Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Dynamic task scheduling modeling in unstructured heterogeneous multiprocessor systems

Hamid TABATABAEE†1, Mohammad Reza AKBARZADEH-T[2,3], Naser PARIZ[3]
([1]*Department of Computer Engineering, Islamic Azad University, Quchan Branch, Quchan, Iran*)
([2]*Center of Excellence on Soft Computing and Intelligent Information Processing, Ferdowsi University of Mashhad, Mashhad, Iran*)
([3]*Department of Electrical Engineering, Ferdowsi University of Mashhad, Mashhad, Iran*)
†E-mail: hamid.tabatabaee@Iauq.ac.ir

Received Aug. 1, 2013; Revision accepted Jan. 17, 2014; Crosschecked Apr. 11, 2014

Abstract: An algorithm is proposed for scheduling dependent tasks in time-varying heterogeneous multiprocessor systems, in which computational power and links between processors are allowed to change over time. Link contention is considered in the multiprocessor scheduling problem. A linear switching-state space-modeling paradigm is introduced to enable theoretical analysis from a system engineering perspective. Theoretical analysis of this model shows its robustness against changes in processing power and link failure. The proposed algorithm uses a fuzzy decision-making procedure to handle changes in the multiprocessor system. The efficiency of the proposed algorithm is illustrated by several random experiments and comparison against a recent benchmark approach. The results show up to 18% average improvement in makespan, especially for larger scale systems.

Key words: Dynamic task scheduling, Fuzzy logic, Genetic algorithms, Unstructured environment, Linear switching state space
doi:10.1631/jzus.C1300204 **Document code:** A **CLC number:** TP301.6

1 Introduction

Many real world applications such as management of projects, human resources, or financial resources as well as distributed computing systems are examples of task scheduling problems. The multiprocessor systems must be both fast and efficient in utilizing the available resources. Each application also poses many challenging issues in its own right. For instance, with continual advances in computer hardware, there are always applications in which more processing power is required than what is available or the needed processor time is not known a priori, and communication links may be in or out. Furthermore, inappropriate scheduling of tasks may lead to failure in exploiting the true potential of a distributed system and offset the gains from parallelization due to excessive communication overhead or under-utilization of resources (Page and Naughton, 2005). In this paper, a novel method is proposed to schedule tasks to a set of heterogeneous processors in a time-varying environment based on a linear switching state space (LS^3) system which considers several of the prominent and frequent time-varying changes in a distributed system.

Task scheduling can be simply defined as the process of allocating dependent tasks to resources so as to minimize the total execution time. This problem is generally NP-complete (Kwok and Ahmad, 1996) and is not easily solvable by traditional methods. Many current approaches have solved it by heuristic and random search methods, but they have adopted constraints that often lead to inefficiency/inapplicability of their methods in the real world. For instance, many researchers have limited the problem to static environments, assuming that the underlying network is fully connected, and/or that there is no link contention in the network. Hence, it is highly desirable to make an accurate model for the task scheduling problem that accounts for many of the above real world limitations. Such a model must be able to appropriately handle real constraints and system uncertainties before it can be used in a real world environment with arbitrary conditions.

Task scheduling is nonlinear due to the many constraints in the environment, tasks and resources that must be considered. In our previous work (Tabatabaee-Yazdi and Akbarzadeh-T, 2013), a static form of this problem was defined in a nonlinear state space and then transformed to a dynamic LS^3 with nonlinear constraints. We focused on static scheduling in which the environment does not change over time and the computation cost of each task on each processor is fixed and known a priori. In this paper, we extend this model to time-varying environments. It is assumed that the environmental parameters such as the computational power of the processor and the link speeds will change over time. Such a time-varying scheduler could be potentially applied to real world environments and help make good decisions based on real-time information from the system. Since the proposed method takes into account these temporal variations in the environment, the scheduling algorithm is revised periodically, different from Crăciun *et al.* (2010) which is based on scheduling events. Note that the word 'dynamic' has two meanings in computer/AI and system engineering. Hence, 'time-varying' is used here for dynamic environments to distinguish its difference in meaning in dynamic systems, i.e., systems that have memory. The major focus of this paper is to show the ability of our previous work (Tabatabaee-Yazdi and Akbarzadeh-T, 2013) to operate in dynamic environments. We compare the LS^3 scheduler with an evolutionary task scheduling approach, i.e., the genetic algorithm (GA) scheduler, developed for static and time-varying environments, proposed by Crăciun *et al.* (2010).

The major contributions of this paper can be distinguished as follows:

1. The problem of task scheduling is modeled in the switching state space.

2. A simple proof is provided for a major characteristic of the proposed method: robustness. Since one of the important factors in control systems is robustness, robust control methods are designed to guarantee system performance/stability as long as the uncertain parameters or disturbances are within a given range. In this paper, the robustness characteristic of the proposed model is examined, and it is shown that the proposed model is applicable in uncertain environments.

3. The proposed method takes into account the variation in the computational power of the processors over time.

2 System model

In a time-varying task scheduling problem, the target problem to be scheduled is represented by a task graph.

Definition 1 (Task graph) A task graph is a directed acyclic graph (DAG) $G=(V, E, \lambda, c)$ representing a program P according to the graph model, where V is the set of tasks in DAG, E is the set of edges in DAG, and λ is the instruction.

Definition 2 (Task) A task is defined as the most basic element where further parallelism is neither necessary nor possible. Hence, all of the instructions for a task execute sequentially. Before the execution of a node (task), all predecessors must be completed and all of its inputs must be received. After the execution of a task, its output is ready to be transferred to its successors immediately (Sinnen *et al.*, 2006).

Most scheduling algorithms employ a strongly idealized model of the multiprocessor system (Yoo and Gen, 2007; Cheng *et al.*, 2009). This model, which we call a classic model, consists of a finite set of processors that are connected by a fixed topology such as a network graph. Local communication in this model has zero cost, and communication can be performed concurrently by an independent communication subsystem. Based on this model, the earliest start time of the node depends only on the completion time of the node's predecessors and the communication time between them.

The classic scheduling model does not consider any kind of contention for communication resources. To make task scheduling contention-aware, and thereby more realistic, the communication network is modeled by a graph, where processors are represented by vertices and edges reflect the communication links. The awareness of contention is achieved by edge scheduling, i.e., scheduling of the edges of the

Table 1 Notations used in this paper

Parameter	Definition	Parameter	Definition
$\boldsymbol{X}$	Vector of state variables	ΔT	Time step
$\boldsymbol{X}[k]$	Vector of state variables in the kth step	$\boldsymbol{U}$	Control vector
Y	System output	V	Set of tasks in DAG
E	Set of edges in the directed acyclic graph (DAG)	n	A node in DAG
$W(n)$	Computation cost of node n	e_{ij}	Edge from node i to node j in DAG
$C(e_{ij}^{k})$	Communication cost of edge e_{ij} in the kth step	Succ(n)	Successor of node n
Pred(n)	Predecessor of node n	P	Set of processors
$t_{\mathrm{s}}(n, p)$	Start time of node n on resource p	$W(n, p)$	Execution time of node n on resource p
$t_{\mathrm{f}}(n, p)$	Completion time of node n on processor p	$t_{\mathrm{s}}^{Q}(n_j)$	Earliest start time of n_j
λ_i	Number of instructions in task i	λ_i^j	The jth instruction of task i (λ_i)
ET_{kj}	Execution time of task j in the kth step	μ	Communication time between tasks
Ψ	Makespan	$t_{\mathrm{f}}(n)$	Completion time of node n
Proc(t)	Processor to which task t is allocated	$\xi=[\xi_{1j}]$	Speed of processor j in cps (clocks per second)

DAG on the links of the network graph, which is similar to scheduling of the edges E on the communication link L. Corresponding to the scheduling of the nodes, $t_{\mathrm{s}}(e, l)$ and $t_{\mathrm{f}}(e, l)$ denote the start and completion time of edge $e \in E$ on link $l \in L$, respectively (Table 1).

3 Related work

Distributed systems present a challenging paradigm for task scheduling and much literature has been devoted to its various aspects. Task scheduling techniques can be divided into two main classes: static and time-varying. In static scheduling, information regarding task execution time and resources (nodes) is assumed to be known beforehand. Dynamic scheduling is based on task assignment during execution, taking into account overloaded and underloaded nodes, with the assumption that, if the load among all nodes is balanced, the overall execution time of the application will be minimized. Hence, according to either centralized or distributed information, a decision has to be made whether or not a task should be transferred and to which node it should be transferred.

There is a vast literature on static task scheduling. Since the focus of the present work is time-varying scheduling, only a few recent studies are highlighted here. The traditional methods that are used to solve this problem in the static mode are heuristic algorithms (list-based scheduling) and random search algorithms (such as GA, simulated annealing, and particle swarm optimization). Shahul and Sinnen (2010) used an A* algorithm to optimally schedule tasks on homogeneous processors. A* is a best-first search algorithm and is guaranteed to find optimal solutions with an admissible and consistent cost function. They proposed a cost function based on three components: computation bottom level, idle time, and data-ready time. In addition, they used a pruning technique to eliminate unpromising sub-trees in the search space. Yoo (2009) minimized the total time and the total number of processors used in scheduling of soft real-time DAGs on homogeneous multiprocessor systems by a multi-objective GA. Kong *et al.* (2008) incorporated particle swarm optimization (PSO) with list scheduling and developed an alternative PSO algorithm for multiprocessor task scheduling. Shin *et al.* (2008) proposed a novel duplication-based algorithm which, unlike most duplication-based heuristics that try to duplicate all possible ancestor nodes of a given join node, duplicates some nodes based on some parameters such as start time and the earliest free slot, without redundant duplications. Their method has two steps. In the first step, a task-priority list is made; in the second step, the best possible processor, i.e., the one that allows the earliest start time, is selected to execute the task with the highest priority.

The longest dynamic critical path (LDCP) algorithm (Daoud and Kharma, 2008) is a list scheduling method that schedules DAGs in heterogeneous multiprocessor systems. In this algorithm, the longest paths from all tasks to the final task are computed, and DAGPs (a DAG that corresponds to each processor) generated. Based on an upward rank attribute, an LDCP is selected and a priority assigned to each task. In the next stage, the task with the highest priority is assigned to a processor that minimizes its completion execution time using the insertion based scheduling policy.

In contrast to static task scheduling, there are relatively few studies on time-varying task scheduling. Prodan and Fahringer (2005) is one of the first to address this problem by using special heuristics at certain scheduling events and a reallocation technique to schedule directed graph based workflows on a grid in which computational and network resources can be dynamically changing. Page and Naughton (2005) presented a scheduling strategy which dynamically schedules independent tasks on heterogeneous processors in a distributed system. They considered dynamic behavior in the environment from two perspectives. First, they allowed tasks to arrive for continuous processing; second, they considered the variability of distributed systems over time. Variability of the distributed systems means a changing communication network and availability of resources over time. They assumed that resources are not dedicated and may execute other tasks. Page *et al.* (2008) introduced a scheduling mechanism considering heterogeneous resources in terms of processing speed, memory, and interconnection properties. In their paradigm, the computational requirements of the tasks are unknown a priori, tasks arrive dynamically for scheduling, and the properties and availability of the resources can vary randomly over time, with unknown statistics. This approach estimates the system resources and task computational requirements dynamically at run-time and adapts the state of the system accordingly.

Also, in Abraham *et al.* (2008), several nature-inspired meta-heuristics were introduced to schedule tasks on computational grids. The main purpose of Page *et al.* (2010) is to use GA to dynamically generate an optimal schedule in order to complete tasks in a minimum period of time while utilizing resources efficiently. Later, in Sivanandam and Visalakshi (2009) and Page *et al.* (2010), GA, simulated annealing (SA), ant colony optimization, and PSO algorithms were combined in a multi-objective manner to inspect the effectiveness of these combinations separately. In contrast to these two studies, in Page *et al.* (2010), two GAs augmented by some heuristics were developed. The heuristics were used to improve performance. In the first GA, two fitness functions were used consecutively, and the second GA tried to duplicate the tasks to reduce the communication overhead. Also, time-varying arrival of the tasks was allowed. In some of these studies such as Crăciun *et al.* (2010) and Page *et al.* (2010), a non-random initial solution was considered. In Page *et al.* (2010) the scheduler uses a heuristic-based approach to find efficient schedules quickly while reducing the probability of resources becoming idle during the schedule-generating phase. This provides GA with reasonable starting solutions, compared to starting with a completely random initial population. Furthermore, Crăciun *et al.* (2010) developed a scheduling mechanism which starts with an initial scheduling pattern produced by GA and then takes the changes in the computational resources over time into account after each scheduling event, in order to adapt the scheduling pattern to the environment. The scheduler also builds an archive of the best schedules ever produced for use in the future. Long *et al.* (2011) proposed an agent-based distributed simulation platform and a distributed optimized scheduling method with partial information. The scheduler also takes a load-balancing factor into account. These similar approaches lack a systematic theoretical analysis support since most of them do not use a systematic model. Instead, they exploit heuristic- or metaheuristic-based methods. Our proposed method, which is discussed in the following, is based on a systematic model.

4 Extended model of task scheduling in the state space

Here we extend our earlier model on static task scheduling (Tabatabaee-Yazdi and Akbarzadeh-T, 2013) to allow for time-varying changes in the environment. Eq. (1) represents the system model in which $\boldsymbol{X}$ is the system state vector which changes over time. For details, please refer to Tabatabaee-Yazdi and Akbarzadeh-T (2013).

$$\boldsymbol{X}[k+1]=\max\left[\boldsymbol{0},\boldsymbol{X}[k]-\Delta T\cdot\mathrm{diag}\left\{\left(\sum_{i}\mathrm{CL}(\lambda_j^i)\frac{1}{\xi}\right)^{-1}\right\}\boldsymbol{U}\right],\tag{1}$$

where $\mathrm{diag}\{\boldsymbol{A}\}$ is the main diagonal of matrix $\boldsymbol{A}$, and $\mathrm{CL}(\lambda_j^i)$ stands for the number of clocks that a processor should use for instruction λ_j^i to be completed.

With Eq. (1), only the clock requirements of instructions in task i and speed of processors are required. In comparison with our earlier model on static task scheduling (Tabatabaee-Yazdi and Akbarzadeh-T, 2013), the required time for completing any task on any processor is not known a priori.

As pointed out in Section 1, we are inspecting the robustness of the proposed method. In the next theorem and sub-theorems, the robustness characteristic of the proposed model is examined, showing that the proposed model is applicable in an uncertain environment.

Theorem 1 (Boundedness) For all D belonging to the set of DAGs with finite height H, $\psi(D)<H(m+Z)<\infty$. In other words, makespan ψ is bounded for all finite DAGs with arbitrary weights of nodes and edges and is a linear multiple of task height H considering the following two conditions:

1. The execution time of each task on any arbitrary processor is bounded.

2. The communication time between any two arbitrary processors is bounded.

Theorem 2 (Robustness in presence of uncertainty on the execution time of tasks) $\forall D\in$DAGs, $\forall T\in$Tasks(d), $\exists\Delta e\in\mathbb{R}$, RET($T$)<$\infty$, RET($T$)=ET($T$)+$\Delta E\Rightarrow\psi<\beta<\infty$, where RET($T$) stands for the run-time execution time of task T, ET(T) is the execution time of task T, and Δe ($\Delta e\in\mathbb{R}$) is the change in execution time due to disturbance. This means that if the execution time of each task change is bounded, makespan change is also bounded.

Theorem 3 (Robustness in presence of uncertainty on the communication time between tasks) $\forall D\in$DAGs, $\forall t_i, t_j\in D$, $\exists\Delta e\in\mathbb{R}$, $c(t_i, t_j)\neq 0$, $\mathrm{Rc}(t_i, t_j)<\infty$, $\mathrm{Rc}(t_i, t_j)=c(t_i, t_j)+\Delta E\Rightarrow\psi(D)<\beta<\infty$, where $c(t_i, t_j)$ represents the communication time between tasks t_i and t_j, $\mathrm{Rc}(t_i, t_j)$ represents the communication time between tasks t_i and t_j, and Δe ($\Delta e\in\mathbb{R}$) is the change in communication time due to disturbance.

In the following, we consider a time-varying multiprocessor system where processors are shared. It means that the utilization of a processor during the execution of a task may change with time. In what follows, we explain our method by using an extended model of task-scheduling in the state space.

5 Proposed method

In this study, we schedule a DAG on an arbitrary time-varying heterogeneous multiprocessor system. In dynamic multiprocessor systems, since processors are shared, utilization of the processors may change. Also, in the assumed system, communication links may be on or off; i.e., a communication link may be unavailable – down – during a period of time and after that it may become available. Our method has two stages: static scheduling and time-varying scheduling. In the first stage, a DAG is scheduled on a nominal multiprocessor system with GA and an initial scheduling solution is generated. The multiprocessor system is at its nominal value when there are no time variations and all processors are available (i.e., no other task runs on them). In the second stage, we operate on the dynamic constraints of the multiprocessor system and, based on the initial scheduling solution, as generated in the first stage, the DAG is scheduled again on the processors. In our model, the utilization of the processors may be reduced or increased. Any changes in the utilization of the processors are monitored and, based on the status of the executing tasks and the utilization of processors, a decision is made about reallocation of tasks from one processor to another. In this section, we first introduce our model and then discuss the proposed scheduling method in static and time-varying environments.

5.1 Multiprocessor model

The assumed multiprocessor model follows task scheduling with a contention-aware model as introduced in Section 2. Its characteristics are defined as follows:

Definition 3 (Parallel system) A target parallel system consists of a set of heterogeneous processors P connected by the communication network TG=

(P, L) and a central scheduler, with the following properties:

1. Processors are shared.
2. Communicating links may be on or off.
3. Local communications have zero costs.
4. Communication is performed by a communication subsystem.
5. The central scheduler monitors the status of the parallel system and makes a decision about executing tasks on processors.

The tasks and edges in the assumed model are scheduled as explained in Section 2. The proposed parallel system may change with time. Changes or disturbances in the parallel system are divided into two categories: changes in utilization of the processors and changes in the presence and absence of links. Changes in utilization of the processors are caused by sharing and changes in the links are caused by link availability.

In the proposed parallel system, a central scheduler exists. The central scheduler has an initial scheduling solution based on changes in the multiprocessor system, and adjusts the initial scheduling solution. The central scheduler is able to preempt tasks from a processor and assign them to another processor, which is called reallocation. The central scheduler has a memory that stores information packets resulting from execution of the tasks. It subsequently sends a task completion report to the expecting processors in order to continue with DAC processing.

5.2 Scheduling method

As stated earlier, scheduling is done in two stages: static scheduling and time-varying scheduling. At the first stage, a GA is used to find the scheduling solution with the lowest makespan ψ. At the second stage, tasks are executed on a dynamic multiprocessor system and if needed, the initial scheduling solution is corrected by the central scheduler to accommodate the uncertainty in utilization of processors and existence or non-existence of links. These steps are schematically illustrated in Fig. 1. The major advantage of scheduling in two stages is that an optimum initial scheduling solution can be ensured and the second stage has no time delay in allocating tasks to processors. In the following, we discuss each stage in detail.

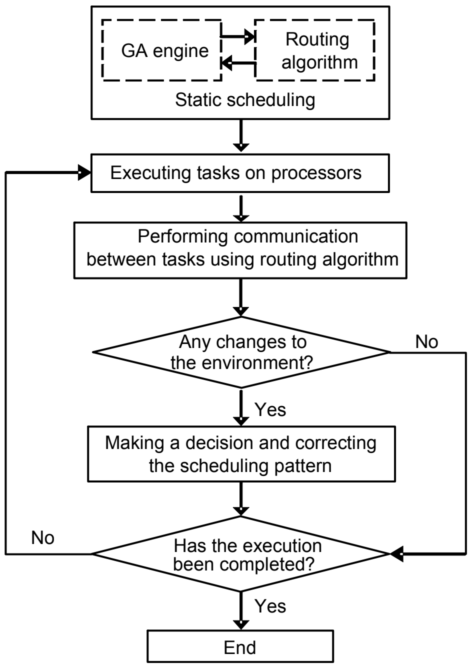

GA engine
Routing algorithm
Static scheduling
Executing tasks on processors
Performing communication between tasks using routing algorithm
Any changes to the environment?
No
Yes
Making a decision and correcting the scheduling pattern
No
Has the execution been completed?
Yes
End

Fig. 1 Schematic of the linear switching state space (LS^3) scheduling mechanism

5.2.1 Static stage

Encoding is an effective stage in GA. In the proposed method, each individual in the population represents a possible schedule. Each character is a mapping between a task and a processor. Each character contains the unique identification number of a task, with S being used to delimit different processor queues, where P_i is processor i. Thus, the number of characters is $N+|P|-1$, where N is the number of tasks in the batch and $|P|$ is the number of processors (Page and Naughton, 2005).

The fitness function of the task scheduling problem is to assign the tasks of a given application to the processors, in order to minimize its schedule length (makespan ψ). To compute the schedule length, precedence-constrained tasks (DAG's nodes) and messages (DAG's edges) are scheduled on the processors and links, respectively. While the start time of a node is constrained by the ready time of its incoming edges, the start time of an edge is restricted by the completion time of its preceding node. The scheduling of the communication between processors further differs from that of a node, in that a communication action might be scheduled on more than one link. A communication between two nodes, which are scheduled on two different but not adjacent processors, uses the communication route of intermediate links between processors. The edge representing this communication must be scheduled on each of the links involved. Algorithm 1 shows the decoding procedure of a chromosome.

Algorithm 1 Decoding a chromosome

```
1  Determine ($n$, proc($n$)), $n \in V$ pairs based on chromosomes
2  For each $n_i \in V$
3    taskready($n_i$)=0
4  End For
5  For each $n_i \in V$ based on upward rank in descending order
6    $m$=proc($n_i$)
7    start_time=taskready($n_i$)
8    Find empty time slot ($t$, $t+w(n_i, m)$), $t \geq$start_time on
       processor $m$
9    $t_f(n_i)=t+w(n_i, m)$
10   For each $n_j \in$succ($n_i$)
11     If proc($n_i$)$\neq$proc($n_j$)
12       $R$=Find the best route between proc($n_i$) and proc($n_j$)
13       Schedule $v(e_{ij})$ on $\mathbb{R}$ and determine $t_f(e_{ij})$
14     End If
15     taskready($n_j$) = max{taskready($n_j$), $T_f$}, where
         $T_f = \begin{cases} t_f(n_i), & \text{if proc}(n_i)=\text{proc}(n_j), \\ t_f(e_{ij}), & \text{otherwise.} \end{cases}$
16   End For
17 End For
```

In line 8, an insertion approach is used to find an empty slot. In this method it is possible, if all the precedence relations of a task are satisfied, for a task to be scheduled between other already scheduled tasks. Thus, in this insertion approach, tasks might be scheduled earlier in a time slot when a processor is running idle and can result in reduction of the schedule length (makespan ψ); in addition, using this approach helps eliminate the idle time from processors (Sinnen, 2007).

After decoding, makespan ψ is calculated from

$$\psi = \max_{n_j \in V} C(n_j), \tag{2}$$

where $C(n_j)$ is the completion time of task j. Due to our encoding and decoding methods, it is not necessary to have a topological order between tasks in each chromosome because precedence relations are considered in the scheduling of each task. Therefore, we do not worry about the changing order of tasks in the chromosome in the crossover and mutation steps.

For the selection step in GA, tournament selection is used. In this method, chromosomes are chosen randomly and the best of them is selected as the parent. After the selection process is completed, the 2D crossover/mutation operator presented in Tabatabaee-Yazdi and Akbarzadeh-T (2013) is used. This operator promotes better recombination and increases the diversity of the population. Fig. 2 shows an example of this operator. As shown in Fig. 2, this operator is able to change the corresponding processor of each task, e.g., tasks 8 and 10 in the first parent and its offspring and tasks 3 and 6 in the second parent and its offspring. In addition to adjusting the diversity of the population, another mutation operator is used. For mutation, elements of a randomly chosen individual in the population are randomly swapped.

In each generation, the population with the current population and current offspring is sorted again based on makespan ψ and only the best N individuals are selected, where N is the population size. The initial population is generated randomly. Population evolves until the maximum number of generations is met.

5.2.2 Time-varying scheduling

In this stage, the central scheduler manages the execution of tasks on processors and the transmission of information packets on links. Since a multiprocessor system changes with time, to reach a low ψ, the central scheduler uses a fuzzy decision-making procedure to correct the scheduling solution. Correction of the scheduling solution in each time step has more overhead in a time-varying environment and therefore the scheduling solution is corrected once every few time steps. Fig. 3 shows the flowchart for time-varying scheduling. This procedure is iterated until all tasks are executed and completed.

In the following, each part of the proposed method is explained in detail.

1. Execute tasks on processors

To execute tasks on processors at each step, the scheduling model based on a state space approach is used. The vector $\boldsymbol{U}$ in which all the elements fall in [0, 1] is equivalent to the utilization of processors. At each step, its values are updated based on changes in utilization of multiprocessor systems.

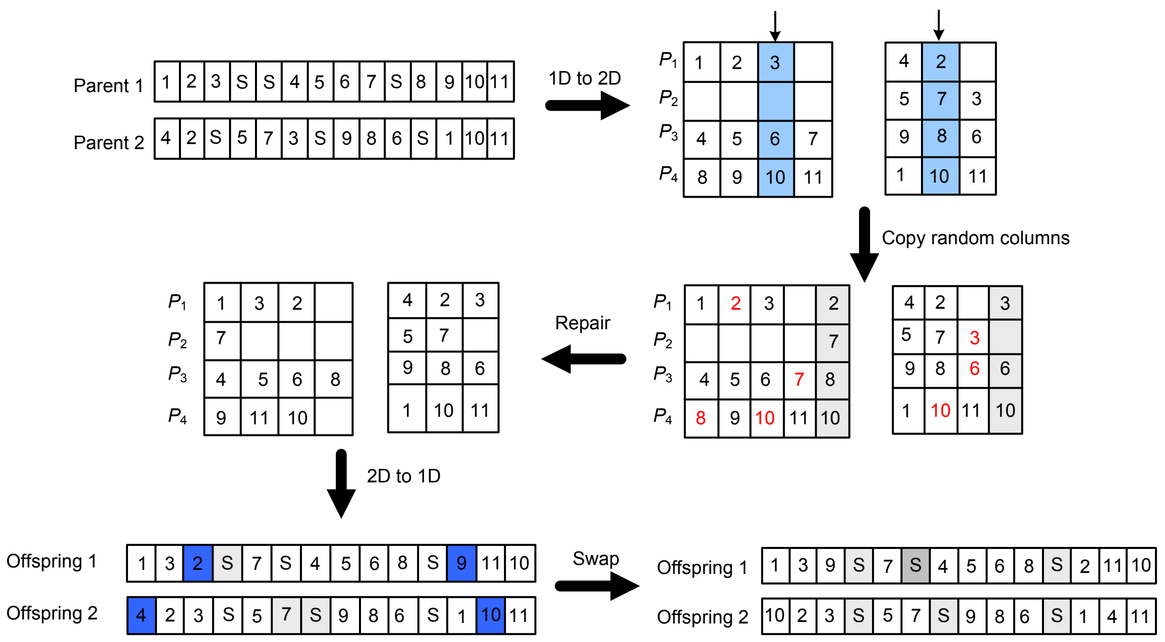


Fig. 2 A two-dimensional Xover/mutation operator example

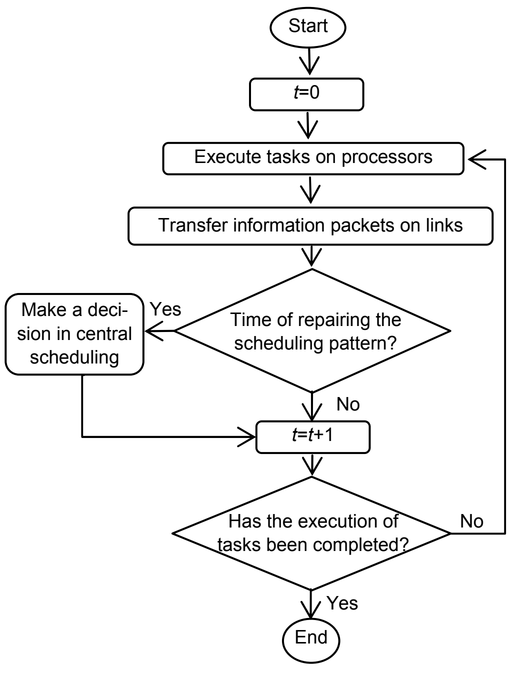

Start
t=0
Execute tasks on processors
Transfer information packets on links
Time of repairing the scheduling pattern?
Yes
Make a decision in central scheduling
No
t=t+1
Has the execution of tasks been completed?
No
Yes
End

Fig. 3 Flowchart of time-varying scheduling

2. Transfer information packets on links

When the execution of any task is completed, its information messages must be sent to the processors which will execute succeeding tasks. For sending information packets, a receiving processor is determined based on the current scheduling solution. After determining the destination processor, it is necessary to find a route between the source and destination processors. Since a multiprocessor system is time-varying and its links change with time, routing is done step by step and on each source processor. In routing mechanism, by knowing the destination processor, the shortest route between the source processor and the destination processor is found. The route's first link is then selected as the next link.

To determine the time required to transfer an information packet through a link, the following equation is used:

$$\text{Step}=\frac{1}{\Delta T}\frac{v(e_{ij})}{\text{Speed}(L_{\text{SD}})}, \tag{3}$$

where $v(e_{ij})$ is the volume of information packets corresponding to edge e_{ij} in kb, Speed(L_{SD}) is the speed of link between processors S and D in kb/s, and ΔT is the time taken for each time step.

If a link is disconnected while passing a packet, the routing mechanism is executed again between the current processor where the packet currently resides and the destination processor.

3. Correct the scheduling solution

The scheduling solution is revised periodically after given time intervals, under the following conditions: (1) The processor has a utilization rate less than a defined threshold; (2) The number of remaining corresponding tasks to the above processor is more than a defined threshold.

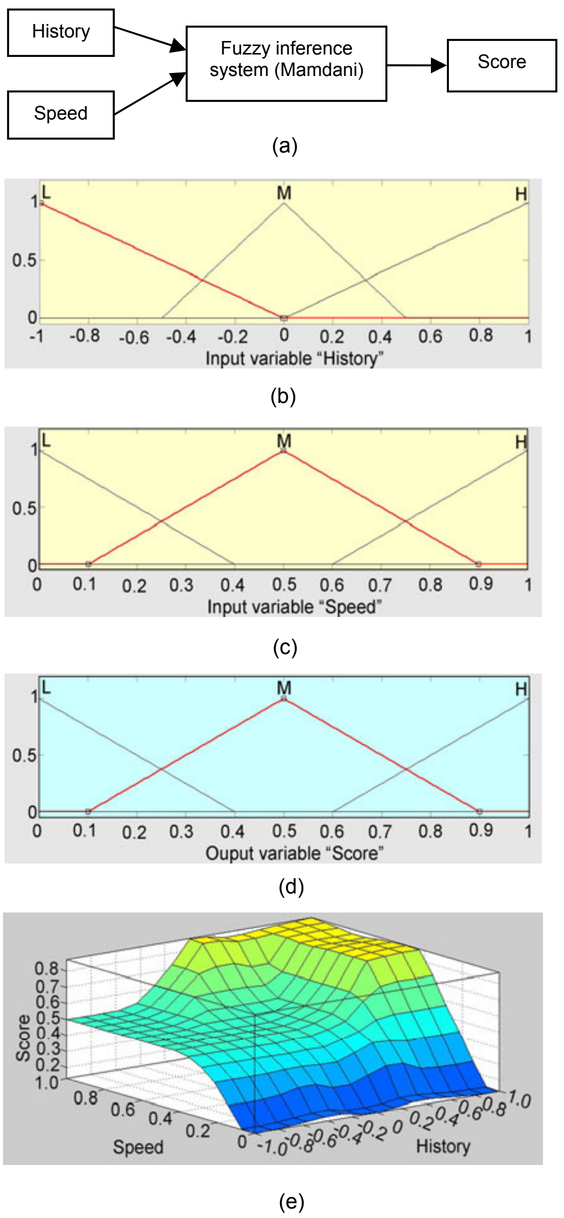
By the first condition, weak processors in terms of utilization are found. The second condition checks the value of task reallocation; when a task is preempted, a new processor is determined to execute it. On this new processor, a task must be executed again from the beginning to the end.

If the above two conditions are satisfied, a new processor must be determined to execute the preempted task. At this time, a fuzzy decision-making approach is used. The proposed fuzzy system is of the Mamdani type and has two inputs and one output. The first input is the history of changes in utilization of the processor, and the second input is the current utilization of the processor. The output is a score that shows the performance of the processor. The fuzzy inference is done for each processor and the processor that has the highest score is selected to execute the preempted task. The area below the profile represents variations in the processor throughput normalized to [−1, 1]. Fig. 4 shows the fuzzy inference system characteristics such as membership functions (MFs) of input and output variables, the control surface, and the fuzzy association memory (FAM) bank.

After fuzzy inference, the processor that has the highest score is selected to execute the preempted task. The central scheduler places the preempted task in the initial processor queue and sends the necessary information to the selected processor. Also, it considers the cost for reallocation of a task from one processor to another. This cost is the time delay that must occur before the task can be executed on the new processor.

6 Experimental results

In this section, we describe the performance of the proposed control algorithm compared with the method in Crăciun *et al.* (2010) on several random examples including task graphs (DAGs) generated using the P-method (Al-Sharaeh and Wells, 1996) in which a random method for generating DAGs is proposed based on a randomly generated adjacency matrix.



History / Speed	L	M	H
L	L	L	L
M	M	M	H
H	M	H	H

(f)

Fig. 4 Fuzzy inference system characteristics
(a) Fuzzy inference system; (b) Membership functions (MFs) of the first input variable, history of changes in utilization of the processor; (c) MFs of the second input variable, current utilization of the processor; (d) MFs of the output variable, score; (e) Control surface of the fuzzy inference system; (f) Fuzzy associative memory bank of the fuzzy inference system

In these experiments, the number of heterogeneous processors, the number of tasks, and sparsity are allowed to vary. The network topology is fully connected for proper comparison with the approach in Crăciun *et al.* (2010). The computation cost of tasks and communication cost between tasks in the task graph are chosen randomly based on a normal distribution (Page and Naughton, 2005; Yoo and Gen, 2007; Yoo, 2009). In the experiments, it is necessary to vary the parameters of a normal distribution in each experiment to justify the computation to communication ratio (CCR). We iterate each experiment 50 times and report the mean and standard deviation of results in each experiment. Table 2 shows the parameters of the experiments.

In these experiments, the sparsity parameter of DAG is fixed to 0.5. Without loss of generality, in the following experiments, we assume that the members of the λ_i^j are equal; i.e., all of the instructions in the ith task need an equal number of clocks to be completed. The scheduling algorithm in Crăciun *et al.* (2010), i.e., the GA scheduler, can be described in two steps. In the first step, a static solution is constructed for starting the scheduling. In this step, a simple GA is used to find the optimal solution which minimizes ψ. After starting the best schedule obtained from the previous step, for every scheduling event which is equivalent to any changes in the list of available processors, the first step is repeated to obtain the new best scheduling solution. During the second step, some of the best schedules are archived periodically to be used as a part of the initial population for the near future call of the GA. The results of the experiments are shown in Table 3. It is obvious that ψ obtained from the scheduler is better than its counterpart in Crăciun *et al.* (2010). Another point of view to be considered for these experiments is the standard deviation of ψ which is also smaller than that in the scheduler in Crăciun *et al.* (2010), especially when the number of processors increases.

This improvement is because the proposed method for correcting the scheduling pattern is simpler than the correction phase of the scheduler in Crăciun *et al.* (2010). To better visualize the comparison of the results in Table 3, they are plotted on a graph indicating percentage improvement of LS^3 to GA (Crăciun *et al.*, 2010) for different CCRs and processor numbers in Fig. 5. Hence, Table 3 and Fig. 5 are equivalent. The difference between the results of the LS^3 scheduler and the GA scheduler in case of CCR=100 is obviously due to the use of the routing

Table 2 Parameters of the experiments

CCR	Computation time (s)		Communication time (s)	
	Mean	Variance	Mean	Variance
0.1	30	4	3	6
1	30	4	30	6
10	30	3	300	2
100	30	3	3000	2

CCR: computation to communication ratio. In all cases, the processor number is in the range of 4–10, and the task number is 200

Table 3 Average and standard deviation of ψ repeated for 50 independent runs

Processor number	Method	Mean				Standard deviation			
		CCR=0.1	CCR=1	CCR=10	CCR=100	CCR=0.1	CCR=1	CCR=10	CCR=100
4	LS^3	43 400.22	9551.83	7105.85	11 369.37	697.1934	126.2441	251.4045	296.8553
	GA	47 420.34	10 606.66	8568.59	15 423.46	806.1842	122.3081	275.1712	357.7091
5	LS^3	40 284.78	8856.08	6485.95	10 118.09	771.9264	154.2187	194.2515	218.4474
	GA	44 304.13	9908.29	8640.94	15 467.27	875.9960	159.3568	351.8517	440.9569
6	LS^3	38 329.51	8406.31	6156.36	9480.79	669.5528	110.6767	147.5880	154.1164
	GA	40 807.88	9084.13	8597.45	15 389.43	791.1898	134.3046	289.5835	349.3945
7	LS^3	36 548.26	7966.81	6086.32	9190.35	704.2015	129.9079	281.4306	340.5310
	GA	39 225.14	8713.48	8023.75	14 041.57	868.5873	158.3808	407.2457	606.7961
8	LS^3	34 984.06	7588.99	5558.70	8338.05	625.7596	137.0898	189.0778	213.6579
	GA	37 890.46	8403.92	7459.35	12 979.27	773.9259	167.8107	340.4172	483.3925
9	LS^3	33 537.43	7153.79	5141.27	7609.08	568.8229	132.4962	144.5210	151.7470
	GA	36 705.04	8020.42	6819.79	11 593.63	725.5341	159.0337	311.4502	420.4577
10	LS^3	32 075.70	6727.91	4937.56	7258.22	558.6696	110.8118	184.7223	181.1807
	GA	35 493.46	7619.16	6239.16	10 481.80	730.1726	145.9182	312.6156	400.1480

algorithm. In this case, the communication overhead is the most influential part of ψ, and hence the optimal routing algorithm can play a significant role.

The percentage improvement is calculated using Eq. (4). Here, the definition of improvement is different from the conventional definition of percentage change since ψ is considered better when it is smaller.

$$\Omega = (\psi_{\text{GA}} - \psi_{\text{LS}^3}) / \psi_{\text{GA}} \times 100\%. \qquad (4)$$

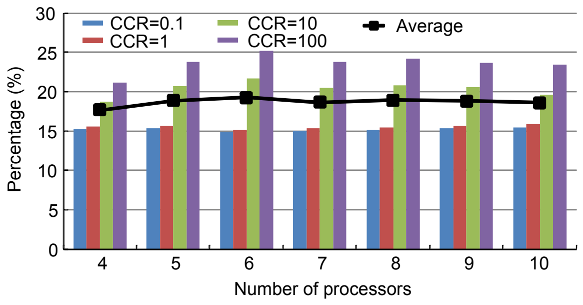


Fig. 5 Percentage improvement of LS^3 to GA (Crăciun *et al.*, 2010) for different CCRs and processor numbers

7 Conclusions and future work

In this paper, we assume a time-varying heterogeneous multiprocessor system in which the computational power of each processor may change over time. Also, in the assumed multiprocessor system, links may turn on or off and there is a link contention problem between them. To schedule a DAG on such a multiprocessor system, we propose an algorithm based on a linear switching space model using a fuzzy heuristic approach. The fuzzy decision-making procedure migrates a scheduled task on one processor to another to achieve a better makespan. This work is an extension of our previous work on task scheduling in static environments. To show the efficiency of LS^3, we design several random experiments and compare performances with those of an evolutionary task scheduling method developed for static and time-varying environments. The results show about 18% average improvement in makespan for large scale systems.

For future work, we aim to consider time-varying task arrival and adaptation to real-time scheduling constraints.

References

Abraham, A., Grosan, C., Liu, H., *et al.*, 2008. Nature inspired meta-heuristics for grid scheduling: single and multi-objective optimization approaches. *In*: Xhafa, F., Abraham, A. (Eds.), Metaheuristisc for Scheduling in Distributed Computing Environments, **146**(3):247-272.

Al-Sharaeh, S., Wells, B.E., 1996. A Comparison of heuristics for list schedules using the Box-method and P-method for random digraph generation. Proc. 28th Southeastern Symp. on System Theory, p.467-471. [doi:10.1109/SSST.1996.493549]

Cheng, S.C., Shiau, D.F., Huang, Y.M., *et al.*, 2009. Dynamic hard-real-time scheduling using genetic algorithm for multiprocessor task with resource and timing constraints. *Expert Syst. Appl.*, **36**(1):852-860. [doi:10.1016/j.eswa.2007.10.037]

Crăciun, C., Zaharie, D., Zamfirache, F., 2010. Evolutionary task scheduling in static and dynamic environments. Proc. IEEE Int. Joint Conf. on Computational Cybernetics and Technical Informatics, p.619-624.

Daoud, M.I., Kharma, N., 2008. A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. *J. Parall. Distr. Comput.*, **68**(4):399-409. [doi:10.1016/j.jpdc.2007.05.015]

Kong, X., Sun, J., Xu, W., 2008. Permutation-based particle swarm algorithm for tasks scheduling in heterogeneous systems with communication delays. *Int. J. Comput. Intell. Res.*, **4**(1):61-70.

Kwok, Y.K., Ahmad, I., 1996. Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. *IEEE Trans. Parall. Distr. Syst.*, **7**(5):506-521. [doi:10.1109/71.503776]

Long, Q.Q., Lin, J., Sun, Z.X., 2011. Agent scheduling model for adaptive dynamic load balancing in agent-based distributed simulations. *Simul. Modell. Pract. Theory*, **19**(4):1021-1034. [doi:10.1016/j.simpat.2011.01.002]

Page, A.J., Naughton, T.J., 2005. Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing. Proc. 19th IEEE Int. Parallel and Distributed Processing Symp., p.152-159. [doi:10.1109/IPDPS.2005.184]

Page, A.J., Keane, T.M., Naughton, T.J., 2008. Scheduling in a dynamic heterogeneous distributed system using estimation error. *J. Parall. Distr. Comput.*, **68**(11):1452-1462. [doi:10.1016/j.jpdc.2008.07.004]

Page, A.J., Keane, T.M., Naughton, T.J., *et al.*, 2010. Multi-heuristic dynamic task allocation using genetic algorithms in a heterogeneous distributed system. *J. Parall. Distr. Comput.*, **70**(7):758-766. [doi:10.1016/j.jpdc.2010.03.011]

Prodan, R., Fahringer, T., 2005. Dynamic scheduling of scientific workflow applications on the grid: a case study. Proc. 20th ACM Symp. on Applied Computing, p.687-694. [doi:10.1145/1066677.1066835]

Shahul, A.Z.S., Sinnen, O., 2010. Scheduling task graphs optimally with A*. *J. Supercomput.*, **51**(1):310-332.

Shin, K., Cha, M., Jang, M., *et al*., 2008. Task scheduling algorithm using minimized duplications in homogeneous systems. *J. Parall. Distr. Comput.*, **68**(8):1146-1156. [doi:10.1016/j.jpdc.2008.04.001]

Sinnen, O., 2007. Task scheduling for parallel systems (1st Ed.). JohnWiley & Sons-Interscience.

Sinnen, O., Sousa, L.A., Sandnes, F.E., 2006. Toward a realistic task scheduling model. *IEEE Trans. Parall. Distr. Syst.*, **17**(3):263-275. [doi:10.1109/TPDS.2006.40]

Sivanandam, S.N., Visalakshi, P., 2009. Dynamic task scheduling with load balancing using hybrid particle swarm optimization. *Int. J. Open Probl. Comput. Math.*, **2**(3): 475-488.

Tabatabaee-Yazdi, H., Akbarzadeh-T, M.R., 2013. The linear switching state space: a new modeling paradigm for task scheduling problems. *Int. J. Innov. Comput. Inform. Contr.*, **9**(4):1651-1677.

Yoo, M., 2009. Real-time task scheduling by multiobjective genetic algorithm. *J. Syst. Softw.*, **82**(4):619-628. [doi: 10.1016/j.jss.2008.08.039]

Yoo, M., Gen, M., 2007. Scheduling algorithm for real-time tasks using multiobjective hybrid genetic algorithm in heterogeneous multiprocessors system. *Comput. Oper. Res.*, **34**(10):3084-3098. [doi:10.1016/j.cor.2005.11.016]