International Journal of Innovative
Computing, Information and Control
Volume **9**, Number **4**, April **2013**

ICIC International ©2013 ISSN 1349-4198
pp. **1651–1677**

THE LINEAR SWITCHING STATE SPACE: A NEW MODELING PARADIGM FOR TASK SCHEDULING PROBLEMS

Hamid Tabatabaee-Yazdi and Mohammad-R. Akbarzadeh-T.

Center of Excellence on Soft Computing and Intelligent Information Processing
Department of Electrical Engineering
Ferdowsi University of Mashhad
Azadi Square, Mashhad 9177948974, Iran
Akbarzadeh@ieee.org

Received January 2012; revised May 2012

Abstract. *Task Scheduling (TS) poses a challenging problem in distributed systems for multiple realms such as multiprocessor systems, flow-shop scheduling and project management problems in which there are multiple tasks and processors (resources), and the problem is to efficiently assign tasks to processors. The importance of this problem can be considered from multiple perspectives such as heterogeneity of processors, computational complexity of reaching a solution as well as theoretical performance analysis. Also, we propose a new modeling paradigm based on system engineering. The proposed switching state space approach opens a possibility of using the extensive theoretical developments that have taken place in this field within the past several decades. In its general form, TS is inherently nonlinear because of its many nonlinear constraints. In this paper, we demonstrate that standard TS can be mapped via nonlinear state space and, through theoretical analysis, show stability of the resulting system for static task scheduling problems. The proposed static scheduling schedules dependent tasks on a heterogeneous multiprocessor system. A suitable transformation is then devised to convert this model to linear switching state space with nonlinear constraints. To illustrate the utility of the model, two scheduling approaches are then presented based on Height Sorted (HS) and Ready Tasks (RT). These two methods are examples that show how this model can be used to reach the stability criteria. Finally, the proposed methods are compared against a conventionally accepted scheduling scheme on several random experiments showing comparative performance.*

Keywords: Modeling, Task scheduling, Static scheduling, Linear switching state space, Stability, Control systems

Nomenclature

Parameters	**Definition**	**Parameters**	**Definition**
X	Vector of state variables	U	Control Vector
$X[k]$	Vector of state variables in kth step	$X[k+1]$	Vector of state variables in $(k+1)$th step
Y	System output	$\boldsymbol{V}$	Set of tasks in DAG
E	Set of edges in DAG	$\boldsymbol{R}$	Set of resources
$T = \lvert\boldsymbol{V}\rvert$	Number (Cardinality) of tasks	$M = \lvert\boldsymbol{R}\rvert$	Number (Cardinality) of resources
$w(n_i)$	Computation cost of node n_i	e_{ij}	Edge from node i to node j in DAG
$C(e_{ij})$	Communication cost of edge e_{ij}	$succ(n_i)$	Successors of node n_i
$pred(n_i)$	Predecessors of node n_i	$w(n_i, r_j)$	Execution time of node n_i on resource r_j
$t_s(n_i, r_j)$	Start time of node n_i on resource r_j	$\theta_{i,j}$	Computation time of task i on resource j
$t_f(n_i, r_j)$	Finish time of node n_i on resource r_j	μ_{ij}	Communication time between task i and task j
A_i, B_i, C_i, D_i	ith Subsystem Matrices	U^l	lth Control vector

1. **Introduction to the Task Scheduling Problem.** Various aspects of task scheduling (TS) have been addressed by many researchers to date. This problem has many real world applications such as in management of projects, human resources, financial resources as well as distributed computing systems. While the vocabulary in these fields

may be different, they all address the same base problem. For instance, *processors* in the realm of computer sciences are the same as the *resources* in the realm of human and financial resources. Here, we adopt the terminology that is common in computer sciences without a loss of general applicability. In general, a multi-processor system must be both fast as well as it must utilize its available resources efficiently to complete the requested tasks. This is while the problem of task scheduling itself may face various challenges such as in processor heterogeneity, interdependent tasks, excessive communication cost, as well as limited computational resources. Finally, inappropriate scheduling of tasks can fail to exploit the true potential of a distributed system and can offset the gains from parallelization due to excessive communication overhead or under-utilization of processing resources [1].

Simple definition of (static) task scheduling problems is allocating dependent tasks to processors while total execution time (makespan) is minimized. In the static task scheduling, all things about tasks and processors are known before execution of tasks in real world; therefore, finding a schedule before execution in real world is possible. This problem is generally NP-complete with only few exceptions as will be discussed later. Even in two simple cases [2], the TS problem is NP such as, 1) scheduling unit-time tasks to an arbitrary number of processors, 2) scheduling one or two time unit tasks to two processors. In these two cases, communication cost is not considered among the tasks. Therefore, solutions to task scheduling problems are generally non-trivial by traditional methods. Until now, many researchers have solved this problem by heuristic and random search methods, but they often consider certain constraints that lead to inapplicability of their methods in the real world. Furthermore, their performance evaluation is often only numeric and lacks theoretical guarantees. Hence, making a general model for task scheduling problems is essential that would be amenable to theoretical and rigorous analysis.

In this paper, we present a new modeling paradigm based on switching systems theory for theoretical modeling and analysis. This new paradigm, because of its rich mathematical framework, provides rigorous tools of analysis to yield concrete results in the field of task scheduling. Here, we focus on static scheduling of dependent tasks on heterogeneous multi-processor systems. It is assumed that tasks are sufficiently decomposed to their integral part such that there is no parallelism within a task and all instructions are executed sequentially. Furthermore, we consider a distributed computing system as a well-known sample for task scheduling problem. This paper is organized as follows. In Sections 2 and 3, the basic task scheduling problem and related work are reviewed for better readability. There are many approaches to the classical task scheduling problem and hence only a few are highlighted that stand out as the major directions in their field. To the author's best knowledge, the present work is the first to address the need for theoretical analysis such as stability from a systems engineering perspective. Section 4 reviews the fundamentals of switching systems paradigm. In Section 5, we then show how task scheduling can be represented in nonlinear state space and consider the stability of the resulting model based on system engineering analysis. This model is then transformed to linear switching state space with nonlinear constraints. In Section 6, we use the proposed model to reach two systematic design strategies (finding appropriate control vectors). These two algorithms are stable since they satisfy stability criteria of the earlier section. In Section 7, we compare results of our methods on 46 different experiments that are each randomly repeated 50 times against HEFT (heterogeneous earliest finish time) [3] and show comparable performance of the proposed approach. Finally, we conclude the paper in Section 8.

2. **Task Scheduling Problem Definition.** In task scheduling, the program to be scheduled is represented by a task graph or so called precedence constraint graph.

Definition 2.1. (Task Graph) [4]. *A task graph is a directed acyclic graph (DAG) $G = (\boldsymbol{V}, \boldsymbol{E}, w, c)$ representing an application P according to the graph model. The nodes (vertices) in $\boldsymbol{V}$ represent the tasks of P and the edges in $\boldsymbol{E}$ represent the communication between the tasks. An edge $e_{ij} \in \boldsymbol{E}$ from node n_i to n_j, n_i, $n_j \in \boldsymbol{V}$, represents the communication from node n_i to node n_j. The positive weight $w(n_i)$ associated with node $n_i \in \boldsymbol{V}$ represents its computation cost and the nonnegative weight $c(e_{ij})$ associated with edge $e_{ij} \in \boldsymbol{E}$ represents its communication cost.*

Before executing a task, all its predecessors must be executed and all of its inputs must be received. After executing a node, its outputs are ready to transfer to their successors, simultaneously [5]. When a task begins to execute on a specified processor, it will continue its execution until completion. In other words, task execution cannot be interrupted by any other task and hence preemption is not allowed.

The set of all direct predecessors of n_i, g, is denoted by $pred(n_i)$ and the set of all immediate successors of n_i, $\{n_x \in \boldsymbol{V} : e_{ix} \in \boldsymbol{E}\}$, is denoted by $succ(n_i)$. Source node is a node $n_i \in \boldsymbol{V}$ without any predecessors, i.e., $pred(n_i) = \varnothing$ and exit node (sink node) is a node $n_i \in \boldsymbol{V}$ without any successors, i.e., $succ(n_i) = \varnothing$ [5].

Scheduling of a DAG is the process of allocating each node to a resource and determining its start time. To describe a schedule S of a DAG $G = (\boldsymbol{V}, \boldsymbol{E}, w, c)$ on a target system consisting of a set R of dedicated processors, the following terms: $t_s(n_i, r_j)$ and $w(n_i, r_j)$ indicate the start time and the execution time of node $n_i \in \boldsymbol{V}$ on processor $r_j \in R$, respectively. Thus, finishing time of a node is calculated by $(n_i, r_j) = t_s(n_i, r_j) + w(n_i, r_j)$. $resource(n_i)$ denotes the processor r to which n_i is allocated. There are two conditions that must be satisfied for all nodes in G to reach a feasible scheduling [5]:

Condition 1 (Dedicated Processor Constraint). Each processor is allocated at most one task at any given time. In other words, for any two nodes n_i, $n_j \in \boldsymbol{V}$, if

$$resource(n_i) = resource(n_j) = r_j \Rightarrow \begin{cases} t_f(n_i, r_j) \le t_s(n_i, r_j), & \text{or} \\ t_f(n_i, r_j) \le t_s(n_i, r_j) \end{cases} \tag{1}$$

Condition 2 (Node Precedence Constraint). For n_i, $n_j \in \boldsymbol{V}$, $e_{ij} \in \boldsymbol{E}$, $r_k \in \boldsymbol{R}$,

$$t_s(n_j, r_k) \ge t_f(e_{ij}), \tag{2}$$

where,

$$t_f(e_{ij}) = t_f(n_i) + \begin{cases} 0 & \text{if } resource(n_i) = resource(n_j) \\ c(e_{ij}) & \text{otherwise} \end{cases} \tag{3}$$

where $t_f(e_{ij})$ is finish time of communication between task i and task j and $t_f(n_i)$ is finish time of node n_i.

A scheduling model consists of an application, a target computing environment, and a performance criterion for scheduling [6-9]. This model is defined in the following, including a generalization towards heterogeneous processors.

One of the important characteristic of a DAG is upward rank that is calculated as follows:

Definition 2.2. (Upward rank). *The upward rank of $n_i \in \boldsymbol{V}$ is recursively defined by*

$$rank_u(n_i) = \overline{w_{n_l}} + \max_{n_j \in succ(n_i)} \left(\overline{c(e_{ij})} + rank_u(n_j) \right), \tag{4}$$

where $succ(n_i)$ is the set of immediate successors of task n_i, $\overline{c(e_{ij})}$ is the average communication cost of edge(i, j) and $\overline{w_{n_l}}$ is the average computation cost of task n_i. For the exit

task n_{exit} (sink node), the upward rank is equal to

$$rank_u(n_{exit}) = \overline{w_{exit}}, \tag{5}$$

The task scheduling problem in this paper has the following conditions:

- Tasks have dependency to each other and are shown by DAG;
- There is communication cost between tasks in DAG (communication cost is considered when two dependent tasks are scheduled on different processors);
- Environment is static and all of information is known a priori;
- Processors are heterogeneous and connected to each other as a fully connected topology;
- There is an independent communication system which passes communication messages between processors.

3. **Related Work.** Distributed systems present a challenging paradigm for TS problem and considerable literature is devoted to its various aspects. However, the existing techniques generally aim to only "solve" the TS problem and the problem is not approached from a modeling perspective. The scheduling problem has been proven to be NP-complete except for a few restricted cases [10]. Indeed, the problem is NP-complete even in two simple cases: (1) scheduling tasks with uniform weights to an arbitrary number of processors [11] and (2) scheduling tasks with weights equal to one or two units to two processors [11]. There are only three special cases for which there exist optimal polynomial-time algorithms. These cases are (1) scheduling tree-structured task graphs with uniform computation costs on an arbitrary number of processors [12]; (2) scheduling arbitrary task graphs with uniform computation costs on two processors [13]; and (3) scheduling an interval-ordered task graph [14] with uniform node weights to an arbitrary number of processors [15].

Traditional methods that are used to solve this problem can be generally categorized as either heuristic (list-based scheduling) or random search (such as genetic algorithm, simulated annealing, and particle swarm optimization). A few of the most significant recent directions in this area are highlighted below. Kong et al. (2008) [16] incorporate Particle Swarm Optimization (PSO), with list scheduling and develop an alternative PSO algorithm for multiprocessor tasks scheduling. Unlike most duplication-based heuristics which try to duplicate all possible ancestor nodes of a given join node, Shin et al. (2008) [17] propose a novel duplication based algorithm which duplicate some nodes based on some parameters such as start time and earliest free slot, without redundant duplications. Duplication is referred to as redundant when it does not lead to an improvement in the performance or a reduction in the schedule length of an individual processor. Their method has two steps. In the first step, task priority list is made; and in the second step, the best possible processor, that is, the one that allows the earliest start time is selected to execute the task with highest priority.

Kuo et al. (2008) [18] minimize makespan in flow-shop scheduling problem and introduce new hybrid particle swarm optimization model named HPSO. They combine random-key (RK) encoding scheme, individual enhancement (IE) scheme, and particle swarm optimization (PSO). Also, Chui et al. (2009) [19] proposed a hybrid of constraint-based reasoning (CBR) mechanism and PSO, called CBPSO, to solve timetables scheduling problems for customer-service departments. When PSO searches the solution space, CBR is utilized to reduce the invalid solution space of particle search.

Choi and Lee (2008) [20] used branch and bound algorithm to minimize the number of tardy jobs in a two-stage hybrid flow shop scheduling. Each job is processed through two production stages in series, each of which has multiple identical parallel machines. The

branch and bound algorithm obtains the lower and upper bounds as well as the dominance properties to reduce the search space. In addition, two-phase heuristic algorithms are suggested to obtain good solutions for large-scale problems within a reasonable amount of computation time.

Yoo (2009) [8] minimizes the total tardiness and total number of processors used in scheduling of soft real-time DAGs on homogeneous multiprocessors systems by a multi-objective genetic algorithm (MoGA). To evaluate chromosomes in a population, Yoo converts multi-objective problem to single objective problem by adaptive weighted approach (AWA). In this approach, the useful information from the current population is used to readjust objective weights and to obtain a search pressure toward a positive ideal point. In contrast, in (2009) [21], Li et al. propose a problem-specific genetic algorithm to handle multiprocessor real-time task scheduling. This work employs not only limited problem-specific information, but it also makes most of such information throughout the evolution of the genetic algorithm.

Hwang et al. (2008) [22] schedule a DAG on multiprocessor system using genetic algorithm to achieve the best scheduling with minimum makespan. They design the new encoding mechanism with a multi-functional chromosome that uses the priority representation, i.e., the so-called priority-based multi-chromosome (PMC). There are also a few works that have focused on the use of linear programming (LP) such as (2008) [23] in which a linear programming driven genetic algorithm is introduced which combines the capabilities of LP and GA and tries to minimize combined cost of all users in a coordinated manner. Furthermore, Moges and Yu (2004) [24] proposed a method in which solutions for an optimal allocation of fraction of loads to nodes in single level tree networks are obtained via linear programming.

Control theory has been used to design adaptive resource management in different applications such as power management, task scheduling, QoS adaptation in Web servers, and load balancing [25]. In [26,27], the control theory is applied to schedule real-time tasks on distributed systems [25]. Lu et al. [26] present a Feedback Control real-time Scheduling (FCS) framework for adaptive real-time systems. They apply a methodology based on control theory to systematically design FCS algorithms to satisfy the transient and steady state performance specifications of real-time systems. In [27], a framework is proposed based on feedback control that incrementally corrects system performance to achieve its target in the absence of initial load and resource assumptions. It allows the designer to specify the desired temporal behavior of system adaptation for a distributed real-time application such as the speed of convergence to desired performance upon load or resource changes. Furthermore, unlike QoS optimization approaches, their solution meets performance guarantees with no accurate knowledge of task execution parameters. This can be considered as a key advantage in a poorly modeled environment.

Marbini and Sacks [28] propose a closed-loop dynamic scheduling algorithm which employs a PID controller to control CPU utilization. In this model, tasks are submitted in a uniform distribution scheme and the controller uses very simple equations to control the feedback error and therefore accepting or rejecting new tasks in order to maintain the CPU utilization in an accepted level. Roux and Déplanche in [29] proposed an extension of T-time Petri net model which takes into account the scheduling of the software tasks distributed over multi-processor hardware architecture. This work is concerned with static priority pre-emptive based scheduling. This extension consists of mapping the way the different schedulers of the system activate or suspend the tasks into the Petri net model. In this work, the underlying hardware architecture is composed of multiple processors that are all static.

Related Work on Control Theory

In contrast to the previous works that usually introduce a specific algorithm, in this paper we propose a general mathematical model based on analytical proof using state space theory that is capable of modelling a wide range of scheduling problems. Here, we consider an analytical comparison between our model and two other similar works that use control theory in their model. In 2002, Lu et al. [26] proposed an architecture which features a feedback control loop that is invoked at every sampling instant k similar to our model. This architecture has a basic scheduler that schedules admitted tasks with a scheduling policy such as Earliest Deadline First (EDF). The scheduler uses two control variables and the controller compares the performance references with corresponding controlled variables to get the current errors. Because this work is mainly developed for real-time systems, another component of this model is QoS actuator which dynamically changes the total estimated requested utilization at each sampling instant k according to the control input variables. This model is mainly developed for real-time systems which have some QoS parameters.

Also in 2002, Marbini and Sacks [28] proposed a simple PID controller that is capable of controlling CPU utilization factor for simple tasks being submitted in a uniformly distributed scheme. The tasks are independent and are generated periodically. There is just one processing unit. The miss ratio of the accepted tasks was periodically monitored and sent to PID. The PID calculates the control signal using control equations. Based on the results of these calculations, another component called Admission Controller (AC) decides whether to accept new tasks or not. While the above approaches succeeded in using control theory for task scheduling, their domain of application remained very limited and cover very simplified versions of scheduling, i.e., addressing only QoS in [26] and only one processing unit in [28].

In contrast to the above, we aim to reach a 'general' model of scheduling in state space by using linear switching systems. This modeling space allows the various constraints and applications of scheduling in real environment to be efficiently and faithfully represented, such as those in resource management. The components of the scheduler are developed in such a way that most of the parameters of the problems can be covered. Another important advantage of the proposed method over these similar works is that the modeling of the constraints and also stability of the method is proven using control theory principals; furthermore all of the components of a task scheduling problem are well mapped to corresponding one in control theory. This leads to a powerful theoretical method applicable to practical problems.

The major contributions of this work can be stated as follows:

- The task scheduling problem is mapped to linear switching state space. This mapping transforms the problem of task scheduling to choosing among several switches, hence simplifying the controller (scheduler) design process.

- The above mapping is a general modeling framework, not just an algorithm, to solve the scheduling problem. In other words, the modeling strategy can be adapted to other variations of the scheduling problem such as job shop and work flow scheduling as well as project management problems.

- The switching state space introduces various theoretical measures of performance, such as system stability in this paper, for the problem of task scheduling.

- The model is capable of supporting multiple constraints in a real environment such as considering the small time steps as well as the optimal routing between processing nodes.

4. **Switching Systems.**

A switching system is a time-variant system that consists of a finite number of subsystems and a logical rule that orchestrates switching between these subsystems. Mathematically, these subsystems are usually described by a collection of

indexed differential or difference equations. In this paper, we focus on switching systems whose subsystems are systems with a collection of discrete-time linear time invariant (LTI) systems [30]:

$$\begin{aligned} X[k+1] &= A_i X[k] + B_i U[k], \quad i \in \{1, 2, 3, \ldots, N\} \\ Y[k] &= C_i X[k] + D_i U[k], \quad i \in \{1, 2, 3, \ldots, N\} \end{aligned} \tag{6}$$

where $x[k]$ is state variable at step k, A_i, B_i, C_i, D_i is ith subsystem matrices of state space, and Z^+ stands for nonnegative integers. The finite set Q is an index set and stands for the collection of discrete modes. The logical rule that orchestrates switching between these subsystems generates switching signals, which are usually described as classes of piecewise constant maps $Z^+ \to Q$. By piecewise constant map, we mean that the switching signal $\sigma(k)$ has a finite number of discontinuities on any finite interval of Z^+. This requirement is always satisfied by discrete-time switching systems because a minimum-stay time threshold helps avoid possible chattering in the switching signal. Chattering in switching systems occurs when switching signal from one subsystem to another is done quickly. In other words, the minimum required time for the system to stay in a specific subsystem is not satisfied [30]. It is usual to reform state space equation of systems to a switching system model; because by this formation, the problem of determining of control vectors transforms to an optimization problem that searches for an appropriate switching controller [31]. Figure 1 shows an example of linear switching system and its formulation.

5. **The Proposed Scheduling Model in State Space.** Solving heterogeneous TS problems by traditional modeling/search approaches is generally NP complete. Hence, a different modeling paradigm based on nonlinear state space paradigm is proposed here.

Consider a general form of a nonlinear system in discrete time is:

$$\begin{gathered} \begin{cases} X[(k+1)\Delta T] = f[X[k\Delta T], U[k\Delta T], k\Delta T] \\ Y[k\Delta T] = g[X[k\Delta T], U[k\Delta T], k\Delta T] \end{cases} \\ k \in \{1, 2, 3, \ldots\}, \quad X \in \mathbb{R}^n, \quad Y \in \mathbb{R}^n, \quad U \in \mathbb{R}^m \end{gathered} \tag{7}$$

where f and g are nonlinear vector functions, X is a vector of state variables, U is control vector, Y is system output and ΔT is the length of time steps. Here, we propose to model TS in the above nonlinear form, Equation (7). Various components of this model are shown in subsequent subsections.

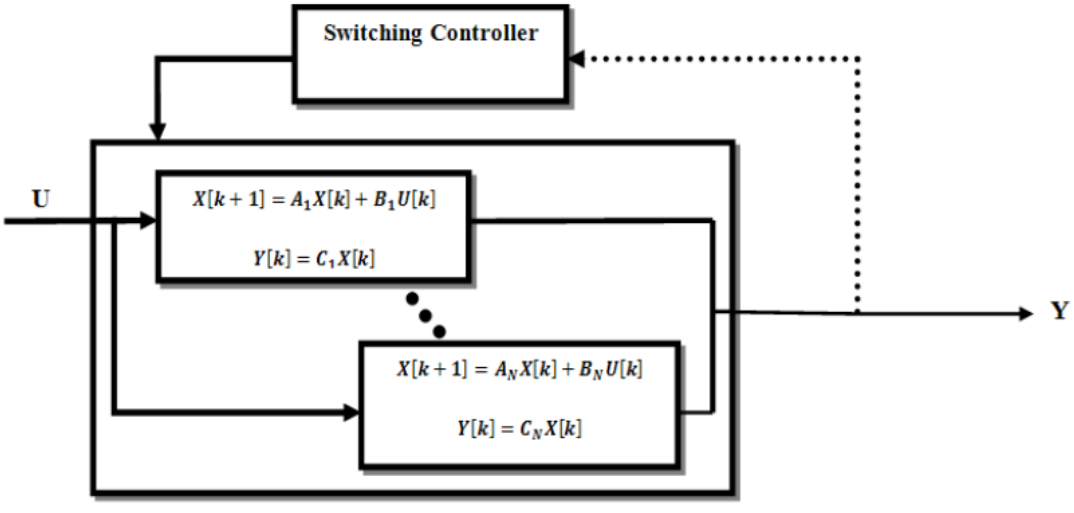


FIGURE 1. Linear switching system

5.1. **Basic definitions in the proposed model.**

• **State variables**

State variables in this model correspond to a vector (X) whose size is equal to the number of tasks, and each element has a value between 0 (the task has completed) and 1 (the execution of task has not started). Hence, each element $x_j[k]$ corresponds to the remaining process of a given task j at kth step.

$$X[k] = [x_1[k] \quad x_2[k] \quad \dots \quad x_T[k]]' \tag{8}$$

where ($T = |\boldsymbol{V}|$) and all of the elements of $X[k]$ are usually initialized to '1'.

• **Control Vector**

Control vector $U[k]$ is a matrix $M \times T$, where $T = |\boldsymbol{V}|$, $M = |\boldsymbol{R}|$, and shows the execution status of each task and each processor in kth time step. Each element in this matrix is between '0' and '1' and depicts available processing power of each processor. The characteristic of this matrix is that at each step there is at most one nonzero element in each row and each column. Element $u_{i,j}$ in kth time step shows the percentage of utilizing processor i in execution of task j. If $u_{i,j}[k] = 1$, task j is said to be fully executed on resource i in kth time step. An important constraint in control vector is that all of the tasks, that are simultaneously executed at kth time step, must be independent from each other. Furthermore, each task can be processed on only one processor, and each processor can process at most one task at any given time.

• **Constraints in designing Controller**

Stability analysis and design of the control algorithm should incorporate the problem constraints. These constraints are summarized as follows:

1. {Initialization} The initial state vector is set to a column of ones, i.e., all of the tasks need to be executed at the beginning.

$$X[0] = [1 \quad 1 \quad \dots \quad 1]' \tag{9}$$

2. All of the elements of the control vector U are chosen in range $[0, 1]$, i.e.,

$$\forall i, j \quad 0 \leq u_{i,j} \leq 1 \tag{10}$$

3. {Constraint of Independence} In each column and each row of control vector, U, there is at most one nonzero element. In other words, there is at most one processor allocated to each task at any given time. Also, that there is at most one task assigned to a processor at any given time.

$$\text{if } \exists u_{i,j} \neq 0 \Rightarrow \{\forall j' \neq j \quad u_{i,j'} = 0, \quad \forall i' \neq i \quad u_{i',j} = 0\} \tag{11}$$

4. {Constraint of Predecessors} Based on precedence relations in DAG, each task can be executed only after execution of its predecessor(s). In other words,

$$\text{if } \exists u_{i,j} \neq 0 \Rightarrow (\forall r \in pred(n_j) x_r = 0) \tag{12}$$

5. {Concurrent Task Independence} Based on precedence relations in DAG, all of tasks that are executed concurrently are independent from each other. In other words,

$$\forall i, j, r, s \quad \text{if } (u_{i,j} \neq 0 \quad \& \quad u_{r,s} \neq 0) \Rightarrow (\mu_{j,s} = 0 \quad \& \quad \mu_{s,j} = 0) \tag{13}$$

6. {Constraint of Communication Cost} If two tasks with precedence relation are scheduled on different processors, non-zero communication cost is considered. Otherwise, communication cost is $\mu_{i,j} = 0$.

7. {Constraint of Task Completion} After completion of task execution, corresponding processor is freed. In other words,

$$\forall i, j \quad \text{if } x_j[k] = 0 \Rightarrow u_{i,j}[k+1] = 0 \tag{14}$$

8. {Constraint of Completeness} All of the given tasks must be assigned (and completed since a task does not free a processor until it is completely executed).

$$\forall j \; \exists i \; \exists k \quad u_{i,j}[k] \neq 0 \tag{15}$$

• **The Non-linear System Function *f***

The function f in system dynamics, Equation (6), relates the defining parameters of the system such as the vector of state variables X, control vector U, time step k, as well as the DAG. For the TS problem, f is defined in discrete time form as follows,

$$X[k+1] = \max\left(0, X[k] - \Delta T * Diag\left(\left[\frac{1}{\theta_{ij}} * U\right]\right)\right) \tag{16}$$

where ΔT is the time step in seconds, $Diag$ is the main diagonal of matrix A, and $\theta = [\theta_{ij}]_{\boldsymbol{T*M}}$ where θ_{ij} shows computation time of task i on processor j. The max operator is chosen to ensure that the state variable does not become negative. In other words, since execution time of each task is different, some tasks may be completed before others during the same discrete time interval and then it is essential to use max operator. Also, the communication cost between tasks (on same or different processors) is shown by μ which is a $T \times T$ matrix. Communication cost is considered in the process of determining control vectors. These control vectors remain the same when the communication between two tasks is running. Once communication is completed, the processors resume their operation as dictated by the control vector. For example, if U^l, lth control vector, is applied in range $[a, b]$ step, next control vector U^{l+1} is applicable if all of its information received. Therefore, U^{l+1} is applied in range $[c, d]$ step which $c \geq b$. During b to c steps, current control vector is still the previous control vector U^l, which does not affect the state of system.

5.2. **System stability.** In control viewpoint, it is necessary to prove some basic theories such as stability that illustrates the transition and final response for system [31-33]. Extending this definition to the task scheduling paradigm enables the theoretical study of such features. In other words, with the present approach, it is now possible to rigorously study "convergence" behavior of scheduling problems. The set of C controllers that stabilize the nominal feedback system (C, P_0) are described by below the Yula Parameterization theorem as follows:

Theorem 5.1. [34] *Let P_0 be an open loop stable plant. Then the set of all controllers stabilizing the nominal feedback system is*

$$C = \left\{C(s) = \frac{Q(s)}{1 - P_{0(s)}Q(s)}\right\} \tag{17}$$

where $Q(s)$ is proper and stable. Yula also introduces a similar theorem for MIMO systems. In following the above work, we introduce here the below theorem to address the convergence and stability of TS problems.

Theorem 5.2. *If control vectors $U^1, U^2, \ldots, U^k$ satisfy all constraints as mentioned in Section 5.1, at step a_k we have*

$$X[a_k] = \max\left(0, X[a_{k-1}] - Diag\left(\left[\frac{1}{\theta}\right] * U^k\right)\right) \tag{18}$$

Proof: We suppose that U^l, $l = 1, 2, \ldots, k$ operates at time slot as

$$0 \leq U^1 \leq a_1, \quad A_1 \leq U^2 \leq a_2, \quad \ldots, \quad a_{k-1} \leq U^k \leq a_k$$

Also, we can write

$$\left[\frac{1}{\theta_{T\times M}}\right] * U_{M\times T} = \begin{bmatrix} \frac{1}{\theta_{11}} & \cdots & \frac{1}{\theta_{1M}} \\ \vdots & \ddots & \vdots \\ \frac{1}{\theta_{T1}} & \cdots & \frac{1}{\theta_{TM}} \end{bmatrix} * \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1T} \\ u_{21} & u_{22} & \cdots & u_{2T} \\ \vdots & \vdots & \ddots & \vdots \\ u_{M1} & u_{M2} & \cdots & u_{MT} \end{bmatrix} \tag{19}$$

$$= \begin{bmatrix} \frac{u_{11}}{\theta_{11}} + \frac{u_{21}}{\theta_{12}} + \frac{u_{31}}{\theta_{13}} + \cdots + \frac{u_{M1}}{\theta_{1M}} & & \cdots & \\ & \frac{u_{12}}{\theta_{21}} + \frac{u_{22}}{\theta_{22}} + \frac{u_{32}}{\theta_{23}} + \cdots + \frac{u_{M2}}{\theta_{2M}} & & \\ & \vdots & \ddots & \vdots \\ & & \cdots & \frac{u_{1T}}{\theta_{T1}} + \frac{u_{2T}}{\theta_{T2}} + \frac{u_{3T}}{\theta_{T3}} + \cdots + \frac{u_{MT}}{\theta_{TM}} \end{bmatrix}$$

And,

$$Diag\left(\left[\frac{1}{\theta_{T\times M}}\right] * U_{M\times T}\right) = \begin{bmatrix} \frac{u_{11}}{\theta_{11}} + \frac{u_{21}}{\theta_{12}} + \frac{u_{31}}{\theta_{13}} + \cdots + \frac{u_{M1}}{\theta_{1M}} \\ \frac{u_{12}}{\theta_{21}} + \frac{u_{22}}{\theta_{22}} + \frac{u_{32}}{\theta_{23}} + \cdots + \frac{u_{M2}}{\theta_{2M}} \\ \vdots \\ \frac{u_{1T}}{\theta_{T1}} + \frac{u_{2T}}{\theta_{T2}} + \frac{u_{3T}}{\theta_{T3}} + \cdots + \frac{u_{MT}}{\theta_{TM}} \end{bmatrix} \tag{20}$$

In Equation (20), each row contains only one non-zero fraction. This is because $u_{ij} = 1$, means task i executes on processor j and based on 3rd constraint, each task can only execute on one processor. Thus, we can write

$$Diag\left(\left[\frac{1}{\theta_{T\times M}}\right] * U_{M\times T}\right) = \begin{bmatrix} \frac{u_{r_1 1}}{\theta_{1 r_1}} & \frac{u_{r_2 2}}{\theta_{2 r_2}} & \cdots & \frac{u_{r_T T}}{\theta_{T r_T}} \end{bmatrix}' \tag{21}$$

where r_i is the index of the processor that executes task i. It should be mentioned that it is possible that $r_i = r_j$, $i \neq j$ where task i and task j are executed in different time slots.

We can expand Equation (16) and write

$$X[a_1] = \max\left(0, \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} - Diag\left(\left[\frac{1}{\theta_{T\times M}}\right] * U^1_{M\times T}\right) * (a_1 - 0)\right) \tag{22}$$

We expand $Diag\left(\left[\frac{1}{\theta_{T\times M}}\right] * U^1_{M\times T}\right)$ based on Equation (21) and we set $\sigma^i_j = \frac{U^i_{r_j j}}{\theta_{j r_j}}$, then

$$X[a_1] = \max\left(0, \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} - a_1 * \begin{bmatrix} \frac{U^1_{r_1 1}}{\theta_{1 r_1}} \\ \vdots \\ \frac{U^1_{r_T T}}{\theta_{T r_T}} \end{bmatrix}\right) = \max\left(0, \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} - a_1 * \begin{bmatrix} \sigma^1_1 \\ \sigma^1_2 \\ \vdots \\ \sigma^1_T \end{bmatrix}\right) \tag{23}$$

and

$$\begin{aligned} X[a_2] &= \max\left(0, X[a_1] - Diag\left(\left[\frac{1}{\theta}\right] * U^2\right)\right) \\ &= \max\left(0, X[a_1] - (a_2 - a_1) * \begin{bmatrix} \sigma^2_1 \\ \sigma^2_2 \\ \vdots \\ \sigma^2_T \end{bmatrix}\right) \\ &= \max\left(0, \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} - a_1 * \begin{bmatrix} \sigma^1_1 \\ \sigma^1_2 \\ \vdots \\ \sigma^1_T \end{bmatrix} - (a_2 - a_1) * \begin{bmatrix} \sigma^2_1 \\ \sigma^2_2 \\ \vdots \\ \sigma^2_T \end{bmatrix}\right) \end{aligned} \tag{24}$$

If we continue, we will achieve

$$X[a_k] = \max\left(0, X[a_{k-1}] - Diag\left(\left[\frac{1}{\theta}\right] * U^k\right)\right) \tag{25}$$

Theorem 5.3. (*Stability*). *If control vectors $U^1, U^2, \ldots, U^k$ satisfy all constraints as mentioned in Section 5.1, we achieve $||X[a_k]|| = 0$, at step a_k.*

Proof: Based on previous theorem, we have

$$X[a_T] = \max\left(1, X[a_{T-1}] - Diag\left(\left[\frac{1}{\theta}\right] * U'\right)\right) \tag{26}$$

That U^l, $l = 1, 2, \ldots, k$ operates at time slot as

$$0 \le U^1 \le a_1, \quad a_1 \le U^2 \le a_2, \quad \ldots, \quad a_{f-1} \le U^f \le a_f$$

where $f \le k$. In following Equation (25), we now expand Equation (26).

$$\begin{aligned} X[a_f] &= \max\left(0, X[a_{f-1}] - Diag\left(\left[\frac{1}{\theta}\right] * U^f\right)\right) \\ &= \max\left(0, \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} - a_1 * \begin{bmatrix} \sigma_1^1 \\ \sigma_2^1 \\ \vdots \\ \sigma_T^1 \end{bmatrix} - (a_2 - a_1) * \begin{bmatrix} \sigma_1^2 \\ \sigma_2^2 \\ \vdots \\ \sigma_T^2 \end{bmatrix} - \cdots - (a_f - a_{f-1}) * \begin{bmatrix} \sigma_1^f \\ \sigma_2^f \\ \vdots \\ \sigma_T^f \end{bmatrix}\right) \\ &= \max\left(0, \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} - \begin{bmatrix} a_1\sigma_1^1 + (a_2 - a_1) * \sigma_1^2 + \cdots + (a_k - a_{k-1}) * \sigma_1^f \\ \vdots \\ a_1\sigma_n^1 + (a_2 - a_1) * \sigma_n^2 + \cdots + (a_k - a_{k-1}) * \sigma_T^f \end{bmatrix}\right) \end{aligned} \tag{27}$$

Based on task scheduling constraints, each task must be executed only at one time slot; thus in each row, only one element is non-zero. We propose task i to execute on time slot i, then $\sigma_i^{s_i} \neq 0$.

We can now write,

$$X[a_f] = \max\left(0, \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} - \begin{bmatrix} (a_{s_1} - a_{s_1-1}) * \sigma_1^{s_1} \\ \vdots \\ (a_{s_f} - a_{s_f-1}) * \sigma_T^{s_f} \end{bmatrix}\right) \tag{28}$$

In the above equation, we can rewrite second statement as:

$$(a_{s_i} - a_{s_i-1}) * \sigma_i^{s_i} = (a_{s_i} - a_{s_i-1}) * \frac{U_{r_i i}^{s_i}}{\theta_{ir_i}} \tag{29}$$

Based on length of each slot is equal to maximum execution time of tasks, we can write

$$(a_{s_i} - a_{s_i-1}) \ge \theta_{ip_i} \Rightarrow (a_{s_i} - a_{s_i-1}) * \frac{U_{r_i i}^{s_i}}{\theta_{ir_i}} \ge 1 \Rightarrow (a_{s_i} - a_{s_i-1}) * A_i^{s_i} \ge 1 \tag{30}$$

And then,

$$X[a_f] = \max\left(0, \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} - \begin{bmatrix} (a_{s_1} - a_{s_1-1}) * A_1^{s_1} \\ \vdots \\ (a_{s_f} - a_{s_f-1}) * A_T^{s_f} \end{bmatrix}\right) = 0 \quad \Rightarrow ||X[a_f]|| = 0 \tag{31}$$

And theorem is proven.

5.3. **Linear function f_L.** In order to simplify a complex model and to benefit from the extensive theoretical results in the field of linear systems, nonlinear equations are usually converted to linear equations [32]. Linear control systems are much easier to design: however, this conversion usually leads to approximation which may be unacceptable. Here, the construct of linear control systems is used to find appropriate linear switching controllers (optimization problem). Nonlinear constraints are considered separately from the linear switching controller; hence the resulting model is exact. By simplifying the model to a linear switching controller, these constraints effectively "reduce" the search space for appropriate controllers [31]. More specifically, we substitute a linear function f_L for its nonlinear counterpart f by considering all of the nonlinearities as nonlinear constraints in the controller design process. The resulting model is then converted it to a switched control problem. For this reason, max and *Diag* operators should be removed from Equation (16). The following subsections address this process.

5.3.1. *Removing Diag operator and simplification.* By the following theorem, *Diag* operator is removed from the product of two general matrices.

Theorem 5.4. *The linear form of $Diag(Q \times U)$, where Q is an $a \times b$ matrix and U is a $b \times a$ matrix, is*

$$Diag(Q \times U) = \begin{bmatrix} q_{11} \cdots q_{1b} & 0 & . & \ldots\ldots & 0 \\ 0 & q_{21} \cdots q_{2b} & 0 & \ldots\ldots & 0 \\ 0 & 0 & q_{31} \cdots q_{3b} & 0\ldots\ldots & 0 \\ . & . & 0 & \ldots\ldots & . \\ . & . & . & \ldots\ldots & . \\ . & . & . & \ldots\ldots & . \\ 0 & . & . & \ldots.0 & q_{a1} \cdots q_{ab} \end{bmatrix} * \begin{bmatrix} u_{11} \\ u_{21} \\ u_{31} \\ \vdots \\ u_{b1} \\ u_{12} \\ u_{22} \\ u_{32} \\ \vdots \\ u_{b2} \\ \vdots \\ u_{1a} \\ u_{2a} \\ u_{3a} \\ \vdots \\ u_{ba} \end{bmatrix} \tag{32}$$

Proof: We can show $Q_{a\times b}$ and $U_{b\times a}$ as vectors:

$$Q_{a\times b} = \begin{bmatrix} \bar{q}^1 \\ \bar{q}^2 \\ . \\ \bar{q}^i \\ . \\ . \\ \bar{q}^a \end{bmatrix}, \quad \bar{q}^i = [\bar{q}^i_1 \quad \bar{q}^i_2 \quad \cdots \quad \bar{q}^i_b] \tag{33}$$

$$U_{b\times a} = [\bar{u}^1 \quad \bar{u}^2 \quad . \quad \bar{u}^j \quad . \quad . \quad \bar{u}^a], \quad \bar{u}^j = \begin{bmatrix} \bar{u^j_1} \\ \bar{u_2}^j \\ \vdots \\ \bar{u}^j_b \end{bmatrix} \tag{34}$$

Therefore, we can write Q as a vector with a rows and one column and also, U as a vector with one row and b columns. We can write

$$\begin{aligned} Diag(Q \times U) &= Diag\left(\begin{bmatrix} \bar{q}^1 \\ \bar{q}^2 \\ . \\ \bar{q}^i \\ . \\ . \\ \bar{q}^a \end{bmatrix} * [\bar{u}^1 \quad \bar{u}^2 \quad . \quad \bar{u^j} \quad . \quad . \quad \bar{u}^a]\right) \\ &= Diag\left(\begin{bmatrix} \bar{q}^1\bar{u}^1 & \cdots & \bar{q}^1\bar{u}^a \\ \vdots & \ddots & \vdots \\ \bar{q}^a\bar{u}^1 & \cdots & \bar{q}^a\bar{u}^a \end{bmatrix}\right) \\ &= [\ \bar{q}^1\bar{u}^1 \quad \bar{q}^2\bar{u}^2 \quad \ldots \quad \bar{q}^a\bar{u}^a\ ] \end{aligned} \tag{35}$$

Also we can write

$$[\ \bar{q}^1\bar{u}^1 \quad \bar{q}^2\bar{u}^2 \quad \ldots \quad \bar{q}^a\bar{u}^a\ ] = \begin{bmatrix} \bar{q}^1 & 0 & . & . & . & . & . & . & 0 \\ 0 & \bar{q}^2 & 0 & . & . & . & . & . & 0 \\ 0 & 0 & \bar{q}^3 & 0 & . & . & . & . & 0 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ 0 & . & . & . & . & . & . & 0 & \bar{q}^a \end{bmatrix} * \begin{bmatrix} \bar{u}^1 \\ \bar{u^2} \\ \vdots \\ \bar{u}^a \end{bmatrix} \tag{36}$$

And finally, we reach

$$Diag(Q \times U) = \begin{bmatrix} q_{11} & \cdots & q_{1b} & 0 & & . & . & . & . & . & 0 \\ 0 & & q_{21} & \cdots & q_{2b} & 0 & & . & . & . & . & 0 \\ 0 & & 0 & & q_{31} & \cdots & q_{3b} & 0 & . & . & . & 0 \\ . & & . & & 0 & & & . & . & . & . & . \\ . & & . & & . & & & . & . & . & . & . \\ . & & . & & . & & & . & . & . & . & . \\ 0 & & . & & . & & & . & . & . & 0 & q_{ab} \end{bmatrix} * \begin{bmatrix} u_{11} \\ u_{21} \\ u_{31} \\ \vdots \\ u_{b1} \\ u_{12} \\ u_{22} \\ u_{32} \\ \vdots \\ u_{b2} \\ \vdots \\ u_{1a} \\ u_{2a} \\ u_{3a} \\ \vdots \\ u_{ba} \end{bmatrix} \tag{37}$$

And theorem is proven.

Based on Theorem 5.4, Equation (16) converts to

$$X[k+1] = \max \left(0, X[k] - \Delta T * \begin{bmatrix} \bar{q}^1 & 0 & . & . & . & . & . & . & 0 \\ 0 & \bar{q}^2 & 0 & . & . & . & . & . & 0 \\ 0 & 0 & \bar{q}^3 & 0 & . & . & . & . & 0 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ 0 & . & . & . & . & . & . & 0 & \bar{q}^T \end{bmatrix} * \begin{bmatrix} \bar{u}^1 \\ \bar{u}^2 \\ \vdots \\ \bar{u}^T \end{bmatrix} \right) \tag{38}$$

where $Q_{T\times M} = \frac{1}{\theta_{T\times M}}$.

5.3.2. *Linearization of* max *operator.* Here, we remove max operator by considering a constraint on designing control vectors. This process is illustrated by the following example.

Example 5.1. *If there are three processors as well as three independent tasks, a possible control vector is*

	Task 1	Task 2	Task 3
Resource 1	1	0	0
Resource 2	0	1	0
Resource 3	0	0	1

FIGURE 2. A possible control vector

Execution time of each task is assumed to be:

- Execution time of first task on first processor is 10 units.
- Execution time of second task on second processor is 5 units.
- Execution time of third task on third processor is 12 units.

Then, it is possible to break the control vector of Figure 2 to several control vectors as in Figure 3.

In other words, we zero out the corresponding tasks in control vector in an ascending order of execution times. By this method, if execution of one task has ended, its corresponding processor is freed, and its corresponding elements of state variables do not become negative.

By considering this new constraint on control vector, we reach a linear model as follows:

$$X[k+1] = X[k] - \Delta T * \begin{bmatrix} \bar{q}^1 & 0 & . & . & . & . & . & . & 0 \\ 0 & \bar{q}^2 & 0 & . & . & . & . & . & 0 \\ 0 & 0 & \bar{q}^3 & 0 & . & . & . & . & 0 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ 0 & . & . & . & . & . & . & 0 & \bar{q}^T \end{bmatrix} * \begin{bmatrix} \bar{u}^1 \\ \bar{u}^2 \\ \vdots \\ \bar{u}^T \end{bmatrix} \tag{39}$$

where $Q_{T\times M} = \frac{1}{\theta_{T\times M}}$.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(a) (b) (c)

FIGURE 3. (a) Contorl vector 1, (b) control vector 2, and (c) control vector 3

5.4. **Switching model in task scheduling problems.** In Equation (39), U is a linear input with non-linear constraints. It is now desirable to find a simple scheme for controller design. By converting this model to a switching linear space model, we can use various optimization/design methods to solve it.

If Equation (39) is written as

$$X[k+1] = A_i X[k] + B_i U[k], \quad i \in \{1, 2, 3, \ldots, N\} \tag{40}$$

Then in switching model,

$$\begin{gathered} (\forall s = 1, 2, 3, \ldots, S \quad i = s \quad A = A_i, \quad B = B_i) \\ X[k+1] = AX[k] + BU_s \end{gathered} \tag{41}$$

where S is the number of sub-systems. Minimum value of subsystems S is equal to the number of levels in DAG; because at each level tasks are independent from each other and then they can possibly execute in parallel. Maximum number of sub-systems is equal to the number of tasks in DAG; i.e., in each time slot, only one task is executed. Therefore, we can write

$$\text{Numbers of Levels of DAG } \leq S \leq \text{ Number of Tasks} \tag{42}$$

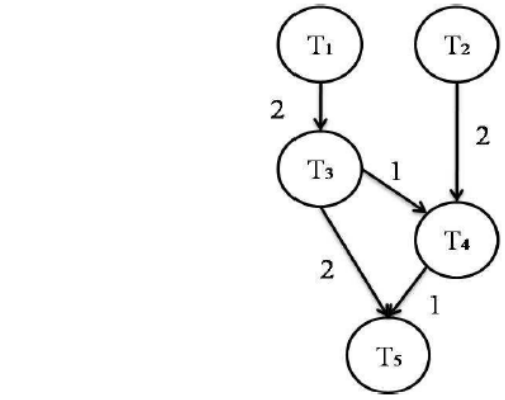
Switching signal is a discrete signal that determines which subsystem is active at kth step. Controller design is equal to determining switching signal and then finding appropriate values for switching indices.

6. **The Control Approaches.** The above proposed model is general and applicable to different scheduling problems such as human resource management and project management. Here, we focus on the problem of task scheduling for distributed systems. Many researchers have to date approached this problem but these approaches are generally heuristic and lack rigorous mechanisms of analysis and guarantee of performance. We assume that all processors are generally completely available and we can use all of their processing power to solve the TS problem; therefore, each element of control vector, U is either '0' or '1'.

In this section, at first we transform a given scheduling problem to state space form. Then we present a systematic scheme to determine control vectors that schedule DAGs with minimum total execution time.

6.1. **Transforming the TS problem in state space form.** DAG and θ matrices are known in the numerical example of Figure 4. In this example, we design control vectors manually. As shown in Table 1, in 25th step ($Time = 25$) all elements of X are zero; therefore, total execution time is 25. Figure 5 shows Gantt chart of this problem.

Table 1 shows state variable values based on (16) on step $k \in [0, 25]$. Figure 6 shows switching signal of numerical example.



(a)

θ Matrix		
	P_1	P_2
T_1	5	8
T_2	11	15
T_3	4	7
T_4	6	7
T_5	1	2

(b)

$0 \le k < 8$: U_1

	T_1	T_2	T_3	T_4	T_5
P_1	0	1	0	0	0
P_2	1	0	0	0	0

$8 \le k < 11$: U_2

	T_1	T_2	T_3	T_4	T_5
P_1	0	1	0	0	0
P_2	0	0	1	0	0

$11 \le k < 15$: U_3

	T_1	T_2	T_3	T_4	T_5
P_1	0	0	0	0	0
P_2	0	0	1	0	0

$15 \le k < 16$: U_4

	T_1	T_2	T_3	T_4	T_5
P_1	0	0	0	0	0
P_2	0	0	0	0	0

$16 \le k < 22$: U_5

	T_1	T_2	T_3	T_4	T_5
P_1	0	0	0	1	0
P_2	0	0	0	0	0

$22 \le k < 24$: U_6

	T_1	T_2	T_3	T_4	T_5
P_1	0	0	0	0	0
P_2	0	0	0	0	0

$24 \le k < 26$: U_7

	T_1	T_2	T_3	T_4	T_5
P_1	0	0	0	0	1
P_2	0	0	0	0	0

$26 \le k$: U_8

	T_1	T_2	T_3	T_4	T_5
P_1	0	0	0	0	0
P_2	0	0	0	0	0

(c)

FIGURE 4. A numerical example: (a) DAG, (b) time table and (c) control vectors

6.2. **A systematic method to determine control vectors.** In the previous subsection, a numerical example illustrates how TS problem can be solved manually; but manual determination of control vectors is not always trivial. In this subsection, we propose that yield similar results to an optimal schedule based on HEFT [3].

TABLE 1. State variable values

k	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
X_1	1	0.875	0.75	0.625	0.5	0.375	0.25	0.125	0	0	0	0	0
X_2	1	0.909	0.818	0.727	0.636	0.545	0.454	0.363	0.272	0.181	0.09	0	0
X_3	1	1	1	1	1	1	1	1	1	0.857	0.714	0.571	0.428
X_4	1	1	1	1	1	1	1	1	1	1	1	1	1
X_5	1	1	1	1	1	1	1	1	1	1	1	1	1
k	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**
X_1	0	0	0	0	0	0	0	0	0	0	0	0	0
X_2	0	0	0	0	0	0	0	0	0	0	0	0	0
X_3	0.285	0.142	0	0	0	0	0	0	0	0	0	0	0
X_4	1	1	1	1	0.833	0.666	0.499	0.332	0.165	0	0	0	0
X_5	1	1	1	1	1	1	1	1	1	1	1	1	0

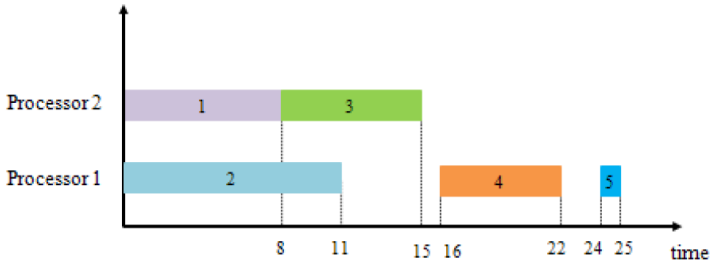


FIGURE 5. Gantt chart for the sample problem

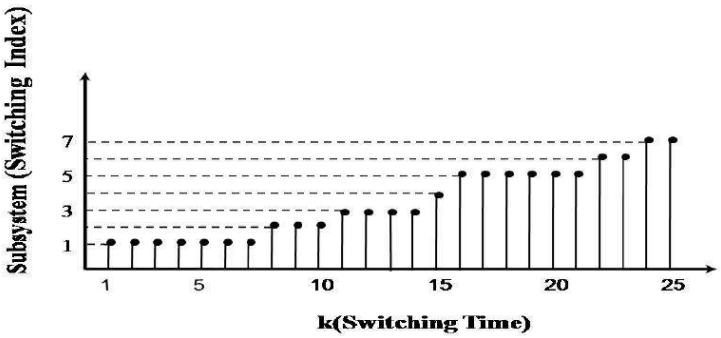


FIGURE 6. Switching signal

6.2.1. *Height sorted (HS) based controller.* At first, we label nodes in DAG based on their height:

$$height(n_i) = \begin{cases} 0 & \text{if } pred(n_i) = \emptyset \\ 1 + \max\limits_{n_j \in pred(n_i)} \{height(n_j)\} & \text{otherwise} \end{cases} \tag{43}$$

Based on its definition, in each control vector, tasks on the same height do not have precedence relation, i.e., they can be executed on different processor control vectors are ordered in ascending order of height values. After grouping, we need to allocate tasks in each group to different processors to minimize total execution time. In this stage, we allocate each task to the processor that can finish it sooner than other processors. The start time and end time of control vector are then derived based on this information (groups and processors). And then the controller is executed. Start time of each slot is equal to the ready time of included tasks and finish time of each slot is equal to maximum finish time of included tasks. As shown in Figure 7, all of the constraints (which are stated in 4.1) on control vectors are satisfied by the algorithm. For instance, in each row in the control vector, there is always only one '1', all of the tasks are executed and also all of precedence relations are considered. Therefore, based on Theorem 5.3, the proposed algorithm is stable.

Time complexity of the first part of HS algorithm is $O(T \log T)$ for sorting tasks and time complexity of the second part is $O(T.M)$, because of lines 8 and 9, where T is number of tasks and M is number of processors. Generally, time complexity of HS algorithm is $O(T.M)$. Figure 8 shows a numerical example that is solved by this method. Figure 7 shows the HS algorithm.

6.2.2. *Ready tasks (RT) based controller.* The second proposed method, similar to the first method, has two phases: grouping and assigning tasks to processors. The difference is that the first method assigns tasks based on their level, whereas the second method assigns tasks based on all of the upward ranks of the remaining tasks. RT can be described

```
1.  ▶ 1. Part:
2.  Determine height of each task (node) in DAG.
3.  Grouping tasks based on their height.
4.  Sorting tasks in each group based on upward rank in
    descending order.
5.  If there is a group with size G that G>M, then break
    down group to subgroups with size M and G%M.
6.  ▶ 2. Part:
7.  for each group g do
8.      for each task t do
9.          Find processor r that finish time of on it
            is lower than other processors.
10.         Schedule t on r.
```

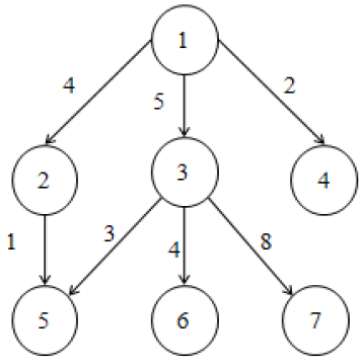
FIGURE 7. HS algorithm

as follows. In the first phase, we determine ready tasks in each step; where a ready task is one that all of its precedence tasks have already been selected or does not have any precedence tasks. At each step, $\alpha \leq G$ tasks are selected to make a group similar to HS, where $G = \min(M,$ number of tasks in ready tasks set). When a ready task is selected, it is removed from the set of ready tasks. Tasks in each group are sorted in descending order based on upward rank. Similar to HS, after the first stage, the processor is determined. Figure 9 shows this algorithm. As shown in Figure 8, all of the constraints (which are stated in 4.1) on control vectors are satisfied by the algorithm. For instance, in each row in the control vector, there is always only one '1', all of the tasks are executed and also all of precedence relations are considered. Therefore, based on Theorem 5.3, the proposed algorithm is stable. Time complexity of the first part of RT algorithm is $O(T \log T)$ for sorting tasks and time complexity of the second part is $O(T.M)$, because of lines 7 and 8, where T is number of tasks and M is number of processors. Generally, time complexity of RT algorithm is $O(T.M)$.

6.2.3. *Comparison among the Optimal Solution, HEFT, HS and RT.* In this subsection, we compare the results of HEFT, HS and RT with the optimum solution. Optimum solution is found by a linear method which finds all of possible schedules and selects schedules with the minimum makespan. Such exhaustive method is clearly computationally expensive; nevertheless it can serve as an excellent benchmark. Hence, this comparison is performed for a small DAG because finding optimum solution is not possible for large DAGs and processor networks due to their time complexity. In other words, this problem is NP-complete and is not trivially solvable by traditional methods when problem size grows. Figure 12 shows DAG and θ matrix. In this example, optimum solution has a makespan of 514. Since it is small, all of the other methods, HEFT, HS and RT, find the optimal scheduling with makespan equal to 514.

One of the best known heuristics in TS problems is Heterogeneous Earliest Finish Time (HEFT) method [3]. In earlier literature, the HEFT algorithm is reported to significantly outperform the other algorithm in terms of average schedule length ratio and speedup [3,35]; therefore, we use it as a benchmark algorithm in this paper. Figure 11 shows HEFT algorithm.

As shown, HEFT has two parts. In the first part, HEFT sorts nodes based on upward rank. In the second part, HEFT selects the task with the highest upward rank value at each step and schedules the selected task on the processor that minimizes its finish time. This step is iterated until all of tasks are scheduled on processors. Time complexity of



(a)

	P_1	P_2	P_3	P_4
T_1	18	11	19	17
T_2	18	10	15	17
T_3	17	11	22	13
T_4	16	12	25	20
T_5	11	11	7	18
T_6	17	8	15	11
T_7	17	16	14	21

(b)

	Height
T_1	0
T_2	1
T_3	1
T_4	1
T_5	2
T_6	2
T_7	2

(c)

Group 1 (height = 0)	1	0	0
Group 2 (height = 1)	2	3	4
Group 3 (height = 2)	5	6	7

(d)

$0 \le k < 11$: U_1

	T_1	T_2	T_3	T_4	T_5	T_6	T_7
P_1	0	0	0	0	0	0	0
P_2	1	0	0	0	0	0	0
P_3	0	0	0	0	0	0	0
P_4	0	0	0	0	0	0	0

$11 \le k < 29$: U_2

	T_1	T_2	T_3	T_4	T_5	T_6	T_7
P_1	0	0	0	1	0	0	0
P_2	0	1	0	0	0	0	0
P_3	0	0	0	0	0	0	0
P_4	0	0	1	0	0	0	0

$29 \le k < 53$: U_3

	T_1	T_2	T_3	T_4	T_5	T_6	T_7
P_1	0	0	0	0	0	0	0
P_2	0	0	0	0	0	0	1
P_3	0	0	0	0	1	0	0
P_4	0	0	0	0	0	1	0

(e)

	Processor	**Start Time**	**Finish Time**
$\mathbf{T_1}$	2	0	11
$\mathbf{T_2}$	2	11	21
$\mathbf{T_3}$	4	16	29
$\mathbf{T_4}$	1	13	29
$\mathbf{T_5}$	3	32	39
$\mathbf{T_6}$	4	29	40
$\mathbf{T_7}$	2	37	53

(f)

FIGURE 8. Solving a numerical example by HS. (a) DAG, (b) θ matrix, (c) height of tasks, (d) classification of tasks based on height value, (e) control vectors, (f) resulted scheduling.

```
1.  ▶ 1. Part:
2.  Determine ready tasks (nodes) in DAG.
3.  Sorting tasks in ready task set based on upward rank
    in descending order.
4.  Select G = min(M, number of ready tasks) tasks
    and make a group with them.
5.  ▶ 2. Part:
6.  for each group g do
7.          for each task t do
8.                  Find processor r that finish time
                    of t on it is lower than other processors.
9.                  Schedule t on r.
```

FIGURE 9. RT algorithm

TABLE 2. Characteristics of random experiments

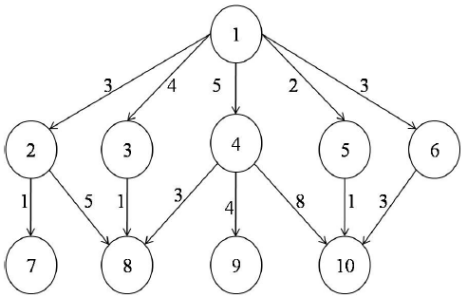
Experiment No	# of Processors	# of tasks	CCR	DAG Sparsity
1	8	{10, 20, 30, 40, 50, 100, 150, 200, 250}	1	0.5
2	8	50	1	{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9}
3	{4, 5, 6, 7, 8, 9, 10}	50	{0.01, 0.1, 1, 10}	0.5

the first part of HEFT algorithm is $O(T \log T)$ for sorting tasks and time complexity of the second part is $O(T.M)$, because of lines 4 and 5 in Figure 11, where T is number of tasks and M is number of processors. Generally, time complexity of HEFT algorithm is $O(T.M)$.

7. **Experimental Results.** In this section, we describe the performance of the proposed HS and RT control algorithms on several random examples including task graphs (DAGs)generated using the P-Method [36]. In this method, DAG is generated based on the probabilistic construction of an adjacency matrix of a DAG. Element a_{ij} of the matrix is defined as 1 if there is a precedence relation from t_i to t_j; otherwise, a_{ij} is zero. The adjacency matrix of acyclic graph is an upper triangular matrix, i.e., all elements on diagonal and lower triangles are zero. Each upper triangular element is determined based on Bernoulli process with parameter p, which represents the probability of success. For each element, when the Bernoulli trial is a success, then the element is assigned a value of one; for a failure the element is given a value of zero. The parameter p can be considered to be the Sparsity of the task graph. With this method, a probability parameter of $p = 1$ creates a totally sequential task graph, and $p = 0$ creates an inherently parallel one. Values of p that lie in between these two extremes generally produce DAGs that possess intermediate structures [36].

To evaluate our methods, randomly generated DAGs with characteristics shown in Table 2, are used. Communication to Computation Ratio (CCR) is a measure that reflects the general scheduling behavior with respect to communication [4]. DAG Sparsity is Bernoulli parameter that is a measure of precedence relations between tasks.

In these experiments, the number of heterogeneous processors is 10 and its topology is fully connected. The computation cost of tasks and communication cost between tasks



(a)

	P_1	P_2	P_3	P_4
T_1	18	11	19	17
T_2	18	10	15	17
T_3	17	11	22	13
T_4	16	12	25	20
T_5	7	18	15	18
T_6	17	18	15	11
T_7	17	16	14	21
T_8	12	12	17	20
T_9	10	12	16	10
T_{10}	12	14	16	13

(b)

$11 \le k < 32 : U_2$

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
P_1	0	0	1	0	0	0	0	0	0	0
P_2	0	0	0	1	0	1	0	0	0	0
P_3	0	1	0	0	0	0	0	0	0	0
P_4	0	0	0	0	0	1	0	0	0	0

$0 \le k < 11 : U_1$

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
P_1	0	0	0	0	0	0	0	0	0	0
P_2	1	0	0	0	0	0	0	0	0	0
P_3	0	0	0	0	0	0	0	0	0	0
P_4	0	0	0	0	0	0	0	0	0	0

$27 \le k < 46 : U_3$

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
P_1	0	0	0	0	1	0	0	0	0	0
P_2	0	0	0	0	0	0	0	1	0	0
P_3	0	0	0	0	0	0	1	0	0	0
P_4	0	0	0	0	0	0	0	0	1	0

(c)

Ready Tasks	Selected Tasks
{1}	{1}
{4,2,6,3,5}	{4,2,6,3}
{5,7,8,9}	{5,7,8,9}
{10}	{10}

(d)

Task	Upward rank	Task	Upward rank
T_1	61.25	T_6	32
T_2	35.25	T_7	17
T_3	32	T_8	15.25
T_4	40	T_9	12
T_5	29.25	T_{10}	13.75

(e)

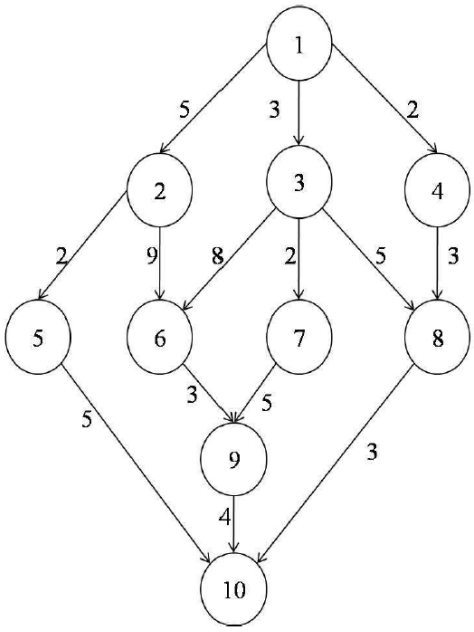
	Processor	**Start Time**	**Finish Time**
T_1	2	0	11
T_2	3	14	29
T_3	1	15	32
T_4	2	11	23
T_5	1	32	39
T_6	4	14	25
T_7	3	29	43
T_8	2	34	46
T_9	4	27	37
T_{10}	1	46	58

(f)

FIGURE 10. Solving a numerical example by RT. (a) DAG, (b) θ matrix, (c) upward rank, (d) grouping of tasks, (e) control vectors and (f) resulting schedule.

1: ► 1. Part:
2: Sort nodes $n_i \in V$ into list L, according to upward rank in descending order.
3: ► 2. Part:
4: **for** each $n \in L$ **do**
5: Find processor $r_j \in R$ that finish time of n on it is lower than other processors.
6: Schedule n on r_j.

FIGURE 11. HEFT algorithm



(a)

	P_1	P_2	P_3	P_4
T_1	100	104	102	101
T_2	102	102	103	101
T_3	105	107	101	100
T_4	102	102	105	106
T_5	103	103	101	101
T_6	103	100	100	101
T_7	101	101	100	102
T_8	103	103	109	102
T_9	108	101	102	100
T_{10}	100	100	101	101

(b)

FIGURE 12. Numerical example. (a) DAG, (b) θ matrix.

in task graph is chosen randomly based on normal distribution similar to [1,8,9]. The parameters of normal distribution have been reported in Table 5. We iterate each experiment 50 times and report mean of results in each experiment. Experimental results are compared with Heterogeneous Earliest Finish Time (HEFT) [3].

As shown in Table 3, the result of proposed controllers are very close to HEFT and in experiments with less than 50 tasks the result of proposed methods are better than HEFT.

In some DAGs more than one source node (node without any predecessor) may exists. For example in the following DAG, nodes 1 and 8 are source nodes because they do not have any predecessors. Sometimes, some of the source nodes are close to sink nodes (nodes without any successor), such as node 8, and therefore have low upward rank. With these DAGs, HEFT operates as follows: it calculates upward rank of all nodes and then sorts nodes in descending order of upward rank. As a result, nodes such as node 8 are located at intermediate levels of the sorted list, resulting in delayed scheduling of them. Although such nodes have low upward rank, they have a lot of impact on scheduling. In HS, since

TABLE 3. Experiment 1 – 50 run average and standard deviation of makespan – Comparing HS, RT and HEFT on DAGs with different number of tasks (CCR = 1, Sparsity = 0.5, Processors = 8)

# Tasks	Parameter	HEFT	HS	RT
10	**Average**	84.00	83.00	84.00
	STD	12.50	10.60	10.90
20	**Average**	171.00	169.00	170.00
	STD	25.52	21.53	22.15
30	**Average**	261.00	258.00	259.00
	STD	38.85	32.87	33.78
40	**Average**	341.00	334.00	336.00
	STD	50.86	42.53	43.88
50	**Average**	437.00	431.00	432.00
	STD	65.19	54.85	56.51
100	**Average**	849.00	872.00	850.00
	STD	126.65	111.03	111.09
150	**Average**	1260.00	1295.00	1257.00
	STD	188.02	164.97	164.34
200	**Average**	1689.00	1734.00	1690.00
	STD	252.04	220.91	221.04
250	**Average**	2118.00	2166.00	2109.00
	STD	316.10	275.85	275.76
300	**Average**	2536.00	2598.00	2533.00
	STD	378.42	330.79	331.21

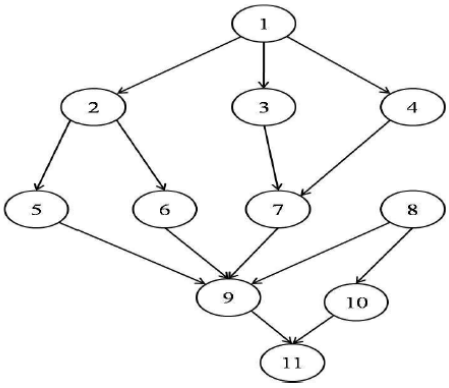


FIGURE 13. A sample DAG

we act based on level, nodes 1 and 8 have same level and therefore node 8 is scheduled sooner than in HEFT. In RT, since we consider all of ready tasks at each step, node 8 is scheduled sooner than in HEFT.

In DAGs with higher number of tasks, the impact of delayed scheduling nodes such as in node 8 is less; therefore, in experiments with a higher number of tasks, although makespan of HEFT is lower than HS and RT, they are close together.

The next experiment is about DAG Sparsity. These experiments are performed for 50 tasks and 10 processors that are fully connected. In addition, each experiment has been iterated 50 times and average value has been reported. Table 5 shows parameters of normal distribution that is used to randomly generate computation and communication

TABLE 4. Experiment 2 – 50 run average and standard deviation of makespan – Compring HS, RT and HEFT on DAGs with different sparsity levels (CCR = 1, Number of tasks = 50, Processors = 8)

Sparsity	Parameter	HEFT	HS	RT
0.1	**Average**	135.88484	150.61433	143.12059
	STD	29.76	33.56	30.61
0.2	**Average**	225.28988	241.29017	228.50856
	STD	49.24	53.76	48.86
0.3	**Average**	289.74098	303.77117	293.61047
	STD	63.32	67.74	62.79
0.4	**Average**	364.52774	381.48965	366.90289
	STD	79.61	85.12	78.54
0.5	**Average**	430.41431	441.56268	428.79773
	STD	93.91	98.59	91.77
0.6	**Average**	482.59026	486.58198	480.88965
	STD	105.21	108.56	102.94
0.7	**Average**	522.23542	529.24895	522.17147
	STD	113.89	118.10	111.85
0.8	**Average**	557.07088	559.5233	557.89501
	STD	121.42	124.90	119.52
0.9	**Average**	574.5813	572.55392	573.84068
	STD	125.27	127.81	122.84

TABLE 5. Experiment 3 – Setting of parameters based on normal distribution

	Computation time		Communication time	
	Mean	**Variance**	**Mean**	**Variance**
Experiments with varying number of tasks	10	4	10	6
Experiments with varying Sparsity value	10	4	10	6
Experiments with CCR = 0.01	100	4	10	6
Experiments with CCR = 0.1	100	4	10	6
Experiments with CCR = 1	20	4	20	6
Experiments with CCR = 10	10	4	100	6

times in DAG. Table 4 shows the result of the experiment. As shown in this table, makespan of HS controllers and HEFT are close together, in all of Sparsity values and sometimes the RT is better than HEFT.

In addition, more experiments are performed as shown in Table 6 Each cell shows the average result of 50 separate execution of experiment with randomly generated DAGs with specified parameters. Computation time and communication time of tasks in experiments are generated randomly based on normal distribution. Table 5 shows parameters of normal distribution function that are used.

TABLE 6. Experiment 3 – 50 run average and standard deviation of makespan repeated for 50 independent runs (Number of Tasks = 50, Sparsity = 0.5)

# Proc	Method	CCR = 0.01		CCR = 0.1		CCR = 1		CCR = 10	
		Average	STD	Average	STD	Average	STD	Average	STD
4	**HEFT**	33613.811	340.31003	12122.97	474.2812	3221.97	103.7077	2523.06	212.9125
	HS	33037.74	358.59624	12221.74	482.745	3301.14	88.4415	5950.41	211.6781
	RT	33116.995	364.52961	12137.05	473.8932	3209.56	102.0932	5726.12	237.8184
5	**HEFT**	31690.596	432.93612	11989.52	551.376	3182.87	133.0228	2453.74	172.7356
	HS	33664.481	417.12938	12071.97	544.0969	3260.24	119.5259	6000.65	280.7522
	RT	32629.978	396.92558	12002.61	548.7594	3167.54	126.4412	5732.21	270.2922
6	**HEFT**	31399.513	373.87099	12053.3	515.0406	3192.24	102.8087	2460.88	141.3361
	HS	33777.223	396.62514	12145.2	510.445	3264.83	104.6349	5970.45	240.0112
	RT	33301.225	389.55232	12069.51	507.9879	3168.5	108.9867	5775.32	286.5996
7	**HEFT**	31166.632	421.9845	11983.04	581.9847	3182.45	107.3619	2566.24	232.5873
	HS	33469.732	484.78769	12069.28	582.9445	3260.53	106.2959	5983.41	273.3193
	RT	32477.883	456.48202	12002.1	583.4138	3174.19	103.52	5745.82	257.5152
8	**HEFT**	31528.373	409.53076	11980.84	553.7696	3197.57	121.3184	2479.69	167.3255
	HS	34056.899	397.22778	12067.02	545.0182	3279.69	118.1766	6005.92	239.7304
	RT	33813.452	445.03422	11991.42	554.3868	3180.94	115.7275	5743.37	195.4504
9	**HEFT**	34149.529	410.56562	12020.58	541.7361	3188.88	126.1868	2434.67	137.639
	HS	33249.595	402.71553	12113.87	537.4327	3270.41	117.8027	5966.56	230.7038
	RT	34092.924	441.36717	12033.55	540.4003	3182.31	127.5147	5713.61	208.4604
10	**HEFT**	32426.103	411.47325	12058.53	575.948	3183.51	114.2389	2491.59	190.4354
	HS	32591.922	414.53706	12155.29	570.4473	3252.69	113.9986	5976.21	244.231
	RT	32976.138	453.14563	12073.84	577.7726	3171.89	113.163	5731.46	200.7092

In these experiments, number of tasks T and Sparsity parameter of DAG are fixed to 200 and 0.5, respectively. Number of fully connected processors and CCR parameter of DAG are changed between 4 to 10 and $\{0.01, 0.1, 1, 10\}$ respectively.

As shown in Table 6, when CCR = 0.1 or CCR = 1, the result of HS and RT are generally comparable with those of HEFT. The results of RT are consistently better than HS due to its global view. In the grouping stage of HS, we only consider height of tasks in DAG that has a local view to DAG and restrict selection of tasks from other levels; while in RT we have a global view and do not consider level of tasks, but consider all of ready tasks. However, when CCR = 10, HEFT exceeds the performance of HS and RT. It should be noted that real world DAG applications with CCR = 10 are less common, because DAGs with CCR = 10 are those where total communication is 10 times that of computation.

8. **Conclusion and Future Works.** Task scheduling and its varieties have a considerable number of significant applications in the functioning of modern society. Hence, many methods have been developed to date, but they are mostly based on heuristics. There are relatively few aspects of TS that has been approached from a theoretical perspective. Here, we propose to address the problem of static task scheduling (TS) by proposing a new general paradigm based on system engineering. This new modeling paradigm is promising due to its extensive theoretical developments. It is different from its earlier competitors, particularly when compared with those with a systems engineering perspective, in terms of its extended generality, general applicability, and translation to design of a controller through solving an optimization problem based on switching systems. We demonstrate how TS can be mapped via nonlinear state space and, through theoretical

analysis, show a test of stability for the resulting system. A transformation is subsequently devised to convert this state space model to linear switching state space with nonlinear constraints. Two systematic methods are then presented to determine control vectors based on this model. Theoretical analysis confirms the asymptotic stability of the closed loop TS model. In TS terms, it is shown that all tasks can be completed with the proposed controller given sufficient time. The proposed HS and RT controllers are then compared against the HEFT scheme on several randomly generated experiments. Results indicate comparative performance particularly when CCR is 0.01, 0.1, and 1.

This paper considers static TS environments as has also been the concern of great many scholars earlier; however most real world problems are defined in dynamic environments (i.e., changes in time) and face preemption. We believe that the proposed framework simply extends to such non pre-emptive and dynamic task scheduling problems. Hence, the model can be exploited for special cases such as concurrent task execution in single-core processors. In the future, we aim to consider these aspects as well as further theoretical analysis by application of system theory perspective to TS.

REFERENCES

[1] A. J. Page and T. J. Naughton, Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing, *Proc. of the 19th IEEE International Parallel and Distributed Processing Symposium*, USA, pp.152-159, 2005.

[2] Y. K. Kwok and I. Ahmad, Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors, *IEEE Trans. Parallel and Distributed Systems*, vol.7, no.5, pp.506-521, 1996.

[3] H. Topcuoglu, S. Hariri and M. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, *IEEE Trans. Parallel and Distributed Systems*, vol.13, no.3, pp.260-274, 2002.

[4] O. Sinnen, *Task Scheduling for Parallel Systems*, John Wiley & Sons-Interscience, 2007.

[5] O. Sinnen, L. A. Sousa and F. E. Sandnes, Toward a realistic task scheduling model, *IEEE Trans. Parallel and Distributed Systems*, vol.17, no.3, pp.263-275, 2006.

[6] S.-C. Cheng et al., Dynamic hard-real-time scheduling using genetic algorithm for multiprocessor task with resource and timing constraints, *Expert Systems with Applications*, vol.36, no.1, pp.852-860, 2009.

[7] K. Shin et al., Task scheduling algorithm using minimized duplications in homogeneous systems, *Parallel and Distributed Computing*, vol.68, no.3, pp.1146-1156, 2008.

[8] M. Yoo, Real-time task scheduling by multiobjective genetic algorithm, *Systems and Software*, vol.82, no.4, pp.610-628, 2009.

[9] M. Yoo and M. Gen, Scheduling algorithm for real-time tasks using multiobjective hybrid genetic algorithm in heterogeneous multiprocessors system, *Computers & Operations Research*, vol.34, no.2, pp.3084-3098, 2007.

[10] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NPCompleteness*, 1st Edition, W. H. Freeman, 1979.

[11] J. Ullman, NP-complete scheduling problems, *Computer System Science*, vol.10, no.2, pp.384-393, 1975.

[12] T. C. Hu, Parallel sequencing and assembly line problems, *Operations Research*, vol.9, no.6, pp.841-848, 1961.

[13] E. G. Coffman and R. L. Graham, Optimal scheduling for two-processor systems, *Acta Informatica*, vol.1, no.3, pp.200-213, 1972.

[14] P. C. Fishburn, *Interval Orders and Interval Graphs*, 1st Edition, John Wiley and Sons Inc., 1985.

[15] C. H. Papadimitriou and M. Yannakakis, Scheduling interval-ordered tasks, *SIAM Jounral of Computers*, vol.8, no.2, pp.405-409, 1979.

[16] X. Kong, J. Sun and W. Xu, Permutation-based particle swarm algorithm for tasks scheduling in heterogeneous systems with communication delays, *Computational Intelligence Research*, vol.4, no.1, pp.61-70, 2008.

[17] K. Shin et al., Task scheduling algorithm using minimized duplications in homogeneous systems, *Parallel and Distributed Computing*, vol.68, pp.1146-1156, 2008.

[18] I.-H. Kuo et al., An efficient flow-shop scheduling algorithm based on a hybrid particle swarm optimization model, *Expert Systems with Applications*, 2008.
[19] C. Chiu, M. J.-J. Wu, Y.-T. Tsai N.-H. Chiu, M. S.-H. Ho and H.-J. Shyu, Constrain-based particle swarm optimization (CBPSO) for call center scheduling, *International Journal of Innovative Computing, Information and Control*, vol.5, no.12(A), pp.4541-4549, 2009.
[20] H.-S. Choi and D.-H. Lee, Scheduling algorithms to minimize the number of tardy jobs in twostage hybrid flow shops, *Computers & Industrial Engineering*, 2008.
[21] Y. Li, Y. Yang, L. Zhou and R. Zhu, Observations on using problem-specific genetic algorithm for multiprocessor real-time task scheduling, *International Journal of Innovative Computing, Information and Control*, vol.5, no.9, pp.2531-2540, 2009.
[22] R. Hwang, M. Gen and H. Katayam, A comparison of multiprocessor task scheduling algorithms with communication costs, *Computers & Operations Research*, vol.35, pp.976-993, 2008.
[23] S. Garg, P. Konugurthi and R. Buyya, A linear programming driven genetic algorithm for meta-scheduling on utility grids, *Proc. of the 16th International Conference on Advanced Computing and Communications*, Chennai, India, pp.19-26, 2008.
[24] M. A. Moges and D. Yu, Grid scheduling divisible loads from multiple sources via linear programming, *Proc. of the 16th IASTED International Conference on Parallel and Distributed Computing and Systems*, Cambridge, MA, USA, 2004.
[25] D. C. Marinescu, Application of control theory to task scheduling on a cloud, *Cloud Computing and Computer Clouds*, University of Central Florida, Orlando, FL, USA, 2012.
[26] C. Lu et al., Feedback control real-time scheduling: Framework, modeling, and algorithms, *Special Issue of Real-Time Systems Journal on Control-Theoretic Approaches to Real-Time Computing*, vol.23, no.1/2, pp.85-126, 2002.
[27] J. Stankovic et al., Feedback control scheduling in distributed real-time systems, *Proc.of the 22nd IEEE Real-Time Systems Symposium*, Charlottesville, VA, USA, 2001.
[28] A. D. Marbini and L. Sacks, Considering control theory in resource management scenarios, *Proc. of London Communications Symposium*, London, England, 2002.
[29] O. H. Roux and A.-M. Déplanche, A T-time petri net extension for real-time task scheduling modeling, *European Journal of Automation*, vol.36, no.7, 2002.
[30] A. Rowhanimanesh, A. Karimpour and N. Pariz, Optimal path planning for controllability of switched linear systems using multi-level constrained GA, *Advances in Intelligent and Soft Computing*, vol.58, pp.399-408, 2009.
[31] Z. Sun, *Switched Linear Systems: Control and Design (Communications and Control Engineering) [Hardcover]*, Springer, 2005.
[32] K. Ogata, *Discrete-Time Control Systems*, 2nd Edition, Prentice Hall, 1995.
[33] C.-T. Chen, *Linear System Theory and Design*, 2nd Edition, Oxford University Press, USA, 1995.
[34] H. Ozbay, *Introduction to Feedback Control Theory*.
[35] G. Q. Liu, K. L. Poh and M. Xie, Iterative list scheduling for heterogeneous computing, *Journal of Parallel and Distributed Computing*, vol.65, no.5, pp.654-665, 2005.
[36] S. Al-Sharaeh and B. E. Wells, A comparison of heuristics for list schedules using the boxmethod and P-method for random digraph generation, *Proc. of the 28th Southeastern Symposium on System Theory*, 1996.