View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Illinois Digital Environment for Access to Learning and Scholarship Repository

# THE DELAY COMPOSITION THEOREM ON PIPELINE SYSTEMS WITH NON-PREEMPTIVE PRIORITY VARYING SCHEDULING ALGORITHMS

BY

YI LU

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Adviser:

Professor Tarek Abdelzaher

## Abstract

The delay composition theorem, by taking into account the fact that pipeline systems allow concurrent execution, expresses the upper-bounded delay of a real-time task as the sum of two summations, where the first one is linear to the number of stages of the system, and the second one is linear to the number of tasks running on the system. The schedulability analysis based on delay composition theorem performs better than traditional analysis techniques. In this paper we break one assumption that has been hold by previous works on delay composition theorem, namely each task has the same relative priority across all stages. We extend the theorem to pipeline systems running non-preemptive scheduling algorithm which may assign different relative priorities to a task on different stages.

## TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION........................................1

CHAPTER 2: RELATED WORKS........................................3

CHAPTER 3: SYSTEM MODEL........................................6

CHAPTER 4: DELAY IN A PRIORITY VARYING PIPELINE (NON-PREEMPTIVE)....8

CHAPTER 5: REDUCTION TO UNIPROCESSOR........................................15

CHAPTER 6: EXAMPLE........................................17

CHAPTER 7: EVALUATION........................................20

CHAPTER 8: CONCLUSION........................................30

REFERENCES........................................31

# CHAPTER 1

# INTRODUCTION

Understanding the behavior of a real-time task and being able to bound its end-to-end delay are fundamental topics in real-time embedded systems. Today as the scale of real-time systems increases with more processing units and tasks involved in, the task of predicting the behavior of real-time tasks becomes even more challenging while urgent. Consider a distributed embedded system comprising of several autonomous processing units, or components, each of which serves certain functionality. A real-time task may consist of a sequence of subtasks which require processing on different components. Since these components are autonomous, each of them could use different algorithms to schedule the subtasks. Therefore, a task may have different priorities on different components. From the perspective of a real-time task, it is executing on a pipeline system in which its priority varies across the stages.

The delay composition theorem (DCT) aims to bound the end-to-end delay experienced by any task on a pipeline as a function of the execution times of higher-priority tasks [Praveen07]. Taking into account the fact that the pipeline system allows concurrent execution, DCT derives and expresses the upper-bounded delay of a task as the sum of two summations, where the first one is proportional to the number of stages of the pipeline, and the second one is proportional to the number of tasks in the set. Although the delay composition theorem is a powerful tool for real-time scheduling analysis, one assumption it makes, that each task is assigned the same relative priority across all stages in the pipeline, limits the scope of application of the theorem. This assumption is also held by later extensions of the theorem to acyclic [Praveen ECRTS08] and cyclic [Praveen 09] distributed systems.

In this paper, we will release this constraint and extend the delay composition theorem to pipeline systems running non-preemptive scheduling algorithms where the relative priority of a task might change across stages. In this paper, we call a scheduling which may assign different priorities to a task on different stages a *priority varying* scheduling algorithm. We

assume that different invocations of a task have the same priority on each stage, and how a task's priority changes across stages is pre-known. No assumption is made on the periodicity of the task set.

The rest of this paper is structured as following: in Chapter 2 we briefly review several related works; in Chapter 3 we describe the system model; in Chapter 4 we present our extended theorem and its proof; in Chapter 5 we show the systematic reduction of a pipeline running non-preemptive priority varying scheduling algorithm to an equivalent uniprocessor to which schedulability analysis techniques developed for uniprocessor can be applied; Chapter 6 presents a specific example of applying the theorem developed in this paper to bound the delay of tasks executing on a pipeline system running non-preemptive priority varying scheduling algorithm; Chapter 7 presents the evaluation of our extended delay composition theorem; we discuss future work and conclude this paper in Chapter 8.

# CHAPTER 2

# RELATED WORKS

## 2.1 A Brief History of Delay Composition Theorem

[1] gives the delay composition theorem for a pipeline system with the assumption that the pipeline is running a preemptive scheduling algorithm with each task assigned the same relative priority across all stages. Compared to offline schedulability tests which break the end-to-end deadline into per-stage deadlines and apply tests for uniprocessor on each stage, the delay composition theorem is much less pessimistic. This is because the worst case assumption made by these scheduling analysis, namely the task of interest and all the other tasks with higher priority will arrive at each stage simultaneously (therefore the higher priority tasks will execute first and delay the task of interest), is not true for every stage. The delay composition theorem was extended in [3] to pipelines running non-preemptive scheduling algorithm.

[4] extends the delay composition theorem to distributed systems that are represented by directed acyclic graphs (DAGs). To derive the expression of delay in a distributed system, the paper uses a reduction-based method: it transforms the distributed system into a pipeline system where the worst case delay of a task cannot be lower than that in the original system. It then applies the theorem already developed in [1] to bound the delay of a task in the reduced pipeline system. Two different expressions are derived for distributed systems running preemptive or non-preemptive scheduling algorithms, respectively. [2] presents the delay composition algebra, which systematically reduces a distributed system (DAG) to an equivalent uniprocessor. The reduced system can be analyzed using existing scheduling techniques for uniprocessors. [5] further extends the delay composition theorem to distributed systems with task graphs that may contain cycles.

In all the works on delay composition theorem so far, it was assumed that each task is assigned the same priority across all stages.

## 2.2 Other Related Works

Real-time scheduling on distributed systems consisting of independent components has been addressed by [8] [9]. The fundemental approach is to divide the end-to-end deadline of tasks into per-stage deadlines and then apply schedulability tests for uniprocessor on each stage independently to determine if the subtasks are schedulable. A set of real-time tasks is declared to be schedulable if all subtasks on all stages are schedulable. In this paper, we call this type of analysis techniques *the traditional per-stage analysis*. The problem of traditional per-stage analysis is that it does not take into account the inherent parallelism in the execution of different stages and assumes a worst arrival pattern on every stage. Therefore, the per-stage analysis will be pessimistic as the number of stages increases.

Holistic analysis [10] uses release jitter and offset of tasks to characterize their arrival patterns on each stage in the distributed system. Particularly, it takes the response time on one stage as jitter for the subsequent stage. By applying this process on each stage successively, an upper bound of delay for each task invocation can be derived. [11] extends the holistic analysis based on response time analysis with the ability of managing heterogeneous distributed systems of components using either fixed priority scheduling (FP) or earliest deadline first (EDF). The paper also integrates optimization techniques for the assignment of priorities [12] and scheduling deadlines [13] into one algorithm called HOSPA (Heuristic Optimized Scheduling Parameters Assignment), which allows the optimization of heterogeneous systems. Unlike the traditional per-stage method, holistic analysis does not divide end-to-end deadlines of tasks into per-stage deadlines. However, by taking the response time on previous stage as the jitter, it implicitly considers that a job is delayed by the same higher-priority jobs on each stage of the pipeline. For this reason, the holistic analysis becomes more pessimistic for pipelines with large number of stages.

Real-time calculus [14] extends the concepts in network calculus [15] to real-time schedul-

ing. This technique characterizes a processing unit by a request function and a capability function. It then derives a request curve that specifies the maximum incoming request within a time interval and a deliver curve that specifies the minimum (or guaranteed) processing within a time interval. By comparing the remaining capability of a processing unit and the demand of a real-time task, it determines whether deadline of the task is met or not.

# CHAPTER 3
# SYSTEM MODEL

Consider a multistage data processing pipeline. A set of real-time tasks arrive at the pipeline and require execution on the system. These tasks can be either periodic or aperordic. Each task has its invocations, which are called *jobs*. In the system, it is possible that different jobs have the same priority, for instance, when they are the invocations of a common task. In this case, we assume there is a tie-breaking rule (e.g. FIFO) among such jobs, and taking that into account, we can still assume that each individual job has its own priority.

By the definition of pipeline, we assume that all jobs will execute on all stages of the pipeline in the same order. Let the number of stages of the pipeline be $N$, then each job executes in the sequence of stage 1, stage 2,. . . , stage $N$. Each job $i$, or $J_i$, arrives at the system at time $A_i$, and has a relative end-to-end deadline $D_i$, which means that $J_i$ has to leave the system by time $A_i + D_i$. Assume that the pipeline system is running a non-preemptive priority varying scheduling algorithm so that each task's priority may change across stages. Let $\rho_{i,j}$ denote the priority of $J_i$ on stage $j$, and its value is between 1 and the number of tasks running on the pipeline. Assume a smaller integer value indicates a higher priority, therefore 1 implies the highest priority. Let $C_{i,j}$ denote the computation time of $J_i$ on stage $j$, also referred as *stage execution time*. Assume that different invocations belonging to each task have the same per-stage priority and stage execution time.

Without loss of generality, let $J_t$ denote the task whose delay we want to bound. Note that in order to reduce the ambiguity that might be raised by naming a task in such a system using explicit integers, in this paper we are using alphabet letters to denote jobs. Let $S$ denote the set of all jobs $J_i$ whose execution intervals $[A_i, A_i + D_i]$ overlap with that of $J_t$ ($[A_t, A_t + D_t]$). S includes $J_t$. Note that a job whose arrival is after the deadline of $J_t$ or whose deadline is before the arrival of $J_t$ has no effect on the delay of $J_t$, and therefore should be ignored.

Let $C_{i,max}$ denote the maximum stage execution time of $J_i$ across all stages, i.e.

$$C_{i,max} = \max_{1 \le j \le N} C_{i,j}$$

Now we are ready to derive an upper bound of delay for $J_t$.

# CHAPTER 4

# DELAY IN A PRIORITY VARYING PIPELINE (NON-PREEMPTIVE)

In this section we derive an upper bound of end-to-end delay a real-time job experiences in a pipeline system running non-preemptive priority varying scheduling algorithm.

To bound the end-to-end delay of $J_t$, let's define the notion *busy execution trace*, which is a sequence of continuous intervals of continuous processing on successive stages of the pipeline that collectively add up to the total delay of $J_t$ [5]. Note that there may exist several execution traces satisfying this definition. In order to reduce the number of different possibilities, let's further require that the processing interval of the trace ends at a job boundary on each stage. This leads to the formal definition of a busy execution trace as following:

**Definition 1**: A *Busy Execution Trace* through all stages of the pipeline is a sequence of contiguous intervals starting with the arrival of $J_t$ on stage 1 and ending with the finish time of $J_t$ on stage $N$, where ($i$) each interval represents a stretch of continuous processing on one stage $j$ of the pipeline, ($ii$) the interval on stage $j$ ends at the finish time of some job on the stage, ($iii$) successive intervals are contiguous in that the end time of one interval on stage $j$ is the start time of the next interval on stage $j+1$, and ($iv$) successive intervals execute on consecutive stages of the pipeline.

Figure 1 illustrates the concept of busy execution trace. Three execution traces (red, green, and blue) are presented in the figure. Among these traces, only the red and the green ones satisfy the definition of busy execution trace. The blue trace fails to satisfy the definition since it runs into idle time and ends before the finish time of $J_t$.

Let's define the end section of a job as following:

**Definition 2**: *The End Section of* $J_i$ (with respect to $J_t$), where $i \neq t$, is the last execution section of $J_i$ that precedes section of $J_t$ in the same continuous execution section. *The End Section of* $J_t$ is its execution section on the $N$-th stage.

In Figure 2, each job's end section is marked with a red diamond. Note that in this paper, we call those sections of a job before its end section *the non-end sections* of this job (with respect to $J_t$). Those execution sections of a job after its end section are NOT its non-end sections. For instance, in Figure 2 we can see that the end section of $J_b$ is on stage 2. Therefore, the execution section of $J_b$ on stage 1 is its non-end section. $J_b$'s execution section on stage 3 is not its non-end section; we simply do not care about it because it can have no effect on the delay of $J_t$.

Let's now define a special case of busy execution trace, the *the earliest traversal trace*, as following:

**Definition 3**: An *Earliest Traversal Trace* is a busy execution trace in which the end of an interval on stage $j$ coincides with the finish time of the *first* non-end section of a job.

The red trace in Figure 1 is the earliest traversal trace. Observe that bounding the end-to-end delay of $J_t$ is equivalent to bounding the length of earliest traversal trace, which can be divided into two complementary categories: ($i$) contribution of non-end sections and ($ii$) contribution of end sections. The following two lemmas give an upper bound for each of them respectively.

**Lemma 1:** The total contribution of non-end sections to the earliest traversal trace is upper-bounded by

$$\sum_{j=1}^{N-1} \max_{J_i \in S} C_{i,j} \qquad \textit{(stage-addative)}$$

*Proof.* Each stage other than the last stage has only **one** *first* non-end section (there is no non-end section at stage $N$). Therefore, there are $N-1$ terms of non-end sections in the earliest traversal trace. By definition, The execution time of the non-end section on stage $j$ is less than or equal to $\max_{J_i \in S} C_{i,j}$. The execution time of all these $N-1$ terms is therefore upper-bounded by $\sum_{j=1}^{N-1} \max_{J_i \in S} C_{i,j}$. □

**Lemma 2:** The total contribution of end sections to the earliest traversal trace is upper-bounded by

$$\sum_{J_i \in S} C_{i,max} \qquad \textit{(job-addative)}$$

*Proof.* Each job has only **one** end section (with respect to $J_t$). By definition, the execution time of $J_i$'s end section is less than or equal to $C_{i,max}$. The execution time of all jobs' end sections is therefore upper-bounded by $\sum_{J_i \in S} C_{i,max}$. □

The following theorem gives the upper bound of $J_t$'s delay in a pipeline running non-preemptive priority varying scheduling algorithm.

**Theorem 1.** *In a $N$-stage pipeline system running a non-preemptive priority varying scheduling algorithm which may assign different relative priority to a job across different stages, the end-to-end delay of $J_t$ can be composed from the execution parameters of other jobs that delay it (denoted by set $S$) as following:*

$$Delay(J_t) \leq \sum_{j=1}^{N-1} \max_{J_i \in S} C_{i,j} + \sum_{J_i \in S} C_{i,max} \tag{1}$$

*Proof.* By adding the contributions of non-end sections (Lemma 1) and end sections (Lemma 2), we get an upper bound of length of the earliest traversal trace, which, by definition, is equivalent to an upper bound of the end-to-end delay of $J_t$. □

This formula shows that bounding the end-to-end delay of a job $J_t$ in a pipeline with non-preemptive priority varying scheduling policy is equivalent to bounding the delay of the same job in a pipeline running non-preemptive scheduling algorithm where each job has the same priority across all stages, with $J_t$ being the lowest priority job in the system. This can be pessimistic, since $J_t$ may now be delayed by any other job, especially jobs with long relative deadline, whose average stage execution time tend to be long as well.

However, if we restrict the range that a job's priority can change from its base priority, then the behavior of jobs can be better predicted. In this case we get a less pessimistic bound than formula 1.

Consider a pipeline system running a set of tasks with a non-preemptive policy. Let $\rho_i$ be the base priority of $J_i$, which can be determined by $J_i$'s period, in the case that all tasks

are periodic and the pipeline is using rate monotonic scheduling. Let $x$ denote the maximum value by which a job's priority could increase or decrease from its base priority, and assume that it is the same for all tasks. Then each job $J_i$'s per-stage priority ranges from $\rho_i - x$ to $\rho_i + x$, inclusively.

To bound the end-to-end delay of $J_t$, we can now divide S into two subsets $S_H$ and $S_L$ (Figure 3), where $S_H$ includes all jobs whose base priorities are in the range $[1,\, \rho_t + x]$, and $S_L$ includes all jobs whose base priorities are in the range $[\rho_t - x,\, n]$, where $n$ is the total number of real-time tasks. Note that $S_H$ and $S_L$ overlap on the jobs whose base priorities are in $[\rho_t - x,\, \rho_t + x]$, so that all possible delay and block cases, including the worst one, are covered.This gives the following corollary:

**Corollary 1**. *In an $N$-stage pipeline system using a non-preemptive scheduling policy with a restricted amount of priority change allowed, the end-to-end delay of a job $J_t$ can be composed from the execution parameters of other jobs that delay it (denoted by set S) as following:*

$$Delay(J_t) \leq \sum_{J_i \in S_H} C_{i,max} + \sum_{j=1}^{N-1} \max_{J_i \in S_H} C_{i,j} + \sum_{j=1}^{N} \max_{J_i \in S_L} C_{i,j} \tag{2}$$

*Proof.* This follows from the non-preemptive pipeline delay composition theorem in [3], with $J_t$ has a fixed priority of $\rho_t$ and each job $J_i$ in $S_H$ and $S_L$ has a fixed priority of $\rho_i$ across all stages. □

For large value of $x$ such that $S_H = S_L = S$, which means there is no restriction on priority change, formula (2) is even more pessimistic than (1) because of the extra term $\sum_{j=1}^{N} \max_{J_i \in S_L} C_{i,j}$ at the end. However, for small value of $x$, we could expect **Corollary 1** to outperform **Theorem 1**. We will see this later in the evaluation section.

Figures

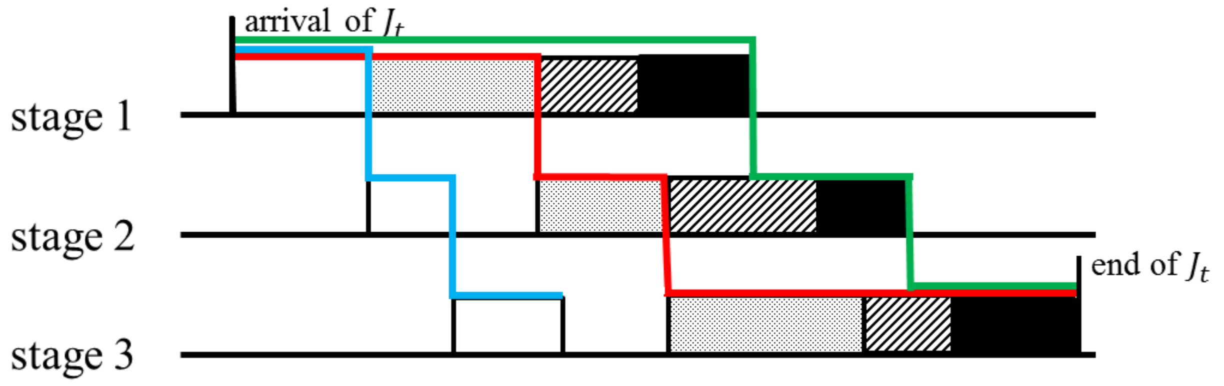


**Figure 1. Three Traces**

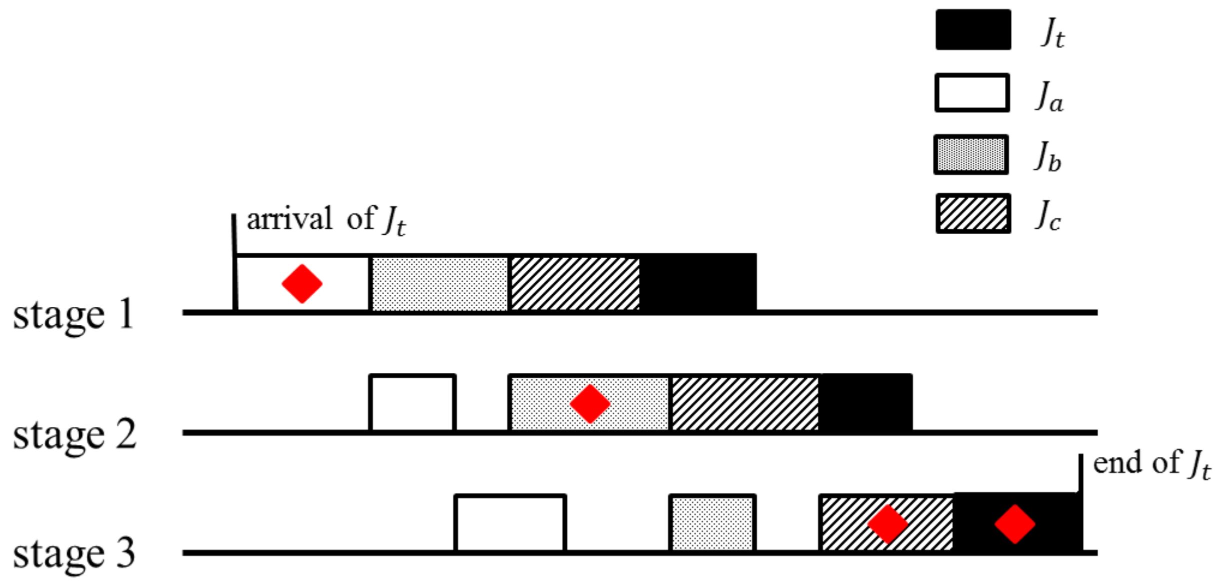


**Figure 2. End/Non-End Sections**

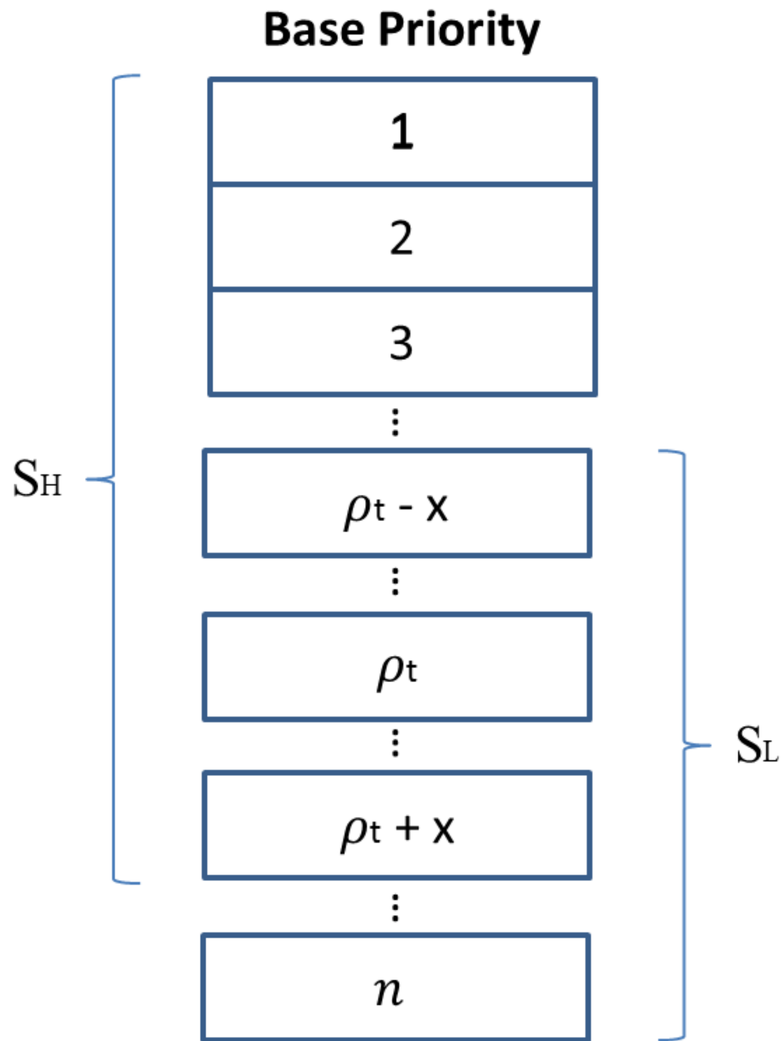


**Figure 3.** $S_H$ **and** $S_L$

# CHAPTER 5

# REDUCTION TO UNIPROCESSOR

Applying the extended delay composition theorem to a pipeline running non-preemtive priority varying scheduling algorithm, the schedulability analysis of the pipeline can be reduced to that of an equivalent uniprocessor system with preemptive scheduling algorithm as following:

• Each job $J_i$ other than $J_t$ is replaced by a uniprocessor job $J_i^*$ of higher priority with execution time $C_i^* = C_{i,max}$ (the maximum stage execution time of $J_i$) and the same deadline as $J_i$ in the original system.

• $J_t$ is replaced by a uniprocessor job $J_t^*$ of the lowest priority with execution time $C_t^* = C_{t,max} + \sum_{j=1}^{N-1} \max_{J_i \in S} C_{i,j}$ (the maximum stage execution time of $J_t$ plus the stage-addative term) and the same deadline as $J_t$ in the original system.

The above transformation is based on **Theorem 1** and note that $J_t^*$ in the reduced system has the lowest priority since it can potentially be delayed by all the other jobs in the original pipeline system. For the special case where only a restricted amount of priority change is allowed, the reduction to a uniprocessor with preemptive scheduling can be done based on **Corollary 1** as following:

• Each job $J_i$ in $S_H$ $(i \neq t)$ is replaced by a uniprocessor job $J_i^*$ with priority $\rho_i^* = max(\rho_t, \rho_i)$, i.e. the higher one of $J_i$ and $J_t$'s base priories, and execution time $C_i^* = C_{i,max}$ (the maximum stage execution time of $J_i$) and the same deadline as $J_i$ in the original system.

• $J_t$ is replaced by a uniprocessor job $J_t^*$ with priority $\rho_t^* = \rho_t$ and execution time $C_t^* = C_{t,max} + \sum_{j=1}^{N-1} \max_{J_i \in S_H} C_{i,j} + \sum_{j=1}^{N} \max_{J_i \in S_L} C_{i,j}$ (the maximum stage execution time of $J_t$ plus the last two terms in formula 2) and the same deadline as $J_t$ in the original system.

If $J_t^*$ is schedulable in the reduced uniprocessor system, so is $J_t$ in the original pipeline system, since the upper bound of delay for $J_t$ is less than the delay of $J_t^*$, which is at least the sum of execution times of all higher-priority jobs $J_i^*$ and itself. Note that we reduce the original pipeline system running a non-preemptive algorithm to a uniprocessor system with

a preemptive algorithm, since we only care about relating the delays of $J_t$ and $J_t^*$ in their corresponding systems.

For periodic tasks, jobs in the reduced uniprocessor that are invocations of the same task can be grouped together when doing analysis. After the reduction, schedulability tests for single processor such as the exact test [7] can be applied to the resultant uniprocessor system. For a set of $n$ periodic tasks with period equal to end-to-end deadline, the exact test for the pipeline system without restriction on priority change would be:

$$R_t^{(0)} = C_t^*$$

$$R_t^{(k)} = C_t^* + \sum_{i=1, i \neq t}^{n} \left\lceil \frac{R_t^{(k-1)}}{P_i} \right\rceil C_i^*$$

In the case that a restriction on range of priority change is enforced, the recursive term would become:

$$R_t^{(k)} = C_t^* + \sum_{i=1, i \neq t}^{m} \left\lceil \frac{R_t^{(k-1)}}{P_i} \right\rceil C_i^*$$

where $m = |S_H| \leq n$, and $S_H$ is the set of all jobs whose base priorities are in the range $[1, \rho_t + x]$. In both cases, $J_t^*$ is schedulable if we reach $R_t^{(k)} = R_t^{(k-1)} < D_t$. Otherwise it is not schedulable.

Other schedulability tests for singe processor can be applied to the pipeline system in a similar manner. We denote the process of reducing a pipeline system to an equivalent uniprocessor and then applying analysis techniques for uniprocessor to determine the schedulability of a task set *the Meta-Schedulability Test*.

# CHAPTER 6

# EXAMPLE

In this section we will use a simple but illustrative example to show that the delay composition theorem based meta-schedulability test can result in a tighter delay bound for real-time tasks running on a distributed pipeline of autonomous components, each of which uses non-preemptive fixed priority scheduling policy.

Consider a three-stage pipeline with two periodic tasks $T_a$ and $T_b$ (Figure 4). The task on top in a stage represents the higher-priority task in that particular stage, i.e. $T_a$ has a higher priority in stage 1 and 3 while a lower priority in stage 2. Let the period of these two tasks, $P_a$ and $P_b$, be equal to their end-to-end deadline, which is 5 units for both of them. For simplicity, let the computation time for each task on every stage be 1 unit. The objective is to estimate the delay and schedulability of each task.

Let us first use holistic analysis [11] to analyze this system. Starting from stage 1, we apply the response time analysis for each task to compute its delay; the task delay computed in one stage will be its jitter in the next stage. The jitters for both tasks, $j_a$ and $j_b$, are initialized to zero.

•*Stage 1* $(\rho_a > \rho_b)$:

$j_a = 0, j_b = 0$

$w_a^{(0)} = C_{a,1} + B_{a,1} = 1 + 1 = 2$

$w_a^{(1)} = C_{a,1} + B_{a,1} = 1 + 1 = 2 = w_a^{(0)}$

Therefore, $R_a = j_a + w_a^{(1)} = 2$

$w_b^{(0)} = C_{b,1} + B_{b,1} = 1 + 0 = 1$

$w_b^{(1)} = C_{b,1} + B_{b,1} + \left\lceil \frac{j_a + w_b^{(0)}}{P_a} \right\rceil C_{a,1} = 1 + \lceil \frac{0+1}{5} \rceil * 1 = 2$

$w_b^{(2)} = 1 + \lceil \frac{0+2}{5} \rceil * 1 = 2 = w_b^{(1)}$

Therefore, $R_b = j_b + w_b^{(2)} = 2$

•*Stage 2* $(\rho_b > \rho_a)$:

$j_a = 2, j_b = 2$

$w_a^{(0)} = C_{a,2} + B_{a,2} = 1 + 0 = 1$

$w_a^{(1)} = C_{a,2} + B_{a,2} + \left\lceil \frac{j_b + w_a^{(0)}}{P_b} \right\rceil C_{b,2} = 1 + \lceil \frac{2+1}{5} \rceil * 1 = 2$

$w_a^{(2)} = 1 + \lceil \frac{2+2}{5} \rceil * 1 = 2 = w_a^{(1)}$

Therefore, $R_a = j_a + w_a^{(2)} = 4$

$w_b^{(0)} = C_{b,2} + B_{b,2} = 1 + 1 = 2$

$w_b^{(1)} = 1 + 1 = 2 = w_b^{(0)}$

Therefore, $R_b = j_b + w_b^{(1)} = 4$

•*Stage 3* ($\rho_a > \rho_b$):

$j_a = 4, j_b = 4$

$w_a^{(0)} = C_{a,3} + B_{a,3} = 1 + 1 = 2$

$w_a^{(1)} = C_{a,3} + B_{a,3} = 1 + 1 = 2 = w_a^{(0)}$

Therefore, $R_a = j_a + w_a^{(1)} = 6 > 5$

Since the response time of $T_a$ exceeds its deadline, we declare that the task set is not schedulable. Now let's use the delay composition theorem based meta-schedulability test to analyze this system. To test the schedulability of $T_a$, we need to first reduce the pipeline into a uniprocessor in which $T_a^*$ has the lowest priority. Following the reduction procedure in section 5, we have a reduced uniprocessor running preemptive policy with 2 tasks: a higher priority task $T_b^*$, with execution time $C_b^* = C_{b,max} = 1$ and period (equal to deadline) $P_b^* = 5$, and a lower priority task $T_a^*$, with $C_a^* = C_{a,max} + \sum_{j=1}^{2} \max_{i \in \{a,b\}} C_{i,j} = 1 + 2 = 3$ and $P_a^* = 5$. Applying the response time analysis on the reduced uniprocessor, we have:

$R_a^{(0)} = C_a^* = 3$

$R_a^{(1)} = C_a^* + \left\lceil \frac{R_a^{(0)}}{P_b^*} \right\rceil C_b^* = 3 + 1 = 4$

$R_a^{(2)} = 3 + \lceil \frac{4}{5} \rceil * 1 = 4 = R_a^{(1)} < 5$

$T_a^*$ meets its deadline in the reduced uniprocessor, so does $T_a$ in the original pipeline system. Similarly, by reducing the pipeline into a uniprocessor with $T_b^*$ being the lower priority task and then applying response time analysis, we will find that $T_b^*$ and $T_b$ are schedulable in their corresponding systems.

Figures

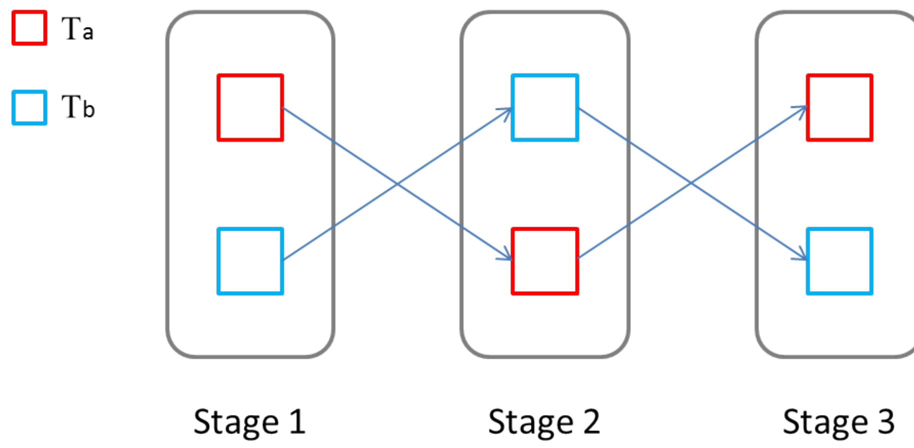


**Figure 4. A Three-Stage Pipeline with Two Real-Time Tasks**

# CHAPTER 7

# EVALUATION

In this section we will evaluated the performance of our meta-schedulability analysis using simulation. The meta-schedulability analysis first reduces the pipeline to an equivalent uniprocessor based on section 6, and then applies schedulability tests for single processor to analyze schedulability of the task set. In our paper we will use the response time analysis (RTA). We will compare the performance of our technique against that of holistic analysis and traditional per-stage analysis, each of which is based on RTA. For each analysis method we implement an admission controller. When a new task arrives, the controller will first tentatively add it to the task set (starting from empty). It will then determine if the current set is schedulable using corresponding schedulability test. If the set is schedulable after the new task arrives, the controller will admit the new task; otherwise the new task will be dropped. The default pipeline system (labeled as 'simulation') in which there is no admission control will just keep adding new tasks until actual deadline missing occurs.

In the rest of this section, we will refer to the average per-stage utilization by the term utilization. Each point in the figures represent the average values obtained from 100 executions of the simulator, with each execution running until any job misses its deadline. In the simulation, we assume all tasks are periodic and each task's period is equal to its end-to-end deadline, which is chosen as $10^{\beta}a$ simulation seconds, where $\beta$ is uniformly varying between 0 and DR (deadline ratio parameter), and $a = 500 * N$ , where $N$ is the number of stages of the pipeline. By choosing deadlines of tasks in this way, the maximum ratio of two tasks' deadlines will be $10^{DR}$. Therefore, the larger the value of DR, the wider the range of tasks' deadlines. The default value of DR is 1.0, which means that task deadlines can differ as much as 10 times. The priority of each task may vary across stages of the pipeline. By default, the per-stage priority of a task is assigned to be a random integer between 1 and n, where n is the number of all tasks. In the special case where each job's priority can only change a restricted amount, we will assign tasks' base priorities according to rate monotonic

scheduling, so that a task with shorter period will have a higher base priority. We assume that different invocations of a task have the same priority on each stage. The stage execution time of each task is chosen based on the task resolution parameter, which measures the ratio of the total execution time of a task over all stages to its deadline. The stage execution time of a task is chosen from a uniform distribution with mean equal to $\frac{D\tau}{N}$, where D is the deadline of the task and $\tau$ is the task resolution (default value being $1:50$). The stage execution time ranges from 10 percent on both sides of the mean.

Let's first see the performance of our analysis in the default case where there is NO restriction on how jobs' per-stage priorities can change.

Figure 5 shows the average per-stage utilization for different number of stages ($N$) of the pipeline with DR equal to 1.0. The utilization for traditional per-stage analysis decreases as the number of stages increases. This is because it assumes a worst-case job arrival pattern at each stage. Specifically, each job at each stage is assumed to be delayed by all jobs with higher priority on that stage, and also delayed by one lower priority job with the maximum execution time on the stage. The holistic analysis does not divide the end-to-end deadline of tasks into per-stage deadline. However, by taking the response time up to current stage as the jitter for the next one, it implicitly assumes that a job is delayed by the same job on every stage of the pipeline where that job has a higher priority. Therefore, the utilization for holistic analysis decreases when there are more stages in the pipeline. For different number of pipeline stages, the utilization for the meta-schedulability test is higher than those of the other two tests and is almost the same across different $N$. This is because the meta-schedulability test takes into account the overlap of execution on different pipeline stages, a feature of the delay composition theorem.

We conducted experiments to see how different values of deadline ratio can affect the utilization of different analysis tools, and plotted the curses along with that of the default system without any admission control in Figure 6. In these experiments all system parameters are the same except the deadline ratio parameter (note that a larger value of DR

indicates a wider range of task deadlines), and the number of pipeline stages is set to its default value of 5. For all four curves in the figure, the utilization increases with DR initially and then decreases after certain value. This is because for larger value of DR, it happens more frequently that only one stage of the pipeline is executing a job with long execution time while other following stages are idle and waiting for the stage to finish processing that job, therefore decreasing the average utilization of the pipeline. The figure shows that the holistic analysis and the traditional per-stage test start to suffer at a smaller DR (around 1.0) than the meta-schedulability test (DR close to 1.2).

We took one step further and tested how different number of stages and deadline ratios affect the utilization for the two analysis methods and plotted the result in Figure 7.

The following experiments evaluate the performance of meta-schedulability analysis in the special case where there is a Restriction on the amount that a task's Priority can Change from its base priority (RPC). The default value of system parameters are the same as before, with N equal to 5, DR equal to 1.0, and $\tau$ equal to $1:50$. Here the base priority of each job is assigned according to rate monotonic scheduling. The maximum amount of priority change $x$ is expressed as a fraction of the total number of tasks running on the pipeline, with default value being $0.1*n$. Therefore, each job $J_i$'s per-stage priorities are uniformly distributed over the range of $[\rho_i - 0.1*n, \rho_i + 0.1*n]$, where $\rho_i$ is $J_i$'s base priority. We break tie among jobs with the same priority on a stage using FIFO. Note that for now the priority distribution is uniform; developing more intelligent and efficient priority distribution can be a potential work in the future.

Figure 8 shows the average per-stage utilization for different number of pipeline stages. As expected, the utilization for meta-schedulability test is almost constant across different values of $N$, while those for the holistic analysis and traditional per-stage test decrease as $N$ increases. A comparison between meta-schedulability tests (RTA) in systems with or without restriction on priority change (Figures 5 & 8) shows that the test in default system where a task's priority changes without restriction has a lower utilization. This is because

the restriction on priority change makes the tasks' behavior more predictable, and we do not have to assume the lowest priority for each job in the reduced uniprocessor when computing its bound of delay, as we did in the default system.

We conducted experiments to measure the utilization of tests for different values of DR and plotted the result in Figure 9.

Figure 10 shows the result of experiments measuring utilization for different values of $x$, which is the maximum amount of priority change from each job's base priority. We did the experiments for x values of 0 (full restriction: no priority change allowed), 10% of $n$, 20% of $n$, 50% of $n$ (half restriction), and $n$ (no restriction). As the value of $x$ increases, jobs' behavior becomes more random and unpredictable, and each job can potentially be delayed by more other jobs. For all tests, this results in a more pessimistic bound of delay and therefore lowers utilization of the pipeline.

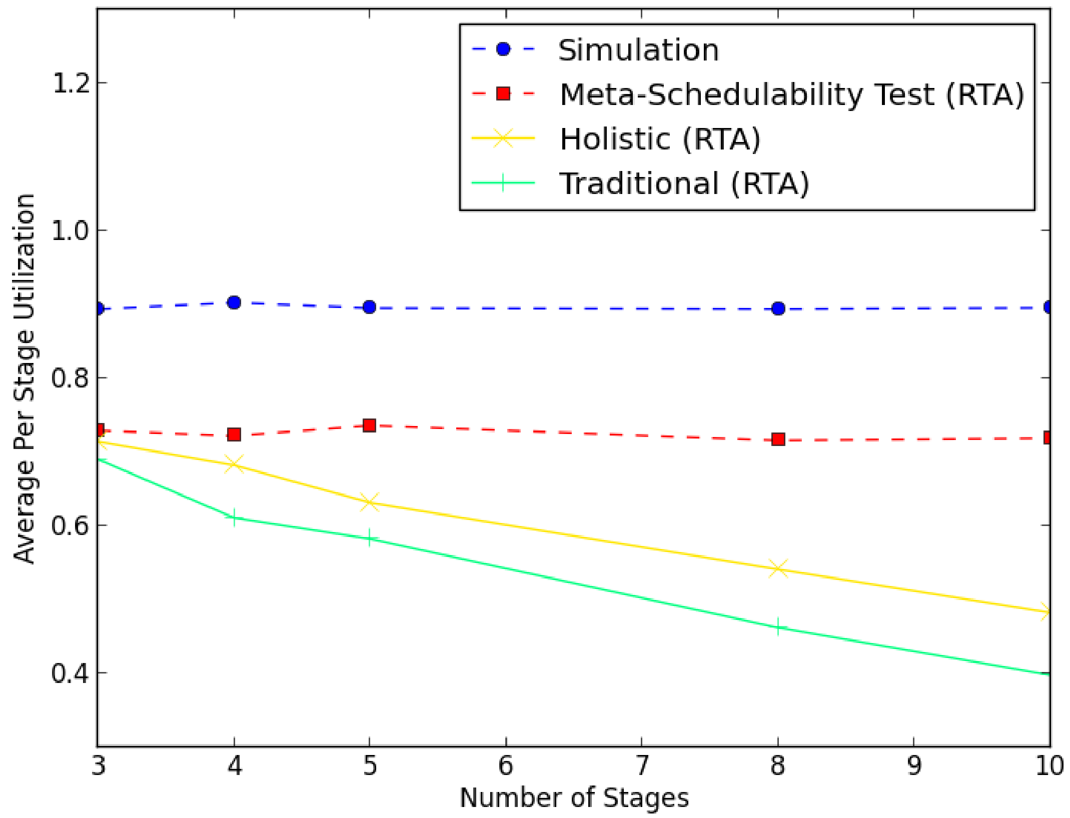


**Figure 5. Utilization for different number of stages in the pipeline**

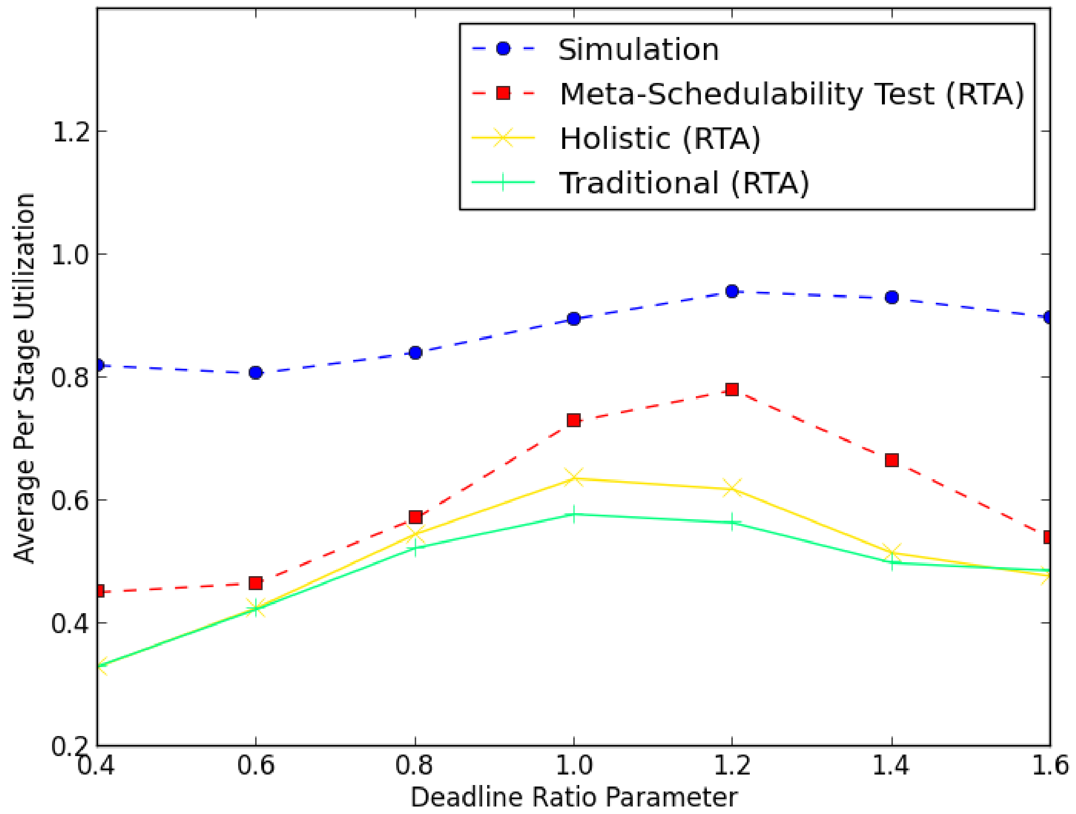


**Figure 6. Utilization for different values of deadline ratio parameter**

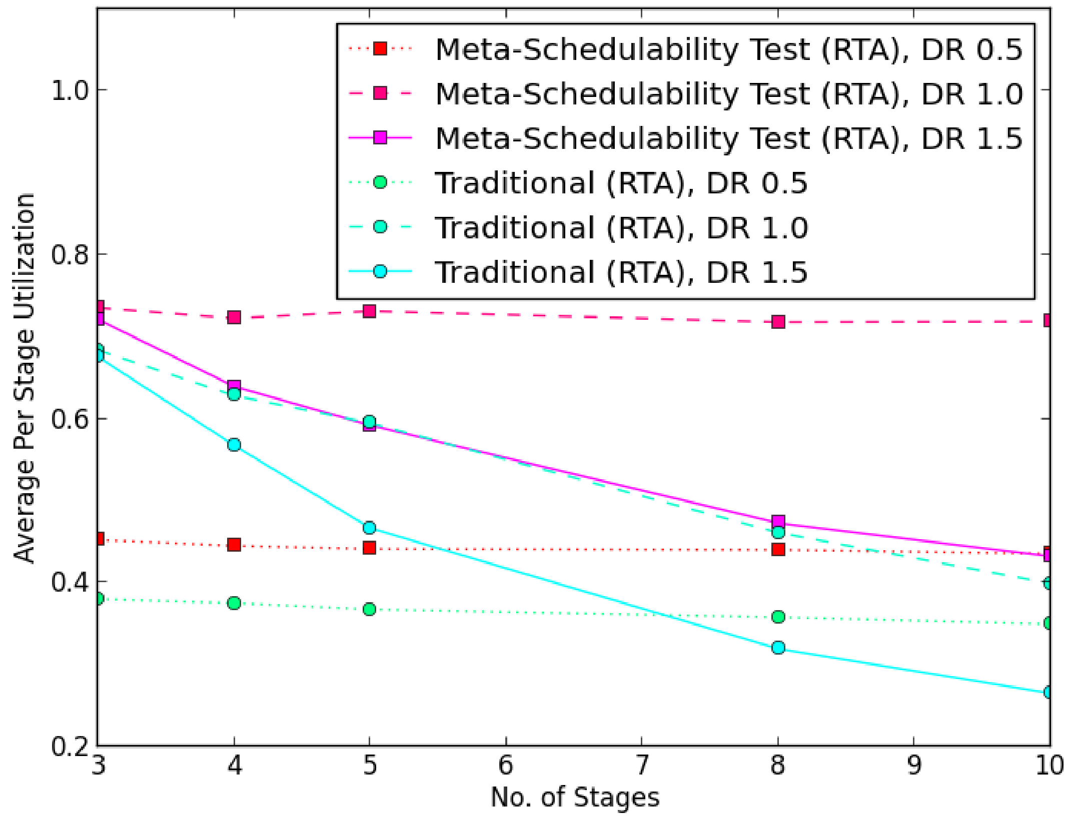


**Figure 7. Utilization for different number of stages and deadline ratios**

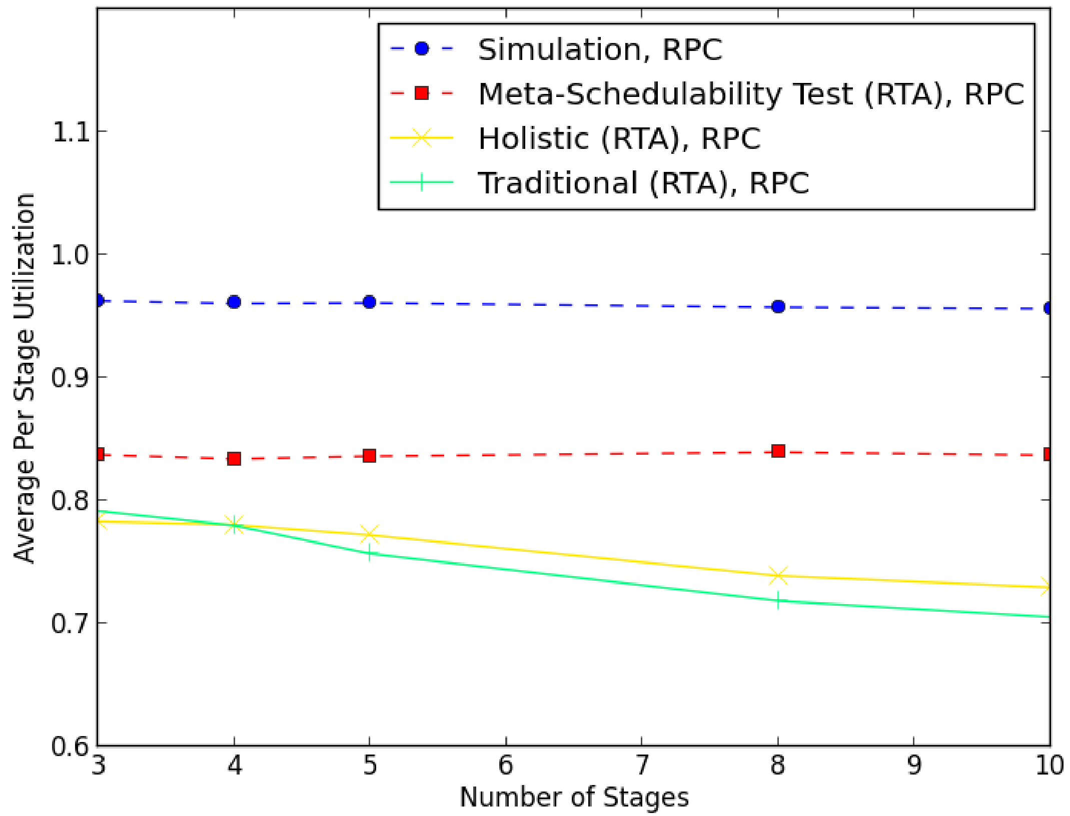


**Figure 8. Utilization for different number of stages in the pipeline under restricted priority change**

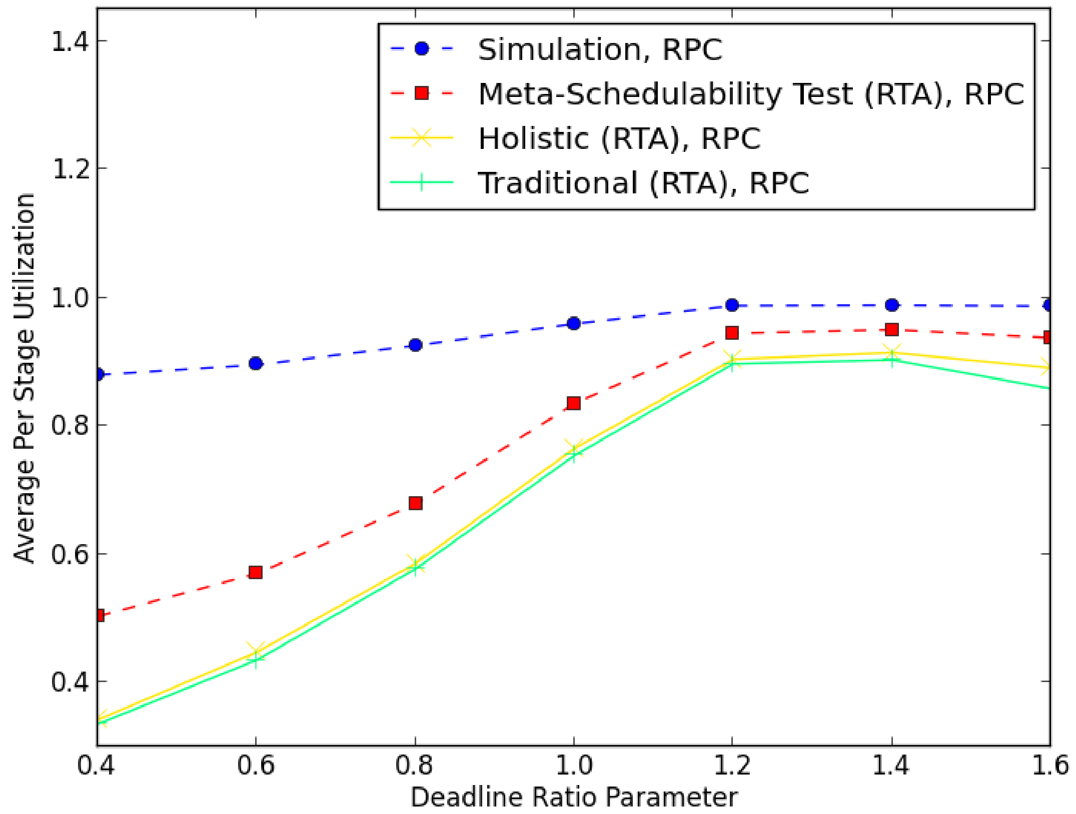


**Figure 9. Utilization for different values of deadline ratio parameter under restricted priority change**

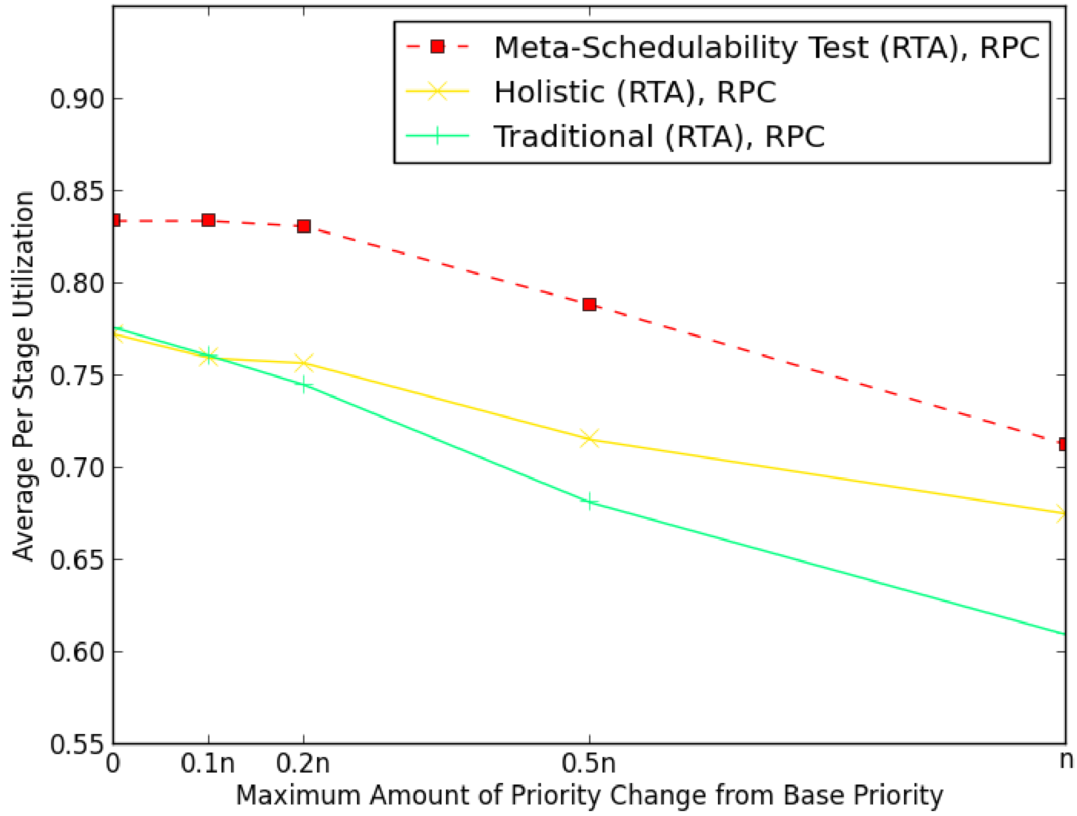


**Figure 10. Utilization for different amount of priority change allowed**

# CHAPTER 8
# CONCLUSION

The growth in research and application of large-scale distributed systems calls for more efficient real-time analysis techniques for the system. Schedulability tests based on the delay composition theorem can be a powerful analysis tool for distributed systems, since the theorem takes parallel execution of real-time tasks into account and therefore is less pessimistic compared to traditional per-stage analysis and holistic analysis. The only limitation of previous works on the delay composition theorem is that it assumes each job has fixed priority across all processing units, which is not a realistic assumption for large distributed systems comprising of autonomous processing units, each of which may use different scheduling policies. From the perspective of a particular real-time task running on the system, it may be assigned different priorities across processing units.

In this paper we broke that assumption held by previous works on the delay composition theorem by extending the theorem to pipeline systems running non-preemptive scheduling algorithms which may assign different priorities to a task on different stages. We further studied how to bound the delay of a task when there is a restriction on the range a task's per-stage priority can change from its base priority. A method for systematically reducing the pipeline system to an equivalent uniprocessor is presented so that analysis tool for uniprocessor, such as the response time analysis, can be applied to the reduced system. We evaluated the performance of real-time analysis based on our extended theorem using simulation, and showed that our analysis technique gives higher average utilization than traditional per-stage analysis and holistic analysis.

Extending the theorem to more general distributed systems with preemptive or non-preemptive scheduling algorithms can be potential work for the future.

# REFERENCES


[1] P. Jayachandran and T. Abdelzaher. A delay composition theorem for real-time pipelines. In *ECRTS*, pages 29-38, July 2007.

[2] P. Jayachandran and T. Abdelzaher. Delay composition algebra: A reduction-based schedulability algebra for distributed real-time systems. In *RTSS*, pages 259-269, December 2008.

[3] P. Jayachandran and T. Abdelzaher. Delay composition in preemptive and non-preemptive real-time pipelines. *Invited to Real-Time Systems Journal: Special Issue on ECRTS'07*, 40(3):290-320, December 2008.

[4] P. Jayachandran and T. Abdelzaher. Transforming acyclic distributed systems into equivalent uniprocessors under preemptive and non-preemptive scheduling. In *ECRTS*, pages 233-242, July 2008.

[5] P. Jayachandran and T. Abdelzaher. End-to-end delay analysis of distributed systems with cycles in the task graph. In *ECRTS*, July 2009.

[6] C. L. Liu and J. W. Layland. Scheduling algorithms for mul- tiprogramming in a hard-real-time environment. *Journal of ACM*, 20 (1): 46-61, 1973.

[7] A. N. Audsley, A. Burns, M. Richardson, and K. Tindell. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering*, pages 284-292, 1993.

[8] B. Kao and H. Garcia-Molina. Deadline assignment in a distributed soft real-time system. *IEEE Transactions on Parallel and Distributed Systems*, 8 (12): 1268-1274, 1997.

[9] S. Chatterjee and J. Strosnider. Distributed pipeline scheduling: End-to-end analysis of heterogeneous multi-resource real-time systems. In *IEEE International Conference on Distributed Computing Systems*, pages 204-211, May 1995.

[10] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Elsevier Microprocessing and Microprogramming*, 40 (2-3): 117-134, 1994.

[11] J. M. Rivas, J. J. Gutirrez, J. C. Palencia, and M. Gonzlez Harbour. Schedulability Analysis and Optimization of Heterogeneous EDF and FP Distributed Real-Time Systems. In *ECRTS*, 2011.

[12] J.J. Gutirrez, and M. Gonzlez Harbour. Optimized Priority Assignment for Tasks and Messages in Distributed Real-Time Systems. Proceedings of 3rd Workshop on Parallel and Distributed Real-Time Systems, Santa Barbara (California), pages 124-132, 1995

[13] J. M. Rivas, J. J. Gutirrez, J. C. Palencia, and M. Gonzlez Harbour. Optimized Deadline Assignment and Schedulability Analysis for Distributed Real-Time Systems with Local EDF Scheduling. 8th International Conference on Embedded Systems and Applications, in *WORLDCOMP*, 2010.

[14] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. Proceedings of the IEEE International Symposium on Circuits and Systems, vol 4, 101-104, 2000.

[15] J.Y. Le Boudec, and P. Thiran. Network Calculus: a theory of deterministic queuing systems for the Internet. Springer, 2001.