Journal of Computer Science and Cybernetics, V.27, N.3 (2011), 218–228

DURATION AUTOMATON IN SCHEDULING PROGRAMS FOR A CLUSTER COMPUTER SYSTEM*

BUI VU ANH

Faculty of Mathematics, Mechanics and Informatics
VNU University of Science

Tóm tắt. Lập lịch tối ưu cho các công việc chạy trên các máy, trong trường hợp tổng quát, là một bài toán khó và không có thuật toán thực hiện trong thời gian đa thức. Các giải pháp tối ưu và xấp xỉ tối ưu chỉ giải quyết cho các trường hợp riêng với các ràng buộc hạn chế. Thuật toán lập lịch trên 1 và 2 máy được công bố ở [4, 16] được xem như những khởi đầu. Trong [16], tác giả giải bài toán *công việc quá hạn* có tính đến thời gian chuẩn bị. [4] giải bài toán *công việc đến hạn* trên trường hợp hai máy, và có thể mở rộng cho trường hợp 3 máy với một số điều kiện trên công việc. Các tác giả khác cũng xem xét các bài toán lập lịch ở các trường hợp riêng như trong [5, 8, 17]. Bài báo ứng dụng mô hình otomat khoảng (Duration Automaton - DA) [1, 2] giải quyết bài toán lập lịch động cho các công việc với thời gian xử lý không chắc chắn trên máy tính ghép cụm - hệ thống gồm nhiều máy tính (nốt tính toán) phối hợp làm việc với nhau. Chúng tôi xét cụm máy tính trong hai trường hợp: các máy giống nhau và khác nhau về cấu hình (tài nguyên của máy tính, một cách hình thức được quy về cấu hình, thể hiện qua thời gian xử lý công việc). Do không biết trước thời gian hoàn thành mỗi công việc, các thuật toán tối ưu đã có sẽ không được áp dụng một cách hiệu quả. Thông qua việc mô hình hóa và sử dụng tiêu chuẩn về thứ tự của otomat khoảng, chúng tôi đề xuất thuật toán lập lịch và thực nghiệm cho thấy có kết quả tốt về thời gian hoàn thành các công việc so với các phương pháp truyền thống như FIFO (hàng đợi tự nhiên), hàng đợi công việc với tiêu chuẩn hoàn thành *nhanh* trước (tiếp cận tham lam), hoàn thành *lâu* trước (tiếp cận an toàn) không đồng bộ.

Abstract. Optimal schedule for works running on machines, in a general case, is a hard problem and there is no complete optimal deterministic algorithm in polynomial time. Optimal and approximated solutions were issued for some specific cases with constraints. One can find the solutions for the cases 1 and 2 machines [4, 16] as the initial algorithms. In [16], author solved *late works* problem using algorithm with setup times included. [4] solve the *due works* problem on two machines, and can be extended for the case of 3 machines with some conditions on works. Other authors looked at schedule problems in specific cases like [5, 8, 17]. In this paper, we apply duration automata [1, 2] to solve the schedule problem dynamically for the works with uncertain processing time in a cluster computer, which is a system consisting of many computers (computing nodes) co-working together. We solve the schedule problem in a cluster with m machines for two cases: all machines are the same and different in configurations (machine's resources are formally considered as a configuration information, represented by the time need to finish the works). Because of uncertainly processing time, tranditional algorithms can not be used effectively. By using DA model with DA's order criterion, we issue schedule algorithms and practically prove to be better in time consuming compare to FIFO (natural order), the fastest first (greedy) and the longest first (safety) methods without synchronized points.

*This paper was supported by Hanoi University of Science (grant TN-10-05)

1. INTRODUCTION

Formal tools are usually used to model systems [1, 2, 3, 9, 10]. DA is a traditional automaton which is augmented with a clock variable and a timed duration constraint on each transition [1, 2]. The labels (or actions) of transitions are separated into three types: *input*, *output*, and *internal* comparing to *internal* and *external* actions of the IO automata [11, 12, 13]. This automaton is less complex than timed automaton [9] but more flexible in reality comparing to IO automata [11]. We have used DA to model component-based real-time systems, real-time objects modeling, and embedded systems with timed and untimed specifications [2], priority networks [1]... In this paper, we use DA to model a cluster computer system where its nodes can be controlled by a master one (head node) or worked independently, and solve the scheduling problem.

Scheduling works on machines, with many kind of constraints on work's order, is a difficult problem and there is no optimal solution in polynomial time. Some special cases had been risen by [4, 16] and there were good solutions (gready approach) but they are really simple to be widely used. Recent reseaches have been moved to the application aspect in serveral cases. Authors in [5] concentrated on the problem of two machines in which works came over the time. There is an improved issue [7] works on two machines with the availability constraint. In [8], authors solved problem on two machines with the view of fuzzy-set theory; and author in [17] developed a $3/2-$algorithm to solve the problem in [16] again. In this paper, we look at the problem with the view of DA, and consider the problem in a different aspect: schedule uncertain processing time works for machines. In special cases, our problems will turn into the problem in [16, 8, 7] when the processing times are defined. We also issue criteria which is used to develop algorithms solving these problems.

2. DURATION AUTOMATON

Definition 2.1. A duration automaton is a tuple $M = \langle S, \Sigma, \triangle, \nabla, q, R, F\rangle$ where:

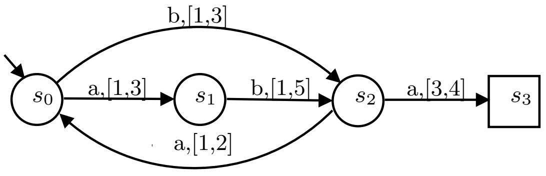
- S is a finite set of states. $q \in S$ is an initial state. In a general case, q can be a set.
- $\Sigma, \triangle, \nabla$ are *internal*, *input* and *output* alphabet of actions (or labels). We denote the set of actions of DA by $\mathcal{A} = \Sigma \cup \triangle \cup \nabla$. There is an *empty* action $\varepsilon \in \Sigma$.
- $R \subseteq S \times \mathcal{A} \times domain \times S$ is a set of transitions, where $domain = \{[l,u], (l,u], [l,u), (l,u) \mid l,\ u \in \mathbf{Z}^+, l \leq u\}$. For each transition $e = (s, a, d, s') \in R$, a will be an output action of s and input action of s' as well. If $s = s'$ then e is a ring.
- $F \subseteq S$ is a set of final (or accepted) states.

We denote by $S(M),\ \Sigma(M),\ \triangle(M),\ \nabla(M),\ R(M),\ F(M)$ the corresponding components of M, and $\mathcal{A}(M) = \Sigma(M) \cup \triangle(M) \cup \nabla(M)$. For $s \in S(M)$, $\Sigma(s),\ \triangle(s),\ \nabla(s)$ are the *internal*, *input* and *output* actions of the state s respectively.

A configuration of M is a couple (s,t), where $s \in S$, $t \in \mathbf{R}^+$ which shows that M reaches the state s and stays there at the time t. So that, initial configuration of M is $(q, 0)$. As the time passes, the changes of M are of the following forms:

- *Time-change*: $(s,t) \xrightarrow{a,\sigma} (s, t+\sigma)$ where $\sigma \in \mathbf{R}^+$, $a \in \Sigma(s)$. Automaton stays at state s and does its *internal* actions.

- *State-change:* $(s,t) \xrightarrow{a,\sigma} (s', t+\sigma)$ where $\sigma \in \mathbf{R}^+, a \in \triangle(s)$, using transition $(s,a,d,s') \in R(M)$, $t+\sigma \in d$. The transition can take place if the time constraint on the arc between s and s' be satisfied. We make an extra assumption that *internal* actions takes no time. It can finish at a small enough time that can be ignored and we only concentrate on input and output actions.



Hình 2.1. Example of duration automaton

Definition 2.2. Let M be a DA and $\widetilde{p}$ be a sequence $(s_0,0),(s_1,d_1),(s_2,d_2),\dots\ (s_n,d_n)$, $\widetilde{p} = (s_i,d_i)_{i=0..n}$ in short, be a sequence of changes.
- If $\widetilde{p}$ satisfies $s_0 = q$ is the initial state, and for each $1 \le i \le n$, $\exists e_i \in R : e_i = (s_{i-1}, a_i, d_i, s_i)$ which makes M move from s_{i-1} to s_i, then it is called a *d-path* from s_0 to s_n in M.
- $\widetilde{p}$ is a d-path. If there is a strictly increasing sequence $t_0 = 0, t_1, t_2, \cdots, t_n$ such that for all $i, 1 \le i \le n$: $t_i - t_{i-1} \in d_i$ then $\overline{p} = (s_0,0),(s_1,t_1),(s_2,t_2),\dots(s_n,t_n)$, $(s_i,t_{i+1})_{i=0..n}$ in short, is called an t-path of $\widetilde{p}$. d-path with t-path and $s_n \in F$ is called a successful path of M. $\widetilde{w} = (a_1,d_1),(a_2,d_2),(a_3,d_3),\dots(a_n,d_n)$, $(a_i,d_i)_{i=1..n}$ in short, where a_i is the action of the transaction e_i of $\widetilde{p}$, is a *d-word* and $\overline{w} = (a_1,t_1),(a_2,t_2),(a_3,t_3),\dots(a_n,t_n)$, $(a_i,t_i)_{i=1..n}$ in short, is called a t-word of $\widetilde{p}$.
- A d-word $\widetilde{w}$ is acceptable (or d-word of M) if there is at least one t-word $\overline{w}$ of $\widetilde{w}$. We say $\overline{w}$ takes M from s_0 to s_n and $\widetilde{w}$ can take M from s_0 to s_n. Each (a_i,d_i) is called an atom. The d-word $\widetilde{w}$ without t-word is called unacceptable.
- The d-word $\widetilde{w}$ is accepted by M if it is a d-word of a successful path. All accepted words of M is called a *d-language* of M, denoted by $\mathcal{L}(M)$.

Definition 2.3. Given a DA $M = \langle S, \Sigma, \triangle, \nabla, q, R, F\rangle$. The projection of M over a real-time value t is $M' = \langle S, \Sigma, \triangle, \nabla, q, R', F\rangle$ where the transitions relationship specified as $R' = \{(s,a,s') | (s,a,d,s') \in R,\ t \in d\}$

As the time passes, the projection operator will give an imagination of DA at a time we observe.

Assume $M = \langle S, \Sigma, \triangle, \nabla, q, R, F\rangle$ is a DA. We add a global clock x, which runs regularly, automatically and can be reset to the initial state, and a function f defined as: for each $e = (s,a,d,s') \in R$, $f(e) = (s,a,s',\lambda,\delta_e)$ where $\lambda = \{x\}$ and $\delta_e = (x \in d)$. Let R' be $\{f(e) \mid \forall e \in R\}$. Because f is an 1-1 function between R and R', so that $T = \langle \Sigma \cup \triangle \cup \nabla, S, q, X, R', \delta, F\rangle$ for $\delta = \{\delta_e,\ e \in R\}$, $X = \{x\}$ is a unique corresponding time automaton accepting the same language as DA. Thus, DA is a specific class of time automaton [9]. We can use time automaton's tools to check for every time properties of DA. As a consequence of [9], we will have the following theorem.

Theorem 2.1. *The reachable and emptiness problems of DA is decidable.*

3. CLUSTER SYSTEM MODELING

A computing node (computer) can be modeled with a DA, where states of the node will be the states of DA, and the changes between states of the node are the arcs of DA. Nodes can have their own internal actions which are separated from one other physically. That results internal actions of nodes belong to themselves and they can do the same (shared) action at a time but independently.

Each node can execute a compiled program (called a *work*) which is modeled as an atom, which can not be divided into the smaller one, (id, d) where id is a work identifier, $d = [l, u]$, $l < u$, $l, u \in \mathbf{Z}^+$. l is an estimated amount of time to finish the work, and u is the maximum amount of time that work must finish. u can be infinitive to show that work can run as long as it needs. The program should finish at time t inside d and (id, t) is called a transition of a node at time t. When a node is ready to receive a work, we said that the node is in $r$eady state. If a node is doing a work, the state will be $b$usy. In case a node can not receive any work, it is in $f$ailure state and can only be $r$eady again by doing $r$eset action. This action can be taken place at any time to make the node from any state to $r$eady state and is not in the action set of any node. Each node has an empty action ε, i. e. the node stays at the $r$eady state (or does its internal actions) which is supposed can be finished at any time. $e$mpty work has the special form $(id, [0, \infty))$, which means that it takes 0 time to finish. Practically, we do not have to care much about this action because it can be considered as internal to that node. The state before the first time in $r$eady is $i$nitial at time $t_0 = 0$.

Definition 3.1. Suppose that there is a computing node M. Let $\widetilde{x} = (id_1, d_1), (id_2, d_2), \cdots,$ (id_n, d_n). If there exists an instance $\overline{x} = (id_1, t_1), (id_2, t_2), \cdots, (id_n, t_n)$ which makes M change from a $r$eady state to a $r$eady state and $t_1 \in d_1$, $t_2 - t_1 \in d_2, \ldots$ $t_n - t_{n-1} \in d_n$ then $\widetilde{x}$ is said to be a word of M. All the word of M is called a language of M and denoted by $\mathcal{L}(M)$

For two words $\widetilde{x}$, $\widetilde{y}$ of M, we denote $\widetilde{x}^\frown\widetilde{y}$ as a word generated by appending $\widetilde{y}$ after $\widetilde{x}$ (connect operator).

Corollary 3.1. *If $\widetilde{x}$, $\widetilde{y}$ are words of M then $\widetilde{x}^\frown\widetilde{y}$ is also a word of M.*

Chứng minh. Assume $\widetilde{x}$ and $\widetilde{y}$ are words of M but $\widetilde{x}^\frown\widetilde{y}$ is not, so that there is a work (id, d) in $\widetilde{x}^\frown\widetilde{y}$ which does not bring M from $r$eady state to $r$eady state. (id, d) in $\widetilde{x}^\frown\widetilde{y}$ implies (id, d) in $\widetilde{x}$ or (id, d) in $\widetilde{y}$. It means (id, d) can not bring M from $r$eady state to $r$eady state in either $\widetilde{x}$ or $\widetilde{y}$ (conflicts with assumption $\widetilde{x}$ and $\widetilde{y}$ are words of M).

By corollary 3.1, if $\widetilde{x}$ and $\widetilde{y}$ are the words which run on M one after another, then we can append $\widetilde{y}$ after $\widetilde{x}$ to make a new compound word and run it on M.

Definition 3.2. (Work relationship) For two works $w = (id, [l, u])$ and $w' = (id', [l', u'])$.

- $w =_{DA} w'$ iff $(l' = l) \wedge (u = u')$.
- w is $s$maller than w' (written as $w <_{DA} w'$ or $w' >_{DA} w$) iff $(\dfrac{u' + l'}{2} > \dfrac{u + l}{2}) \vee (\dfrac{u' + l'}{2} = \dfrac{u + l}{2} \wedge l < l')$.
- If $w =_{DA} w'$ and $w <_{DA} w'$, we write $w \leq_{DA} w'$ or $w' \geq_{DA} w$.

Theorem 3.1. $\leq_{DA}$ *is a totally ordered in* $\mathcal{A} \times$ *domain and* $((\mathcal{A} \times domain), \leq_{DA})$ *is a lattice whenever domain is bounded.*

Chứng minh. Given three works $x = (id_x, [l, u])$, $y = (id_y, [l', u'])$ and $z = (id_z, [l'', u''])$ of M.
- ***Reflexive:*** $x = x$ because $(l = l) \ \wedge \ (u = u)$.
- ***Antisymmetric:*** If $(x \leq_{DA} y)$ and $(y \leq_{DA} x)$ then $l' = l$ and $u = u'$, implies that $x =_{DA} y$
- ***Transitive:***
$x \leq_{DA} y \ \Rightarrow (\dfrac{u'+l'}{2} > \dfrac{u+l}{2}) \ \vee \ (\dfrac{u'+l'}{2} = \dfrac{u+l}{2} \ \wedge \ l < l')$. We have 4 cases:

$y \leq_{DA} z \ \Rightarrow (\dfrac{u''+l''}{2} > \dfrac{u'+l'}{2}) \ \vee \ (\dfrac{u''+l''}{2} = \dfrac{u'+l'}{2} \ \wedge \ l' < l'')$

- $(\dfrac{u'+l'}{2} > \dfrac{u+l}{2})$ and $(\dfrac{u''+l''}{2} > \dfrac{u'+l'}{2}) \Rightarrow (\dfrac{u''+l''}{2} > \dfrac{u+l}{2}) \Rightarrow x \leq_{DA} z.$
- $(\dfrac{u'+l'}{2} > \dfrac{u+l}{2})$ and $(\dfrac{u''+l''}{2} = \dfrac{u'+l'}{2} \ \wedge \ l' < l'') \Rightarrow (\dfrac{u''+l''}{2} > \dfrac{u+l}{2}) \Rightarrow x \leq_{DA} z.$
- $(\dfrac{u'+l'}{2} = \dfrac{u+l}{2} \ \wedge \ l < l')$ and $\Rightarrow (\dfrac{u''+l''}{2} > \dfrac{u'+l'}{2}) \Rightarrow (\dfrac{u''+l''}{2} > \dfrac{u+l}{2}) \Rightarrow x \leq_{DA} z.$
- $(\dfrac{u'+l'}{2} = \dfrac{u+l}{2} \wedge l < l')$ and $(\dfrac{u''+l''}{2} = \dfrac{u'+l'}{2} \wedge l' < l'') \Rightarrow \dfrac{u+l}{2} = \dfrac{u''+l''}{2} \wedge l < l''$ $\Rightarrow x \leq_{DA} z.$

Because of *domain* with points l, u are in $\mathbf{Z}$ (bounded), so that in a set of works, we can find max and min ones. Thus, ($(\mathcal{A} \times domain), \leq_{DA}$) is a lattice.

Note: If $u = l$ then $\leq_{DA}$ and $<_{DA}$ becomes traditional order $\leq$ and $<$.

Definition 3.3. (*W*ord comparison) Suppose $\widetilde{x} = (id_i, [l_{x_i}, u_{x_i}])_{i=1..n}$ and $\widetilde{y} = (id_j, [l_{y_j}, u_{y_j}])_{j=1..n}$ are words of a node M. We define:

$$\widetilde{x} =_{DA} \widetilde{y} \Leftrightarrow \begin{cases} \widetilde{x} \in \mathcal{L}(M) \Leftrightarrow \widetilde{y} \in \mathcal{L}(M) \\ (t_{l_x} = t_{l_y}) \ \wedge \ (t_{u_x} = t_{u_y}) \end{cases}$$

$$\widetilde{x} <_{DA} \widetilde{y} \Leftrightarrow \begin{cases} \widetilde{x} \in \mathcal{L}(M) \Leftrightarrow \widetilde{y} \in \mathcal{L}(M) \\ (\dfrac{t_{u_x} - t_{l_x}}{2} < \dfrac{t_{u_y} - t_{l_y}}{2}) \\ \text{or } (\dfrac{t_{u_x} - t_{l_x}}{2} = \dfrac{t_{u_y} - t_{l_y}}{2}) \ \wedge (t_{l_x} < t_{l_y}) \end{cases}$$

where $t_{l_x} = \sum_i l_{x_i}, \ t_{l_y} = \sum_j l_{y_j}, \ t_{u_x} = \sum_i u_{x_i}, \ t_{u_y} = \sum_j u_{y_j}$
When $\widetilde{x} <_{DA} \widetilde{y}$ or $\widetilde{x} =_{DA} \widetilde{y}$, we have $\widetilde{x} \leq_{DA} \widetilde{y}$.

Lemma 3.1. *The relation* $\leq_{DA}$ *is a totally ordered.*

The proof can be done in the same way as in the proof of the theorem 3.1.

Lemma 3.2. *If* $\widetilde{x}$, $\widetilde{y}$, $\widetilde{z}$ *are words of a node* M, ε *is an empty word then*
- $\widetilde{x}^\frown\widetilde{y} =_{DA} \widetilde{y}^\frown\widetilde{x}$
- $\widetilde{x}^\frown\widetilde{y}^\frown\widetilde{z} =_{DA} (\widetilde{x}^\frown\widetilde{y})^\frown\widetilde{z} =_{DA} \widetilde{x}^\frown(\widetilde{y}^\frown\widetilde{z})$
- $\widetilde{x}^\frown\varepsilon =_{DA} \varepsilon^\frown\widetilde{x} =_{DA} \widetilde{x}$

Because we do not consider the priority order of words, lemma 3.2 shows that if $\widetilde{x}$ and $\widetilde{y}$ are two words of the node M then we can run $\widetilde{x}$ before $\widetilde{y}$ or $\widetilde{y}$ before $\widetilde{x}$. Moreover, we can group some words together to make a new one.

Lemma 3.3. *Suppose* $\widetilde{x},\ \widetilde{y},\ \widetilde{z}$ *are words of* M.
- *If* $\widetilde{x} \leq_{DA} \widetilde{y}$ *and* $\widetilde{y} \leq_{DA} \widetilde{z}$ *then* $\widetilde{x} \leq_{DA} \widetilde{z}$
- *If* $\widetilde{x} \leq_{DA} \widetilde{y}$ *then* $\widetilde{x}\frown\widetilde{z} \leq_{DA} \widetilde{y}\frown\widetilde{z}$ *and* $\widetilde{z}\frown\widetilde{x} \leq_{DA} \widetilde{z}\frown\widetilde{y}$

A cluster computer system (cluster in short) consists of some computing nodes connected by a network and worked together with or without the control of a master node. The actions belong to a group of nodes (share actions) can be synchronized. Formally, the system can be modeled as a parallel product of DAs. Let $M_1, M_2, \cdots,\ M_n$ be DAs corresponding to nodes. We call parallel product of DAs a cluster on $M_1, M_2, \cdots, M_n$, written as $\mathbf{M} = M_1 \times M_2 \cdots M_n$. A configuration of $\mathbf{M}$ is a tuple $\mathcal{C} = (c_1, c_2, \cdots c_n)$ where $c_i,\ i = 1..n$ is a configuration of M_i. Configuration $\mathcal{C}$ of $\mathbf{M}$ is:
- *initial* if $\exists c_i,\ 1 \leq i \leq n$ is the initial configuration of M_i and others are in *initial* or *ready* states.
- *ready* if $\forall i,\ i = 1..n,\ c_i$ are the *r*eady states of M_i.
- *busy* if $\exists i, 1 \leq i \leq n : c_i$ is the *b*usy state, others are not in *f*ailure states of correlative M_i.
- *failure* if $\exists i, 1 \leq i \leq n : c_i$ is the *f*ailure state of M_i.

Suppose $\mathbf{M} = M_1 \times M_2 \times \cdots M_n$ is a cluster. Let $id \in \cup_{i=1}^{n}\mathcal{A}(M_i)$ and $dom(id) = \{i \mid id \in \mathcal{A}(M_i)\}$. Transition $((s_1, t_1), \ldots, (s_n, t_n)) \xrightarrow{id,\sigma} ((s'_1, t'_1), \ldots, (s'_n, t'_n))$, where $\sigma \geq 0$ and $id \in \cup_{i=1}^{n}\mathcal{A}(M_i) \cup \{\varepsilon\}$ is a transition of $\mathbf{M}$ if there is a transition $(s_i, id, d_i, s''_i) \in R(M_i)$ satisfying $t_i + \sigma \in d_i$ for any $i \in dom(id)$

$$(s'_i, t'_i) = \begin{cases} (s_i, t_i + \sigma) & \text{if } i \notin dom(id) \\ (s''_i, t_i + \sigma) & \text{if } i \in dom(id) \text{ and } \exists (s_i, id, d_i, s''_i) \\ & \in R(M_i) \text{ where } t_i + \sigma \in d_i \end{cases}$$

Definition 3.4. A sequence of transitions leads cluster from a *r*eady state to the other *r*eady state is called a word of the cluster.

For two words with different length, we can add *e*mpty works to the end of the shorter to make it as long as the longer. So we always can assume that two words have the same length, starting with not *e*mpty works in every comparison.

Definition 3.5. (*S*chedule) Suppose that there is a set of works $X = \{(i, [l_i, u_i])_{i=1..n}\}$ and a cluster $M = M_1 \times M_2 \times \cdots \times M_m$. A permutation of X: $\widetilde{x} = (i, [l_i, u_i])_{i=1..n}$ is called a schedule for the works and $t(\widetilde{x}) = [t_l(\widetilde{x}), t_u(\widetilde{x})]$ is the duration time requirement for $\widetilde{x}$ where $t_l(\widetilde{x}) = \sum_{i=1}^{n} l_i,\ t_u(\widetilde{x}) = \sum_{i=1}^{n} u_i$.

Let $Sh = (sh_1, sh_2, \ldots, sh_m),\ sh_i \in \{0,1\}^n$ where $sh_i[j] = 1$ if work j is arranged to run on node M_i, otherwise $sh_i[j] = 0$ is called a works schedule on $\mathbf{M}$ and $\widetilde{x}_i = [(j, [l_j, u_j]) \mid sh_i[j] = 1, j = 1..n]$ is a node schedule. The time needed by a cluster schedule is the time duration needed to finish all cluster schedule $T(Sh) = [T_l, T_u]$:

$$T_l = \min_{i=1..m}\{t_l(\widetilde{x}_i)\},\ T_u = \max_{i=1..m}\{t_u(\widetilde{x}_i)\}$$

Definition 3.6. Suppose that $\widetilde{x}$ is a word with length n and m is the number of machines. The work j of $\widetilde{x}$ runs on the machine i takes the constraint d_{ij}. $Sh = (sh_1, sh_2, \cdots, sh_m)$ is a schedule.
- Sh is a good schedule if:
+ $\sum\limits_{i=1}^{m} sh_i[j] = 1$ for $j = 1..n$.
+ if $sh_i[j] = 1$ then $d_{ij} = \min\limits_{k=1..m} d_{kj}$ (using $\leq_{DA}$ order).
- sh_i is a non zero vector, means that $\exists k : sh_i[k] = 1$. Schedule $Sh' = (sh_1, \cdots, sh'_i, \cdots, sh'_j, \cdots, sh_m)$ is conclused by moving a work k^{th} in sh_i to sh_j. Sh' is better than Sh if $T(Sh') <_{DA} T(Sh)$. Sh will be (one of) the best schedules if there is no better one.

4. SCHEDULING ALGORITHMS

4.1. Machines are the same in configuration

• **Problem:** Assume that there is a set of n works $\widetilde{x} = (j, [l_j, u_j]),\ j = 1..n$ where l_j is at least time work j can be finihed and u_j is the deadline of time that work must finish or being failure since a node takes that work. Given a time value t, scheduling to finish as many works in $\widetilde{x}$ as possible (late work, maximun time flow) using one node: $\max\limits_{\sum_{j=1}^{k} t_j \leq t} k$

where t_j is the time work j finishes.
Algorithm 1: Late work problem
Input: m works in the queue $\widetilde{x}$.
Output: Done works and consumed time.

```
1. Sort all works in $\widetilde{x}$ increasing by $\leq_{DA}$ order.
2. Mark the all works undone and set $time = 0$.
3. Start from the work (at position) $i = 0$ in $\widetilde{x}$.
4. While ($time < t$) and ($i < m$)
       + Do the work $i$ and get consumed time $ctime$.
       + If ($time + ctime \leq t$) then mark the work $i$ is done and try the work $i + 1$.
       + Otherwise, the work $i$ is failure and break.
5. Return $time$ and done works.
```

This problem is solved by selecting $s$mall works to do first (greedy approach) and minimize waiting time (total and on average). The algorithm can be expanded for the case $m > 1$ nodes with the same configurations by finding first $r$eady node k before delivering the work to k. In that case, we will have m time variables denoted by $time[j],\ j = 1..m$ which are used to sum the time consumed by each node. The condition in while loop (step 4) will change to $max(time[]) < t$ and the condition in if statement will change to $time[j] + ctime \leq t$.

Most of schedulers only use the criterion on maximum time to finish a work. If that amount of time passes but the work does not finish (because of wrong estimated or some unawared reasons), that node still returns to $r$eady state and leaves the work undone. That results all the schedule or the works after a tracked point will have to do again. Our criterion will make the estimate progress more flexible, accept some wrong estimation in time. Further more, if a job was submitted with open access policy, nodes of the cluster can be seen and accessed, we can send works directly to nodes. If computing nodes are managed by a $h$ead node then every access must be allowed. In this case, we can not see nodes inside and the cluster will be considered as a compound node (represented by $h$ead node); we have to schedule works for one node.

4.2. Machines are different in configurations

• **Problem:** Assume that there is n works $\widetilde{x} = (j, [l_j, u_j]),\ j = 1..n$, where l_j is at least time work j can be finihed and u_j is the deadline of time that work must finish or being failure since a node takes that work. Schedule to finish works in $\widetilde{x}$ using $m \geq 2$ nodes in a as small time as possible. If $time[i]$ is the time node i finish all its works in a schedule S then we have to find: $\min\limits_{S} \max\limits_{i} time[i]$.
Assume that work $j,\ j = 1..n$, running on node $i,\ i = 1..m$, takes constraint d_{ij}.

Algorithm 2: Scheduling works on $m \geq 2$ general nodes
Input: $\widetilde{x}$ is a queue of works, and m computing nodes.
Output: All done works and time consumed the schedule.

```
1. Enable all nodes. Set consumed times on nodes to 0.
2. Find a good schedule $S$ on $\widetilde{x}$ for nodes. $\widetilde{x}_i$ are works for nodes $i,\ i = 1..m$ (work-queue $i$).
3. Sort $\widetilde{x}_i,\ i = 1..m$, ascending by $\leq_{DA}$ order.
4. For any enabled node $i$ with empty work-queue, re-schedule $S$ as following:
   + Try undone work $j$ (from small to big when running on $i$), suppose $j$ is in the $\widetilde{x}_k$ with at least 2 works.
   + If $d_{kj} \leq_{DA} T(\widetilde{x}_k)$ then move $j$ from $\widetilde{x}_k$ to $\widetilde{x}_i$.
   Otherwise, try an other work.
   + If there is no movable work, disable node $i$.
   + Repeat re-schedule process until there is no enabled node with empty work-queue.
5. Deliver the works in $\widetilde{x}_i$ to node $i$ if $i$ is $r$eady and update the time consumed by node $i$ until there is an empty work-queue on enabled node.
6. Repeat step 4 and 5 until all node schedules are empty.
7. Return the maximum consumed time on nodes.
```

A smaller work will be done first on node that work consumes the smallest time, so that it will minimum the waiting time of works. In case there is any free node, it can share work from other if the time the free node needs to finish a share work is smaller than the time the node to be shared finishs all its works (include the work will be shared). Each time we move a work from one queue to another, the required time by the schedule for that node will be reduced by the time that work requires on the to be shared node. When there is no more movement, no more share is needed. This approach gives the dynamic optimal on both flow and make span time of the schedule.

5. EXPERIMENT

We performed 10 tests on the same input data, and got the average value with the number of machines are 5, 10, 20, 50, 100 respectively. Initial time l of each work is between 0 and 50, time upper bound u and actually finish time of each work are different in 4 cases:

- *Case 1*: Time upper bound and finish time are the same, it means $l = u = t$ where t is an actually finish time.
- *Case 2*: Upper limit is initial time plus a random number between 0 and 25 (50% of initial time).
- *Case 3*: Upper limit is initial time plus a random number between 0 and 50 (100% of initial time).

5.1. Machines are the same

We compare schedules using $<_{DA}$ order with schedule using FIFO, $l$ongest first and $s$hortest first approach. After scheduling, works will be delivered to nodes using Round-Robin method. We run each test for 10 times on 10000 works. Earning points for each case about time among DA and FIFO schedule, $l$ongest-first, $s$hortest-first as follow. We omit case 1 because it will show the same result in all schedules. Tables show the time percentage saved using DA approach with $<_{DA}$ criteria comparing to the others.

Case 1: $l \in [0, 50], u = l * 1.25$			
Machines	DA/Fifo (%)	DA/U 1st (%)	DA/L 1st (%)
5	0.38	4.7	4.0
10	1.05	1.33	1.12
20	1.86	2.56	1.74
50	5.89	7.64	6.23
100	12.28	14.23	12.28

Case 2: $l \in [0, 50], u = l * 1.5$			
Machines	DA/Fifo (%)	DA/U 1st (%)	DA/L 1st (%)
5	0.24	0.27	0.20
10	0.61	0.70	0.60
20	1.55	1.55	1.56
50	4.73	4.31	4.81
100	8.27	8.22	8.19

5.2. Machines are different

We performed 10 tests on 1000 works for 3 cases and get the average. We compare result between FIFO queue, good schedule and DA schedule with dynamic re-arrangement works. This is the time consumed in each case.

6. CONCLUSION

We have used DAs to model and define the behavior of cluster systems, languages that accepted by the system and proposed algorithms to solve scheduling problem for uncertain precessing time works. In the future, we can develop algorithms to schedule programs with synchronization, shared actions, develop tools to schedule work for clusters using DA.

Case 1: $l \in [0, 50], l = u = t$			
Machines	Fifo schedule	Good schedule	DA schedule
5	4932.0	1937.0	1819.0
10	2489.0	597.0	544.0
20	1305.0	200.0	172.0
50	558.0	63.0	44.0
100	281.0	35.0	19.0

Case 2: $l \in [0, 50], u = l * 1.25$			
Machines	Fifo schedule	Good schedule	DA schedule
5	6597.1	4153.1	2927.3
10	3315.3	1100.2	763.3
20	1680.3	388.1	257.4
50	707.3	93.8	61.1
100	382.5	33.0	20.2

Case 3: $l \in [0, 50], u = l * 1.5$			
Machines	Fifo schedule	Good schedule	DA schedule
5	6956.8	5457.0	3443.7
10	3489.7	1469.5	929.4
20	1752.5	549.9	280.4
50	739.3	102.0	64.9
100	408.3	35.0	20.1

REFERENCES

[1] Bui Vu Anh, Nondeterministic duration automata in modeling priority network, *Proceeding of National Conference on Discovery Knowledge from Data*, Vietnam, 5-6, August 2009 (vol. 210156B00) 315-325.

[2] Dang Van Hung, Bui Vu Anh, Model checking component based systems with black-box testing, *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)*, 2005 (ISBN ĨSSN:1533-2306, 0-7695-2346-3).

[3] Michael Merritt, Time-Constrained automaton, *Proceedings of 2nd International Conference on Concurrency Theory (Concur'91) vol 527 of LNCS*, Springer-Verlag, Berlin, August 1991 (408-423).

[4] S.M. Johnson, *Optimal two- and three-stage production schedules with setup times included*, Naval Res. Log. Quart. I, 1954 (61-68).

[5] Bo Chen, and Arjen P. A. Vestjens, and Gerhard J. Woeginger, On-line scheduling of two-machine open shops where jobs arrive over time, *Journal of Combinatorial Optimization* **1** (1996) 355-365.

[6] John Noga and Steve Seiden, "Scheduling two machines with release times", 1998.

[7] Joachim Breit, "An improved approximation algorithm for two-machine flow shop scheduling with an availability constraint", 2004.

[8] Sanja Petrovic and Xueyan Song, *A new approach on two-machine flow shop problem with uncertain processing time*, University of Maryland, College Park, USA, 2003 (110–115).

[9] Rajeev. Alur and David L. Dill, A theory of Timed Automata, *Proceedings of 17th International Colloquium on Automata, Languages, and Programming 1990 and Proceedings of the REX workshop "Realtime: theory in practice" 1991; Theoretical Computer Science* **1**26 1994 (183–235).

[10] Rajeev. Alur and Thomas A. Henginger, A really temporal logic, *Proceedings of 30th IEEE Symposium on Foundation of Computer Science (FOCS 1989)*, pp. 164-169 (Extended version appeared in *The Journal of the ACM* **41** (1994) 181–204.

[11] Nancy Lynch and Mark Tuttle, An introduction to input/output automata, *CWI-Quarterly* **2** (3) (September 1989) 219–246.

[12] Dilsun K. Kaynar, Nancy Lynch, Roberto Segala, Frits Vaandrager *The Theory of Timed I/O Automata*, 2005.

[13] N. A. Lynch, R. Segala, and F. W. Vaandrager, Hybrid I/O automata, *Information and Computation* **1**85 (1) (2003) 105–157.

[14] M. Sorea, "Bounded model checking for timed automata", CSL Technical Report SRICSL-02-03, 2002.

[15] T. C. E. Cheng and V. S. Gordon, Batch delivery scheduling on a single machine, *Journal of the Operational Research Society* **4**5 (10) (1994) 1211–1215.

[16] J. M. Moore, An n jobs, one machine sequencing algorithm for minimizing the number of late jobs, *Management Sciences* **1**5 (1968) 102–109.

[17] V. A. Strusevich, A. J. A. Van De Waart, R. Dekker, A 3=2 algorithm for two-machine open shop with route-dependent processing times, *Journal of Heuristics* **5** (1999) 5–28.

Received on January 10 - 2011
Revised on September 15 - 2011