase TPT</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>5x5</td>
<td>21x21</td>
<td>1556x1556</td>
</tr>
</tbody>
</table>

As a comparison, let us look at the largest matrix size when main memory thrashing is ignored. Notice that without paging activities, the largest matrix size happens when \(K=C_{\text{max}}\). With paging activities, the largest matrix usually happens when \(K=1\). This is because some jobs may be waiting in the internal processor queue and we have to record the phases where they will resume as well. Further, the increase on complexity is much higher than the case of adding a data disk.
Outline

- Motivation
- Problem definition
- Analytical model
- Numerical results
- Cross-validation with simulation results
- Conclusions and future work
### Numerical Results

- Without any overhead, maximum throughput increases monotonically with $K$ and $C$.

- (a) $\overline{\theta_d}=1$, i.e., CPU-bound; (b) $\overline{\theta_d}=2$, mixed; (c) $\overline{\theta_d}=4$, I/O-bound

Using the full model, we examine the influence of memory thrashing on the optimality of $K$ and $C$. First, we assume no overhead. It can be seen that maximum throughput monotonically improves with the increase of $K$ and $C$. Indeed, this is true for different types of jobs.
No Paging and Negligible Cache Thrashing

- $K$ and $C$ should be as large as possible

First, we show that without cache thrashing and main-memory thrashing, maximum throughput increases monotonically with $C$. 
**No Paging, but with Cache Thrashing**

- **If CPU is a bottleneck, cache thrashing degrades performance**

When cache thrashing is considered, system performance degrades if CPU is a bottleneck and K is large. So an optimal K value may exist under cache thrashing. We can still choose a large C since no paging is assumed.
With Paging but no Cache Thrashing

- Assume page fault rate being a linear function of $c$
  \[ \gamma(c) = \gamma_0 \cdot \max(0, c-2) \text{ with } \gamma_0 = 25 \]

On the other hand, when paging overhead is considered, an optimal value of $C$ often exists.
We also obtain a counter-intuitive result. In particular, we got improved performance under some paging activities for configurations with $K < C$. As shown in this figure, the curve with paging activities outperforms the one without paging activities.

the upper curve is not the one without main-memory thrashing, but the one with main-memory thrashing.
Why?
- Paging activities achieve round-robin if $K<C$
  - Pay extra paging overhead
- A way to model restricted round-robin?

Our explanation is that long jobs tend to be interrupted because of paging activities. So short jobs can go through quickly. In other words, paging activities achieve some effect of round-robin.

To verify this point, we choose several different paging disk service rates and examine how maximum throughput changes with the increase of page fault rate. It can be seen that with a fast paging disk, the performance of the configuration with $K<3$ does improve by increasing page fault rate. And the performance approaches to that of the configuration with $K=3$. This means that our model may be applied to analyze restricted round-robin. Notice that inter page-fault times are like time-slice values, although it assumes exponential time-slice values.

To further examine whether our model reflects the performance of restricted round-robin, we also compare with simulation results.
Comparison with Simulation Results

- Analytical results should agree with simulation ones
  - Fast paging disk: $\mu_g = 20000$

- Partial Match using Fixed $\Delta$

Now, let us look at the comparison of analytical results and simulation ones. Recall that our model assumes exponential time-slices. In our simulation, we first use deterministic time-slices.
We also simulated restricted round-robin using exponential time-slice. This time, we get an exact match as shown in the figure. For example, the green one and the red one can hardly be distinguished. As such, we can say that our analysis regarding the counter-intuitive result is correct.
Summary

- Presented a generic analytical model
  - Handling main-memory thrashing differently from cache thrashing
- Demonstrated existence of optimal $K$ and $C$ values due to memory thrashing
- Obtained a counter-intuitive result
  - Paging activities help sometimes
  - It suggests a way to model restricted round-robin with exponential time-slice values
- Obtained cross-validation between simulation and analytical results

In conclusions, …
To extend this work, there are several directions that can be pursued.

So this is the end of my talk. Any questions?