CS499 Senior Thesis: Delay Composition Algebra applied to tasks conforming to Processor Quotas

Abhishek Pradhan*
Department of Computer Science
University of Illinois at Urbana-Champaign

Tarek Abdelzaher†
Department of Computer Science
University of Illinois at Urbana-Champaign

Abstract

Delay Composition Algebra is a technique used to analyze the real time schedulability of a task set over a distributed system. This paper will explore methods in which we can apply the Delay Composition Algebra to systems where real time tasks sets are limited to a certain quota within the individual nodes of the distributed system. We will define two configurations in which the delay caused by the imposed quota affects the real time task set and derive methods in which the configuration along with the task set can be reduced to a standard Delay Composition Algebra problem. The result of the reduction will describe a systematic method of performing schedulability analysis on a set of real time tasks that operate only under the alloted quota.

Keywords: delay composition algebra, quotas, real time systems

1 Introduction

1.1 Background

The Delay Composition Algebra uses a set of simple operators to systematically convert a set of real-time tasks on a distributed system into a set of tasks on a uniprocessor system in order to perform schedulability analysis. [Jayachandran and Abdelzaher 2008a] The method is derived from the fact that the worst-case delay imposed by one task on another is combined sub-additively due to the effect of pipelining on distributed systems. [Jayachandran and Abdelzaher 2007] We are also granted the ability to extend this framework to various types of task sets due to the fact that this method does not make any assumptions about scheduling policy other than the fact that tasks are assigned the same priority across nodes.

1.2 Motivation

There is an increasing number of applications that are modeled as real time tasks that often coexist with non real time applications. One result of such systems is that the scheduling algorithm results in what is seen to be a round robin algorithm where time quanta are heterogeneously assigned to real time and non real time tasks. [Deng et al. 1997] In such systems the time quantum assigned to a real time task is defined to be quota that it operates under. With the advent of distributed computing we can see that real time tasks would similarly be shared with non real time tasks paving the need to perform schedulability analysis on such systems. Schedulability analysis on distributed systems can be simplified into a systematic process as shown previously. [Jayachandran and Abdelzaher 2008a] It is due to reason that we will build upon this mechanism in order to perform schedulability analysis on tasks that are limited to quotas on distributed systems.

*e-mail: apradha2@illinois.edu
†e-mail: zaher@illinois.edu

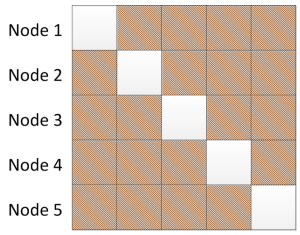

Figure 1: *In the second configuration we note that the real time quota is synchronized such that it is viewed as a pipelined task traveling through the distributed system.*

1.3 Objective

There are two configurations that we will define based on the means in which the delay caused by processor quotas affects the real time tasks. The first configuration we will look at is when the processor quotas across the various nodes in the distributed system are not synchronized, in effect we will look at the worst case condition for examples of such delays. For the second configuration we will look at when processor quotas are synchronized across nodes; in this case we will look at the scenario where the quota travels through the system in order to allow simulate pipelined behavior, in effect trying to produce the best case scenario. Figure 1 illustrates the method in which we define the synchronized scenario.

2 Delay Composition Algebra

2.1 Introduction

Delay Composition Algebra uses a simple set of operators used to reduce a real-time task set on a distributed system into a uniprocessor task set. We will first give a brief overview of how delay composition algebra works in order to provide a background for the methods that are used in this paper, the detailed analysis and development of these methods are described in a previous paper. [Jayachandran and Abdelzaher 2008a]

2.2 Data Representation

Given a real-time task set, there are two primary methods of data representation. The first one describes the structure and arrangement of nodes in the distributed system including the flow of information which is represented by a directed acyclic graph. [Jayachandran and Abdelzaher 2008b] The second is used to describe the task set on each node which presents itself as an n by n matrix where n is the number of tasks that appear in the system. [Jayachandran and Abdelzaher 2008a] The elements of this matrix are defined to be the delay terms where element (i, k) represents the delay that task J_i imposes on J_k.

The delay term is a tuple defined to be $(q_{i,k}, r_{i,k})$ where $q_{i,k}$ is the defined to be maximum delay that task J_i imposes on J_k within that node and is called the $max\ term$. The second term in the tu-

i,k	J_1	J_2	.	.	J_n
J_1	$(q_{1,1}, r_{1,1})$	$(q_{1,2}, r_{1,2})$	.	.	$(q_{n,n}, r_{n,n})$
J_2	$(q_{2,1}, r_{2,1})$	$(q_{2,2}, r_{2,2})$	.	.	$(q_{n,n}, r_{n,n})$
.	.				
.	.				
J_n	$(q_{n,1}, r_{n,1})$	$(q_{n,2}, r_{n,2})$	.	.	$(q_{n,n}, r_{n,n})$
	s_1	s_2	.	.	s_n

Figure 2: *The operand used in the Delay Composition Algebra thats contains the delay term matrix.*
[Jayachandran and Abdelzaher 2008a]

i,k	J_1	J_2	J_3	J_4
J_1	$(C_{1,j}, 0)$	$(C_{1,j}, 0)$	$(0,0)$	$(C_{1,j}, 0)$
J_2	0	$(C_{2,j}, 0)$	$(0,0)$	$(C_{2,j}, 0)$
J_3	0	0	$(0,0)$	$(0,0)$
J_4	0	0	0	$(C_{4,j}, 0)$
	$C_{1,j}$	$max(C_{1,j}, C_{2,j})$	0	$max(C_{1,j}, C_{2,j}, C_{4,j})$

Figure 3: s_k *term defined using preemptive scheduling.*
[Jayachandran and Abdelzaher 2008a]

ple $r_{i,k}$ is called the $accumulator\ term$ and is used for keeping a track of delay that is caused by a task that is not represented in that node. [Jayachandran and Abdelzaher 2008a] A term described as s_k which is in the last row of the operand (shown in Figure 2) is used to describe the delay imposed on job J_k independent of the number of jobs in the system. Given preemptive scheduling this term is defined to be $s_k = \max_{i \le k} q_{i,k}$ and for non-preemptive scheduling we have $s_k = \max_i q_{i,k} + \max_{i>k} q_{i,k}$. [Jayachandran and Abdelzaher 2008a; Jayachandran and Abdelzaher 2008c] Figures 3 and 4 are examples of these definitions respectively.

2.3 Operators

The objective of using the Delay Composition Algebra is to take a distributed system and reduce it to a single processor. As such there are two operators that are defined in order to aid in the reduction of a distributed task set into one or more uniprocessor task sets. This reduction allows us to use previously defined bounds and research in order to determine the schedulability of the task set.

2.3.1 PIPE Operator

Given A and B where there exists an edge from A to B and A has exactly one outgoing edge, we are able reduce these two nodes

i,k	J_1	J_2	J_3	J_4
J_1	$(C_{1,j}, 0)$	$(C_{1,j}, 0)$	$(0,0)$	$(C_{1,j}, 0)$
J_2	0	$(C_{2,j}, 0)$	$(0,0)$	$(C_{2,j}, 0)$
J_3	0	0	$(0,0)$	$(0,0)$
J_4	0	0	0	$(C_{4,j}, 0)$
	$C_{1,j} + max(C_{2,j}, C_{4,j})$	$max(C_{1,j}, C_{2,j}) + C_{4,j}$	0	$max(C_{1,j}, C_{2,j}, C_{4,j})$

Figure 4: s_k *term defined using non-preemptive scheduling.*
[Jayachandran and Abdelzaher 2008a]

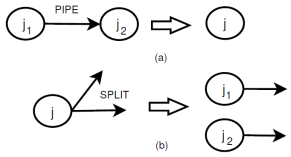

Figure 5: *PIPE and SPLIT operator and the equivalent stages of their result.*
[Jayachandran and Abdelzaher 2008a]

$$\begin{pmatrix} & J_1 & J_2 \\ J_1 & (q^A_{1,1}, r^A_{1,1}) & (q^A_{1,2}, r^A_{1,2}) \\ J_2 & (0,0) & (q^A_{2,2}, r^A_{2,2}) \\ & \cdots & \\ & s^A_1 & s^A_2 \end{pmatrix} PIPE \begin{pmatrix} & J_1 & J_2 \\ J_1 & (q^B_{1,1}, r^B_{1,1}) & (q^B_{1,2}, r^B_{1,2}) \\ J_2 & (0,0) & (q^B_{2,2}, r^B_{2,2}) \\ & \cdots & \\ & s^B_1 & s^B_2 \end{pmatrix}$$

$$= \begin{pmatrix} & J_1 & J_2 \\ J_1 & (max(q^A_{1,1}, q^B_{1,1}), \; max(r^A_{1,1}, r^B_{1,1})) & (max(q^A_{1,2}, q^B_{1,2}), \; max(r^A_{1,2}, r^B_{1,2})) \\ J_2 & (0,0) & (max(q^A_{2,2}, q^B_{2,2}), \; max(r^A_{2,2}, r^B_{2,2})) \\ & \cdots & \\ & s^A_1 + s^B_1 & s^A_2 + s^B_2 \end{pmatrix}$$

Figure 6: *PIPE Operator*
[Jayachandran and Abdelzaher 2008a]

into a single node C by using the PIPE operator. As described by [Jayachandran and Abdelzaher 2008a] the definition for PIPE is as follows where C = A PIPE B:

1. $\forall i,k : q^C_{i,k} = max(q^A_{i,k}, q^B_{i,k})$
2. $\forall i,k : r^C_{i,k} = max(r^A_{i,k}, r^B_{i,k})$
3. $\forall k : s^C_k = s^A_k + s^B_k$

2.3.2 SPLIT Operator

The SPLIT operator is used when we have multiple outgoing and no incoming edges for a given node A. (we can use the PIPE operator to eliminate all incoming edges) Given l outgoing edges the node A is split into l different nodes, these are replicas of A except for the fact that the columns for tasks that do not follow the given edge are zeroed out. Finally for each matrix i of these new l matrices we update the accumulator term values for all higher priority tasks that existed in A but do not exist in i; the new accumulator value is defined as $(0, q_{i,k} + r_{i,k})$ where $(q_{i,k}, r_{i,k})$ was the value in the original matrix for A. The formal definition for the SPLIT operator is defined in [Jayachandran and Abdelzaher 2008a]. Figure 7 describes the usage of the SPLIT operator.

2.3.3 Non-Acyclic Task Sets

If we are given a non-acyclic task set a CUT operator as defined in [Jayachandran and Abdelzaher 2008a] can be used to modify the task set such that it is acyclic.

3 Processor Quotas

Given the overview of Delay Composition Algebra, we can now look at using and extending this method by applying it to cases where real time tasks are limited to a certain quota on the processor. As mentioned previously we will look at three configurations in this

$$SPLIT \left(\begin{array}{c|cc} & J_1 & J_2 \\ \hline J_1 & (q^A_{1,1}, r^A_{1,1}) & (q^A_{1,2}, r^A_{1,2}) \\ J_2 & (0,0) & (q^A_{2,2}, r^A_{2,2}) \\ & \cdots\cdots\cdots\cdots\cdots & \\ & s^A_1 & s^A_2 \end{array} \right) \equiv$$

$$\left(\begin{array}{c|cc} & J_1 & J_2 \\ \hline J_1 & (q^A_{1,1}, r^A_{1,1}) & (0,0) \\ J_2 & (0,0) & (0,0) \\ & \cdots\cdots\cdots\cdots & \\ & s^A_1 & 0 \end{array} \right), \left(\begin{array}{c|cc} & J_1 & J_2 \\ \hline J_1 & (0,0) & (0, q^A_{1,2} + r^A_{1,2}) \\ J_2 & (0,0) & (q^A_{2,2}, r^A_{2,2}) \\ & \cdots\cdots\cdots\cdots\cdots & \\ & 0 & s^A_2 \end{array} \right)$$

Figure 7: *SPLIT Operator*
[Jayachandran and Abdelzaher 2008a]

scenario and reduce its analysis into a Delay Composition Algebra problem.

3.1 Unsynchronized Quotas

3.1.1 Problem Description

The case of unsynchronized quotas is classified by the fact that there is an assumption that the delay on each node is independent of the other nodes. It is here that we will make no assumption about the processor quota assigned on each node and take into account a worst case scenario for processor quotas, thus given a distributed system we will handle a delay component for each node.

3.1.2 Method

The approach taken by this method classifies the delay components as independent tasks running through the system, given the highest priority. Given a node A we will define the delay component for that node as D^A, it will be defined by the amount of time on the node that is not dedicated to the real time tasks that we are analyzing d^A. This analysis is defined by assigning n additional rows to the final operand matrix where n is the number of nodes in the system. The delay element in the matrix for each D^A_i, J_k pair will be defined as $(0, d^A_i)$ if job J_k was present in that node in the distributed system. If this was not the case then the delay element d^A_i is defined to be 0 resulting in $(0,0)$, as that processor quota does not affect the corresponding job. This is illustrated in Figure 8. Once the final matrix has been constructed we are able to reduce it to a task set as per the method described in [Jayachandran and Abdelzaher 2008a].

3.1.3 Derivation

The method for handling unsynchronized quotas is obtained from the fact that the delay caused by the imposition of a processor quota is modeled as a high priority task. We begin by observing that on a uniprocessor task set we are able to model the delay caused by a processor quota by adding an extra task and assigning it as the highest priority task within that task set. On a distributed system in the worst case scenario we make the assumption that this delay will affect us on each node that the tasks have to travel through. It is in this case that the delay on each node will be wholly independent of the other nodes and due to this reason could be variable for each node. Thus to approach the solution we will model the delay caused by processor quotas as an independent task on each node.

In order to create this model we note that we must for each node PIPE the delay task into the node and then SPLIT it as it does not

$$\left(\begin{array}{c|ccccc} i,k & J_1 & J_2 & . & . & J_n \\ \hline D^A_1 & (0, d^A_1) & (0, d^A_1) & . & . & (0, d^A_1) \\ D^B_2 & (0, d^B_2) & (0, d^B_2) & . & . & (0, d^B_2) \\ . & . & & & & \\ . & . & & & & \\ D^N_n & (0, d^N_n) & (0, d^N_n) & . & . & (0, d^N_n) \\ J_1 & (q_{1,1}, r_{1,1}) & (q_{1,2}, r_{1,2}) & . & . & (q_{n,n}, r_{n,n}) \\ J_2 & (q_{2,1}, r_{2,1}) & (q_{2,2}, r_{2,2}) & . & . & (q_{n,n}, r_{n,n}) \\ . & . & & & & \\ . & . & & & & \\ J_n & (q_{n,1}, r_{n,1}) & (q_{n,2}, r_{n,2}) & . & . & (q_{n,n}, r_{n,n}) \\ \hline & s_1 & s_2 & . & . & s_n \end{array} \right)$$

Figure 8: *Applying delay composition algebra to unsynchronized quotas.*

travel travel through the rest of the distributed system. Given a node A within a distributed system with a task set J_1 through J_n, we first introduce the quota delay by PIPEing it into the node.

$$A = \left(\begin{array}{c|ccccc} i,k & J_1 & J_2 & . & . & J_n \\ \hline J_1 & (q_{1,1}, r_{1,1}) & (q_{1,2}, r_{1,2}) & . & . & (q_{n,n}, r_{n,n}) \\ J_2 & (q_{2,1}, r_{2,1}) & (q_{2,2}, r_{2,2}) & . & . & (q_{n,n}, r_{n,n}) \\ . & . & & & & \\ . & . & & & & \\ J_n & (q_{n,1}, r_{n,1}) & (q_{n,2}, r_{n,2}) & . & . & (q_{n,n}, r_{n,n}) \\ \hline & s_1 & s_2 & . & . & s_n \end{array} \right)$$

$$PIPE$$

$$D^A = \left(\begin{array}{c|c} i,k & D^A_0 \\ \hline D^A_0 & (d^A_0, 0) \\ \hline & s_0 \end{array} \right)$$

The operation A PIPE D^A gives us the following operand matrix:

$$A \; PIPE \; D^A =$$

$$\left(\begin{array}{c|ccccc} i,k & D^A_0 & J_1 & J_2 & . & . & J_n \\ \hline D^A_0 & (d^A_0, 0) & (d^A_0, 0) & (d^A_0, 0) & . & . & (d^A_0, 0) \\ J_1 & (0,0) & (q_{1,1}, r_{1,1}) & (q_{1,2}, r_{1,2}) & . & . & (q_{n,n}, r_{n,n}) \\ J_2 & (0,0) & (q_{2,1}, r_{2,1}) & (q_{2,2}, r_{2,2}) & . & . & (q_{n,n}, r_{n,n}) \\ . & (0,0) & . & & & & \\ . & (0,0) & . & & & & \\ J_n & (0,0) & (q_{n,1}, r_{n,1}) & (q_{n,2}, r_{n,2}) & . & . & (q_{n,n}, r_{n,n}) \\ \hline & s_0 & s_1 & s_2 & . & . & s_n \end{array} \right)$$

Next we perform a SPLIT operation as we do not want the delay task to propagate through the rest of the system. Thus we obtain SPLIT(A PIPE D^A):

$$A' = \left(\begin{array}{c|ccccc} i,k & J_1 & J_2 & . & . & J_n \\ \hline D^A_0 & (d^A_0, 0) & (d^A_0, 0) & . & . & (d^A_0, 0) \\ J_1 & (q_{1,1}, r_{1,1}) & (q_{1,2}, r_{1,2}) & . & . & (q_{n,n}, r_{n,n}) \\ J_2 & (q_{2,1}, r_{2,1}) & (q_{2,2}, r_{2,2}) & . & . & (q_{n,n}, r_{n,n}) \\ . & . & & & & \\ . & . & & & & \\ J_n & (q_{n,1}, r_{n,1}) & (q_{n,2}, r_{n,2}) & . & . & (q_{n,n}, r_{n,n}) \\ \hline & s_1 & s_2 & . & . & s_n \end{array} \right)$$

We are only concerned about A' since once the delay task has been SPLIT we do not need reference it for schedulability analysis. It is

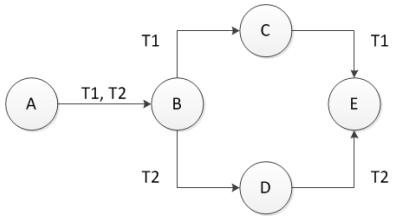

Figure 9: *Unsynchronized Quota Example*

noted that this sequence of operations applied to through all nodes in the system will result in the matrix described earlier and portrayed in Figure 8.

3.1.4 Example

The following example illustrates the usage of this method, we will use the five stage preemptive system as shown in Figure 9. This system is shown to have two tasks defined as T_1 and T_2. We will define the quota delay component and the computation time of each task to be one computation time unit.

Using Delay Composition Algebra we first use the method described in [Jayachandran and Abdelzaher 2008a] to reduce the distributed system into a uniprocessor task set to obtain the following operand matrix.

$$\left(\begin{array}{c|cc} i,k & T_1 & T_2 \\ \hline T_1 & (1,0) & (1,1) \\ T_2 & (0,0) & (1,0) \\ \hline & 3 & 3 \end{array}\right)$$

We now add the delay elements for each of the nodes to the matrix.

$$\left(\begin{array}{c|cc} i,k & T_1 & T_2 \\ \hline D^A & (0,1) & (0,1) \\ D^B & (0,1) & (0,1) \\ D^C & (0,1) & (0,0) \\ D^D & (0,0) & (0,1) \\ D^E & (0,1) & (0,1) \\ T_1 & (1,0) & (1,1) \\ T_2 & (0,0) & (1,0) \\ \hline & 3 & 3 \end{array}\right)$$

It is noted that D^C, T_2 and D^D, T_1 are $(0,0)$ since T_1, and T_2 are not present on those stages. We can now perform schedulability analysis on this task set by reducing it to classical uniprocessor by following the method as described in [Jayachandran and Abdelzaher 2008a]. Once this task has been performed we can use classical uniprocessor techniques such as [Audsley et al. 1993] to determine the schedulability of the task set.

3.2 Synchronized Quotas

3.2.1 Problem Description

The second case we will look at involves time synchronized quotas across the distributed system. This synchronization as illustrated in Figure 1 is defined to be one where the quota for the real time tasks travels through the system in order to ensure pipelined behavior. We will define this behavior to be one where given nodes A and B in the distributed system, where there exists a directed edge from A to B, the real time quota once completed on node A immediately begin on node B. For now we will also assume that there will be no SPLIT operations in the distributed system.

$$\left(\begin{array}{c|cccccc} i,k & D & J_1 & J_2 & . & . & J_n \\ \hline D & (d,0) & (d,0) & (d,0) & . & . & (d,0) \\ J_1 & (0,0) & (q_{1,1},r_{1,1}) & (q_{1,2},r_{1,2}) & . & . & (q_{n,n},r_{n,n}) \\ J_2 & (0,0) & (q_{2,1},r_{2,1}) & (q_{2,2},r_{2,2}) & . & . & (q_{n,n},r_{n,n}) \\ . & . & & & & & \\ . & . & & & & & \\ J_n & (0,0) & (q_{n,1},r_{n,1}) & (q_{n,2},r_{n,2}) & . & . & (q_{n,n},r_{n,n}) \\ \hline & s_0 & s_1 & s_2 & . & . & s_n \end{array}\right)$$

Figure 10: *Synchronized Quotas Method: The operand matrix for each node is modified to include the delay task.*

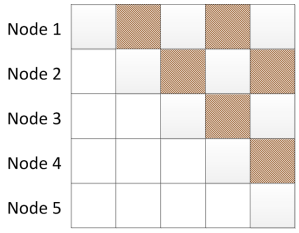

Figure 11: *If the delay component is modeled as a periodic task we do not need to perform any extra analysis in order to determine the schedulability of the task set.*

3.2.2 Method

The approach taken by this method views the delay caused by the quota as a periodic task traveling through the distributed system. Given a distributed system we will define the delay component to be D. We note that for each node A in the distributed system we will add task D as an additional row, defined to be the highest priority task. The operand element $(d,0)$ will be used to represent the time that is not allotted to the real time task set where d is the delay caused in time computation units. This will allow us to view the delay as a task traveling through the distributed system. We will then perform the reduction of the distributed system into a uniprocessor task to perform schedulability analysis as shown in [Jayachandran and Abdelzaher 2008a].

3.2.3 Derivation

The derivation for this method is obtained from the principal that the delay component is modeled as a periodic task. Given the fact that the quota is synchronized in order to allow the emulation of pipelined behavior we note that in Figure 11 we are able to view the delay component as providing an uninterrupted pipelined pattern for those tasks that complete within the allotted quota. We note that in such analysis we need to calculate the delay caused to those tasks that are unable to complete on a single node in the allotted quota, as the other tasks will flow through the system in an interrupted fashion. Due to this case if we model the delay as a periodic task we automatically handle the calculation of this delay using the tools of analysis we already have available to us. A note to make here is that there is a slight degree of pessimism imposed by the fact that we assume that delay quota affects the real time task set in at least one node. This is due to the fact that we make use of the result described in [Jayachandran and Abdelzaher 2007] in order to perform an elegant reduction.

4 Evaluation : Pending Section

This section will be used to evaluate the accuracy and degree of pessimism imposed by this method. I plan to use the same simulation technique as in the Delay Composition Algebra Paper.

References

AUDSLEY, N., BURNS, A., RICHARDSON, M., TINDELL, K., AND WELLINGS, A. J. 1993. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal 8*, 284–292.

DENG, Z., LIU, J.-S., AND SUN, J. 1997. A scheme for scheduling hard real-time applications in open system environment. In *Real-Time Systems, 1997. Proceedings., Ninth Euromicro Workshop on*, 191 –199.

JAYACHANDRAN, P., AND ABDELZAHER, T. 2007. A delay composition theorem for real-time pipelines. In *Real-Time Systems, 2007. ECRTS '07. 19th Euromicro Conference on*, 29 –38.

JAYACHANDRAN, P., AND ABDELZAHER, T. 2008. Delay composition algebra: A reduction-based schedulability algebra for distributed real-time systems. In *Real-Time Systems Symposium, 2008*.

JAYACHANDRAN, P., AND ABDELZAHER, T. 2008. Transforming distributed acyclic systems into equivalent uniprocessors under preemptive and non-preemptive scheduling. In *Real-Time Systems, 2008. ECRTS '08. Euromicro Conference on*, 233 –242.

JAYACHANDRAN, P., AND ABDELZAHER, T. 2008. Delay composition in preemptive and non-preemptive real-time pipelines. *Real-Time Syst. 40* (December), 290–320.