Abstract- Real-time multi-process scheduling is commonly used in many control situations, where it is important to achieve job completions within specific time intervals. This paper investigates the potential improvement that might be achieved when additional information (notably the residual computing time of each process) is used in the scheduling process. First, a simulation study was completed in order to demonstrate the potential benefit of the approach. The results, over many alternative job processing time distributions, clearly show a significant benefit. Second, by using a discrete distribution as a representation/approximation of each job’s distribution, a technique is developed to achieve similar beneficial results using a practical, and time cost efficient scheduler. The discrete distributions can be derived from the computation structures for each process. The results show that the approach can be effective in real time scheduling situations.

Keywords: real time scheduling, simulation and testing, residual time, computation structure software model, coefficient of variation.

1 Introduction

There are many applications of multiprocess scheduling where job deadlines are desired. In our case, we consider soft deadlines, where some allowance can be made for process execution times which exceed the specified deadline. We assume that the processes are known and available a-priori, such that their run-time distribution (probability density function) on a dedicated processor is known or can be evaluated. These applications may be process control situations, where a processor is supporting a number of time-sensitive applications, each with a desirable deadline. Alternatively, the number of requests satisfied per second by a web server is one of the major metrics considered in a high performance system. Today, web servers cannot respond immediately to the numerous requests for files (or HTTP pages) they receive. Typically requests are stored in a buffer or a queue and then processed in the order of their arrivals using a
FCFS policy. The FCFS scheduling model which is used in most of the existing web servers does not provide Quality of Service (QoS) because the completion time of a request depends on how many jobs are already in the queue. As the FCFS policy serves jobs in the order of job arrival there is no concern for job size variation. A system in which deadlines are assigned and applied could provide some control and order in the serving process and thus provide a better service. Our study has not attempted this application, but rather has developed a generic solution approach and evaluation which can be applied to these types of situations.

In this paper we consider the case where the probability density function (pdf) of the job service time is known or approximated. We assume that the coefficient of variation of service time, $C^2 \geq 1$. The goal is to maximize the number of jobs that meet their deadlines, with the emphasis on an approach which can be used in practical scheduling applications. Our particular research contribution is in the using the estimated residual time of each process to be scheduled as a criterion for selecting jobs for immediate service. By definition, residual time is the expected remaining execution time of a task after receiving a service time of $t$ time units. In this paper we have developed, and evaluated the performance of, a Residual Time Based (RTB) scheduling algorithm which uses both job residual time information and deadlines to select the next job to be executed.

First we compared the performance of scheduling strategies, namely First Come First Served (FCFS), Round Robin (RR), Earliest Deadline First (EDF) with the Residual Time Based (RTB) algorithm [12] in scheduling jobs with highly variable job service time distributions. We report results from discrete event simulations comparing these different scheduling policies for a wide range of well-known (Markovian) continuous job size
distributions (coefficient of variation, $c^2$, ranging from less than 1 to greater than 1). The RTB approach is shown to provide a higher percentage of deadline makers than the other approaches. Second, we considered a discrete approximation approach which could be applied to virtually any distribution, and developed and tested its application to the scheduling problem in a time cost effective manner.

Section 2 of the paper gives an overview of related research. In section 3 we provide required mathematical background along with some function definitions. Section 4 describes the scheduling model and outlines the RTB policy. Section 5 describes the simulation model and the outcome of the simulation experiments. Section 6 describes a practical approach to apply RTB and its experimental evaluation. The last section concludes the work.

2 Related Research

Given the enormous amount of literature available, in the field of real time scheduling any survey can only scratch the surface. Moreover, a large number of scheduling approaches are based upon radically different assumptions making their unified comparison difficult [16, 18]. At the highest level the scheduling paradigm may be divided into two major categories: real time and non real time. Within each of the categories scheduling techniques may also vary depending on whether one has a precise knowledge of job processing time or only its pdf.

In [18], authors identify several classes of scheduling algorithms for real time systems. The majority of real time scheduling algorithms reported in the literature, perform static scheduling and hence have limited applicability since processing time of all tasks is not known a-priori. Recently many scheduling algorithms [17] have been proposed to dynamically schedule a set of tasks where each task is assumed to require deterministic processing time, which is their worst-case execution time. However, many real time applications have stochastic
processing time with large variability. It can be impractical to assume a worst case execution time for each task when the processing time variance is high as in most computer applications, because on average the system will be highly underutilized. Liu and Layland [15] formally studied priority-driven algorithms for periodic tasks. They focused on the problem of scheduling periodic tasks on a single processor and proposed two preemptive algorithms, namely, Rate-Monotonic (RM), and Earliest Deadlines First (EDF). Our present work focuses on non-periodic real time jobs.

In [3] and [4] they used SRPT scheduling. It is well known from queueing theory that SRPT is an optimal algorithm which minimizes queueing time [20, 21]. However SRPT requires knowing the size of the job beforehand. Most of the time job sizes in computer systems are not known a priori, for instance, an operating system does not know how long a process will need to run, in request for database transaction where the response to the web server are created on the fly: such dynamic web requests processing times are not known ahead of time, and a router does not know the length of a flow in the network. In such a case, the default queueing discipline is First-Come-First-Served (FCFS).

The number of requests (jobs) satisfied per second by a web server is one of the major metrics to be considered for high performance web server system. FCFS is used in today’s router [2]. Investigation on the Internet overload control strategy is reported by Cao and Nyberg [4] used a tandem queueing network to predict web server performance metrics and was validated through measurements and simulation.

Recent studies have revealed that network traffic exhibits multiple time scale traffic [10]. Most of the transactions are short such that a tiny fraction of the largest flows constitutes more than half of the total load [9] implying a highly variable job size distribution. The web and
data servers receive a continuous stream of request with processing times, which vary over several orders of magnitude [11]. Highly varying job demands are characterized by $C^2 > 1$. If $C^2 < 1$ one should stick to FCFS. In FCFS short jobs suffer from longer delays due to the presence of longer jobs in front. When $C^2 > 1$ and the exact processing time of each job is not known in advance, as in many computer system applications, processor sharing (PS) is the way to go. Last Come First Served Preemptive Resume (LCFSPR), where an arriving job preempts the job presently being served and continues until it finishes or is preempted by a newer arrival, produces the same mean system time as PS [1]. The structure is stack rather than a queue. However the system time is much more varied, and small jobs may actually get stuck in the queue for longer times.

In practice, however, PS must be implemented by time-slicing. For example, RR, a common time-slicing scheme, serves a queue of awaiting jobs by turns. Specifically, a job is chosen from the front of the queue and it is served for at most a time-slice. If it completes its service time requirement during the time-slice, it releases the processor and leaves the system immediately. Otherwise, it is preempted after the allocated time-slice and put at the end of the queue, awaiting further service. As a variation of RR a newly arriving job can be allowed to preempt the one presently in service. Then there are several options for the interrupted job after the new job finishes. Either it can lose the rest of its time slice (the system moves to the next job in the ring); or receive the remaining time in its slice; or receive a whole new time slice. We have studied these variants of RR by extensive simulations, and reported in a previous paper [19].

Scheduling both real-time and non real-time tasks under load requires knowing how long a task is going to run. One can determine the latest time the task must start executing if one
has the knowledge of the exact time of how long a task will run. However, in most of the computer applications it is not possible to know in advance exactly how long a job is going to execute until it is complete. In this paper, we investigated this issue and illustrated a detailed method to estimate job execution time using residual time information. Further we developed a dynamic scheduling strategy, which includes both the deadline and the estimated execution time (residual time). A job arrives with a service time and deadline requirement. As time evolves, both: (1) the residual service time, depending on the amount the job has already serviced, and (2) the time remaining until the deadline, change. To the best of our knowledge no prior research has investigated the use of job residual time in scheduling.

3 Terminology and Mathematical Background

Let $X$ be a random variable denoting the processing time of a request. In probability theory for any service time distribution the probability that the service time is no more than $x$ is called the Probability Distribution Function (PDF), $F(x)$. Its derivative $f(x) = \frac{dF(x)}{dx}$ is called the probability density function (pdf). The mean, $\bar{x} = E[X]$, variance is $\sigma^2$, squared coefficient of variation, $C^2 = \sigma^2 / \bar{x}^2$. If job demands vary widely then $C^2 \geq 1$. The Reliability function

$$R(x) = P(X > x) = \int_{x+}^{\infty} f(x) dx,$$

where $R(x) + F(x) = 1$. The expected execution time,

$$\bar{x} = E[X] = \int_{0}^{\infty} xf(x) dx = \int_{0}^{\infty} R(x) dx$$

Conditional reliability, $R_t(x)$, is the probability that the task executes for an additional interval of duration of at least $x$ given it has already executed for time $t$ [6].
\[ R_t(x) = \frac{R(t+x)}{R(t)} \]. By definition, the task residual time \( rm(t) \) given that the task has already been executed for \( t \) time units is;

\[ rm(t) = \int_0^\infty R_t(x) = \int_0^\infty \frac{R(t+x)}{R(t)} \, dx \quad (3.1) \]

It is well known that a single server First Come First Serve (FCFS) policy yields a mean \textit{system time} (total time a job spent in the system including any waiting time for service), given by the Pollaczek-Khinchine (P-K) formula:

\[ E[T] = \frac{\bar{x}}{1 - \rho} + \frac{\bar{x}\rho}{1 - \rho} \left( \frac{C^2 - 1}{2} \right) \quad (3.2) \]

The equation states that the job mean system time (which is called mean turn-around or mean response time), \( E[T] \), depends only on their arrival rate \( \lambda \), and the mean, \( \bar{x} \), and variance, \( \sigma^2 \), of the service time distribution. In the formula the job size variation is expressed in terms of the squared coefficient of variation, \( C^2 = \sigma^2 / \bar{x}^2 \). Where the utilization is \( \rho = \lambda / \bar{x} \). Clearly, the larger the \( C^2 \) is, the longer is the \textit{system time}. Because of this issue there has been an increases interest in processor sharing (PS), which brings the mean \textit{system time} to that of \( C^2 = 1 \) \cite{5},

i.e., \( E[T_{ps}] = \frac{\bar{x}}{1 - \rho} \)

It is clear, if \( C^2 < 1 \) it is beneficial to use FCFS which yield mean system time less than \( \bar{x} / (1 - \rho) \). As soon as \( C^2 \) becomes bigger than 1 the value for \( E[T] \) starts getting larger than that for \( E[T_{ps}] \). The appendix contains the specific models/equations for various distributions. These are: Uniform, Exponential, Two Branch Hyper-exponential, Erlangian-n, Four Stage Hyper Erlangian, Three Branch Hyper Exponential, and Power Tail.
4 Scheduling Structure

4.1 Task model

Jobs are assumed to be independent, identically distributed, and preemptable. They are characterized by the following: task arrival time, $A_i$; task execution time, $E_i$; task deadline, $D_i$; task start time, $S_i$. Task $i$ is ready to execute as soon as it arrives, regardless of processor availability. Tasks may have to wait for some time, $W_i$, before they start executing for the first time due to scheduling decision. As a result, we have

$$S_i = A_i + W_i, \text{ where } W_i \geq 0$$

Risk factor: We define a function called risk factor ($rf$). As soon as a process $i$ arrives, time starts running out before it reaches its deadline. At clock time $C_T$, the time left before deadline is $D_i - (C_T - A_i)$. Let $rm(t)$ be the expected remaining execution time (residual time) required to finish the given process $i$, when it has already received service for $t$ time units. This $rm(t)$, has to fit in the time interval $D_i - (C_T - A_i)$ to meet the deadline. Let us define the risk factor for process $i$ after receiving service for $t$ time units, as follows:

$$rf_i = rm(t) / (D_i - C_T - A_i)$$

In other words, $rf_i = \frac{\text{residual time}}{\text{remaining time before deadline}}$

The $rf$ is inversely proportional to the difference between request deadline and the clock time and increases as the request approaches to its deadline. As soon as the clock time passes the deadline $rf$ becomes negative if the request has not yet been completed. The goal of the present research is to increase the number of jobs meeting the deadline. After the clock time passes the deadline, the request is considered to be less important compared to jobs that are very close to
deadline but have not yet crossed the deadline. Therefore, jobs with positive risk factor are
given higher priority than jobs with negative risk factor.

4.2 RTB Scheduling Algorithm

Jobs submitted from clients join a queue. Each job is associated with a processing time,
the duration of which follows a distribution function, which can either be deterministic or non-
deterministic. A job $i$ will have a deadline $D_i$ which represents a soft timing constraint on its
completion. The scheduling strategy includes both the deadline and the estimated remaining
execution time (residual processing time). As time evolves, both (1) the residual processing
time $rm(t)$, depending on the amount of time it has already executes, and (2) the time remaining
until the deadline, changes. Note, the server CPU is being shared by several concurrent jobs.
The server scheduler will pick a request to process from the queue depending on the scheduling
policy. Once picked, each request $i$ will be processed for a small time slice called time quantum,$\Delta t$. At the end of $\Delta t$ the executing request is either complete or partially executed. A partially
executed request stays in the queue. As soon as a request arrives, time starts running out as it
reaches the deadline. At clock time $C_i$ the time left before deadline is $D_i - (C_T - A_i)$, where $A_i$ is
the request arrival time. Let $rm(t)$ be the remaining processing time for request $i$ to be finished,
which has to fit in the time interval $D_i - (C_T - A_i)$ to meet the deadline.

In this paper we developed a Residual Time Based (RTB) scheduling approach, a
dynamic scheduling policy which uses residual times, $rm(t)$, as the criteria for selecting jobs
for immediate service. $rm(t)$ is the expected remaining execution time of a request after
receiving a service time of $t$ time units. If one can keep track of the service time, $t$, which an
individual request has already received, one can estimate $rm(t)$ [5].
Algorithm: Outline

In the Residual Time Based (RTB) dynamic scheduling algorithm, the risk factor, \( rf \), (of request \( i \)), is used as a measure to choose the next request to be processed [13], at the end of time slice. Jobs in the request set are maintained in the order of decreasing \( rf \). The outline of the RTB algorithm is given below:

1) When/if a request arrives, place it in the queue.

2) Initially and for each quantum conclusion;
   a) If the executing request has completed, remove it from the queue.
   b) compute remaining time, \( rm(t) \) and risk factor, \( rf \) for each request in the queue
   c) select the request that has the highest positive \( rf \), otherwise the most negative \( rf \) to execute.

Note, the risk factor increases positively as the request approaches to its deadline. As soon as the clock time passes the deadline the risk factor becomes negative. Since, the goal in the present research is to increase the number of jobs satisfied before the deadline, jobs with positive \( rf \) are given higher priority than the job with negative \( rf \).

4.3 Other Scheduling Algorithms

This section describes the other service disciplines we included. They are: First Come First Service (FCFS), Round-Robin (RR), Earliest Deadline First (EDF).

FCFS (First Come First Served):

The tasks are allowed to join the arrival pool as soon as they arrive. The scheduler selects the first job in the queue and executes it until completion. This policy does not need any estimation of execution time. Smaller jobs may have to wait long time if a longer job happens to be ahead of it in the queue.
RR (Round Robin):

The scheduler selects the first job in the queue and executes for a given time-slice (time-quantum). At the end of the time slice the job (if not finished) is appended at the back of the queue. Thus, the jobs share the processor time in a Round Robin fashion.

EDF (Earliest Deadline First):

Earliest Deadline First (EDF) is a deadline driven algorithm, which can be applied for real time jobs. Jobs are arranged in a queue with a descending order of their deadline time. We have implemented EDF as a processor-sharing algorithm. The scheduler selects the first request in the queue and executes for a given time-slice (time-quantum). At the end of the time-slice the request is inserted in the queue so they remain sorted in a descending order of their deadline. The jobs keep on sharing the processor until they are done.

5 Simulation Study

5.1 Job Processing Time Model

Job execution (processing) time was modeled by a stochastic distribution (e.g., uniform, exponential, hyper-exponential, Erlangian, power tail). The choice of the specific job size distribution is motivated to represent job streams where the jobs have high variability (most of them are short and a tiny fraction of them are large). Distributions with $C^2 > 1$ represents high variability job sizes and hyper-exponential, hyper-Erlangian, power tail are a few cases that can have high $C^2$.

5.2 Job Size distribution Types and Their Properties

Various cases of request processing time distributions have been investigated to present a wide variety of $C^2$: 
- Uniform, $C^2 = 1/3$.
- Exponential, $C^2 = 1$.
- 2 branch hyper exponential, $C^2 = 10$
- 4 stage Hyper Erlangian, $C^2 = 10$
- 3 branch hyper exponential, $C^2 = 10$
- Power tail, $C^2 = \infty$.

Figures 1 - 5 show the residual time behavior of the five stochastic distributions from the above list (please see the appendix for individual residual time expressions). All the other distributions except uniform use the exponential distribution as a building block. The residual time of the exponential distribution is a constant (mean service time = $1/\mu$), independent of time. Figure 1 shows the plot of residual time for 2-branch hyper-exponentially distributed service time for $p_1 = .0477, p_2 = .9523, T_1 = 10.475, T_2 = .525$. At time $t = 0$, $rm(t)$ is 1 (= the mean). As time grows, $rm(t)$ increases and approaches to $\max[T_1, T_2^-]$. Figure 3 depicts the case of hyper-Erlangian, where the residual time at first decreases, then increases greatly and then decreases gradually to $\max[T_1, T_2^-]$.

Figure 1. Residual time for a two branch hyper exponential service time distribution
Figure 2. Residual time for uniform service time distribution.
5.3 Simulation Results and Analysis

Several stochastic discrete event simulators were constructed to implement the operation of RTB, FCFS, RR and EDF policies. The simulation code was first developed in the Microsoft visual C environment. Then it was executed in CygWin using a Linux 48 bit random number generator. The aggregate request arrival is considered to have exponentially distributed inter-arrival times (i.e., a Poisson arrival process). For each chosen algorithm and a given distribution simulation run of $10^9$ job samples was conducted. Below we describe the comparative performance of RR, FCFS and RTB algorithms in a single processor system. Job deadline was assumed to be 4 times of the mean service time and mean service time is considered to be one. For each type of service time distribution the run time is generated randomly so that they follow...
a mean of one. Without loss of generality, we have used normalized service time for each distribution.

**Validation of the Simulator:** Several test cases were run to validate the performance of the simulator. As a first set of tests, the simulator has been set to run a FCFS algorithm with exponential arrival time, and exponential service time. The output results were compared with the analytical results: process turn-around time calculated from the Pollaczez-Khinchin formula [5, 7] for M/M/1/FCFS and for $\rho = .8$ is $3\frac{2}{3}$. Our simulation result gives 3.66.

Our RTB algorithm is a Processor-Sharing algorithm. So, we further verified the simulator by running the common Round Robin Processor-Sharing algorithm for M/M/1. Again, process turn-around time can be calculate from the Pollaczez-Khinchin formula to be 5 for M/M/1/RR queue and $\rho = .8$. Our simulation yields the job turn-around for the same scenario as 4.985. For any M/G/1 queue RR should yield the same value (=5) for the system time as obtained from M/M/1 [7]. Table 1 shows the turn-around time obtained from the simulation. The simulation results are presented in two subsections as follows: first we analyze the results of four distributions with different variances and second we analyze the results of different distributions with same variance.

<table>
<thead>
<tr>
<th>Distribution names</th>
<th>Turn-around time</th>
</tr>
</thead>
<tbody>
<tr>
<td>uniform</td>
<td>4.99</td>
</tr>
<tr>
<td>exponential</td>
<td>4.985</td>
</tr>
<tr>
<td>hyper exponential</td>
<td>4.96</td>
</tr>
<tr>
<td>power tail</td>
<td>4.89</td>
</tr>
</tbody>
</table>
5.3.1 Distributions with Different Variances

In this section we describe the comparative performances of RR, FCFS, RTB and EDF algorithms in a single processor system for four different service time distributions with different values for $C^2$, namely, uniform ($C^2 = 1/3$), exponential ($C^2 = 1$), two branch hyper exponential distribution with $C^2 = 10$, and power tail ($C^2 = \infty$). Below we describe the comparative performance of Table 2.a and 2.b summarizes the comparative performance of all the four job service time distributions. The results for each of the distributions are itemized into the following subsections.

Figure 6 depicts our simulation results for three different scheduling policies namely, FCFS, RR and RTB when the job size distribution is considered to be uniformly distributed. The graph shows the fraction (normalized) of the number of jobs that have been processed by time $t$, after they arrived. The x-axis represents the request turn-around time. The y-axis represents the normalized number of jobs completed on or before the corresponding turn-around time, shown by the x-axis. The cusp of the RTB graph indicates the request deadline, which is 4 times the mean service time. For uniform distribution, there is not much variability within the request size. FCFS appears to be a better choice than RR in the sense that a higher fraction of number of jobs finishes earlier when they are serviced using FCFS policy as compared to when they are serviced using RR policy. This again validates the fact that using FCFS is detrimental when $C^2 > 1$. Instead, a processor-sharing algorithm should be used to improve system performance, whether the performance parameter is the system time (non real time case) or the number of jobs meeting deadline (real time case).

As one may observe in figure 6 that the plot for RTB is rising faster than that for FCFS and RR policy, indicating that more jobs (87.4 %) can be processed before the deadline when they are scheduled by the RTB policy as opposed to either FCFS (66.6% jobs satisfied before
deadline) or RR (58.8% jobs satisfied before deadline) policy (see table 2a and 2b for the numbers. Moreover, the plot for FCFS rises above the plot for either RTB or RR as the turnaround time passes the deadline. This indicates that longer jobs (those that did not have an opportunity to meet deadline), can get a better chance to finish beyond the deadline when they are serviced by either RTB or RR. In the world of real time jobs one of the major goals is to increase the number of jobs meeting the deadline, which is accomplished better if the RTB scheduling policy is used instead FCFS or RR.

Figure 7 represents the simulation results for exponentially distributed request size. By comparing figure 6 (uniform distribution) and figure 7 (exponential distribution) we see that as the request size variability increases in exponential distribution \((C^2=1)\) a PS policy like RR outperforms FCFS. Moreover, RTB algorithm outperforms both RR and FCFS algorithms in terms of the number of jobs finishing before deadline. Percentage of jobs satisfied before deadline for FCFS, RR and RTB policies are 55.9, 65.4 and 85.99, respectively.

In hyper exponential distribution there are three free parameters, namely, \(p_1\), \(T_1\) and \(T_2\) [5]. In this simulation \(p_1\) is considered to be .0477, whereas, \(T_1\) and \(T_2\) are considered to be 10.475 and .525, respectively. The parameter values are chosen so to have \(C^2 = 10\) to represent more short jobs and fewer longer jobs. Now the jobs need to be serviced by a PS algorithm. In figure 8a (hyper exponential distribution) we observe that RTB outperforms both FCFS and RR in satisfying request deadlines. The RR algorithm can satisfy more jobs (78%) than FCFS (37.9%) algorithm (see table 1a and table 1b). When jobs with high variability is processed using our developed RTB algorithm, 93.75% of them are able to be processed before their deadlines.
Figure 6 Normalized number of jobs completed vs. turn-around time, t for uniform job service time distribution and for FCFS, RR, and RTB algorithms.

Figure 7 Normalized number of jobs completed vs. turn-around time, t for exponential job service time distribution and for FCFS, RR, and RTB algorithms.

Figure 8a Normalized number of jobs completed vs. turn-around time, t for hyper exponential job service time distribution and for FCFS, RR, and RTB algorithms.

Figure 8b shows the simulation results comparing the Earliest Deadline First (EDF) algorithm with RTB and FCFS algorithm for hyper exponential distribution. EDF is a deadline driven
algorithm, which can be applied for real time jobs. Jobs are arranged in a queue with a
descending order of their deadline time. We have implemented EDF as a processor-sharing
algorithm. The scheduler selects the first job in the queue and executes for a given time-slice
(time-quantum). At the end of the time-slice the job is inserted in the queue so they remain
sorted in a descending order of their deadline. The jobs keep on sharing the processor until they
are done. The nature of the plot for EDF algorithm is similar to that of FCFS algorithm. In other
words, EDF algorithm behaves quite the same as FCFS algorithm.

For job service time following a power tail distribution, the job queue length grows
towards infinity when jobs are serviced using the FCFS policy. Thus, simulation falls apart. As
a result, we do not have any simulation result for the FCFS policy. Figure 9 shows the
simulation results for the power tail distribution. Table 2b shows that more jobs (93.5 %) can be
processed before deadline when they are scheduled by RTB policy as opposed to RR policy
(80.9 % jobs). From table 1a we observe that the turn-around time (=3.67) for power tail
distribution is the least.

5.3.2 Different Distribution with Same Variance

In this section we compare the results from three different job service time distributions
with same variance, $C^2 = 10$. The distributions are, namely, hyper exponential, hyper-
Erlangian, and three-branch hyper exponential. Note: these distributions have same $C^2$ here but
the nature of their residual times is different (see section 6.2). The results for hyper exponential
job service time distribution have already been presented in the previous section (Figure 8). In
this section we present results for hyper-Erlangian and three-branch hyper exponential.
The simulation results, where job service times follow hyper-Erlangian distribution with $C^2 = 10$ is shown in figure 10, the values for probabilities and the corresponding times; $p_1, p_2, T_1$ and $T_2$, are .0477, .952, 6.12026 and .218281, respectively [5]. Table 3a compares the job turn-around time obtained from P-K formula and simulation results for three different distributions but with same $C^2 = 10$. Table 3b depicts the improvement of RTB over FCFS and RR, for the same three different distributions which shows that more than 93% jobs can meet their deadline when they are scheduled by RTB algorithm, whereas, only 80.7% jobs can meet their deadline if they are serviced using RR policy. FCFS policy can meet the deadlines for less than 34% jobs only. From table 3b the turn-around time obtained from the P-K formula is 23 for all of the three distributions. Our simulation yields turn-around time for FCFS algorithm in the range of 22.23 to 23.3 for the three different distributions. Turn-around times for the RR algorithm ranges from 4.92 to 4.97, which is very close to 5. For each distribution, RTB algorithm always gives lesser values for turn-around times than RR algorithm.

![Hyper Exponential Distribution](image)

Figure 8b Normalized number of jobs completed vs. turn-around time, t for hyper exponential job service time distribution and for FCFS, RR, RTB and EDF algorithms.
These results showed the Residual Time Based (RTB) scheduling approach offers deadline-meeting advantages in a broad range of situations. Thus one can conclude confidently that RTB algorithm satisfies more jobs meeting their deadlines for all of the examined distributions with $C^2 = 10$. (implying higher job size variability).

| Table 2a Comparison of job turn-around time obtained from the P-K formula and the simulation results, for different job service time distributions |
|-----------------|-----------------|-----------------|-----------------|
| Distribution types | Turn-around time calculated from P-K formula | Turn-around time from simulation |
| uniform | 3.666 | 3.66 | 4.99 | 4.48 |
| exponential | 5 | 4.95 | 4.825 | 4.94 |
| hyper exponential | 23 | 22.23 | 4.96 | 4.098 |
| power tail | ----- | 4.89 | 3.67 |
Table 2b Improvement of RTB over FCFS and RR

<table>
<thead>
<tr>
<th>Distribution type</th>
<th>% of task satisfying $D$ in</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FCFS</td>
</tr>
<tr>
<td>uniform</td>
<td>66.6</td>
</tr>
<tr>
<td>Exponential</td>
<td>55.9</td>
</tr>
<tr>
<td>Hyper exponential</td>
<td>37.9</td>
</tr>
<tr>
<td>Power Tail</td>
<td>---</td>
</tr>
</tbody>
</table>

Table 3a Comparison of job turn-around time obtained from the P-K formula and the simulation results, for different job service time distributions with $C^2 = 10$

<table>
<thead>
<tr>
<th>Distribution types</th>
<th>Turn-around time calculated from P-K formula</th>
<th>Turn-around time from simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Turn-around time calculated from P-K formula</td>
<td>FCFS</td>
</tr>
<tr>
<td>Hyper exponential</td>
<td>23</td>
<td>22.23</td>
</tr>
<tr>
<td>Hyper Erlangian</td>
<td>23</td>
<td>23.3</td>
</tr>
<tr>
<td>Hh 3 branch hyper exponential</td>
<td>23</td>
<td>22.64</td>
</tr>
</tbody>
</table>

Table 3b Improvement of RTB over FCFS and RR, for different job service time distributions with $C^2 = 10$

<table>
<thead>
<tr>
<th>Distribution types</th>
<th>% of task satisfying $D$ in</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FCFS</td>
</tr>
<tr>
<td>hyper exponential</td>
<td>37.9</td>
</tr>
<tr>
<td>Hyper-Erlangian</td>
<td>33.9</td>
</tr>
<tr>
<td>3 branch Hyper exponential</td>
<td>37.4</td>
</tr>
</tbody>
</table>

6 Practical Solutions for Implementation

The simulation results presented so far assess the viability of using job residual time in a scheduling situation. There is a clear improvement when the residual time approach is used. Next we develop a practical solution to apply the RTB algorithm, by representing service time
distributions by a discrete time-probability distribution, which can be used for virtually any case.

6.1 Discrete Representation for Job Service Time

We use the Computation Structure Model [22], to describe the detailed time-execution behavior of any computation as a discrete distribution, where the execution time performance can be approximated by the following set of time, $T$ and probability, $P$ vectors. The Computation Structure approach provides a procedure which derives a discrete distribution of the time-probability behavior for a computation from its software language. It should also be noted that a discrete distribution can also be used for cases where data size alternatives and likelihoods lead to different, predictable execution times (e.g. for data lookup and delivery cases).

$$T = [t_1, t_2, .........., t_N]$$  \hspace{1cm} $$P = [p_1, p_2, .........., p_N]$$

where,  \hspace{1cm} $t_i < t_j \text{ if } i < j$ \hspace{1cm} and \hspace{1cm} $\sum_{i=1}^{N} p_i = 1$

The set $t_1, t_2, .........., t_N$ covers all represented execution times of a program, where $p_i$ is the associated probability. In other words, during an execution the software will take any one of the times $t_i$ to finish execution, with $p_i$ probability. The properties are:

- Expected execution time, $E[T] = \sum_{i=1}^{n} p_i t_i$
- Variance, $\sigma^2 = \sum_{i=1}^{N} p_i (t_i - E[T])^2$
- Coefficient of variation $C^2 = \frac{\sigma^2}{E[T]^2} = \frac{E[T^2]}{E[T]^2} - 1$

Now we consider the scheduling structure and derive formulae for calculating the expected residual time $rm(t)$, given a time $t$ has already passed.
6.2 Scheduling Structure for Job and Timing Control

Before the start of execution, job residual time is denoted by \( rm(0) \). From the time of arrival until the beginning of the task execution, residual time remains unchanged as \( rm(0) \), where \( rm(0) = E[T] = \sum_{i=1}^{N} p_i t_i \). The software program will take any one time \( t_i \) to complete execution, where \( t_i \) is an element of vector \( T \). Assume the job has executed for \( t_{e_1} \) time units. The remaining execution time can be calculated as follows:

**Starting point:**

\[
T = [t_1, t_2, \ldots, t_N] \\
P = [p_1, p_2, \ldots, p_N]
\]

**Step 1:**

\[
T_2 = [t_1 - t_{e_1}, t_2 - t_{e_1}, \ldots, t_n - t_{e_1}] \\
P_2 = [p_1, p_2, \ldots, p_N]
\]

*explanation:* Some of the smaller entries in \( T_2 \), where \( t_i \leq t_{e_1} \), may be negative or zero, indicating that, those paths were not taken in the execution. Next, these distribution components need to be removed and the distribution be normalized. Let \( p_c = \sum_{t_i - t_{e_1} \leq 0} p_i \). The RHS adds all the values of probability for which the corresponding time is either 0 or negative. \( 1 - p_c \) gives the summation of the probabilities where the corresponding time has positive entry in \( T_2 \) vector.

**Step 2:**

Rewrite \( T_2 \) and \( P_2 \) as \( T_2' \) and \( P_2' \) respectively, to normalize the distribution.
**Explanation:** From $T_2$ remove those entries that have zero or negative time and rename $T_2$ as $T_2'$. From $P_2$ remove the corresponding entries, normalize the entries in $P_2$, and rename $P_2$ as $P_2'$, where

\[
T_2' = [t_1', t_2', \ldots, t_{r_2}']
\]

\[
P_2' = [p_1', p_2', \ldots, p_{r_2}']
\]

Let us assume that the $j^{th}$ element in vector $T_2$ has the first nonzero entry. Here, $t_1' = t_j - t_{e_1}$ and $t_{r_2}' = t_n - t_{e_1}$, respectively, in $T_2$ vector. Similarly, $p_1' = p_j/(1 - p_c)$ and $p_{r_2}' = p_n/(1 - p_c)$, respectively, in $P_2$ vector. The remaining time distribution is complete. $T_2$ and $P_2$ vector depicts the remaining time distribution after elapsed time $t_{e_1}$. This procedure can be applied iteratively as successive time intervals elapsed.

**Step 3:**

Update the mean remaining time,

\[
rm(t) = \sum_{i=1}^{r_j} p_i t_i
\]

at the $r_j^{th}$ cycle of iteration, where,

\[
p_i \in P_{r_j}\text{ and } t_i \in T_{r_j}'.
\]

We assume that the program will be scheduled to execute for one quantum at a time, and $t$ (elapsed time) is a multiple of a single quantum $\Delta t$. Assume that the accumulated time, $t$ is greater than $t_i$, i.e., $t_i < t \leq t_{i+j}$ where $t_i$ and $t_{i+j}$ are elements of vector $T$ and $i = 0 \ldots N - 1$, $t_0 = 0, 1 \leq j \leq N - 1$, and $i + j = N$. At the end of next quantum the job will have executed for $\Delta t + t$ time units.

There can arise two possible cases in between the times of $t$ and $\Delta t + t$:
Theorem 1

For any $i$:

Case 1) when none of the $t_i$s occur between $t$ and $\Delta t + t$ i.e., where $t_i < t < t_{i+1}$ and $\Delta t + t \leq t_{i+1}$, the distribution remains the same between the time interval, and the residual time can be updated by equation 6.1 as follows:

$$rm(t + \Delta t) = rm(t) - \Delta t$$

(6.1)

Case 2) when one or more of the $t_i$s fit between $t$ and $\Delta t + t$ i.e., where $t_i < t < t_{i+1}$ and $\Delta t + t > t_{i+1}$, $t_{i+j-1} < \Delta t + t \leq t_{i+j}$ and $i + j \leq N$. The distribution changes between the time interval. And the residual time can be updated by equation 6.2 as follows:

$$rm(t) = \frac{rm(0) - \sum_{j=1}^{i} p_j t_j}{1 - \sum_{j=1}^{i} p_j} - t$$

(6.2)

Proof Case 1:

As time progresses, at the end of each quantum, $\Delta t$, the remaining time of the task being executed decreases. At the end of the first quantum, the expected remaining time

$$rm(t + \Delta t) = rm(t) - \Delta t$$

provided $\Delta t \leq t_1$. Let us consider an r length sequence of quanta, $t = r\Delta t$, where $t \leq t_1$ (or $r \leq t_1 / \Delta t$). Now, the time vector becomes

$$T = [t_i - t] \quad \forall i$$

Thus the expected remaining time after $t$ is:

$$rm(t) = \sum_{i=1}^{N} p_i(t_i - t)$$

or

$$rm(t) = rm(0) - t, \text{ for } 0 < t \leq t_1$$

(6.3a)

since $\sum_{i=1}^{N} p_i = 1$.

After an additional quantum, $rm(t + \Delta t) = \sum_{i=1}^{N} p_i(t_i - (t + \Delta t)$.
Finally, after rearranging the terms and substituting, we have

\[ rm(t + \Delta t) = rm(t) - \Delta t \]  

(6.3b)

Note that equation (6.3a) is a special case of equation (6.3b). In general, at any time \( t + \Delta t \), \( rm(t + \Delta t) \) can be obtained by subtracting \( \Delta t \) from the expected remaining time at \( t \), \( rm(t) \), using equation (6.3b) as long as the distribution remains unchanged from accumulated time \( t \) to \( (\Delta t + t) \).

**Q.E.D**

**Proof: Case 2**

We obtain a general equation for \( rm(t) \) in the process of proving case no. 2. When the accumulated time \( t \) crosses time \( t_i \) (an element of T), the number of elements in the time vector reduces to \( N - i \). The distribution changes from the previous one and then remains the same until \( t > t_{i+1} \). Within the time interval of \( t_i < t \leq t_{i+1} \) equation 6.3b can be used to compute the remaining time. As time crosses an element of time vector, we need a different expression to compute remaining time, which is derived as in the following:

Assume the accumulated time falls in between \( t_i \) and \( t_{i+1} \) such that \( t_i < (\Delta t + t) \leq t_{i+1} \). The new normalized distribution at time \( \Delta t + t \) is then:

\[
T_{NEW} = [t_{i+1} - (\Delta t + t), \ldots, t_N - (\Delta t + t)]
\]

\[
P_{NEW} = \left[ \frac{p_{i+1}}{1 - \sum_{j=1}^{i} p_j}, \ldots, \frac{p_N}{1 - \sum_{j=1}^{i} p_j} \right]
\]

The summation of the elements \( P_{NEW} \) must be unity. Elements of \( P_{NEW} \) are obtained by normalizing \( p_j \), for \( j = i + 1, \ldots, N \) by dividing the relevant elements of vector \( P \) by \( 1 - \sum_{j=1}^{i} p_j \).

\[
rm(t + \Delta t) = \sum_{k=i+1}^{N} \left( \frac{p_k}{1 - \sum_{j=1}^{i} p_j} \right)(t_k - (\Delta t + t))
\]
Finally, after some algebraic manipulation we obtain the following equation.

\[
rm(t + \Delta t) = \frac{rm(0) - \sum_{j=1}^{i} p_j t_j}{1 - \sum_{j=1}^{i} p_j} - (\Delta t + t) \quad \text{for} \quad t_i < (\Delta t + t) \leq t_{i+1} \quad (6.4)
\]

*Expected remaining time for case 2 is proved, Q.E.D.*

Note: Equation (6.4) can be used to calculate the expected remaining execution time at any accumulated execution time $\Delta t + t$ by simply identifying $t_i$ and $t_{i+1}$ where $t_i < (\Delta t + t) \leq t_{i+1}$. It is not required to keep track of the remaining time for accumulated time less than $\Delta t + t$.

**Residual Time Estimation Algorithm (ResTEA):**

We have designed an algorithm (ResTEA) that uses equation 6.2 to compute the expected remaining execution time of a job for scheduling application when job execution time is estimated by the $T$ and $P$ vector. We assume a dynamic quantum size which may change during scheduling operation. The steps of ResTEA and its complexity analysis will make it
clear that the approach does not require the re-calculation and storage of the remaining time vector.

**Description of ResTEA:**

The following description of the steps of “ResTEA” show how efficiently residual time calculation can be achieved by simply updating the prior results (e.g., differentially). Steps 1 through 3 are executed only once to initialize. Steps 4 through 7 can be executed repeatedly to compute remaining time at any value of accumulated time towards the execution of the job. Furthermore, using a differential calculation approach the time cost can be significantly reduced.

1. Set the vector size $N$, quantum size $\Delta t$, and accumulated time $t$.
2. Initialize $T$, $P$, and the product vector $PT$.
3. Compute, $rm(0)$, the expected remaining time at accumulated time 0 as $rm(0) = \sum_{i=1}^{N} p_{i}t_{i}$
4. Increment accumulated time $t$ by a quantum so $t = t + \Delta t$
5. If $t \leq t_{i}$ compute $rm(t) = rm(0) - t$
   Else
6. Search for the $t_{i}$ element such that $t_{i} < t \leq t_{i+1}$ and remember this $i$
7. Re-compute $rm(t)$, the remaining time at accumulated time $t$ using equation 6.2 and a differential approach.
   (a) Compute the following summation, $\sum_{j=1}^{i} p_{j}t_{j}$
   (b) Compute numerator of Eq. 6.2 as $num = rm(0) - \sum_{j=1}^{i} p_{j}t_{j}$
   (c) Compute denominator of Eq. 6.2 as $deno = 1 - \sum_{j=1}^{i} p_{j}$
   (d) Finally, compute $rm(\Delta t + t)$ as $rm(\Delta t + t) = \frac{num}{deno} (\Delta t + t)$

Next we provide a theorem and its proof on the complexity of using equation 6.2.
Theorem 2

The real time execution complexity of “ResTEA” in differentially computing expected remaining execution time, \( rm(t) \), is \( O(1) \) and is, therefore, independent of the size of the vector.

Proof:

Complexity

Along with the process of proving the theorem we analyze the complexity of equation (6.2). As each quantum is passed, \( \Delta t + t \) is computed. If the distribution of the time vector \( T \) does not change before and after the quantum i.e., in between accumulated time \( t \) and \( \Delta t + t \), equation (5.1) is still valid for computing \( rm(t) \). It requires only one subtraction to compute current \( rm(t + \Delta t) \) from previous \( rm(t) \). So, the cost is \( O(1) \). On the other hand, suppose the distribution of the time vector \( T \) changes if \( t_i \) falls in between \( t \) and \( t + \Delta t \). Now equation (6.2) is used. The numerator of the fraction part of equation (6.2) has two parts: \( rm(0) \) and \( \sum_{j=1}^{i} p_j t_j \). \( rm(0) \) is computed only once, during initialization and even before scheduling (offline). So the computation cost of \( rm(0) \) is not considered. (Note \( p_j t_j \) for \( i=1\ldots N \) is also done while computing \( rm(0) \)). To obtain the numerator, \( rm(0) - \sum_{j=1}^{i} p_j t_j \), only one term, \( p_j t_j \), is subtracted from the prior iteration’s numerator. Thus, the numerator has just one subtraction. To obtain \( 1 - \sum_{j=1}^{i} p_j \) in the denominator of the fraction part of equation (6.2), only one term, \( p_j \), is subtracted from, the previous iteration’s denominator. As in numerator the denominator just has one subtraction. Next is the division. Moreover, \( t \) is updated by \( \Delta t + t \) with just one addition. Finally, there is one more subtraction for \( \Delta t + t \). Thus, it is clear that \( rm(t + \Delta t) \) can be updated from \( rm(t) \) with just 5 operations altogether. Therefore,
the cost is $O(1)$. Thus, the cost of computing $rm(t + \Delta t)$ from $rm(t)$ is always $O(1)$ and is independent of the original vector size of the distribution. \textit{QED}

It is to be noted that in a dynamic situation, at the end of each quantum, the residual time can be updated from the previous quantum to the next quantum by having 5 extra operations (as shown above). Thus complexity of updating the residual time is $O(1)$. That allows the scheduling decision to be made quickly, which is desirable in practical real time applications.

\textbf{6.3 Examples of Test Results}

In the following example we show specific situations and explain the use of equation 6.1 or 6.2 to compute $rm(t)$ dynamically. Let the execution time and probability vector be as follows: $T = [1 \ 2 \ 4 \ 8]$ and $P = [.2 \ .4 \ .1 \ .3]$. Therefore, $rm(0) = \sum_{i=1}^{4} p_i t_i = 3.8$

Let the time slice, $\Delta t$, be .3 time-units. When accumulated time $t = \Delta t = .3$ or $t = 2 \times \Delta t = .6$ the distribution still remains the same as the original, since $t \leq t_i$. Therefore, using equation (5.1) one can obtain $rm(.3) = rm(0) - .3 = 3.5$. Similarly, using equation (6.2) once again $rm(.6) = rm(.3) - .3 = 3.2$. When accumulated time $t = 4 \times \Delta t = 1.2$, the element $t_i = 1$ of the original $T$ vector falls in between $t = 3 \times \Delta t = 0.9$ and $t = 4 \times \Delta t = 1.2$. The time vector reduces to $T_{new} = [.8 \ 2.8 \ 6.8]$. The new probability vector after normalization becomes $P_{new} = [.5 \ .125 \ .375]$. 
Whereas, at $t = 3 \Delta t = 0.9$ the distribution stays unchanged. Thus, $rm(.9)$ can be computed as:

\[ rm(.9) = rm(.6) - .3 = 2.9. \]

But at $t + \Delta t = 4 \times \Delta t = 1.2$ equation (6.1) will not produce the real expected remaining time. Instead equation (6.2) needs to be used.

Therefore, we have $rm(1.2) = \frac{3.8 - .2}{1 - .2} - 1.2 = 3.3$

Note: $rm(1.2)$ increases from $rm(.9)$. The reason behind this is that the distribution changes between time .9 and time 1.2 (as time passes). Our method of computing expected remaining time uses the new changed distribution with the new time vector as $T_{new}$. Figure 11 shows the behavior of the residual time for the given example. At time $t = 0$, $rm(0) = 3.8$. The graph of $rm(t)$ shows jumps at 3 different times, when the accumulated time $t$ crosses $t_i$ (element of $T$ vector, $t_i = 1, 2, 4$) for $i = 2 \ldots N - 1$. Because the distribution changes each time the accumulated time crosses $t_i$.

### 6.4 Simulation Results using Discrete Distributions

Here we present the simulation results comparing FCFS, RR and RTB scheduling algorithms for job distributions represented by the $T$ and $P$ vector. Aggregate job arrivals follow the Poisson process. Here the vector size is 10. The time vector components were sampled from a hyper exponential distribution with $C^2 = 10$. The probability vector components were sampled from a uniform distribution and normalized to achieve a sum of unity. The average $C^2$ over all the simulation was 8.3 and $\rho = .95$. The job deadline is 4. For the RTB policy the residual time is updated using ResTEA as described in section 6.2. Figure 12 depicts the fraction of the number of jobs that have finished by time $t$, after they arrived, where the times are normalized by dividing by their deadlines. There are three different graphs representing FCFS, RR and RTB algorithms. The graph with the knee represents the RTB algorithm and the knee indicates the
job deadlines. About 89.6% and 67.5% of the jobs meet their deadlines if they are serviced by RTB and RR policy, respectively.

Figure 11 Remaining time vs. accumulated time

Figure 12 Fraction of the number of jobs vs. turn around time, $t$, for FCFS, RR, and RTB algorithm.

Figure 13 Fraction of the number of jobs vs. turn-around time, $t$, for RTB algorithm at $\rho = .8$ for three different deadlines.
FCFS is not a processor-sharing algorithm. Thus when FCFS is used long jobs occupy the CPU for longer time. The shorter jobs have to wait till the completion of all jobs that arrived before them. On the other hand RR is a PS algorithm. Every job waiting to be served is allowed to execute for a time slice in a round robin fashion. RR performs better than FCFS. The results for discrete distributions are similar to those found in the earlier simulation study presented in section 5.3.

6.5 Experiments with Deadline

Figure 13 illustrates the scheduler behavior if the deadline is reduced. Sometimes, a user may attempt to get a better likelihood of making his/her deadline by requesting a smaller value for the deadline than what is needed (e.g., use a “safety factor”). In this simulation we assumed that the real deadline is 39, and all users requested an earlier value (either 9 or 4). Interestingly, the results show that when an earlier deadline is requested, a lower percentage of real deadline satisfaction is achieved than if the users did not use a safety factor. While this may seem counterintuitive, reflection reveals that the rules for handling post deadline cases will effectively lower the priority of processes that miss the artificial deadline but could still make the real deadline, helping to cause this effect.

7 Conclusions

We proposed and evaluated, through extensive simulations, a scheduling algorithm which uses job residual time computing time information for real time jobs. The results demonstrate significant performance improvement in percentage of jobs finishing before their deadlines, especially when job size variability is high ($C^2 = 10$, e.g.), if they are serviced using the RTB policy, in comparison with any of the other policies considered (RR, FCFS, EDF). Further, the RTB algorithm shows consistent improvement over a wide spectrum of
distributions, which shows the algorithm to have robustness. In the process of selecting the next job, RTB gives higher priority to jobs which are close to their deadline, as opposed to those which have already passed the deadline, which enables the approach to increase the probability of jobs meeting deadlines.

We also developed and tested (by simulation) a practical and time cost effective technique to compute job residual time differentially, when job service time distributions are represented by a discrete time-probability distribution, which can be used for virtually any case, including web server applications. These results support the practicality of using the approach in real-time applications.

8 References


Appendix:

a. **uniform distribution**: Here the task execution time follows the uniform distribution with
\[ pdf \ f(x) = \frac{1}{2T} \quad \text{for} \quad 0 \leq x < 2T. \]
The reliability function, \( R(x) = \frac{2T-x}{2T} \). The conditional reliability, \( R_{s}(x) = \frac{R(t+x)}{R(t)} = \frac{2T-(x+t)}{2T-t} \). The residual time \( rm(t) = \int_{0}^{\infty} \frac{R(t+x)}{R(t)} dx = \frac{2-t}{2} \)

b. **exponential distribution**: The well known exponential distribution has pdf, \( f(x) = \mu e^{-\mu x} \) and PDF \( F(x) = 1 - e^{-\mu x} \), the reliability function, \( R(x) = e^{-\mu x} \), the mean \( E(x) = \int_{0}^{\infty} xf(x) dx = \int_{0}^{\infty} x \mu e^{-\mu x} dx = \frac{1}{\mu} \). Where, \( \frac{1}{\mu} \) is referred to as the mean service time. Thus, the reciprocal of the mean service time, which is \( \mu \) in this case, is called the service rate. The residual time at time \( t \) can be shown to be equal to \( rm(t) = \frac{1}{\mu} \) (using equation 3.1). Due to the memory-less property of the exponential distribution, the residual time, \( rm(t) \), does not change with time. Note: all the other distributions examined here use exponential distribution as their base.

c. **two branch hyper-exponential**: The two branch hyper-exponential distribution with pdf, \( f(x) = p_1 \mu_1 e^{-\mu_1 x} + p_2 \mu_2 e^{-\mu_2 x} \) and reliability function, \( R(x) = p_1 e^{-\mu_1 x} + p_2 e^{-\mu_2 x} \) describes the service times of jobs whose execution takes one of the two branches. One branch has probability \( p_1 \) and the other has probability \( p_2 \) where \( (p_1 + p_2 = 1) \).

![Figure A1 : Two branch hyper-exponential](image)

There are three free parameters, namely \( p_1, \mu_1 \) and \( \mu_2 \), (where \( T_i = 1/\mu_i \)) are defined in [5] as follows:
\[ \gamma = \frac{C^2 - 1}{2}, \quad p_1 = \frac{2}{C^2 - 1}, \quad p_2 = 1 - p_1, \quad T_1 = \bar{x} \left(1 + \sqrt{p_2 \gamma / p_1} \right), \quad T_2 = \bar{x} \left(1 + \sqrt{p_1 \gamma / p_2} \right) \]
The hyper exponential distribution can be used when it is expected or desired that $C^2 > 1$.

Substituting the expression for $R(t)$ and $R(t + x)$ in equation 2.1 and after some algebraic manipulation we have

$$rm(t) = \frac{p_1 T_1 e^{-\lambda T_1} + p_2 T_2 e^{-\lambda T_2}}{p_1 e^{-\lambda T_1} + p_2 e^{-\lambda T_2}}$$

<table>
<thead>
<tr>
<th>Distributions</th>
<th>rm(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform</td>
<td>$(2 - \tau)/2$</td>
</tr>
<tr>
<td>Exponential</td>
<td>$1/\mu$</td>
</tr>
<tr>
<td>Two branch hyper-exponential</td>
<td>$\frac{p_1 T_1 e^{-\lambda T_1} + p_2 T_2 e^{-\lambda T_2}}{p_1 e^{-\lambda T_1} + p_2 e^{-\lambda T_2}}$</td>
</tr>
<tr>
<td>Erlangian-3</td>
<td>$\frac{1}{\mu} \left[ \frac{3 + 2 \sigma + (\eta \sigma)^2}{1 + \sigma + (\eta \sigma)^2} \right]$</td>
</tr>
<tr>
<td>4 stage hyper Erlangian</td>
<td>$\frac{p_1 T_1 [2 + \lambda T_1 e^{-\lambda T_1}] + p_2 T_2 [2 + \lambda T_2 e^{-\lambda T_2}]}{p_1 (1 + 1/T_1) e^{-\lambda T_1} + p_2 (1 + 1/T_2) e^{-\lambda T_2}}$</td>
</tr>
<tr>
<td>3 branch hyper-exponential</td>
<td>$\frac{p_1 T_1 e^{-\lambda T_1} + p_2 T_2 e^{-\lambda T_2} + p_3 T_3 e^{-\lambda T_3}}{p_1 e^{-\lambda T_1} + p_2 e^{-\lambda T_2} + p_3 e^{-\lambda T_3}}$</td>
</tr>
<tr>
<td>Power tail</td>
<td>$1 + \frac{t}{\alpha - 1}$</td>
</tr>
</tbody>
</table>

Table A1 Residual time expressions of various distributions.

d. **Erlangian-n:** When the service time distribution is represented by $n$ identical exponential servers in cascade (one at a time), it is termed as Erlangian-$n$.

![Figure A2. Erlangian-n distribution](image-url)
The pdf of Erlangian-n is known to be, \( f(x) = \frac{\mu^n x^{n-1}}{(n-1)!} e^{-\mu x} \). In this paper we have considered an Erlangian-3 whose reliability function \( R(x) = \left[ 1 + \mu x + \frac{(\mu x)^2}{2} \right] e^{\mu x} \).

The residual time at time \( t \),

\[
\frac{1}{\mu} \left[ \frac{3 + 2 \mu t + (\mu t)^2}{1 + \mu t + (\mu t)^2} \right]
\]

**e. four stage Hyper Erlangian**: A four stage Hyper Erlangian distribution has two branches with probability \( p_1 \) and \( p_2 \), respectively. Each branch has two identical exponential servers in cascade (Erlangian-2). The service rates of the two servers in branch \( i \) is \( \mu_i \) each, where \( i = 1, 2 \).

The reliability function \( R(x) = p_1 (1 + \mu_1 x) e^{-\mu_1 x} + p_2 (1 + \mu_2 x) e^{-\mu_2 x} \).

There are three free parameters, namely \( p_1 \), \( \mu_1 \) and \( \mu_2 \), where \( T_i = 1/\mu_i \). For the definition of those parameters we refer [5].

**f. three-branch hyper-exponential**: A three-branch hyper-exponential distribution with pdf, \( f(x) = p_1 \mu_1 e^{-\mu_1 x} + p_2 \mu_2 e^{-\mu_2 x} + p_3 \mu_3 e^{-\mu_3 x} \) describes the service times of jobs whose execution takes one of three exponential branches, with service rates as \( \mu_1, \mu_2 \) and \( \mu_3 \), respectively. \( p_1 \), \( p_2 \) and \( p_3 \) are the probabilities of each of the branches, respectively, with \( p_1 + p_2 + p_3 = 1 \). The reliability function \( R(x) = p_1 e^{-\mu_1 x} + p_2 e^{-\mu_2 x} + p_3 e^{-\mu_3 x} \).

**g. power tail**: A random variable \( X \) has a power tail distribution if its reliability function \( R(x) \) satisfies the following: \( R(x) = cx^{-\alpha} \), where \( \alpha \) and \( c \) are positive constants, that is, the reliability function drops off as a power of \( x \) for large \( x \). The tail refers to the behavior of \( R(x) \) when \( x \) is very large. Considering the asymptotic property of the reliability function one can get the density function as \( f(x) = -\frac{R(x)}{dx} = \frac{c \alpha}{x^{\alpha+1}} \). For a given \( \alpha \) it follows...
\( E(X^l) = \int x^l f(x)dx = \infty \) for \( l \leq \alpha \). Therefore, if \( 0 < \alpha \leq 1 \), \( E(X) = \infty \), a classical queueing analysis falls apart. The most encountered situation is \( 1 < \alpha \leq 2 \) for which the random variable \( X \) has a finite mean but infinite variance. Here we have considered \( \alpha = 1.4 \). The well-known power-tail distribution is also known as the Pareto distribution for which the reliability function is defined as follows:

\[
R(x) = \frac{1}{(1 + ax)^\alpha} \quad \text{for} \quad t \geq 0 \text{ and } a > 0.
\]

The reliability function with mean 1 can be found to be:

\[
R(x) = \frac{1}{(1 + x/\alpha)^\alpha}
\]

The residual time

\[
rm(t) = \int_0^\infty \frac{R(t + x)dx}{R(t)}
\]

\[
= \frac{1}{(1 + at)^\alpha} \int_0^\infty \frac{1}{(1 + a(t + x))^{\alpha}}dx \quad \text{by substitution}
\]

After integrating by parts we obtain the expected time remaining given the job has received service for time \( t \) as;

\[
rm(t) = 1 + \frac{t}{\alpha - 1}
\]