9

Parallel Application Scheduling Model Based on Petri Net with Changeable Structure[1]

Xiangang Zhao, Caiying Wei, Manyun Lin,
Xiaohu Feng and Wei Lan
National Satellite Meteorological Center
China

1. Introduction

In Parallel computing environments, each user can submit his job that is represented as a workflow composed of tasks that require multiple types of computational resources. How to develop a mechanism that ensures the success of these workflows is a challenging issue because the resources they use are dynamic and heterogeneous.

In order to schedule these workflows conveniently, a model is needed to describe them in a simple, intuitive way. Script-based method is a simple way to describe workflows. However, because those scripts often consist of so many elements with complex syntax that the users cannot understand them quickly. The graphic description for a workflow is an intuitive way, such as directed acyclic graph (DAG) and Petri Net. Compared to script-based descriptions, DAG is easier to use and more intuitional. However, DAG offers only a limited expressiveness [1], e.g. loops cannot be expressed directly. Moreover, as DAG only has a single node type, data flowing through the net cannot be modeled easily.

Petri net [2] is a modeling tool used for modeling discrete, dynamic, parallel and asynchronous system. Because of the function of simple graphical description and interpretation ability, Petri net is widely used for system modeling and performance analysis in recent years. Many researches have already introduced this method to model workflow [3-5].

In this paper, we model scheduling nets and job nets based on Petri Net techniques. To be convenient to analyze the performance of parallel jobs and to make net models compact and intuitional, we separate the scheduling net from the job net and model them respectively. A hierarchical colored Petri Net is proposed for the scheduling net that is designed into four levels according to the granularity of parallel applications. The hierarchical scheduling model makes each level scheduling only pay attention to its responsibility and it can reduce the structure complexity of the scheduling net at the same time. This paper also designs a extended Petri net with changeable structure for the job net model, which can change its structure dynamically according to the real-time state of running job. This model supports the mergence and division of subtasks and has ability to deal with the abnormity of subtasks. The models are validated with reachability tree techniques and their performances are analyzed with transition trees.

[1] This work is supported by the project of Research on Regional Observation Technology about Slow-roll Stabilization Satellite (GYHY201006046).

2. Related work

Jia Yu and Rajkumar Buyya et al[6-8] have done many researches on workflow in parallel environments. They propose a taxonomy that characterizes and classifies various approaches for building and executing workflows on Grids. They model workflow applications as a DAG and present many algorithms to address scheduling optimization problems in workflow applications based on QoS constraints. The emphases of their researches are the development of workflow management systems and scheduling algorithms.
BPEL4WS[9] builds on top of XML and web services specifications and provides a rich method for modeling web services based on the description of business process. However, its representation is script-based and it only composes workflows from web services.
Jin Hai et al[10] propose a workflow model based on colored Petri net for grid service composition, in which the image date transmission was taken as requirements for the service flow to improve the efficiency of settling service flow and reduce tasks' execution time. However, the model does not take the failure of task into consideration and does not support dynamic structures.
A general scheduling framework[11] modeled by Petri net is proposed, which locates on the layer of Grid scheduler and is used for independent tasks in computational Grid.
A three-level scheduling scheme[12] is proposed based on a high-level timed Petri net. The scheme divides Grid scheduling into three levels: Grid scheduler, Local Scheduler and Home Scheduler. It constructs different Petri net models for these levels. However, this scheme only focuses on independent tasks. In order to deal with the scheduling problem of task that consists of a set of communicating subtasks, an extended timed Petri net model[3] is proposed. Based on composition and reduction of Petri nets, the model can reduce the complexity of model and solve the state explosion problem in reachability analysis of Petri nets. But this model does not concern about the abnormity of running tasks and has no ability to change structure dynamically.

3. Definitions of extended Petri Nets

The jobs are dynamic and hierarchical in a parallel environment. According to these characteristics, we design two types of enhanced Petri Net, which are extended from the original Petri Net.
Definition 1 A Hierarchical Color Petri Net (HCPN) is designed into 9-tuple.

$$HCPN = \{P,T;F,D,C,I,O,K,M_0\}$$

1. P is a finite set of places.
2. T is a finite set of transitions and $T = \{T_s \cup T_c\}$, where T_s is a set of simple transitions, T_c is a set of complex transitions and $(P \cup T \neq \varnothing)\Lambda(P \cap T = \varnothing)$.
3. F is a finite set of arcs and $F \subseteq (P \times S) \cup ((S \times P))$.
4. D is a finite set of colors.
5. C is a finite set of color functions. $C : P \cup T \rightarrow \psi(D)$, where $\psi(D)$ is the power set of colors.
6. I and O are the input and output arc functions respectively.

$\forall(p,t) \in (P \times T) \Rightarrow I(p,t) \in \left[C(p)_{MS} \to C(t)_{MS}\right]_L$

$\forall(t,p) \in (T \times P) \Rightarrow O(t,p) \in \left[C(t)_{MS} \to C(p)_{MS}\right]_L$

When a transition needs to consume all tokens in a place, the input arc function is $I(p,t) = \gamma C(p)$.

7. K is a set of capacity functions. $K : P \to N \cup \omega$, $N = \{1,2,3,\cdots\}$ and ω denotes infinite.
8. $M_0 : P \to D_{MS}$ is the initial token marking. $\forall p \in P : M_0(p) \in C(p)_{MS}$, where $C(p)_{MS}$ is the multiple set of the color tokens in p.

Definition 2

1. ${}^{\bullet}p = \{< t,p >| t \in T\}$, $p^{\bullet} = \{< p,t >| t \in T\}$,

   ${}^{\bullet}p^{\bullet} = \{{}^{\bullet}p \cup p^{\bullet}\}$.
2. ${}^{\otimes}p = \{t |< t,p >\in F\}$, $p^{\otimes} = \{t |< p,t >\in F\}$.
3. ${}^{\otimes}t = \{p |< p,t >\in F\}$, $t^{\otimes} = \{p |< t,p >\in F\}$.
4. ${}^{\otimes\otimes}t = \{t' \mid p \in {}^{\otimes}t \wedge < t',p >\in F\}$,

   $t^{\otimes\otimes} = \{t' \mid p \in t^{\otimes} \wedge < p,t' >\in F\}$.
5. In $HCPN$, the firing rules of transition is

   $\forall p \in {}^{\otimes}t : M(p) \geq O(p,t) \wedge \forall p \in t^{\otimes}$:

   $M(p) + I(p,t) \leq K(p)$.
6. There exist only one p_k and one p_l in $HCPN$, which satisfy the condition: ${}^{\otimes}p_k = \varnothing \wedge p_l^{\otimes} = \varnothing \wedge p_k \neq p_l$. p_k and p_l are called Beginning Place and End Place of $HCPN$ respectively.

Definition 3

Any complex transition in $HCPN$ can be extended to a subnet. The subnet of complex transition t_i is defined as:

$S\_HCPN_i = \{P_i, T_i; F_i, D_i, C_i, I_i, O_i, K_i, M_0^i\}$.

1. $P_i = \{p_{begin}^i, p_{end}^i, P_i'\}$ is a set of places. P_i' is the set of inner places in S_HCPN_i. p_{begin}^i and p_{end}^i are additional places used to denote the beginning and end places of S_HCPN_i.

   ${}^{\bullet}p_{begin}^i = \varnothing$, $(p_{end}^i)^{\bullet} = \varnothing$;

   $C(p_{begin}^i) = \{\cup_{k=1}^m C(p_k) \mid p_k \in {}^{\otimes}t_i\}$;

   $C(p_{end}^i) = \{\cup_{k=1}^m C(p_k) \mid p_k \in t_i^{\otimes}\}$.
2. $F_i = \{(p_{begin}^i)^{\bullet} \cup {}^{\bullet}p_{end}^i \cup F_i'\}$ is a set of arcs. F_i' is the set of inner arcs in S_HCPN_i.
3. T_i, D_i, C_i, I_i, O_i, K_i and M_0^i are the sets of transitions, colors, color functions, input arc functions, output arc functions, capacity functions and initial token marking respectively.

Definition 4

A Petri Net with changeable structure is designed to11-tuple.

$$CSCTPN_k = \left\{P_k, T_k; F_k, D_k, C_k, I_k, O_k, \Gamma_k, Q_k, M_0^k, M_0^{k-1}\right\}$$

4. P_k, T_k, F_k, D_k, C_k, I_k, and O_k are the sets of places, transitions, arcs, colors, color functions, input arc functions and output arc functions respectively after the structure of $CSCTPN$ is changed k times.
5. 2) Γ_k is a set of times after $CSCTPN$ have changed k times. Its element is defined as $< t_i, \tau_b, \tau_l, \tau_d >$, which denotes the earliest start time, the latest start time and duration time of transition t_i are τ_b, τ_l, and τ_d respectively. $\Gamma_b(t_i) = \tau_b$, $\Gamma_l(t_i) = \tau_l$, $\Gamma_d(t_i) = \tau_d$.
6. Q_k is a set of cost after $CSCTPN$ have changed k times. Its element is defined as $< t_i, q_r, q_m >$, which denotes that the maximal cost of transition t_i is q_m and the real cost is q_r. $Q_r(t_i) = q_r$, $Q_m(t_i) = q_m$, and $q_r \le q_m$
7. M_0^k is the initial token marking after $CSCTPN$ have changed k times and $M_0^{-1} = M_0^0$.

4. Parallel application scheduling model

In this section, we propose a four-level scheduling model firstly according to the characteristics of parallel jobs. Then, Parallel job net is designed based on Petri Net with changeable structure and the conversion rules of the job net are presented at the same time.

4.1 Four-level scheduling net

In a Parallel environment, users use resources by submitting their applications. A user application is called a parallel job that can implement some functions specifically. A parallel job is usually composed of many steps and each step has certain input and output sets. Each step is called a subjob that can be divided into two types: computing subjob and data transferring subjob. A computing subjob needs to transfer its inputs firstly and then perform computing operation, so a computing subjob can be divided into transferring tasks and computing tasks. Similarly, a data transferring subjob often has many data inputs and it can be divided into many transferring tasks. A data transferring task has only one input and one output. The input data of a computing task is already transferred to local computing node. Because a data resource may have many replicas that locate on different nodes, to speed up the transfer a data transferring task can be divided into many subtasks according to the number of replicas and the QoS requirements of the user. Each subtask transfers a part of data from different replicas. If a computing task can be processed in parallel we call it a parallel computing task, otherwise we call it an unparallel computing task. A parallel computing task can be divided into many subtasks that run on different computing node. According to job, subjob, task and subtask, the parallel allocation scheduling model is designed into four levels: job scheduling net, subjob scheduling net, task scheduling net and subtask scheduling net. Only subtasks use computing or data resources directly, so all resource allocations take place in subtask scheduling net.

4.1.1 Job scheduling net.

The job scheduling net mainly manages the states of jobs. Its function includes job selection and monitoring. There are four states of a job: waiting, running, completed and failed. When all subjobs of a job are completed, the job is completed. If any subjob failes, the state of the job is failed. When a job has running subjobs and has no failed subjobs, the state of the job is running. The job scheduling net is modeled based on *HCPN*, which is shown as Fig.4-1. The detailed definition is shown as follows:

$HCPN = \{P,T;F,D,C,I,O,K,M_0\}$

1. $P = \{p_i \mid 1 \le i \le 7\}$;
2. $T = \{t_i \mid 1 \le i \le 7\}$, t_1 : select a job for running; t_2 : start a job; t_3 : select a job for monitoring; t_4 : check the states of jobs, which is a complex transition; t_5 : mark a job; t_6 : return a completed job; t_7 : return a failed job.
3. $D = \{d_i \mid 1 \le i \le 8\}$, d_1 : jobs submitted by users; d_2 : jobs waiting to be started; d_3 : running jobs; d_4 : jobs waiting for being checked; d_5 : jobs validated to run normally; d_6 : completed jobs; d_7 :failed jobs; d_8 : returned jobs; d_9 : tokens used for restricting the number of jobs that are running at the same time.
4. $C(p_1) = \{d_1\}$, $C(p_2) = \{d_2\}$, $C(p_3) = \{d_3\}$, $C(p_4) = \{d_4\}$, $C(p_5) = \{d_5, d_6, d_7\}$, $C(p_6) = \{d_9\}$, $C(p_7) = \{d_8\}$; $C(t_1) = \{d_1\}$, $C(t_2) = \{d_2, d_9\}$, $C(t_3) = \{d_3\}$, $C(t_4) = \{d_4\}$, $C(t_5) = \{d_5\}$, $C(t_6) = \{d_6\}$, $C(t_7) = \{d_7\}$.
5. $K(p_1) = K(p_2) = m$, $K(p_3) = n$, $K(p_4) = K(p_5) = 1$, $K(p_6) = n$, $K(p_7) = \omega$. $M_0 = \{k,0,0,0,0,n,0\}$, where $1 \le k \le m$.

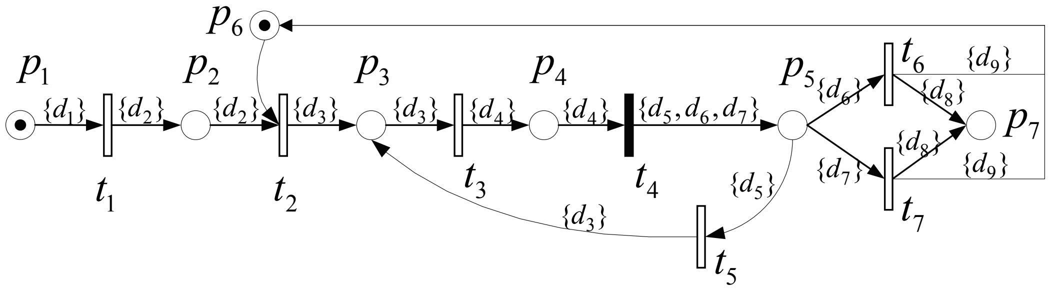

Fig. 4.1. Job Scheduling Net

4.1.2 Subjob scheduling net.

The subjob scheduling net is a subnet of the job scheduling net, which is extended from the complex transition t_4. The subjob scheduling net mainly manages the states of subjob and is used to analyze jobs, create subjobs, order the running sequence of subjobs and monitor the states of subjobs. The subjob scheduling net is shown as Fig.4-2, which is modeled based on *S_HCPN* and the detailed definition is shown as follows:

$$S\_HCPN_4 = \{P_4, T_4; F_4, D_4, C_4, I_4, O_4, K_4, M_0^4\}$$

1. $P_4 = \{p_{begin}^4, p_{end}^4, p_i^4 \mid 1 \le i \le 10\}$.
2. $T_4 = \{t_i^4 \mid 1 \le i \le 15\}$, t_1^4 : check whether a job is initialized; t_2^4 : analyze a job net; t_3^4 : get the set of running subjobs; t_4^4 : start subjobs; t_5^4 : select a subjob for monitoring; t_6^4 : monitor subjobs, which is a complex transition; t_7^4 : creat a token for clearing out all subjobs; t_8^4 : mark a normal subjob; t_9^4 : mark a completed subjob; t_{10}^4 : check whether all subjobs of the job have already been checked in this scheduling round; t_{11}^4 : creat a token for selecting a subjob; t_{12}^4 : mark a failed job; t_{13}^4 : check whether all subjobs of the job have already accomplished. t_{14}^4 : mark a completed job; t_{15}^4 : mark a normal job.

3. $D_4=\{C(p^4_{begin})\cup C(p^4_{end})\cup d^4\}$, where
$d^4=\{d^4_i \mid 1\le i\le 15\}$, d^4_1: initialized jobs; d^4_2: uninitialized jobs; d^4_3: subjob nets; d^4_4:running subjobs; d^4_5: subjobs waiting to be checked; d^4_6: subjobs running normally; d^4_7: completed subjobs; d^4_8: failed subjobs; d^4_9: marked subjobs; d^4_{10}: jobs whose subjobs have been checked completely in this scheduling round; d^4_{11}: jobs whose subjobs have been checked incompletely in this scheduling round; d^4_{12}: tokens for selecting a subjob; d^4_{13}: tokens for clearing out all subjobs; d^4_{14}: jobs whose subjobs have accomplished completely; d^4_{15}: jobs whose subjobs have accomplished incompletely.
4. $C(p^4_{begin})=\{d_4\}$, $C(p^4_{end})=\{d_5,d_6,d_7\}$,
$C(p^4_1)=\{d^4_1,d^4_2\}$, $C(p^4_2)=\{d^4_3\}$, $C(p^4_3)=\{d^4_4\}$, $C(p^4_4)=\{d^4_5\}$, $C(p^4_5)=\{d^4_6,d^4_7,d^4_8\}$, $C(p^4_6)=\{d^4_9\}$, $C(p^4_7)=\{d^4_{10},d^4_{11}\}$, $C(p^4_8)=\{d^4_{12}\}$, $C(p^4_9)=\{d^4_{13}\}$, $C(p^4_{10})=\{d^4_{14},d^4_{15}\}$; $C(t^4_1)=\{d_4\}$, $C(t^4_2)=\{d^4_2\}$, $C(t^4_3)=\{d^4_1\}$, $C(t^4_4)=\{d^4_3\}$, $C(t^4_5)=\{d^4_4,d^4_{12}\}$, $C(t^4_6)=\{d^4_5\}$, $C(t^4_7)=\{d^4_8\}$, $C(t^4_8)=\{d^4_6\}$, $C(t^4_9)=\{d^4_7\}$, $C(t^4_{10})=\{d^4_9\}$, $C(t^4_{11})=\{d^4_{11}\}$, $C(t^4_{12})=\{d^4_4,d^4_{13}\}$ $C(t^4_{13})=\{d^4_{10}\}$, $C(t^4_{14})=\{d^4_{14}\}$, $C(t^4_{15})=\{d^4_{15}\}$.
5. $K(p^4_3)=n$ and others are 1.
6. $M^4_0=\{1,0,0,0,0,0,0,0,0,0,0,0\}$.

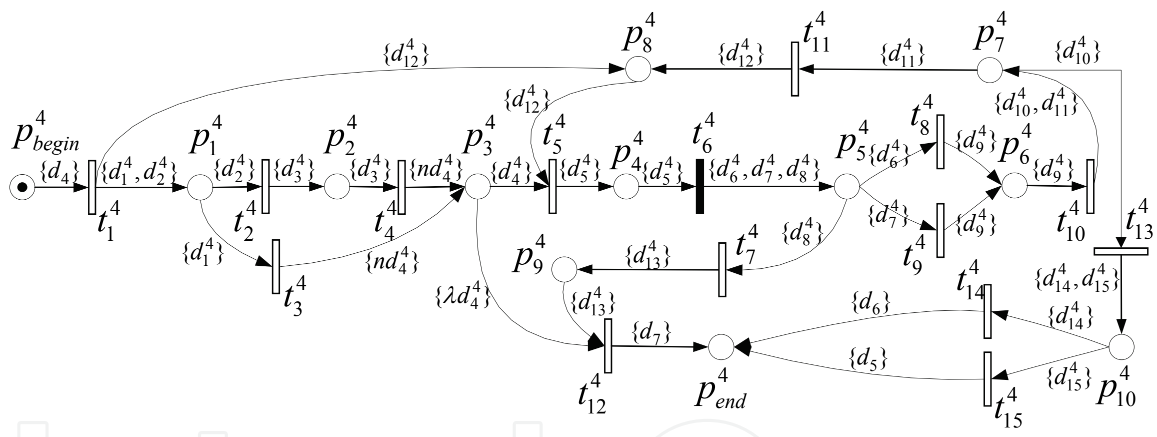

Fig. 4.2. Subjob Scheduling Net

4.1.3 Task scheduling net.

The task scheduling net is a subnet of the subjob scheduling net, which is extended from the complex transition t^4_6. The task scheduling net mainly manages the states of tasks and is used to analyze subjobs, create tasks, order the running sequence of task and monitor the states of tasks. A task scheduling net is shown as Fig. 4-3, which is modeled based on *S_HCPN* and the detailed definition is shown as follows:

$$S\_HCPN_{4,6}=\left\{P_{4,6},T_{4,6};F_{4,6},D_{4,6},C_{4,6},I_{4,6},O_{4,6},K_{4,6},M^{4,6}_0\right\}$$

1. $P_{4,6}=\{p^{4,6}_{begin},p^{4,6}_{end},p^{4,6}_i \mid 1\le i\le 10\}$.

2. $T_{4,6} = \{t_i^{4,6} \mid 1 \le i \le 15\}$, $T_4 = \{t_i^4 \mid 1 \le i \le 15\}$, $t_1^{4,6}$: check whether a subjob is initialized; $t_2^{4,6}$: transform a subjob scheduling net into a job scheduling net; $t_3^{4,6}$: get the set of running tasks; $t_4^{4,6}$: start tasks; $t_5^{4,6}$: select a task for monitoring; $t_6^{4,6}$: monitor tasks, which is a complex transition; $t_7^{4,6}$: create a token for clearing out all tasks; $t_8^{4,6}$: mark a normal task; $t_9^{4,6}$: mark a completed task; $t_{10}^{4,6}$: check whether all tasks of the subjob have already been checked in this scheduling round; $t_{11}^{4,6}$: create a token for selecting a task; $t_{12}^{4,6}$: mark a failed task; $t_{13}^{4,6}$: check whether all tasks of the subjob have already accomplished. $t_{14}^{4,6}$: mark a completed subjob; $t_{15}^{4,6}$: mark a normal subjob.
3. $D_{4,6} = \{C(p_{begin}^{4,6}) \cup C(p_{end}^{4,6}) \cup d^{4,6}\}$, where
   $d^{4,6} = \{d_i^{4,6} \mid 1 \le i \le 15\}$, $d_1^{4,6}$: initialized subjobs; $d_2^{4,6}$:uninitialized subjobs; $d_3^{4,6}$: task net; $d_4^{4,6}$: running tasks; $d_5^{4,6}$: tasks waiting to be checked; $d_6^{4,6}$: tasks running normally; $d_7^{4,6}$: completed tasks; $d_8^{4,6}$: failed tasks; $d_9^{4,6}$: marked tasks; $d_{10}^{4,6}$: subjobs whose tasks have been checked completely in this scheduling round; $d_{11}^{4,6}$: subjobs whose tasks have been checked incompletely in this scheduling round; $d_{12}^{4,6}$: tokens for selecting a task; $d_{13}^{4,6}$: tokens for clearing out all tasks; $d_{14}^{4,6}$: subjobs whose tasks have accomplished completely; $d_{15}^{4,6}$: subjobs whose tasks have accomplished incompletely.
4. $C(p_{begin}^{4,6}) = \{d_5^4\}$, $C(p_{end}^{4,6}) = \{d_6^4, d_7^4, d_8^4\}$, $C(p_1^{4,6}) = \{d_1^{4,6}, d_2^{4,6}\}$, $C(p_2^{4,6}) = \{d_3^{4,6}\}$,
   $C(p_3^{4,6}) = \{d_4^{4,6}\}$, $C(p_4^{4,6}) = \{d_5^{4,6}\}$, $C(p_5^{4,6}) = \{d_6^{4,6}, d_7^{4,6}, d_8^{4,6}\}$, $C(p_6^{4,6}) = \{d_9^{4,6}\}$,
   $C(p_7^{4,6}) = \{d_{10}^{4,6}, d_{11}^{4,6}\}$, $C(p_8^{4,6}) = \{d_{12}^{4,6}\}$, $C(p_9^{4,6}) = \{d_{13}^{4,6}\}$, $C(p_{10}^{4,6}) = \{d_{14}^{4,6}, d_{15}^{4,6}\}$;
   $C(t_1^{4,6}) = \{d_5^4\}$, $C(t_2^{4,6}) = \{d_2^{4,6}\}$, $C(t_3^{4,6}) = \{d_1^{4,6}\}$, $C(t_4^{4,6}) = \{d_3^{4,6}\}$, $C(t_5^{4,6}) = \{d_4^{4,6}, d_{12}^{4,6}\}$,
   $C(t_6^{4,6}) = \{d_5^{4,6}\}$, $C(t_7^{4,6}) = \{d_8^{4,6}\}$, $C(t_8^{4,6}) = \{d_6^{4,6}\}$, $C(t_9^{4,6}) = \{d_7^{4,6}\}$, $C(t_{10}^{4,6}) = \{d_9^{4,6}\}$,
   $C(t_{11}^{4,6}) = \{d_{11}^{4,6}\}$, $C(t_{12}^{4,6}) = \{d_4^{4,6}, d_{13}^{4,6}\}$, $C(t_{13}^{4,6}) = \{d_{10}^{4,6}\}$, $C(t_{14}^{4,6}) = \{d_{14}^{4,6}\}$, $C(t_{15}^{4,6}) = \{d_{15}^{4,6}\}$.
5. $K(p_3^{4,6}) = n$ and others are 1.
6. $M_0^{4,6} = \{1,0,0,0,0,0,0,0,0,0,0,0\}$.

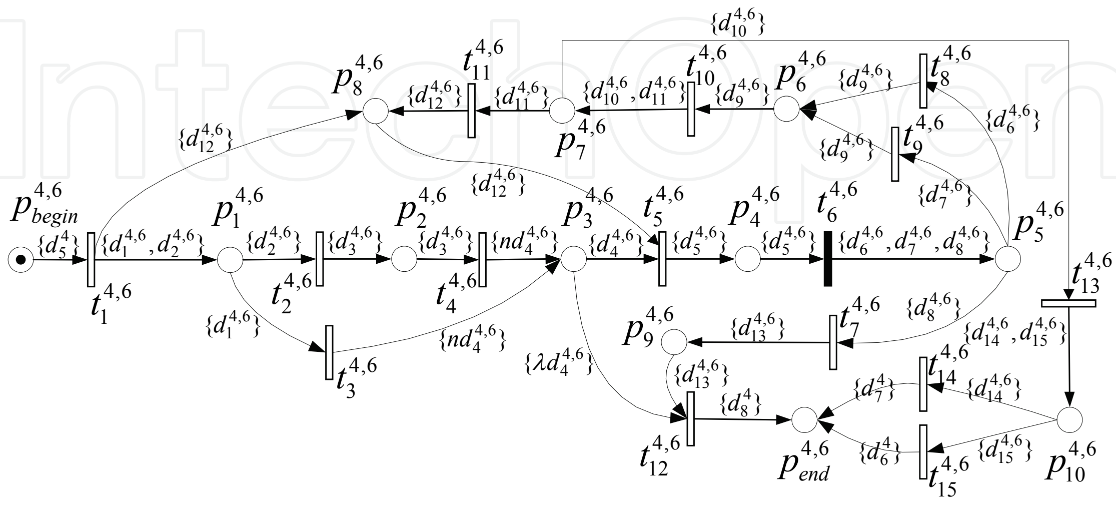

Fig. 4.3. Task Scheduling Net

4.1.4 Subtask scheduling net.

The subtask scheduling net is a subnet of the task scheduling net, which is extended from the complex transition $t_6^{4,6}$. The subtask scheduling net mainly manages the states of subtasks and is used to analyze tasks, create subtasks, allocate and reallocate resources, order the running sequence of subtask and monitor the states of subtasks. The subtask scheduling net is shows as Fig.4-4, which is modeled based on *S_HCPN* and the detailed definition is shown as follows:

$$S\_HCPN_{4,6,6} = \left\{P_{4,6,6}, T_{4,6,6}; F_{4,6,6}, D_{4,6,6}, C_{4,6,6}, I_{4,6,6}, O_{4,6,6}, K_{4,6,6}, M_0^{4,6,6}\right\}$$

1. $P_{4,6,6} = \{p_{begin}^{4,6,6}, p_{end}^{4,6,6}, p_i^{4,6,6} \mid 1 \le i \le 14\}$.
2. $T_{4,4,6} = \{t_i^{4,4,6} \mid 1 \le i \le 21\}$, $t_1^{4,4,6}$: check whether a task is initialized; $t_2^{4,4,6}$: get the set of running subtasks; $t_3^{4,4,6}$: search resources; $t_4^{4,4,6}$: select resources; $t_5^{4,4,6}$: create subtasks; $t_6^{4,4,6}$: start subtasks; $t_7^{4,4,6}$: select a subtask for monitoring; $t_8^{4,4,6}$: check a subtask; $t_9^{4,4,6}$: mark a completed subtask; $t_{10}^{4,4,6}$: mark a normal subtask; $t_{11}^{4,4,6}$: check whether all subtasks of the task have already been checked in this scheduling round; $t_{12}^{4,4,6}$: create a token for selecting a subtask; $t_{13}^{4,4,6}$: reallocate a subtask; $t_{14}^{4,4,6}$: start subtasks after reallocation; $t_{15}^{4,4,6}$: mark subtasks running normally after reallocation; $t_{16}^{4,4,6}$: create a token for clearing out all subtasks; $t_{17}^{4,4,6}$: mark a failed subtask; $t_{18}^{4,4,6}$: check whether all subtasks of the task have already accomplished. $t_{19}^{4,4,6}$: mark a normal task; $t_{20}^{4,4,6}$: mark a completed task; $t_{21}^{4,4,6}$: mark a failed task.
3. $D_{4,4,6} = \{C(p_{begin}^{4,4,6}) \cup C(p_{end}^{4,4,6}) \cup d^{4,4,6}\}$, where
   $d^{4,4,6} = \{d_i^{4,4,6} \mid 1 \le i \le 15\}$, $d_1^{4,6,6}$: uninitialized tasks; $d_2^{4,6,6}$: initialized tasks; $d_3^{4,6,6}$: running subtasks; $d_4^{4,6,6}$: resource list; $d_5^{4,6,6}$: selected resources; $d_6^{4,6,6}$: subtasks that have no inadequate resources; $d_7^{4,6,6}$: subtask net; $d_8^{4,6,6}$: tokens for selecting a subtask; $d_9^{4,6,6}$: subtasks waiting for monitoring; $d_{10}^{4,6,6}$: subtasks running normally; $d_{11}^{4,6,6}$: completed subtasks; $d_{12}^{4,6,6}$: subtasks running abnormally; $d_{13}^{4,6,6}$: marked subtasks; $d_{14}^{4,6,6}$: tasks whose subtasks have not been checked completely in this scheduling round; $d_{15}^{4,6,6}$: tasks whose subtasks have been checked completely in this scheduling round; $d_{16}^{4,6,6}$: tasks whose subtasks have already been accomplished; $d_{17}^{4,6,6}$: tasks whose subtasks have not been accomplished completely; $d_{18}^{4,6,6}$: subtasks reallocated successfully; $d_{19}^{4,6,6}$: subtasks reallocated unsuccessfully; $d_{20}^{4,6,6}$: subtasks running normally after reallocation; $d_{21}^{4,6,6}$: tokens for clearing out all subtasks.
4. $C(p_{begin}^{4,6,6}) = \{d_5^{4,6}\}$, $C(p_{end}^{4,6,6}) = \{d_6^{4,6}, d_7^{4,6}, d_8^{4,6}\}$, $C(p_1^{4,6,6}) = \{d_1^{4,6,6}, d_2^{4,6,6}\}$, $C(p_2^{4,6,6}) = \{d_3^{4,6,6}\}$, $C(p_3^{4,6,6}) = \{d_4^{4,6,6}\}$, $C(p_4^{4,6,6}) = \{d_5^{4,6,6}, d_6^{4,6,6}\}$, $C(p_5^{4,6,6}) = \{d_7^{4,6,6}\}$, $C(p_6^{4,6,6}) = \{d_9^{4,6,6}\}$, $C(p_7^{4,6,6}) = \{d_{10}^{4,6,6}, d_{11}^{4,6,6}, d_{12}^{4,6,6}\}$, $C(p_8^{4,6,6}) = \{d_{13}^{4,6,6}\}$, $C(p_9^{4,6,6}) = \{d_{14}^{4,6,6}, d_{15}^{4,6,6}\}$, $C(p_{10}^{4,6,6}) = \{d_8^{4,6,6}\}$, $C(p_{11}^{4,6,6}) = \{d_{18}^{4,6,6}, d_{19}^{4,6,6}\}$, $C(p_{12}^{4,6,6}) = \{d_{21}^{4,6,6}\}$, $C(p_{13}^{4,6,6}) = \{d_{20}^{4,6,6}\}$, $C(p_{14}^{4,6,6}) = \{d_{16}^{4,6,6}, d_{17}^{4,6,6}\}$;
   $C(t_1^{4,6,6}) = \{d_5^{4,6}\}$, $C(t_2^{4,6,6}) = \{d_2^{4,6,6}\}$, $C(t_3^{4,6,6}) = \{d_1^{4,6,6}\}$, $C(t_4^{4,6,6}) = \{d_4^{4,6,6}\}$, $C(t_5^{4,6,6}) = \{d_5^{4,6,6}\}$, $C(t_6^{4,6,6}) = \{d_7^{4,6,6}\}$, $C(t_7^{4,6,6}) = \{d_3^{4,6,6}, d_8^{4,6,6}\}$, $C(t_8^{4,6,6}) = \{d_9^{4,6,6}\}$,

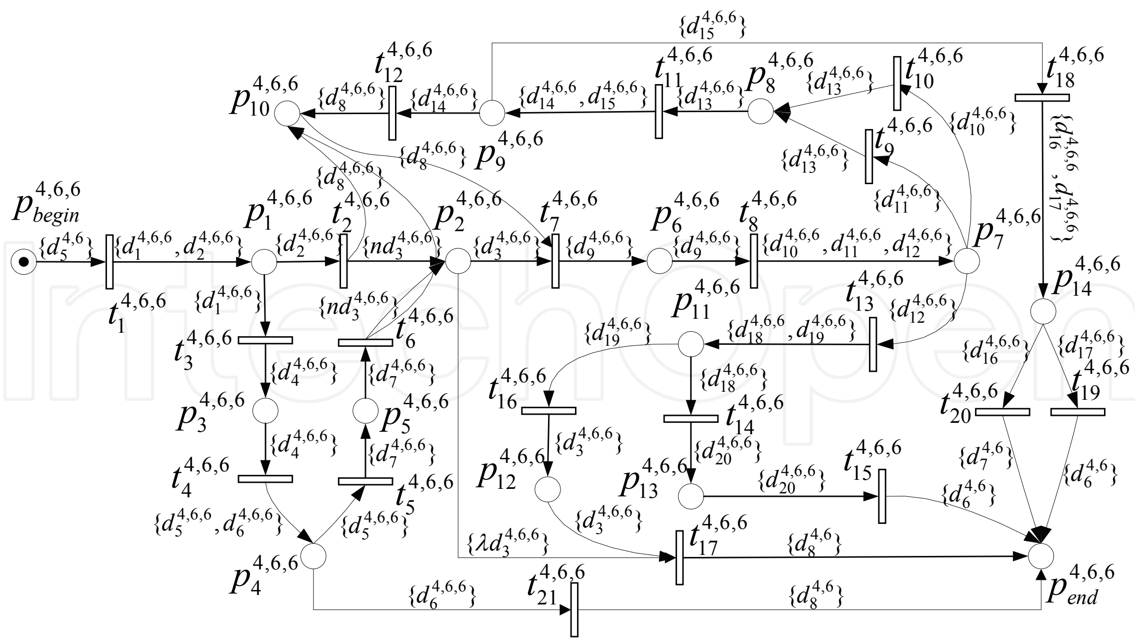

Fig. 4.4. Subtask Scheduling Net

$C(t_9^{4,6,6}) = \{d_{11}^{4,6,6}\}$, $C(t_{10}^{4,6,6}) = \{d_{10}^{4,6,6}\}$, $C(t_{11}^{4,6,6}) = \{d_{13}^{4,6,6}\}$, $C(t_{12}^{4,6,6}) = \{d_{14}^{4,6,6}\}$, $C(t_{13}^{4,6,6}) = \{d_{12}^{4,6,6}\}$, $C(t_{14}^{4,6,6}) = \{d_{18}^{4,6,6}\}$, $C(t_{15}^{4,6,6}) = \{d_{20}^{4,6,6}\}$, $C(t_{16}^{4,6,6}) = \{d_{19}^{4,6,6}\}$, $C(t_{17}^{4,6,6}) = \{\lambda d_3^{4,6,6}, d_{21}^{4,6,6}\}$, $C(t_{18}^{4,6,6}) = \{d_{15}^{4,6,6}\}$, $C(t_{19}^{4,6,6}) = \{d_{17}^{4,6,6}\}$, $C(t_{20}^{4,6,6}) = \{d_{16}^{4,6,6}\}$, $C(t_{21}^{4,6,6}) = \{d_6^{4,6,6}\}$.

5. $K(p_2^{4,6,6}) = n$ and others are 1.
6. $M_0^{4,6,6} = \{1,0,0,0,0,0,0,0,0,0,0,0\}$.

4.2 Job Net

The job net describes the flow of parallel application submitted by users, which is a kind of workflow net and defines the relation of each step strictly. The job net is modeled based on $CSCTPN$ and its detailed definition is shows as follows:

$$CSCTPN_k = \left\{P_k, T_k; F_k, D_k, C_k, I_k, O_k, \Gamma_k, Q_k, M_0^k, M_0^{k-1}\right\}$$

$CSCTPN_0$ is the initial structure of the job net. In this level, we only concern about the time limit, cost, input and output of a job, which are defined as follows:

1. $P_0 = \{p_{in}, p_{out}\}$, they are places of input and output respectively;
2. $T_0 = \{t_0^1\}$, there is only a transition that denotes the whole process of the job;
3. $D_0 = \{d_{in}, d_{out}\}$, they are the input and output of the job;
4. $\Gamma_0 = \{< t_0^1, \tau_b, \tau_l, \tau_d >\}$, it denotes the time limit of t_0^1;
5. $Q_0 = \{< t_0^1, q_r, q_m >\}$, it denotes the cost limit of t_0^1.

There are four types of data in the job net: remote data, local data, outer data and inner data. For a job, outer data has already existed in the parallel environment and it is not produced by the job. The data produced temporarily by the jobs is called inner data.

In the job net, a user needs to indicate the maximum cost and the deadline of his job. In addition, the user needs to estimate the cost and durable time of each subjob according to his experiences, which are shown in the description file of the job. This is helpful to assign the cost and time characteristics of subjobs. Otherwise, the assignment methods of these values are the same with tasks and subtasks. For tasks and subtasks, we need to assign their cost and time characteristics dynamically within their limits according to the allocation results. The cost is proportional to the transferring traffic and the computing load [13,14] in our assignation strategy. Compared with cost assignation, time assignation is more complex than cost assignation. We refer to a method[15] to assign the times of tasks and subtasks within fixed-time constraints.

4.2.1 Subjob net.

A job consists of many subjobs that have own inputs, outputs and operations. The subjob net is $CSCTPN_1$ that is built by analyzing the description file of a job.

$$CSCTPN_1 = \left\{P_1, T_1; F_1, D_1, C_1, I_1, O_1, \Gamma_1, Q_1, M_0^1, M_0^0\right\}$$

1. P_1 : a set of places for inputs and outputs of subjobs.
2. $T_1 = J_t \cup J_c$, where J_t is a set of transferring subjobs and J_c is a set of computing subjobs.
3. $D_1 = \{d_i \mid 1 \le i \le 4\}$, where d_1 : remote data from outside; d_2 : local data from outside; d_3 : remote data from inside; d_4 : local data from inside.
4. Γ_1 : a time set of subjobs, $\forall t_1^i \in T_1(\Gamma_l(t_1^i) + \Gamma_d(t_1^i) \le \Gamma_l(t_0^1) + \Gamma_d(t_0^1))$.
5. Q_1 :a cost set of subjobs, $\sum_{i=1}^{m} Q_m(t_1^i) = Q_m(t_0^1)$, where m is the number of transitions in T_1 .

4.2.2 Task net.

The task net is $CSCTPN_2$ that is built by decomposing the subjob net. $CSCTPN_2$ is defined as follows.

$$CSCTPN_2 = \left\{P_2, T_2; F_2, D_2, C_2, I_2, O_2, \Gamma_2, Q_2, M_0^2, M_0^1\right\}$$

1. P_2 : a set of places for inputs and outputs of tasks.
2. $T_2 = T_t \cup T_{npc} \cup T_{dpc} \cup T_{dnpc}$, where T_t : a set of data transferring tasks; T_{npc} : a set of computing tasks that can not run in parallel; T_{dpc} : a set of computing tasks whose data inputs can be divided; T_{ndpc} : a set of computing tasks whose data input can not be divided.
3. $D_2 = \{d_i \mid 1 \le i \le 4\}$, where d_1 : remote data from outside; d_2 : local data from outside; d_3 : remote data from inside; d_4 : local data from inside.
4. Γ_2 : a time set of tasks, $\forall t_2^i \in T_2(\Gamma_l(t_2^i) + \Gamma_d(t_2^i) \le \Gamma_l(t_0^1) + \Gamma_d(t_0^1))$.
5. Q_2 : a cost set of tasks, $\sum_{i=1}^{m} Q_m(t_2^i) = Q_m(t_0^1)$, where m is the number of transitions in T_2 .

According to the input and output, a subjob can be divided into several tasks. Based on the types of subjobs, the division rules are defined as follows.

1. data transferring subjob

A data transferring subjob often has many data inputs and they may need to be transferred at the same time. In order to be convenient to deal with them, a data transferring subjob needs to be divided into many tasks and each task has only one data input. The division result is shown as Fig.4-5. The process satifies these conditions:

$$\sum_{r=1}^{s+k} Q_m(t_2^{i,r}) = Q_m(t_1^i),$$

$$\exists h(1 \le h \le (s+k) \wedge \Gamma_b(t_2^{i,h}) \le \Gamma_l(t_1^i))$$

$$\forall h(1 \le h \le (s+k) \to \Gamma_b(t_2^{i,h}) \ge \Gamma_b(t_1^i))$$

$$\max(\Gamma_l(t_2^{i,1}) + \Gamma_d(t_2^{i,1}), \cdots, \Gamma_l(t_2^{i,s+k}) + \Gamma_d(t_2^{i,s+k})) \\ \le \Gamma_l(t_1^i) + \Gamma_d(t_1^i))$$

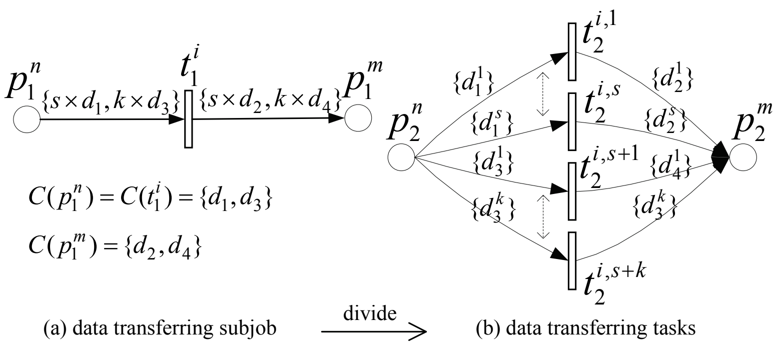

Fig. 4.5. Division result of data transferring subjob

2. computing subjob

Generally, a computing subjob has remote data inputs and these data need to be transferred to local node firstly. The division result for a computing subjob is show as Fig.4-6 and it satisfies these conditions:

$$\sum_{r=1}^{s+k} Q_m(t_2^{i,r}) + Q_m(t_2^i) = Q_m(t_1^i)$$

$$\exists h(1 \le h \le (s+k) \wedge \Gamma_b(t_2^{i,h}) \le \Gamma_l(t_1^i))$$

$$\forall h(1 \le h \le (s+k) \to \Gamma_b(t_2^{i,h}) \ge \Gamma_b(t_1^i))$$

$$\Gamma_b(t_2^i) \le \max(\Gamma_l(t_2^{i,1}) + \Gamma_d(t_2^{i,1}), \cdots, \Gamma_l(t_2^{i,s+k}) + \Gamma_d(t_2^{i,s+k})) \le \Gamma_l(t_2^i)$$

$$\Gamma_l(t_2^i) + \Gamma_d(t_2^i) \le \Gamma_l(t_1^i) + \Gamma_d(t_1^i)$$

4.2.3 Subtask net.

Subtask net is $CSCTPN_i$ ($i \ge 3$)that is built by decomposing task net, which is defined as follows.

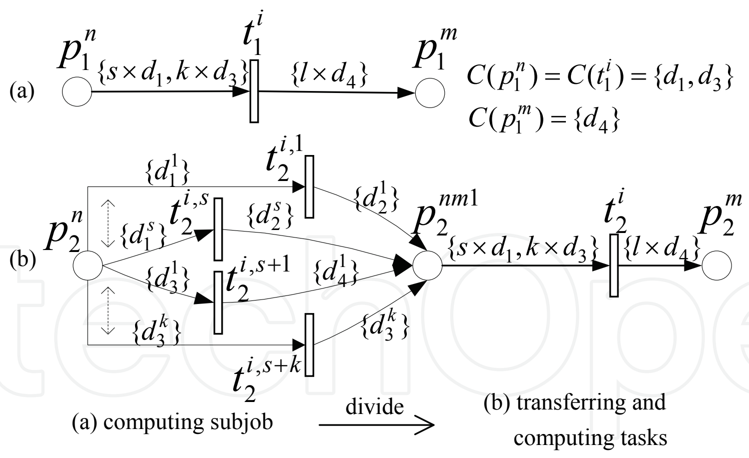

Fig. 4.6. Division result of computing subjob

$$CSCTPN_i = \left\{P_i, T_i; F_i, D_i, C_i, I_i, O_i, \Gamma_i, Q_i, M_0^i, M_0^{i-1}\right\}$$

1. P_i : a set of places for inputs and outputs of subtasks.
2. $T_i = J_t \cup J_c$, where J_t : a set of transferring subtasks; J_c : a set of computing subtasks.
3. $D_i = \{d_i \mid 1 \le i \le 5\}$, where d_1 : remote data from outside; d_2 : local data from outside; d_3 : remote data from inside; d_4 : local data from inside; d_5 : computing resources.
4. Γ_i : a time set of subtasks, $\forall t_2^i \in T_i(\Gamma_l(t_2^i)+\Gamma_d(t_2^i) \le \Gamma_l(t_0^1)+\Gamma_d(t_0^1))$.
5. Q_i : a cost set of subtasks, $\sum_{i=1}^{m} Q_m(t_2^i) = Q_m(t_0^1)$, where m is the number of transitions in T_i .

According to the number of data replicas or computing resources, a task can be divided into many subtasks. Based on the types of tasks, the division rules are defined as follows.
1. tasks transferring outer data
Outer data may have many replicas in a parallel environment, so a task transferring outer data can transfer parts of data from different replicas firstly in order to reduce the total transferring time. Then, these parts of data are merged into one by a merging subtask. Though a merging subtask is a computing subtask here, we do not allocate computing resource for it specially and it runs on the node that the data lies on. The detailed division is shown as Fig.4-7. t_2^i is a task transferring outer data. $t_3^{i,r}(1 \le r \le k)$ denotes the transferring subtasks running in parallel. t_3^i is the merging subtask. $R_k(d_1^j)$ denotes the task uses k replicas of d_1^j , and $R_k(d_1^j) = \{R_k^i(d_1^j) \mid 0 \le i \le k\}$. $E_k(b_m^j)$ denotes data b_m^j is divided into k pieces and $E_k(b_m^j) = \{E_k^i(b_m^j) \mid 0 \le i \le k\}$. The division satisfies these conditions:

$$\sum_{r=1}^{k} Q_r(t_3^{i,r}) + Q_r(t_3^i) \le Q_m(t_2^i)$$

$$\exists h(1 \le h \le k \wedge \Gamma_b(t_3^{i,h}) \le \Gamma_l(t_2^i))$$

$$\forall h(1 \le h \le k \rightarrow \Gamma_b(t_3^{i,h}) \ge \Gamma_b(t_2^i))$$

$$\Gamma_b(t_3^i) \le \max(\Gamma_l(t_3^{i,1})+\Gamma_d(t_3^{i,1}),\cdots,\Gamma_l(t_3^{i,k})+\Gamma_d(t_3^{i,k})) \le \Gamma_l(t_3^i) \quad \Gamma_l(t_3^i)+\Gamma_d(t_3^i) \le \Gamma_l(t_2^i)+\Gamma_d(t_2^i)$$

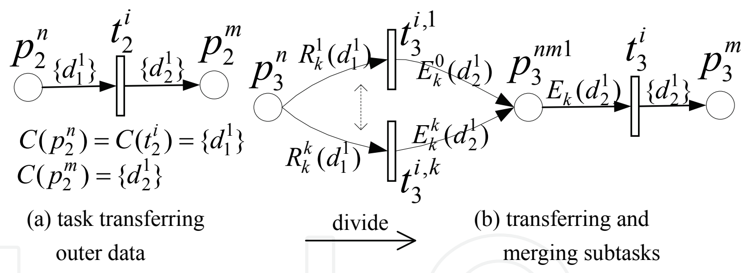
$C(p_2^n) = C(t_2^i) = \{d_1^1\}$
$C(p_2^m) = \{d_2^1\}$
(a) task transferring outer data
divide
(b) transferring and merging subtasks

Fig. 4.7. Division result of task transferring outer data

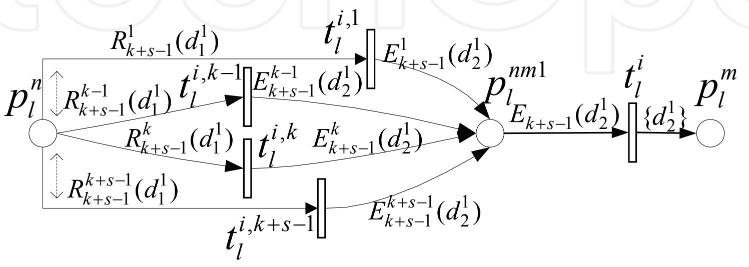

Fig. 4.8. Structure change of subtask net when subtasks transferring outer data become abnormal

When a subtask becomes abnormal because the data resources it uses are out of service or their performances decrease, this subtask needs to be reallocated in order to ensure it can be accomplished on time. The reallocation can lead to the structure change of the subtask net. The detail is shown as Fig.4-8. The number of subtasks that the abnormal subtask is divided into is s. The division accords with these conditions:

$$\sum_{r=1}^{k+s-1} Q_r(t_l^{i,r}) + Q_r(t_l^i) \le Q_m(t_2^i)$$

$$\exists h (1 \le h \le (k+s-1) \wedge \Gamma_b(t_l^{i,h}) \le \Gamma_l(t_2^i))$$

$$\forall h (1 \le h \le (k+s-1) \rightarrow \Gamma_b(t_l^{i,h}) \ge \Gamma_b(t_2^i))$$

$$\Gamma_l(t_3^i) \le \max(\Gamma_l(t_l^{i,1}) + \Gamma_d(t_l^{i,1}), \cdots, \Gamma_l(t_l^{i,k}) + \Gamma_d(t_l^{i,k})) \le \Gamma_l(t_3^i) \qquad \Gamma_l(t_3^i) + \Gamma_d(t_3^i) \le \Gamma_l(t_2^i) + \Gamma_d(t_2^i)$$

2. tasks transferring inner data

Inner data is produced by computing subtasks and it has no replicas. Therefore, a task transferring inner data has only one subtask. When its resource is out of service, the task fails because there are no other resources to use. Its division result is shown as Fig.4-9.

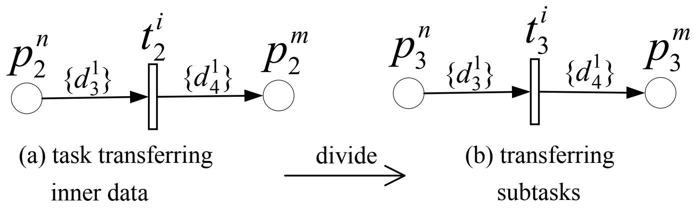

Fig. 4.9. Division result of task transferring inner data

3. parallel computing tasks

There are two kinds of parallel computing tasks: tasks whose data inputs can be divided and tasks whose data inputs can not be divided. According to the number of computing resources and QoS requirements, the former can be divided into many subtasks that only compute parts of the input data. Because these subtasks run on different nodes, the input data needs to be transferred into local node firstly. The detailed process is shown as Fig.4-10. t_2^i is a parallel computing task. $t_3^{i,r,h}(1 \le r \le s, 1 \le h \le k)$ are the transferring subtasks running in parallel. $t_3^{i,r}(1 \le r \le s)$ are the computing subtasks running in parallel. t_3^i is a merging subtask. The division accords with these conditions:

$$\sum_{r=1}^{s} Q_r(t_3^{i,r}) + \sum_{r=1}^{s}\sum_{h=1}^{k} Q_r(t_3^{i,r,h}) + Q_r(t_3^i) \le Q_m(t_2^i)$$

$$\exists h,u(1 \le h \le s \wedge 1 \le u \le k \wedge \Gamma_b(t_3^{i,h,u}) \le \Gamma_l(t_2^i))$$

$$\forall h,u(1 \le h \le s \wedge 1 \le u \le k \rightarrow \Gamma_b(t_3^{i,h,u}) \ge \Gamma_b(t_2^i))$$

$$\Gamma_b(t_3^i) \le \max(\Gamma_l(t_3^{i,1}) + \Gamma_d(t_3^{i,1}), \cdots, \Gamma_l(t_3^{i,s}) + \Gamma_d(t_3^{i,s})) \le \Gamma_l(t_3^i)$$

$$\Gamma_l(t_3^i) + \Gamma_d(t_3^i) \le \Gamma_l(t_2^i) + \Gamma_d(t_2^i)$$

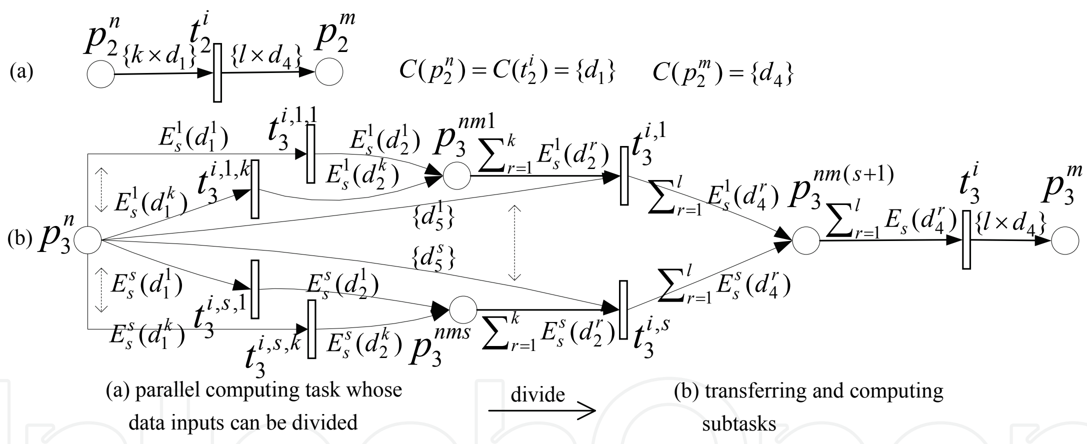

Fig. 4.10. Division result of parallel computing task whose data inputs can be divided

When one subtask of a parallel computing task becomes abnormal because the computing resource it uses is out of service or its performance decreases, this subtask needs to be reallocated in order to ensure it can be accomplished on time. The reallocation can lead to the structure change of the subtask net. The detail is shown as Fig.4-11. s is the number of subtasks that the abnormal subtask is divided into according to the number of computing resources and QoS requirements. The division accords with these conditions:

$$\sum_{r=1}^{v+s-1}\sum_{h=1}^{k} Q_r(t_z^{i,r,h}) + \sum_{r=1}^{s+v-1} Q_r(t_z^{i,r}) + Q_r(t_z^i) \le Q_m(t_2^i)$$

$$\exists h,u(1 \le h \le (s+v-1) \wedge 1 \le u \le k \wedge \Gamma_b(t_z^{i,h,u}) \le \Gamma_l(t_2^i))$$

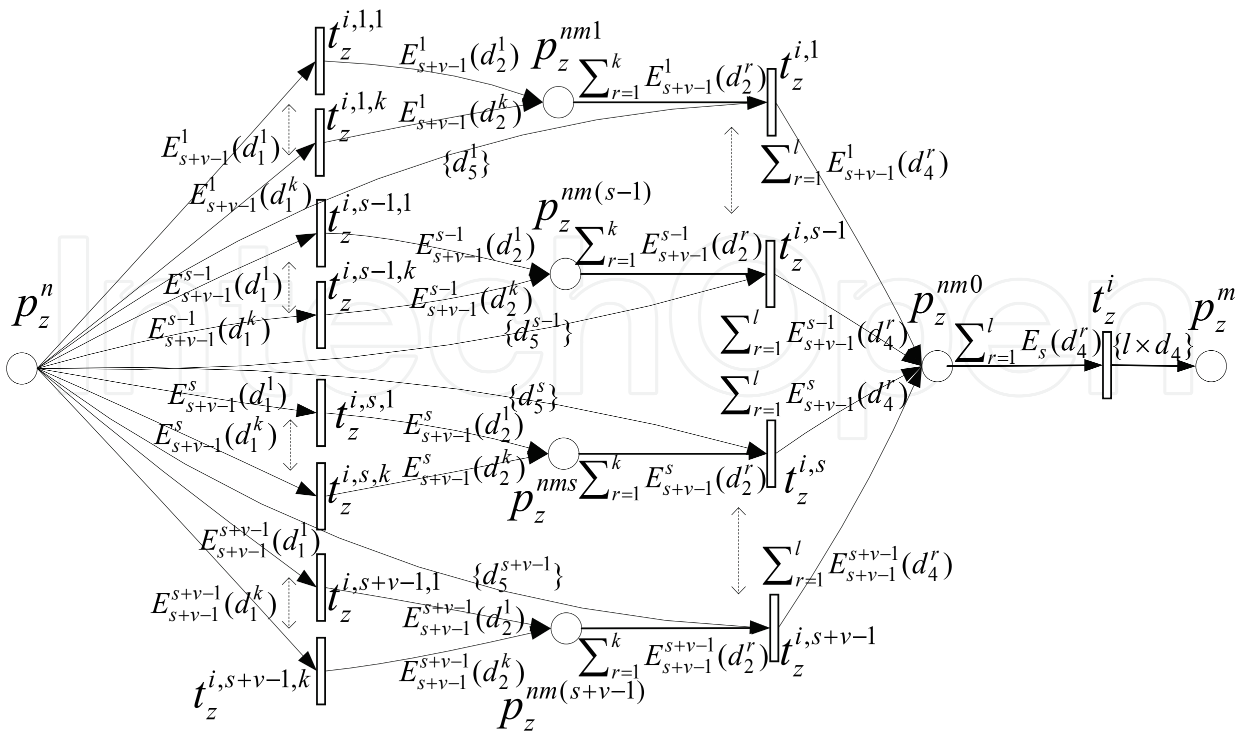

Fig. 4.11. Structure change of the subtask net when parallel computing subtasks become abnormal

$$\exists h,u(1 \le h \le (s+v-1) \wedge 1 \le u \le k \wedge \Gamma_b(t_z^{i,h,u}) \ge \Gamma_b(t_2^i))$$

$$\forall h(1 \le h \le (s+v-1) \rightarrow \Gamma_b(t_z^{i,h}) \le \max(\Gamma_l(t_z^{i,h,1}) + \Gamma_d(t_z^{i,h,1}),\cdots,\Gamma_l(t_z^{i,h,k}) + \Gamma_d(t_z^{i,h,k})) \le \Gamma_l(t_z^{i,h}))$$

$$\Gamma_b(t_z^i) \le \max(\Gamma_l(t_z^{i,1}) + \Gamma_d(t_z^{i,1}),\cdots,\Gamma_l(t_z^{i,s}) + \Gamma_d(t_z^{i,s})) \le \Gamma_l(t_z^i)$$

$$\Gamma_l(t_z^i) + \Gamma_d(t_z^i) \le \Gamma_l(t_2^i) + \Gamma_d(t_2^i)$$

The division of a parallel computing task whose data inputs can not be divided is similar to a task whose data inputs can be divided. The difference between them is that the input data of transferring subtask is the data or parts of it.

4. computing tasks that cannot run in parallel

A computing task that can not run in parallel only has a subtask and Fig.4-12 shows the division result.

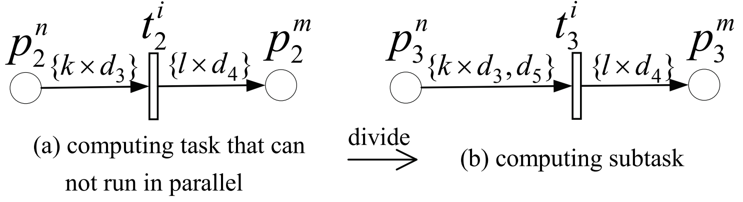

Fig. 4.12. Division result of computing task that can not run in parallel

Within an unparallel computing task, when a subtask becomes abnormal because the computing resource it uses is out of service or its performance decreases, this subtask needs to be reallocated in order to ensure it can be accomplished on time. For unparallel computing tasks, the reallocation can not lead to the structure change of the subtask net, but it can result in the states change. The detail is shown as Fig.4-13. Because the computing node is replaced, the data needs to be transferred to a new computing node. t_z^i is the new computing subtask and $t_z^{i,r}(1 \le r \le k)$ denotes transferring subtasks. The change accords with these conditions:

$$\sum_{r=1}^{k} Q_r(t_z^{i,r}) + Q_r(t_z^i) \le Q_m(t_2^i)$$

$$\exists h(1 \le h \le k \wedge \Gamma_b(t_3^{i,h}) \le \Gamma_l(t_2^i))$$

$$\forall h(1 \le h \le k \rightarrow \Gamma_b(t_3^{i,h}) \ge \Gamma_b(t_2^i))$$

$$\Gamma_b(t_z^i) \le \max(\Gamma_l(t_z^{i,1}) + \Gamma_d(t_z^{i,1}), \cdots, \Gamma_l(t_z^{i,k}) + \Gamma_d(t_z^{i,k})) \le \Gamma_l(t_z^i)$$

$$\Gamma_l(t_z^i) + \Gamma_d(t_z^i) \le \Gamma_l(t_2^i) + \Gamma_d(t_2^i)$$

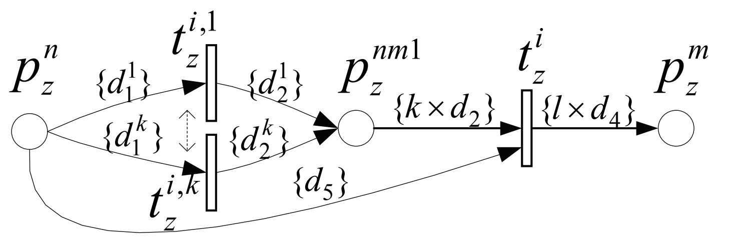

Fig. 4.13. Structure change of subtask net when computing subtasks that can not run in parallel become abnormal

5. Analysis and optimization

In this section, we adjust the structure of subtask net firstly in order to optimize the process of subtasks. Then, we analyze the validity of the scheduling net and the job net. Finally, we analyze the performance of the job net.

5.1 Structure optimization

In order to keep the consistency of the model and make the process of structure change clear and intuitive, we divide parallel computing tasks in standard way. However, this way results in redundant data transfer within a subjob and this part of subtask net need to be optimized further.

Suppose the number of remote data inputs in a computing subjob t_1^i is k. t_1^i has remote data inputs $\{d_1^r \mid 1 \le r \le k\}$ and a computing task t_2^i whose subtasks are $t_3^{i,r}(1 \le r \le s)$. h_a^b is the number of replicas of number b data that number a subtask uses. Within a subjob, the optimization result is shown as Fig.5-1. Compared with the former, the total number of reduced transitions is k, the saved money is $\sum_{r=1}^{k} Q_r(t_2^{i,r})$ and the reduced time is $\max(\Gamma_d(t_2^{i,1}),\cdots,\Gamma_d(t_2^{i,k}))$.

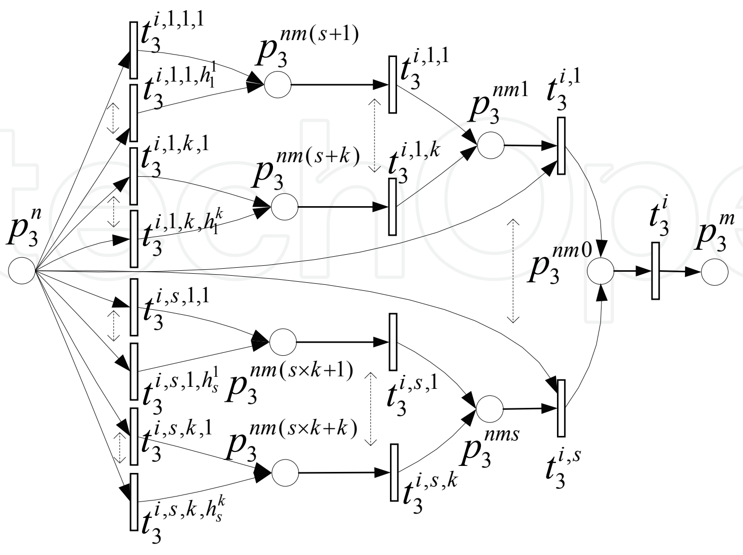

Fig. 5.1. Structure optimization of parallel computing subtasks

5.2 Validity analysis

Validity analysis is necessary for a model based on Petri Net to ensure the success of model in practice. For the scheduling net, we analyze its structure to verify its correctness. For the job net, besides structure analysis we also need to analyze its time reachability to validate that the time limits of transitions are reasonable.

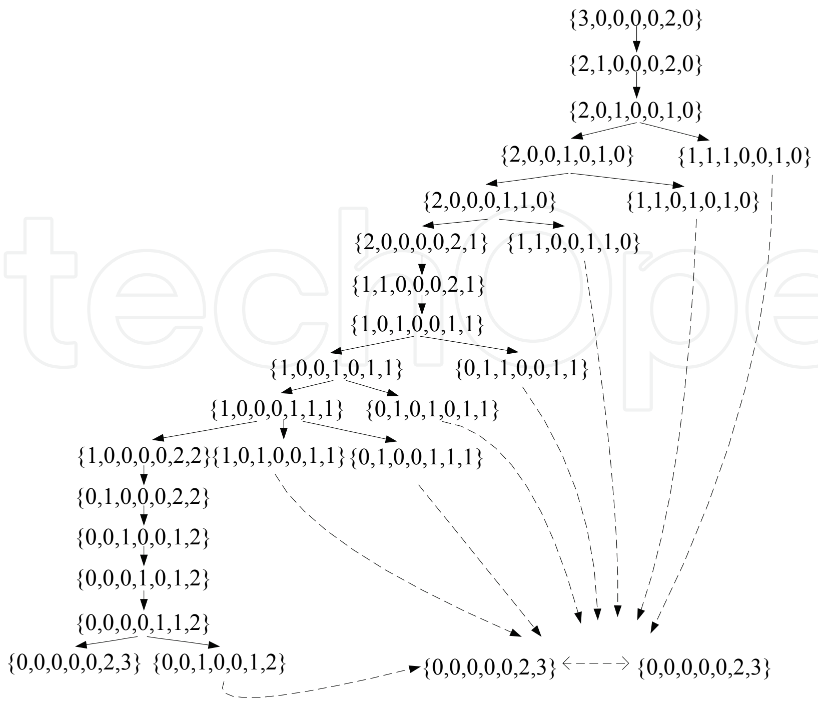

Fig. 5.2. A sample of reachability trees

The top level of scheduling net is a job scheduling net, which consists of circular structures and cannot stop without outside force. The end place and other places in the job scheduling net can have tokens at the same time. The maximal number of jobs that the job scheduling net can schedule simultaneously is $K(p_3)$ and these jobs are classified by the net according to their states.

The three lower levels of the scheduling net are workflow nets, which are driven by the job scheduling net and only schedule one subjob, one task or one subtask at the same time. The job net is also a workflow net and its validity [16] is described as follows:

1. For each state M reachable from state i, there exists a firing sequence leading from state M to state o.
2. State o is the only state reachable from state i with at least one token in place o.
3. There are no dead transitions in net.

The main work in structure analysis is reachability analysis. We can build reachability trees [17, 18] for the scheduling net and the job net to validate their reachability and three conditions above. There are too many reachability trees, so we only list a sample of them here. Fig.5-2 shows a sample of reachability tree built according to a job scheduling net, which has 3 jobs in the beginning and the maximal number of running jobs that scheduler can deal with simultaneity is 2. Therefore, M_0={3,0,0,0,0,2,0} and the end state is {0,0,0,0,0,2,3}. For a job scheduling net with M_0={m,0,0,0,0,n,0} , its reachability tree is similar to Fig.4-15. The root of this tree is {m,0,0,0,0,n,0} and all leaves are {0,0,0,0,0,n,m}. The number of tokens in the tree satisfies these conditions:

$$|C(p_1)_{ms}|+|C(p_2)_{ms}|+|C(p_3)_{ms}|+|C(p_4)_{ms}|+|C(p_5)_{ms}|+|C(p_7)_{ms}|= m$$

$$|C(p_6)_{ms}|+|C(p_3)_{ms}|+|C(p_4)_{ms}|+|C(p_5)_{ms}|= n$$

$$0 \le |C(p_2)_{ms}|, |C(p_4)_{ms}|, |C(p_5)_{ms}| \le 1$$

$$|C(p_3)_{ms}|, |C(p_6)_{ms}| \le n$$

Time reachability is that the time requirements of transitions are satisfied within time limitations. After analyzing the validity of structure, we can validate the time reachability easily. In the job net, if $\forall t \in T(\forall t' \in {}^{\otimes\otimes}t \rightarrow \Gamma_l(t')+\Gamma_d(t') \le \Gamma_l(t)+\Gamma_d(t))$, the time reachability of the net is satisfied. Otherwise, the time reachability is not satisfied.

Validity analysis is necessary for a model based on Petri Net to ensure the success of model in practice. For the scheduling net, we analyze its structure to verify its correctness. For the job net, besides structure analysis we also need to analyze its time reachability to validate that the time limits of transitions are reasonable.

The top level of scheduling net is a job scheduling net, which consists of circular structures and cannot stop without outside force. The end place and other places in the job scheduling net can have tokens at the same time. The maximal number of jobs that the job scheduling net can schedule simultaneously is $K(p_3)$ and these jobs are classified by the net according to their states.

The three lower levels of the scheduling net are workflow nets, which are driven by the job scheduling net and only schedule one subjob, one task or one subtask at the same time. The job net is also a workflow net and its validity [16] is described as follows:

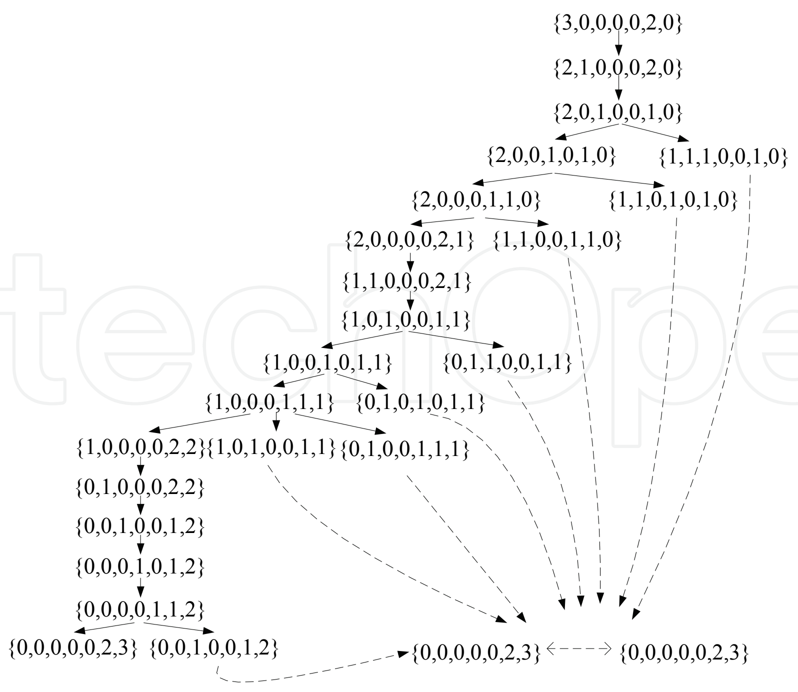

Fig. 5.3. A sample of reachability trees

1. For each state *M* reachable from state *i* , there exists a firing sequence leading from state *M* to state *o.*
2. State *o* is the only state reachable from state *i* with at least one token in place *o*.
3. There are no dead transitions in net.

The main work in structure analysis is reachability analysis. We can build reachability trees [17, 18] for the scheduling net and the job net to validate their reachability and three conditions above. There are too many reachability trees, so we only list a sample of them here. Fig.4-15 shows a sample of reachability tree built according to a job scheduling net, which has 3 jobs in the beginning and the maximal number of running jobs that scheduler can deal with simultaneity is 2. Therefore, M_0={3,0,0,0,0,2,0} and the end state is {0,0,0,0,0,2,3} . For a job scheduling net with M_0={m,0,0,0,0,n,0} , its reachability tree is similar to Fig.5-3. The root of this tree is {m,0,0,0,0,n,0} and all leaves are {0,0,0,0,0,n,m} . The number of tokens in the tree satisfies these conditions:

$$|C(p_1)_{ms}|+|C(p_2)_{ms}|+|C(p_3)_{ms}|+|C(p_4)_{ms}|+|C(p_5)_{ms}|+|C(p_7)_{ms}|= m$$

$$|C(p_6)_{ms}|+|C(p_3)_{ms}|+|C(p_4)_{ms}|+|C(p_5)_{ms}|= n$$

$$0 \le |C(p_2)_{ms}|, |C(p_4)_{ms}|, |C(p_5)_{ms}| \le 1$$

$$|C(p_3)_{ms}|, |C(p_6)_{ms}| \le n$$

Time reachability is that the time requirements of transitions are satisfied within time limitations. After analyzing the validity of structure, we can validate the time reachability easily. In the job net, if $\forall t \in T(\forall t' \in {}^{\otimes\otimes}t \to \Gamma_l(t') + \Gamma_d(t') \le \Gamma_l(t) + \Gamma_d(t))$, the time reachability of the net is satisfied. Otherwise, the time reachability is not satisfied.

5.3 Validity analysis

Performance analysis mainly analyzes the time and cost characteristics of a job net. The total cost of a job net is the cost sum of all transitions in the net: $\sum Q_r(t)$. If $\sum Q_r(t) = Q_r(t_0^1) \le Q_m(t_0^1)$, the cost allocation succeeds, otherwise the cost allocation fails and the job net can not run correctly.

In order to be convenient to analyze time characteristics of a job net, we propose a transition tree algorithm that translates the transitions in a job net into a transition tree. The conversion rules are shown as follows:

1. The root of a transition tree is t_{root} whose time limits and cost are 0;
2. p_{begin} is the beginning place in the job net and all transitions in $p_{begin}^{\otimes}$ are the leaves of t_{root}.
3. For each leaf t, find $t^{\otimes\otimes}$ and all transitions in $t^{\otimes\otimes}$ are the leaves of t;
4. repeat step 3 until each leaf t satisfies the condition: $t^{\otimes\otimes} = \varnothing$.
5. The tree with root t_{root} is the corresponding transition tree of the job net.

According to the transition tree, it is convenient to analyze the time characteristics of the job net and optimize the allocation process for subtasks.

The maximum number of serial transitions is $D_T - 1$, where D_T is the depth of the transition tree.

To reduce the waiting time of transitions, for each transition $t({}^{\otimes\otimes}t = \{t_i \mid 1 \le i \le k\})$ in a job net, let $\Gamma_l(t) = \Gamma_b(t) = \max(\Gamma_l(t_1) + \Gamma_d(t_1), \cdots, \Gamma_l(t_k) + \Gamma_d(t_k))$.

Suppose there are s leaves $\{t'_r \mid 1 \le r \le s\}$ in a transition tree and each leaf has a path from it to root : $p(t'_i) = \{t_i^r \mid 1 \le r \le l_i\}$,where l_i is the number of transitions in $p(t'_i)$. Each path has a total time of transitions: $\Gamma(t'_i) = \sum_{r=1}^{l_i} \Gamma_d(t_i^r)$, the total durable time of the net is $\Gamma(t'_r)$, $\Gamma(t'_r) = \max(\Gamma(t'_1), \cdots, \Gamma(t'_k))$. The corresponding path of t'_r is the key path and l_r is the number of transitions on the key path.

The key path decides the total durable time of a job net and it is important for subtask allocation optimization. For each subtask, we should choose those resources with high performance. All computing subtask on the key path can run on the same node and this can reduce the data transferring time.

6. Conclusions and future work

This paper proposes two different models for the scheduling net and the job net based on the idea that the scheduling net is separated from the job net. This method makes models compact and intuitional. In addition, the separation benefits the analysis of the job net and the scheduling net respectively. According to the granularity of parallel applications, the scheduling net is designed to four levels, which is convenient to deploy distributed schedulers in parallel environment and is beneficial to the management of different parallel application granularities. Based on Petri Net with changeable structure, the job net model can change its structure dynamically according to the allocation results or states of jobs. Therefore, the model supports dynamic mergence and division of subtasks and can deal with the abnormity of subtasks. We validate the scheduling net and the job net using

reachability tree technologies. In addition, a transition tree algorithm is designed for analyzing the performances of the job net and optimizing the allocations of subtasks according to the key path in the job net.
In the future, we will optimize these models further and put emphasis upon researching algorithms used for optimizing resource allocations.

7. References

[1] M. Alt, S. Gorlatch, A. Hoheisel, H. W. Pohl, "Using High-Level Petri Nets for Hierarchical Grid Workflows," presented at Second IEEE International Conference on e-Science and Grid Computing, Amsterdam, The Netherlands 2006, pp. 13-13.
[2] M. Silva,L. Recalde, "Petri nets and integrality relaxations: A view of continuous Petri net models," *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, vol. 32, pp. 314-327, 2002 (4).
[3] H. Yaojun,L. Xuemei, "Modeling and Performance Analysis of Grid Task Scheduling Based on Composition and Reduction of Petri Nets," 2006, pp. 331-334.
[4] S. Distefano, A. Puliafito, M. Scarpa, "GridSPN: a grid-based non Markovian Petri nets tool," presented at IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise, 2005, pp. 331-336.
[5] X. Jiefeng, W. Zhaohui, C. Huajun, "Distributed Petri Net for Knowledge Base Grid reasoning," presented at IEEE International Conference on Systems, Man and Cybernetics, Hangzhou, China, 2003, pp. 593-597 vol.1.
[6] J. Yu, R. Buyya, C. K. Tham, "QoS-based Scheduling of Workflow Applications on Service Grids," presented at Proceedings of the 1st IEEE International Conference on e-Science and Grid Computing Melbourne, Australia, 2005.
[7] J. Yu,R. Buyya, "A Taxonomy of Scientific Workflow Systems for Grid Computing," *Special Issue on Scientific Workflows, SIGMOD Record*, vol. 34, pp. 44-49, 2005 (3).
[8] J. Yu,R. Buyya, "A Budget Constrained Scheduling of Workflow Applications on Utility Grids using Genetic Algorithms," presented at Workshop on Workflows in Support of Large-Scale Science, Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing Paris, France, 2006.
[9] BPEL4WS, "http://www.ebpml.org/bpel4ws.htm,"
[10] J. Hai,W. Shuzhen, "Grid workflow model based on colored Petri net," *J . Huazhong Univ. of Sci. & Tech. (Nature Science Edition)*, vol. 34, pp. 39-41, 2006 (7).
[11] Z. Hu, R. Hu, W. Gui, J. Chen, "General scheduling framework in computational Grid based on Petri net " *Journal of Central South University of Technology*, vol. 12, pp. 232-237, 2005
[12] H. Yaojun, J. Changjun, L. Xuemei, "Resource scheduling model for grid computing based on sharing synthesis of Petri net," 2005, pp. 367-372 Vol. 1.
[13] M. Dobber, R. van der Mei, G. Koole, "Effective Prediction of Job Processing Times in a Large-Scale Grid Environment," presented at 15th IEEE International Symposium on High Performance Distributed Computing, 2006, pp. 359-360.
[14] Z. Yuanyuan, S. Wei, Y. Inoguchi, "Predicting Running Time of Grid Tasks based on CPU Load Predictions," presented at 7th IEEE/ACM International Conference on Grid Computing, Barcelona, Spain, 2006, pp. 286-292.

[15] C. Jinjun,Y. Yun, "Assigning Local Fixed-time Constraints in Grid Workflow Systems," presented at Fifth International Conference on Grid and Cooperative Computing Workshops. WSGE '06. , Changsha, China, 2006, pp. 227-234.
[16] Z. Liang, "Research on Workflow Patterns based on Petri nets," presented at 2006 IEEE Conference on Robotics, Automation and Mechatronics, Bangkok, 2006, pp. 1-6.
[17] J. Mu Der,P. Mao Yu, "Augmented reachability trees for 1-place-unbounded generalized Petri nets," *IEEE Transactions on Systems, Man and Cybernetics, Part A*, vol. 29, pp. 173-183, 1999 (2).
[18] Y. Ru, W. Wu, C. N. Hadjicostis, "Comments on "A Modified Reachability Tree Approach to Analysis of Unbounded Petri Nets"," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 36, pp. 1210-1210, 2006 (5).